

# Table of Contents

## Introduction

Statement of Purpose

Disclaimer

Confidentiality

Overall Goals for Big Bang

Challenges for Big Bang

The Challenge to Provide a Minimum Number of Customer Releases

The Challenge to Support and Enhance New Hardware

The Software Integration Challenge

The Challenge to Support and Enhance 1 Megabyte Macintoshes

Illustrations

Macintosh Genealogy

Decision Tree

Big Bang Key Features

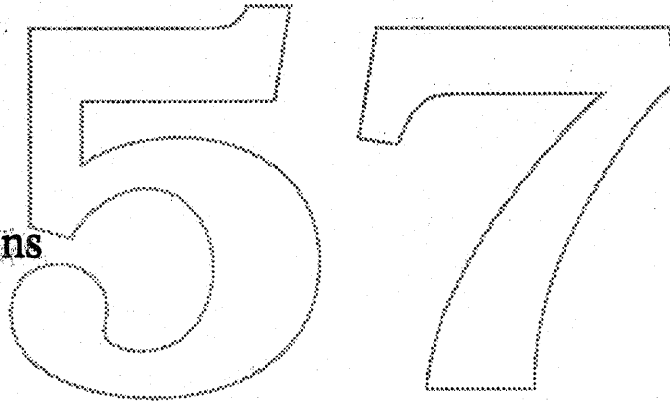
Operating System

Imaging

Desktop Services

System Integration

Where we will be



## Project Descriptions

## Schedules

## ERSs

### A. Graphics

32-Bit Quickdraw

Skia

Monitors CDev

Video Software Madness - The Horror Continues

Harpo/Topanga Video Software Support

Video Configuration ROM Software Specification

\*Video Configuration ROM Software Specification - Delta Guide

### B. Fonts

Bass 1.0 (Intelligent Font Sealing Technology for Macintosh) External Reference Specification v. 1.2

Bass System Software Interface (sub-ers)

The Bass Project Font Instruction Set and Interpreter Sub ERS

\* Denotes change since last release of table of contents.

## C. Printing

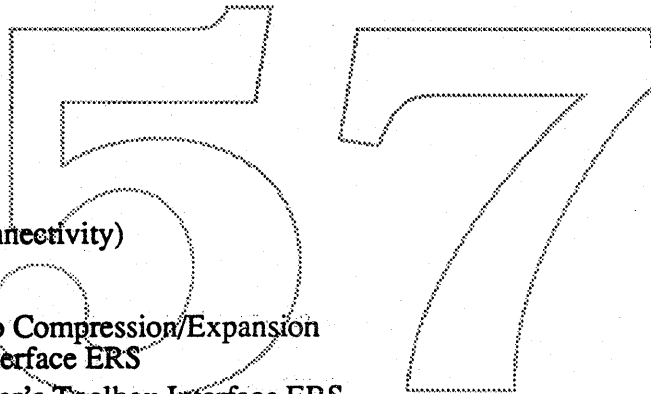
LasweWriter 6.0 ERS Version 0.2  
Ginsu Project ERS Version 0.1  
Ginsu User Interface ERS Version 0.3  
The Ginsu Architecture Just-Poured-Concrete Version  
Imaging Engine, Ginsu Project Proposal Version 1.0d8  
Ginsu Application Interface January 13, 1989

## D. International

Boffin (a.k.a. Layout Manager)  
Everest (a.k.a. Language Manager)  
Script Manager 3.0  
TextEdit 3.0  
InfoWorld (a.k.a. Multiscript System)  
Script System Core  
Unicodes/Uniform Character Codes

## E. Toolbox

Toolbox Overview  
Diet Coke  
Glass Plus  
Snarfman (a.k.a. SQL Connectivity)  
Esperanto (a.k.a. DDIF)  
Sound  
MACE (Macintosh Audio Compression/Expansion)  
Help Manager Human Interface ERS  
Help Manager Programmer's Toolbox-Interface ERS



## F. Finder & Applications

Finder Overview  
Multi-Disk Installer  
Furnishings 2000/New Finder  
The Blue Interface Group - New Chooser Design  
MacroMaker 1.1  
Esprit Support  
PictWhap  
Pat  
CyberBash  
FontUtility 2.0  
CloseView 1.1

\* Denotes change since last release of table of contents.

## G. CPUs & ROMs

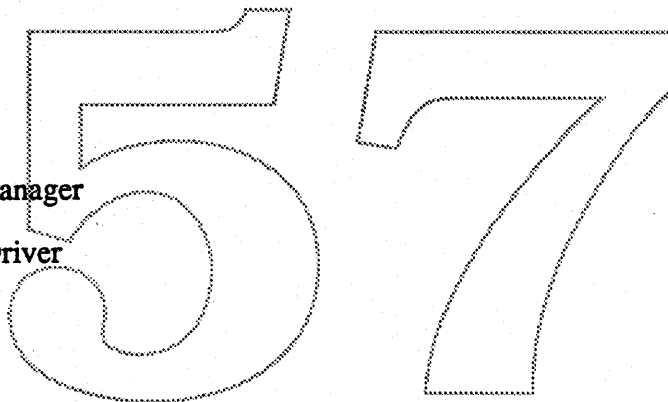
- Esprit ROM
- Cobra II/Spin ROM
- Jaws (a.k.a. Universal ROM)
- FourSquare/F19 ROM
- Remote Booting
- Gestalt

## H. OS Kernel

- File Systems Management
- MS-DOS External File System
- HFS Enhancements
- \*MacVM, Virtual Memory for Macintosh
- MultiFinder 6.1
- \*MultiFinder 7.0
  - Interprocess Communication Facility
- System Segmentation
- Desktop Manager

## I. I/O

- Asynchronous SCSI Manager
- Time Manager
- HDSC Setup/HDSC Driver
- SCSI Bus Rover
- Slot Manager
- IOP Manager
- IOP SWIM Driver
- IOP ADB Driver
- E-Disk Driver
- DebugUtil
- Device Manager
- Serial Driver



## J. Networking & Communications

- \*AppleShare 1989 - The Holy Grail
- 976 (Remote AppleTalk Network)
- Calvin and Hobbes
- AppleTalk 2.0
- Serial IOP ERS

\* Denotes change since last release of table of contents.

## \*Product Plan

### Notes

Dependencies

\* Denotes change since last release of table of contents.

57

# The Blue Book

1/19/89

by: Sheila Brady  
Amy Rapport  
Maura McNamara  
The Blue Group  
Charlie Oppenheimer

## Statement of purpose

The Blue Book is a collection of documents that describe the Blue Macintosh System Software strategy and the contents of "Big Bang". Big Bang is the System Software release slated to ship in October of 1989. In addition to Big Bang there are features discussed which will ship shortly before or after October. Much of this introductory section has been gleaned from Charlie Oppenheimer's Blue System Software Product Plan. The complete Product Plan, which goes into depth on these and other issues, can be found in the back of this book.

## Disclaimer

Big Bang is in the development phase and the contents of the documents you will be reading will be updated as the work progresses. Please digest this information with an understanding that these are evolving documents. Your input can be very valuable to the project so be sure and chime in!

## Confidentiality

The entire contents of this book is confidential. Please do not copy or discuss the contents with anyone outside of Apple. If there are Apple employees who would profit from this information, call Helen Miller at extension 4-6718 for a copy.

## Overall Goals for Big Bang

The Big Bang release is an evolutionary release. Our goals are to:

- Answer developers' needs with more functionality
- Build an EVEN MORE user friendly system
- Improve performance and efficiency of our system for the customer base in general
- Continue to support the full Macintosh line (1 megabyte, floppy-only systems and up)
- Maintain compatibility for the existing applications base by
  - Adding functionality invisibly and painlessly where possible
  - Requiring application rewrite only to take advantage of new features
  - Avoiding changing the programming model

With Big Bang we hope to continue to provide a friendly and responsive system software package for our existing customer base. We are stretching the existing boundaries of Macintosh System Software with incremental additions for more powerful internal integration (New Finder, Ginsu). At the same time we are positioning the Mac to leverage off of the capabilities of the outside world through the addition of SQL API, the Language Manager, IPC, etc. We are laying the foundation for Pink's arrival with coordinated Blue/Pink efforts in Printing, the Finder, and International. Last but not least, we are attempting to complement the new CPUs with features such as Extended Memory. We are attempting to add more functionality without unreasonably burdening the smaller systems.

## Challenges for Big Bang

There are a number sticky problems to reconcile in the Big Bang release.

- Provide a minimum set of customer releases
- Support, and enhancement of new Hardware
- Integration of Software features to provide the most holistic product
- Support for the 1Mb world

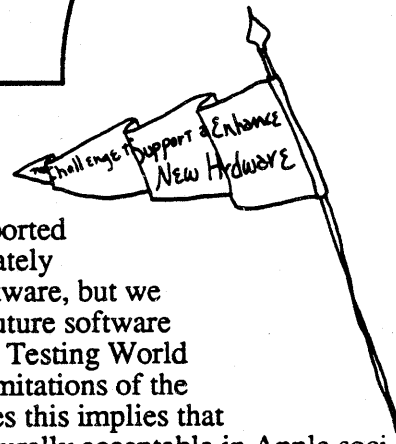
### The Challenge to Provide a Minimum Number of Customer Releases

Macintosh System Software has been requested to not put out too many releases every year. Extra releases cause havoc with the customer base, and in the field. This puts us in a challenging position, because most new pieces of hardware need complementary pieces of system software. We also want to minimize the dependency that other projects (CPUs in particular) have upon us, so that we can be driven in our designs by the best software solution, and not by the best market window for a particular CPU. Our tactic so far to deal with this problem is to build Hardware-only support releases that are publicized as releases that are only required to support the new hardware. These are releases that still need the full bank of tests run, yet those that will not require major bug fixing revisions to all the pieces of the software. Our testing will then be minimized since major areas of the system should be stable. This should improve turn-around time for those releases. Minimizing the number of releases is a constant challenge as Hardware and Software schedules ebb and flow with the vagaries of engineering and marketing opportunities.



### The Challenge to Support and Enhance New Hardware

A variety of new CPUs are racing (plodding? marching?) to completion now. We must map their likely completion dates against our proposed release timeframes. This is a complicated equation. We must make sure that the H/W is completely supported in the matching system software release. It must also be adequately tested. We have to avoid holding up the hardware with our software, but we also have to continue to add our software functionality to the future software releases. We have to maximize the effort made in the Software Testing World across all CPUs. And we know from past releases about the limitations of the number of disks that can be built in parallel. (none.) Sometimes this implies that the hardware will wait for the software, which is so far not culturally acceptable in Apple society.



Outlined here are the currently known considerations that we have taken into account in resolving our future release strategy.

- 1) It takes 6 months from alpha ROM to shipping with a Golden System Disk.
- 2) The Cobra II ROM should be done earlier than FourSquare and Spin, and will therefore not be a 'Universal' ROM for those machines.
- 3) Cobra II alpha is Feb. 15 '89, and Golden System Disk is August '89.
- 4) Spin will be much later (Jan 1 '90) due to remote booting. (If remote booting is to happen, we

- need to get resources committed now!)
- 5) FourSquare schedule is uncertain, and is hard to plan for at this time.
  - 6) Since FourSquare and Spin and F19 occur in late '89- early '90, they should be targeted for the first 'Universal ROM, and they should contain all Big Bang features. They should ship with the Big Bang system disk.
  - 7) The above conclusions make using a July Antares System Disk for Esprit & Cobra II difficult. We propose delay of Esprit to match the Cobra II schedule so Antares can cover both.

Noting the above conditions, we have two viable approaches for our release strategy over the next 10-12 months. As the hardware and software schedules become more certain, we will select the best course of action.

**Approach #1**

<u>System Disk</u>	<u>CPU's supported</u>	<u>Features</u>	<u>Date</u>
6.0.4	Harpo & Cobra II	H/W support only	July/August '89
7.0		Big Bang	October '89
7.0.1	F-19 and/or 4 square	H/W support	January '90

This approach implies 3 System disks in 1 year. (Maura - 1 disk is in January!)

or..

**Approach #2**

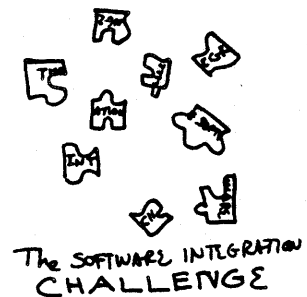
<u>System Disk</u>	<u>CPU's supported</u>	<u>Features</u>	<u>Date</u>
6.1	Harpo & Cobra II	Extended Mem* & H/W	July/August '89
7.0	F-19 and/or 4 square	Big Bang & H/W support	January '90

\*extended memory-not true virtual memory

This proposal constitutes 2 System disks in 6 months. 6.0.4 becomes 6.1 in this scenario due to the inclusion of extended memory

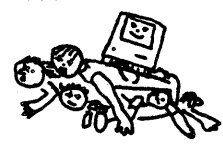
**The Software Integration Challenge**

We are making some drastic and aggressive changes to the internals of the Blue System software. For these changes to work, and for the system to play together in a reasonable manner in it's final release, we need to pay particular attention to integration issues early in the game. We have dependency meetings weekly, and the project leaders and managers are trying to identify and track all the open issues across the groups. We hope to head off some of the problems that would normally appear at the end of the project by carefully thinking through these issues at the beginning. This is still the most difficult technical challenge of this project.



**The Challenge to Support and Enhance 1 Megabyte Macintoshes**

The Big Bang system will likely need two megabytes to operate effectively with medium to large applications like HyperCard, PageMaker, FullWrite, PixelPaint, and so on. One-megabyte system support is required because of the large installed base of one-megabyte systems and because over 50% of systems forecasted for late 1989 sales will still be one-megabyte configurations. In addition, we hope to continue to have one system software platform for all Macintoshes.



A single software platform means that

- our customers don't have to choose which system software to install,
- our developers can write for only one platform (not 1 meg & > 1 meg systems),
- our field service people don't have to support two system software sets in the field,
- our factory doesn't have to build and track two systems,
- our documentation people don't have to document two systems,
- our Networking groups don't have to write two types of networking packages,
- our testing group doesn't have to test two systems,
- our international groups don't have to localize two systems, and
- Software Engineering doesn't have to support two sets of sources.

Unfortunately, though one system for both 1 meg and > 1 meg machines might be desirable, it might also be impossible. With the additional capabilities that we are adding, there may be no way to fit the system into the 256K allocated on the 1 megabyte machine.

We plan to explore all possible methods of providing full support to the 1 meg systems before reducing their set of system software functionality and finally giving up and going to a dual system. The exact set of functions and exact implementation to be delivered to one-megabyte system users is still being discussed. Segmentation is one scheme under investigation that may help the 1 Mb world. This is a key decision that will be shaped as we proceed with development.

The following diagrams illustrate the Macintosh family tree and a look at the bifurcation (two systems) strategy questions. The Macintosh Genealogy chart is a straightforward portrayal of the evolution of the Macintosh family. The next diagram is a yes-no flow chart describing our evaluation of the methods to support 1 megabyte machines.

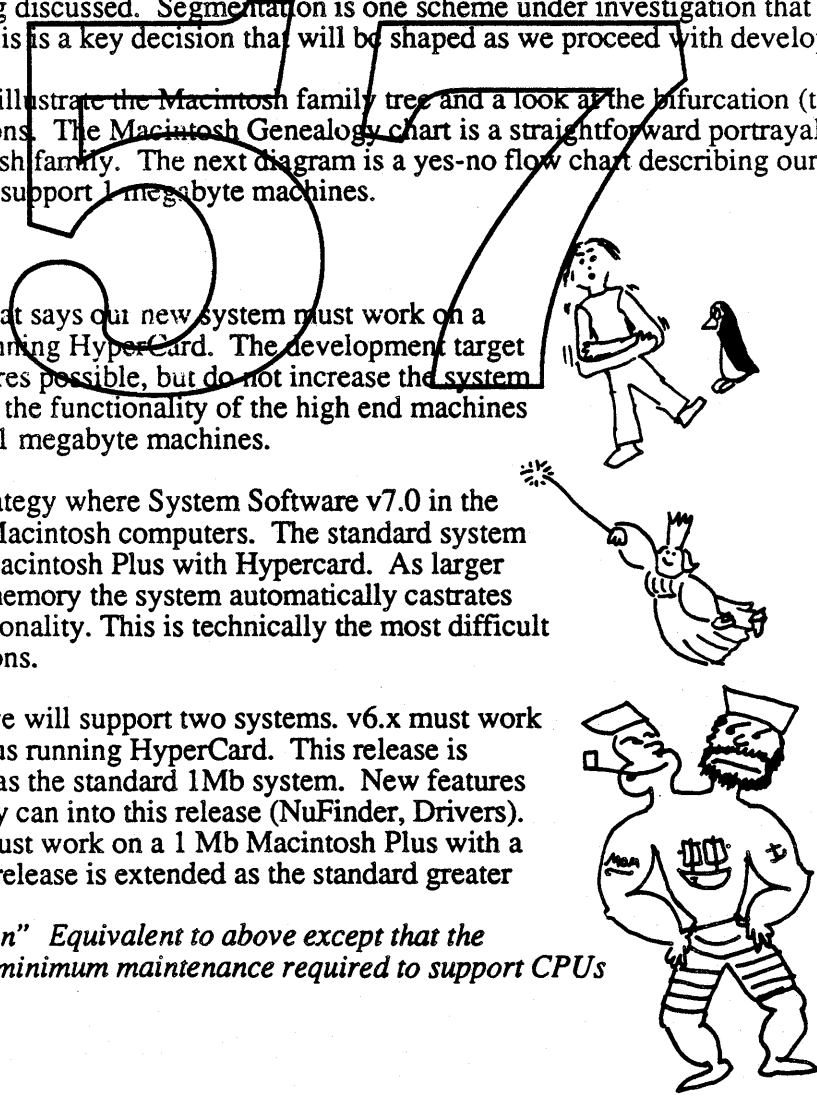
### Legend

1) **Freeze** is a strategy that says our new system must work on a 1 Mb Macintosh Plus running HyperCard. The development target is: Add all the new features possible, but do not increase the system size. This would restrict the functionality of the high end machines by the limitations of the 1 megabyte machines.

2) **Good Fairy** is the strategy where System Software v7.0 in the standard system for all Macintosh computers. The standard system must work on a 1 Mb Macintosh Plus with Hypercard. As larger applications run out of memory the system automatically castrates itself by removing functionality. This is technically the most difficult of all the possible solutions.

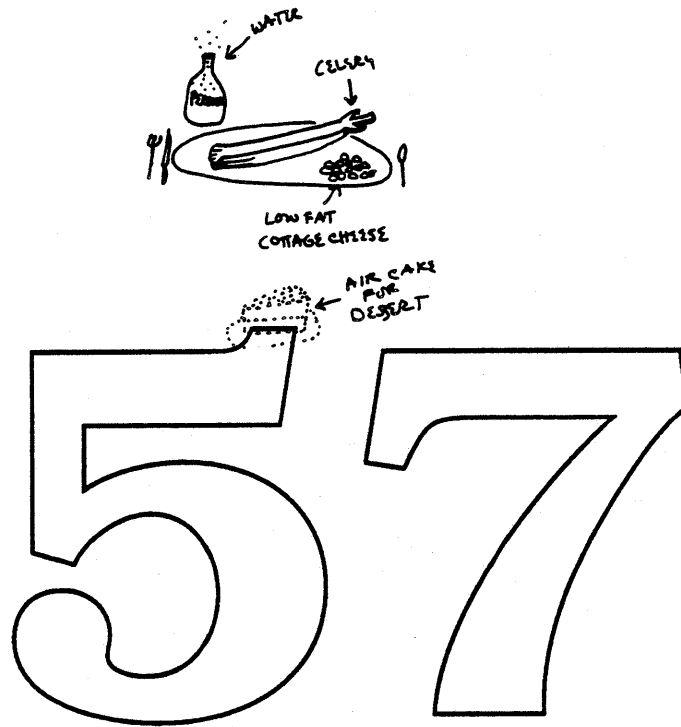
3) **Bifurcation** means we will support two systems. v6.x must work on a 1 Mb Macintosh Plus running HyperCard. This release is maintained and shipped as the standard 1Mb system. New features must be retrofitted if they can into this release (NuFinder, Drivers). System Software v7.0 must work on a 1 Mb Macintosh Plus with a 512K application. This release is extended as the standard greater than 1Mb system.

*Option: "Diet Bifurcation" Equivalent to above except that the v6.x release exists with minimum maintenance required to support CPUs*

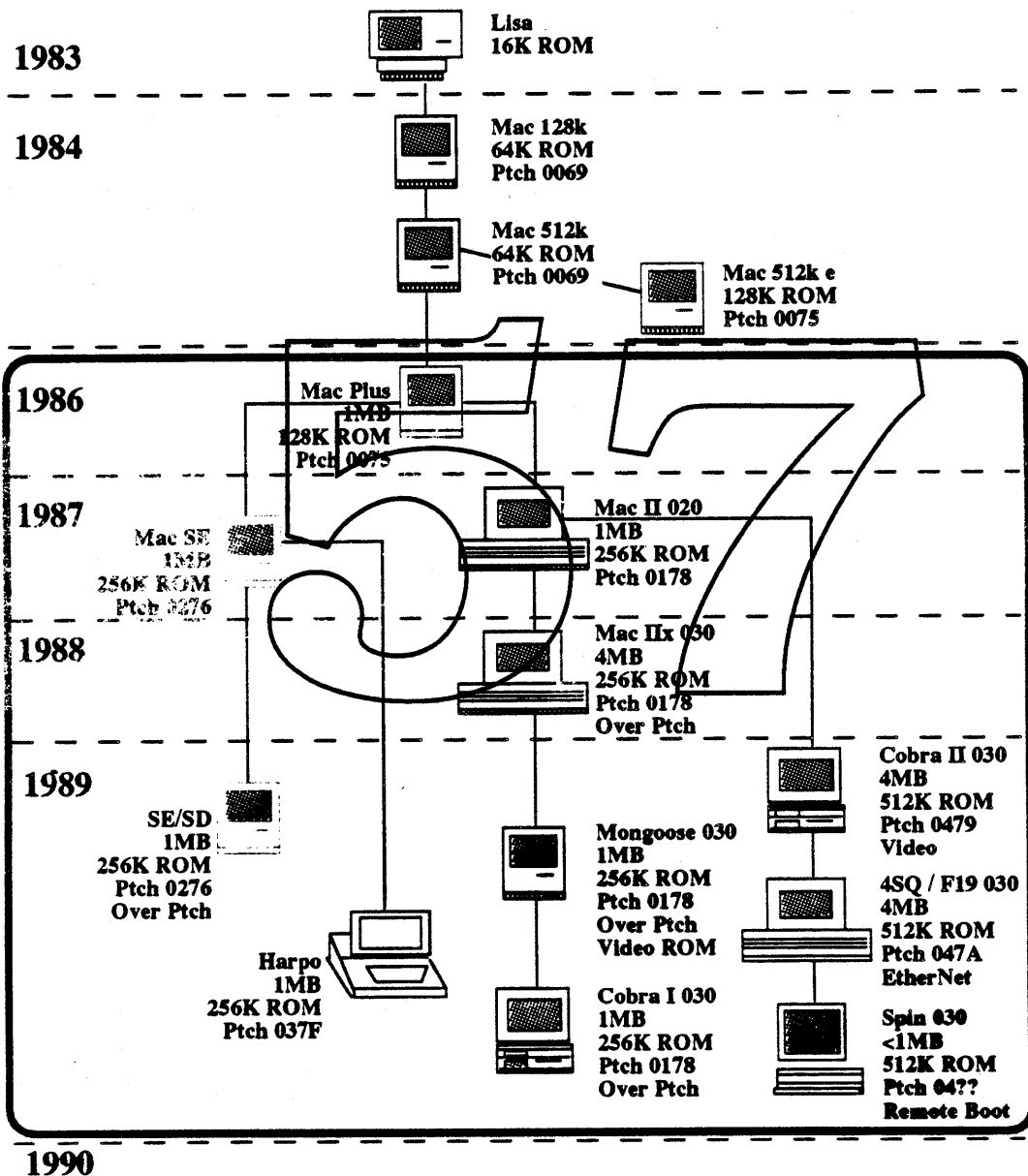


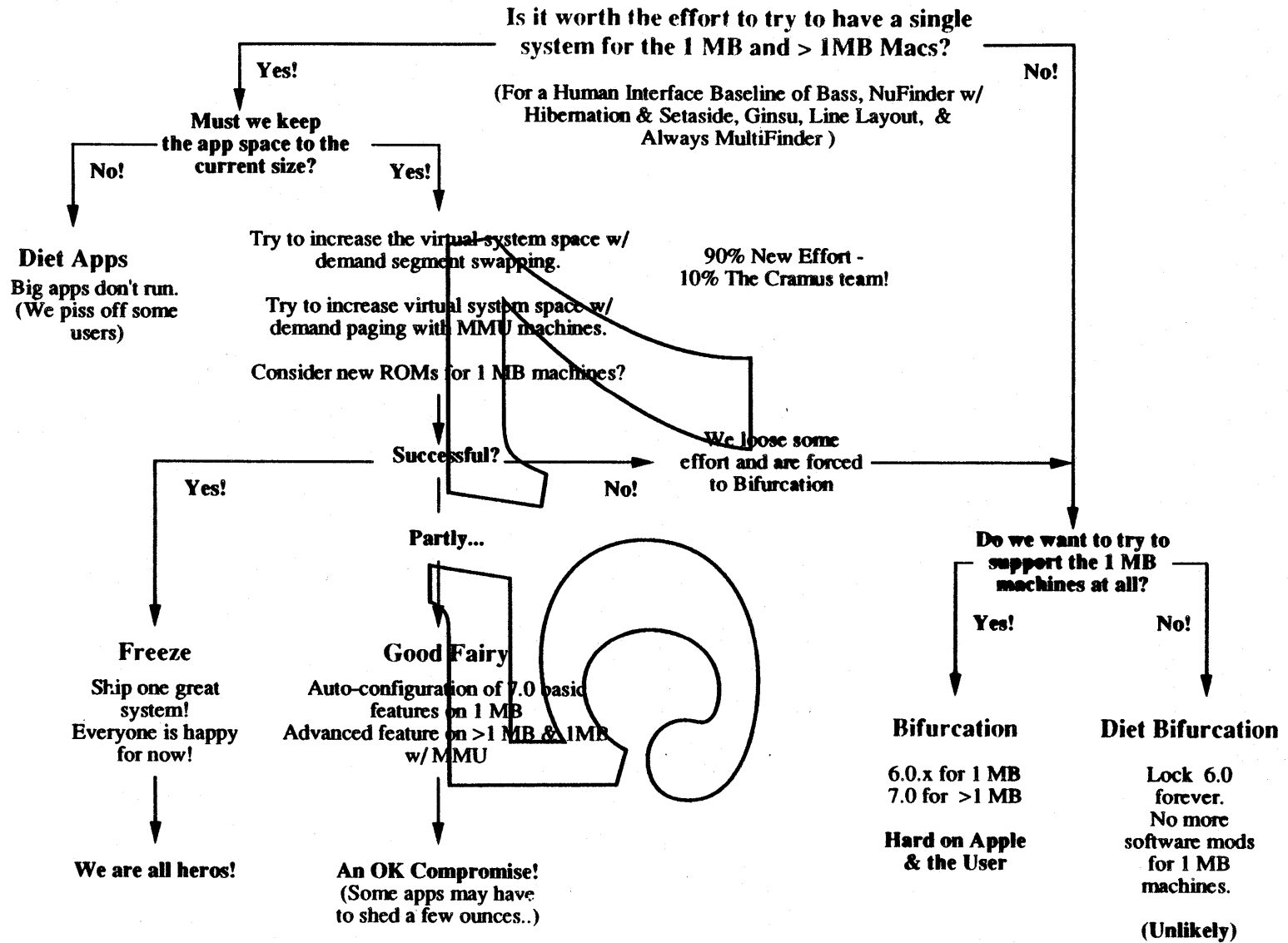


4) **Diet Applications** System v6.x is replaced by v7.x. System v7.x is the standard system for all Macintosh computers. This system should work on a 1 Mb Macintosh Plus with a 512K application. Huge applications (primarily HyperCard) have to run in approximately 512K if they want to run on one megabyte machines which means that they may have to go on a diet.



# Macintosh Genealogy





## Big Bang key features

Macintosh system software covers these four functional areas:

### Operating System

The two priorities for operating systems development in FY89 are 32-Bit Addressing and Interprocess Communications (IPC). 32-Bit Addressing will allow our 68020/030 machines to accommodate more than 8 megabytes of address space, which will support larger applications and more concurrently active applications. IPC will provide the means for applications to send messages to one another and to leverage off each other's functionality. IPC will also act as an enabling technology for user scripting and will allow developers to write more modular applications. Virtual memory is the next highest development priority.

### Imaging

We will have two releases of new imaging functionality during FY89. In the spring, we will ship 32-bit QuickDraw and a color PostScript LaserWriter driver as installable extensions to System 6.0.3. This will extend Apple's lead in color functionality and provide for photo-quality color on high-end systems. With Big Bang, Apple will deliver an open-format outline font system, the "Layout" manager, and several important extensions to QuickDraw. Outline fonts will give users more flexibility in dealing with font sizes and styles. The layout manager will give developers a standard way to attractively layout text on a line. The highest priority extensions beyond these are those that cannot reasonably be added by developers on an application-by-application basis.

### Desktop Services

Desktop services covers those functions that deliver services directly to (or most closely impact) the user. It includes the Finder, User Interface Toolbox, User Utilities, and Application Utilities. With Big Bang we want to simplify system setup and system management functions as well as to provide new tools to give applications additional power. New Finder is our highest priority in Desktop Services. New Finder will unify most system functions into one consistent interface and will provide both more power in dealing with large volumes of files and the hooks for Finder expansion. Other important 1989 Desktop Services projects include a new Installer program (easier system setup), the Language Manager (standard text-processing services like spell-checking and thesaurus for applications), an API for access to remote SQL databases, user interface extensions, and Interapplication Communications ("hotlinks" for applications).

### System Integration

The two big efforts in System Integration for 1989 are the new Print Architecture (code named Ginsu), which will allow Macintosh to support a greater range of output devices and will provide improved device control, and the File System Manager in combination with the DOS File System, which will provide Macintosh desktop access to MS-DOS disks with Superdrive. (Currently the MS-DOS project is unstaffed)

### Where we will be

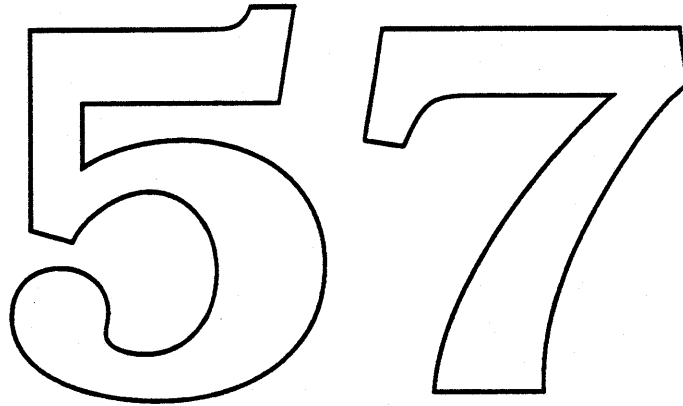
With the delivery of Big Bang, we will have delivered a major step forward in the evolution of Macintosh system software. We will reap these important benefits:

- Macintosh will have the ability to support larger, more complex applications. (32-Bit Addressing)
- We will see applications beginning to work cooperatively. (IPC, IAC)
- We will have the best color support in our product class. (32-Bit QuickDraw, Color PostScript Driver)
- Users will have much greater flexibility in dealing with font sizes and styles. (Outline

Fonts)

- Macintosh will be easier to set up and use. (NewFinder, Installer 3.0)
- Spell-checking and other text services will be a pervasive Macintosh application feature. (Language Manager)
- Many applications will support dynamic data insertions or "hot links." (IAC)
- Applications will begin to make use of remote SQL databases. (SQL Interface)
- There will be a greater range of output device choices for Macintosh. (NewPrint Architecture)
- Non-Macintosh file systems will be better integrated with Macintosh than on their native machines. (File System Manager)

With this set of expanded features, we will have major product introduction opportunity.



57

**Schedules will be distributed at a later date.**

57

57



# Video Software Madness - The Horror Continues

Product Engineering- Graphics Software Group  
January 19, 1989

## Overview

This document describes a number of modifications that Apple is making to the specification for all Macintosh II video drivers, many of which are directed toward improving support for Jackson Pollack devices (16- and 32-bit chunky pixels, and extended frame buffers). This paper is not intended as a full implementation specification, but as a preliminary introduction to the general direction of the changes. Information in this document is subject to change.

*If you don't like scary movies or the system bomb box, skip ahead to "The Solution" overleaf. I'm headed there now...*

## The Nightmare

So you're cranking along at 3AM trying to put those final touches on your MacWorld demo. You were skeptical of the marketing manager's request that you add an Australian sunset splash screen to your boot block configuration program, but it seemed like a good idea after you found out you'd get a high ticket video card, and, since nobody could find a picture, you'd have to go to Australia to take the picture yourself. So you got the picture, got the card, and installed Jackson Pollock software so you could use it. And it looked fantastic, too. Until now. SysError city. And for some reason (not *our* fault), you can't boot off your trusty hard disk. So you do what you'd always do- whip out that handy System 6.0.3 floppy and boot your machine up. But hey, what happened? The system starts booting but then it crashes! You try other disks- always the same problem. Finally you yank out that new 32-bit video card and you're able to boot again. But you'll never trust that darn 32-bit stuff again...

## The Problem

Hopefully this will never happen to you. But if it does, here's why everything seems to go wrong.

In a nutshell, the problem is due to the Mac II CPU ROM's 24-bit addressed nature. Mac IIs normally run in 24-bit addressing mode for compatibility with existing Mac software. For normal displays, the 1MB address space available for a video card frame buffer was reasonable, since Color QuickDraw could only support displays of 8-bits/pixel or less. With the advent of Jackson Pollack software, and support for 32-bit chunky pixels, the frame buffer size limitation becomes a major problem, severely limiting the possible display size. To alleviate this problem, the Jackson Pollack software switches the hardware to a true 32-bit addressing mode at blit time, allowing frame buffers to occupy up to 16MB of the 68020 address space. The video configuration ROM data structures were extended to notify the CPU at startup that a Jackson Pollack frame buffer was to be accessed in 32-bit mode, primarily by setting a Slot Manager flag bit that specified a 32-bit device base address be constructed in that card's Device Control Entry (DCE) and in this device's gDevice data structure.

This all worked pretty well, with the one sticky point being the fact that it was potentially dangerous for one of these 32-bit addressed frame buffers to be accidentally accessed in 24-bit address mode, since that would translate into a wild write into unprotected space. Outside of applications that directly access frame buffers (highly discouraged, and pretty difficult to do anyway), the only problems for these cards would arise if a 32-bit accessed card were the boot device, since the "Welcome to Macintosh" message would be written before Jackson Pollack had been patched in. A SecondaryInit mechanism was added to the Slot Manager to allow this message to be suppressed, and to allow a 32-bit accessed card to be the sole card in a system.

Releases of Jackson Pollack prior to version d3.3 did not include support for SecondaryInits, whose absence was the cause of the crashing system in the scenario above. Had this code been in place, the card would never have been activated, preventing the crash, but rendering the card useless, even though the lower pixel depths were, at least in theory, accessible.

The generation of cards which lack SecondaryInit are particularly troublesome. Setting the f32BitMode unconditionally generates a 32-bit DCE base address well before there is any way to tell if Jackson Pollack software will be loaded. If the software was not present, then the first write to the frame buffer is potentially catastrophic. As our poor programmer found out above, the majority of the bootable floppies in the world do *not* include the Jackson Pollack patch and easily throw the system into this inoperable condition. Although this is a System configuration problem, it creates the appearance of bad hardware or faulty software, which are both highly user-UNfriendly situations.

## The Solution

By modifying the configuration ROM structure to guarantee that Jackson Pollack is installed before enabling the 32-bit accessed video modes most of these shortcomings are annulled. Video cards that support direct pixels often include support for 1- or 8-bit indexed modes that do not need to be accessed in 32-bit mode. By enabling one of these 24-bit addressing compatible modes at startup, the decision to switch to a 32-bit addressed mode can be deferred until SecondaryInit which can positively identify the presence of Jackson Pollack software. With this addition, any card that has at least one 24-bit addressable video mode can work on any Macintosh II. If that machine has Jackson Pollack software, then additional features may be available. This card will also work when booted with any system disk.

This enhancement is accomplished with a few additions to the configuration ROM, some extensions to the Slot Manager, and a bit more code in Jackson Pollack. In addition to the sRsrc lists that contain all modes that are available with 32-bit addressing (which are unchanged), a new video sRsrc list that contains only modes that can be accessed in 24-bit mode (and doesn't have f32BitMode on) is included in the sRsrc directory. On future machines that have Jackson Pollack and the new Slot Manager in ROM, this new sRsrc list is unnecessary and can just be deleted by the PrimaryInit code. For current CPU ROMs, the 32-bit addressed sRsrcList is excised to prevent the problems mentioned at the beginning of this document.

System patches, Jackson Pollack and the new Slot Manager are loaded next,

then the SecondaryInit code is run. As part of the SecondaryInit, the card can test for Jackson Pollack (compare trap vector \$AB03 equal to `_Unimplemented`)<sup>1</sup> and, if present, can delete the 24-bit addressed `sRsrcList`. At this point, the 32-bit `sRsrc` list can be added back into the system by another new Slot Manager call, `_sInsertSRTRec`. If this was the boot device, the Slot Manager takes care of updating the information in the DCE; other video devices are not yet open. If the Jackson Pollack patches are not loaded, then the SecondaryInit is not run, and the card uses only the 24-bit compatible `sRsrcList`. Notice that under this model, cards that are written in full compliance with the original Jackson Pollack driver specification or that have no 24-bit addressable modes will also work, although they will be completely inactive in non-Jackson Pollack systems. Existing minor space-only cards remain unchanged.

Here's how the card configuration sequence looks in a little more detail:

- 1) The machine powers up and runs CPU diagnostics.
- 2) PrimaryInit code from each slot ROM is executed in ascending order.
  - a) The PrimaryInit code calls `_sVersion` (a new slot manager trap) to determine the revision level of the ROM Slot Manager. If the version returned is 2 or greater, then this card is plugged into a machine that has the new Slot Manager and Jackson Pollack resident in ROM, and can leave the video `sRsrcList` that includes 32-bit addressed modes directly and return `seSuccess (=1)` in the `seStatus` field of the `sExec` block.
  - b) If the new Slot Manager is not in ROM, then `_sVersion` will return an error code (because it was unimplemented). If the video card has an `sRsrcList` that is compatible with 24-bit addressing, then it should leave that list installed and return `seStatus=seSuccess`.
  - c) If the card has no 24-bit compatible video modes, it should return the special value that signifies that it should be invalid until SecondaryInit (`seStatus=$8001`). If Jackson Pollack is not present, then this card will not be useable.
  - d) Normal hardware initialization and screen graying should be performed at this time.
- 3) The boot card's driver is opened and its `gDevice` is created.

<sup>1</sup> It's unlikely that the SecondaryInit would be called if Jackson Pollack was not present since they are bundled together, however, it's safer to verify that all is well.

- 4) The happy Mac and "Welcome to Macintosh" messages are displayed and the system begins to load MacsBug and System Patches.
- 5) Jackson Pollack and the new Slot Manager are loaded.
- 6) SecondaryInits are executed in ascending slot order.
  - a) The presence of Jackson Pollack is verified.
  - b) If a 24-bit sRsrc list was passed from PrimaryInit, it is removed.
  - c) The 32-bit sRsrc list is added in via `_sInstallSRTRec`.
  - d) Any additional initialization is performed.
  - e) SecondaryInit returns `seStatus=seSuccess`.
- 7) The Slot Manager fixes any system data structures affected by the switch.
- 8) The remaining video drivers are opened.
- 9) INITs are executed. The first `InitGraf` (in the INIT calling routine) causes the screen configuration to be initialized based on the `scrn` resource. This sets all displays to their previous depths.
- 10) The first application is launched.

All the nitty-gritty details about how to find the values to pass will be covered in the full specification.

## More Information About Drivers in Direct Mode

The original 32-bit drivers paper did not include specifics on extensions to the driver in support of direct mode devices. Here's that information.

**SetMode** – The `SetMode` call is extended to allow 16- and 32-bit/pixel modes to be selected. When a direct pixel mode is selected, in addition to changing the screen depth, the color table is written with black->white ramps (black at the low end of the CLUT, reversed from grayscales in indexed mode where white has pixel value zero). The current gamma table should be applied during this operation. All video modes in an `sRsrc` list should have the same base address.

**SetEntries** – The `SetEntries` control call is never called by the ROM for direct or fixed device types, and applications that call the `SetEntries` control call should test the target `gDevice`'s `gdType` before making this call (the Color Manager's high-level `SetEntries` call honors this convention). If

SetEntries is called while in direct mode, it should return a `ctlBad` error code. There is no way to directly set an arbitrary CLUT location with a particular RGB when in direct mode, although color table animation effects are possible when using the `SetGamma` call. `SetEntries` is constrained in this way because its color table must be kept as part of the `gDevice` device `pixMap` for Jackson Pollack to correctly map colors, and, in direct devices, the destination color table is implied, not explicit.

SetGamma - In the photorealistic applications that are the mainstay of direct color devices, gamma correction is particularly important to avoid oversaturated output. The gamma data structures are unchanged for direct video modes. The `SetGamma` call is also unchanged when the driver is in a non-direct mode. When in direct mode, the `SetGamma` call not only sets the gamma correction data table, but sets the CLUT with the same linear grayscale ramps set in the `SetMode` call, again applying the gamma correction. Unlike indexed mode, note that the `SetGamma` call need not be followed by a `SetEntries` call (but can be) for the new Gamma table to take effect. This routine can be used for limited psuedo-color table animation, but applications that use this feature take responsibility for the screen effects.

In addition to the definition of gamma correction for direct devices, the `SetGamma` call has also undergone some general extensions for all video drivers. In particular, if `NIL` is passed as the gamma table pointer, then the driver should reset its default gamma correction table. If `-1` is passed as the gamma table pointer, the driver should set a linear uncorrected gamma table. In both cases, the driver's internal copy of the gamma table should be updated to match the latest state.

The new `Monitors` `cdev` will allow a user to specify the default gamma table resource to be applied on a device-by-device basis. Third parties can supply this resource in an extensions file similar to a `cdev` containing device specific data.

Due to its omission from the current documentation, gamma information is a little hard to come by. Contact DTS for the latest version of *Gamma Correction in the Macintosh II* which explains gamma correction and the data structures.

## Other Fun Stuff

Here are a few other features which are coming soon to a Jackson Pollack release near you. These features are available to all graphics cards, not just direct devices.

Video mode families - Currently, the system allows only one video sRsrc list to be present per gDevice. With the Jackson Pollack release, cards will be able to have multiple sRsrc lists available which describe alternative modes which the card can support. A new Slot Manager routine, `_sSetsRsrcState`, will allow the card software to specify inactive sRsrc lists which are not "seen" by standard Slot Manager calls such as `_sGetNextsRsrc` unless a special flag is set. The new Monitors cdev will allow selection among the active and inactive sRsrc lists. This feature will allow a user to select between underscan and overscan modes on an NTSC display, for example which was not possible before. Since the Foolbox is rather intolerant of desktop size changes, the new selection will not be realized until the next reboot.

The implementation of this feature is very similar to the method described above for 24->32-bit addressing switching. For current ROMs, all video sRsrc lists except the currently chosen video sRsrc list are deleted in PrimaryInit. If the new Slot Manager is present and SecondaryInit is executed, then the inactive modes can be added back in with `_sInsertSRTRec` and then inactivated with `_sSetsRsrcState`. If the PrimaryInit detects the presence of the new Slot Manager in ROM (with `_sVersion`), then the invalidation can be performed in PrimaryInit.

When selecting the lists to delete, insert, or inactivate, it is important to follow this simple rule: All inactivated sRsrcs presented in Monitors should be displayable on the current monitor connected to the system. Users should not be able to select an alternate video mode that would render the screen unviewable. For example, if a card supported the Apple HiRes RGB display, underscan RS170, and overscan RS170, if an Apple RGB monitor were connected, both RS170 resources should be deleted. If an RS170 display were connected, then the non-interlaced list should be deleted and one of the interlaced modes inactivated. Cards intended for use with multisync monitors could inactivate modes of different timings, but should verify the presence of a compatible monitor if possible.

scrn invalidation - Due to popular demand, a video card will be able to invalidate the scrn resource programmatically. Typically, this is most useful to cards that support different sized monitors or multiple screen sizes on a single monitor. When the PrimaryInit code of the card senses that the monitor connected to the card has changed, it can invalidate the scrn resource by calling `_Invalscrn`. When QuickDraw detects that this bit is set, it will disregard the current scrn configuration, until a new scrn resource is built by Monitors. As with the video mode family support, scrn invalidation is targeted toward cards that can identify the type of monitor connected.

Card Names and Icons - New versions of the Monitors cdev will display a manufacturer-specific icon embedded in the configuration data structures as well as the board name string. To include a developer icon, include an OSLstEntry whose spID is `sRsrcIcon` and whose offset points to a standard icon/mask set. To include a color icon (if you've got the room), include an OSLstEntry whose spID is `sRsrcCicn`. This entry's offset points to an image of a cicn resource of size 32\*32. You may want to consider how this icon will look in various pixel depths. Also be sure that the board name and version strings are up to date (i.e., don't leave "Toby Frame Buffer Card - Beta 7.0" in your ROM unless you want to look stupid).



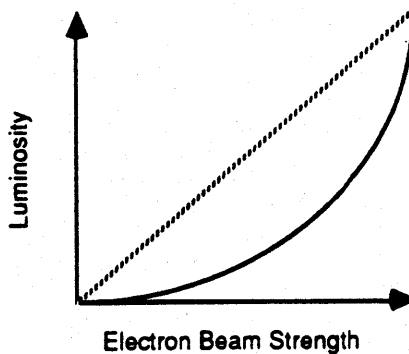
# Gamma Correction in the Macintosh II

Product Engineering - Graphics Software Group  
January 19, 1989

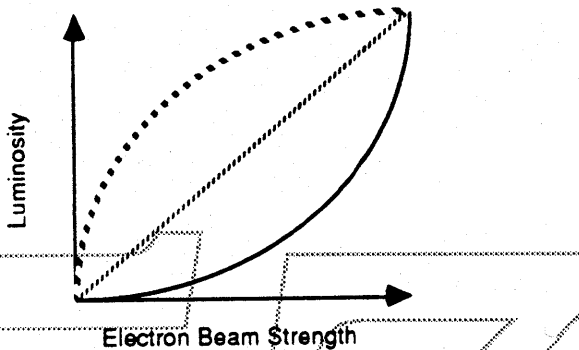
An important part of the device-independent graphics model of Color QuickDraw is that all colors specified by applications are *absolute* specifications; that is, from the application's view, a single color specification will appear as a uniform color across devices which have different color response. In the multiple-screen Mac II video environment, these differences in color response must be handled by the system since the application does not "see" the different screens, and does not have a chance to perform screen-by-screen corrections. The act of linearizing the color response (or gray-scale response) is called *gamma correction*, and is currently performed by the video driver for each display device configured in the system. This document describes the data structures and implementation of gamma correction in the Macintosh II.

## What is Gamma Correction?

The phosphors on the face of the monitor tube luminesce when struck by the beam of the electron gun sweeping out the scanlines. By increasing the intensity of the beam, the phosphor dot luminesces more brightly, and by reducing the intensity of the beam, the phosphor glows less brightly. Unfortunately, the output of the phosphor is not directly proportional to the impinging beam strength. The response of the phosphors usually looks something like this:



where the dotted line shows ideal linear response, and the solid line approximates the observed response of a normal phosphor. This response characteristic is due to physical phenomena and can be described via a gamma function. Subjectively, this causes the colors on the screen to appear darker than expected. Based on this behavior, an inverse gamma correction function can be applied to compensate for this non-linear response:



Here, the solid line again represents the uncorrected phosphor response, the large dotted line is the inverse gamma function, and the fine dotted line is the resultant, linearized color response.

The appropriate inverse gamma function can be determined mathematically, given the appropriate data about the phosphors, however, in the current Mac II cards, the data is determined empirically, using a photometer to measure the response characteristic on a calibrated system. Gamma correction values determined by mathematical means are supported by the gamma data structures (see below), but must be implemented by the driver that this table is intended for.

In the current implementation of the video drivers, gamma correction is applied to requested absolute colors immediately before they are set in the color look up table (CLUT), by the SetEntries control call. More specifically, some number of high-order bits are extracted from the red, green, and blue channels and used as an index into tables of corrected values. These values are then placed into the hardware, yielding corrected output. On the Macintosh II Video Card (the TFB card), the high eight bits of each channel is used to reference the gamma table.

There are a number of minor shortcomings of this implementation. First, there is not absolute symmetry between the SetEntries control call, which sets

the CLUT, and the GetEntries status call, which reads the CLUT hardware, since the gamma correction took place as part of the SetEntries call. Also, the uncorrected values are generally unrecoverable (although a copy of the absolute colors are always available in the GDevice structure). Finally, it is most desirable to extract more bits as an index to the gamma table than the number of bits of color information that will be set in the CLUT to avoid a loss of color resolution after correction. For example, the TFB card has an eight-bit per channel CLUT, but only uses the most significant eight bits of the (sixteen-bit) channel information to perform the gamma lookup. At lower intensities, the gamma correction increases the distance between adjacent values, and as a result, on the TFB card, some dynamic range is lost at the lower intensities. This could be corrected by extracting nine or ten bits of channel information rather than eight and using a larger gamma correction table, but this option was declined to reduce gamma table size.

## The GammaTbl Data Structure

The structure itself has been a bit of a mystery, as it is not defined in either *Inside Macintosh, Volume 5* or *Cards and Drivers*. This is the structure:

```

record GammaTable of
    gVersion      : integer;           (gtab version, currently 0)
    gType         : integer;           (drHwId value)
    gFormulaSize  : integer;           (size of formula data, below)
    gChanCnt      : integer;           (# of component channels)
    gDataCnt      : integer;           (# of values per channel)
    gDataWidth    : integer;           (size of data in tables)
    gFormulaData  : array [0.. gFormulaSize] of byte; (data for
                                                gamma calculation formula)
    gData         : array [0.. gDataCnt] of byte; (gamma
                                                correction lookup tables)
end;
```

In this structure, `gVersion` represents the gamma table format version, which is 0 for all current cards. The `gType` field holds the `drHwId` value for this board, to identify the board that this table was measured for. Note that this means that a single gamma table can't directly be shared between two different cards, even if they both have the same CLUT response curve (which is usually linear). This allows the data in

the gamma table to be in an appropriate form for varying hardware (i.e., a card could have four-bit/channel DACs and might prefer gamma data in the range \$0..\$F rather than \$0..\$FF).

`gFormulaSize` defines the number of bytes occupied by the `gFormulaData` field. On Apple's current video cards, gamma correction is performed by modifying the values loaded into the CLUT by the `SetEntries` control call to approximate linear response on the display. On these systems, the gamma correction table acts as a final lookup data table that translates the requested color into closest available linearized level. These gamma table values are determined empirically by measuring the output of a calibrated display. More sophisticated systems may choose an alternative to this simple lookup mechanism, for instance, calculating gamma correction factors based on a mathematical response function. By default, the TFB card uses a single correction table for all three channels. No calculations are performed on the incoming color table other than simple lookup. Cards can remember the specific monitor configuration at the beginning of the `gFormulaData` field, allowing it to identify and use only gamma tables developed for the currently connected monitor.

`gChanCnt` is the number of lookup tables in `gData`, below. If there is more than one channel of gamma correction data, then the R, G, and B tables follow each other respectively at the end of the structure.

`gDataCnt` is the number of discrete lookup values included in each of the channel's correction table. It is always equal to  $2^{gDataWidth}$ , but refers to number of bytes that this channel's data occupies.

`gDataWidth` describes the number of significant bits of information available in each entry in a channel's correction table. The data always appears as `gDataWidth` bits, right-justified in a field which is the next larger number of bytes than `gDataWidth`. Since it is rare to have devices have more than 8-bits of CLUT resolution, virtually all devices pack their correction data into bytes.

`gData` is actual correction table data itself. If there is more than one channel's information, each table follows the next in R,G,B order. The standard tables included in Apple's driver have only one table, which is

applied to all three output channels. Since Pascal cannot express variable size fields in record structures, the independent channels are not individually named.

## 'gama' Resource Format

In addition to the RAM data structure for gamma tables covered above, there is a standard resource format for gamma table resources. Like many other resource templates, the gamma structure is an image of the RAM form stored in resource format. There are no changes.

## Using Gamma Correction

Gamma correction is always applied by the TFB video driver. At driver open time, the driver is usually initialized with a linear (non-correcting) gamma table. When `_InitGraf` is called, the 'scrn' screen configuration resource is read from the system file. This resource (described in *Inside Macintosh, Volume 5*) includes information about the size and orientation of the different monitors configured into the system, including their last video mode (pixelsize), color table, and gamma table. If there is no 'gama' resource id specified, or the specified id is not present, then a default gamma table, 'gama'=0 is loaded from the system file and used (this is the table calculated for the TFB card). If the specified resource is found, then the appropriate resource is loaded, and a control call is issued to the driver to make this the current gamma table. Unfortunately, there is currently no tool to allow the 'gama' id to be set short of modifying the 'scrn' resource directly (*Note - the new Monitors cdev will provide this facility*). To facilitate the usage of the gamma table, there are two calls in the standard video driver routines that set the gamma table (control call 4, `SetGamma`) and retrieve the pointer to the current gamma table (status call 6 on TFB rev 2 drivers and up). These calls simply take and return a pointer to a `GammaTbl` structure.

## Driver Functionality :

**SetGamma / Control code = 4 :** This routine sets the card's gamma correction table by copying the supplied table to the driver's private table. The `csParams` field of the control parameter block has a pointer to a gamma table block (`csGTable`). This block consists solely of a pointer to the gamma table data structure. For devices in an indexed pixel mode, this routine merely loads the gamma data into the driver's private copy for application on the next `SetEntries` call. Since direct devices do not respond to the `SetEntries` call, when in these modes `SetGamma` generates a linear ramp in each channel then writes these values into the CLUT, applying the gamma correction at that time. Unlike normal ramps in indexed mode, the ramp applied in direct mode places level 0 in the lowest position of the CLUT and \$FF in the highest address (for 8-bit DACs).

As an enhancement, a number of shortcut dummy parameters for the `SetGamma` routine improve useability. If `NIL` is passed in the `SetGamma` Control parameter block as the new table address, then the driver will set the gamma to the initial gamma table, which was loaded at driver Open time. If `POINTER(-1)` is passed in the control parameter block, then the driver will set the correction table to be linear and uncorrected. Use of this option will guarantee that all possible gray levels that the card can produce are available, however, images drawn on this screen are not being gamma corrected.

# Harpo/Topanga Video Software Support

David Fung  
Graphics Software Group  
January 6, 1989

## Overview

This document describes the system software environment in support of the Harpo and Topanga display systems. It includes a brief description of the hardware, the specifics of the software implementation, and an analysis of some implications of the hardware design on software and the user interface.

## Hardware Description

Harpo is the 68000-based portable Macintosh. It features a 640\*400 active matrix LCD display built into the case. The Topanga video converter is an optional add-on the Harpo which allows the Harpo to drive a variety of CRTs, specifically, 67Hz non-interlaced timing compatible with the Mac II Hi-Res Monitors (640\*480 pixels), NTSC displays (512\*400), and PAL (640\*480) displays, selected by physical switches. Although Topanga includes an internal frame buffer, this frame buffer is not accessible from the CPU bus, and always displays the same data as is present in the Harpo CPU. Unlike typical Mac II video hardware, the Topanga produces actual composite video signals rather than timing-compatible RGB signals (which must be encoded to composite video to be displayed on a television set). In addition, composite output of the Topanga is always convolved to reduce flicker on these interlaced displays.

Topanga does not include any software-accessible control registers. As such, software cannot identify the presence of a Topanga, the type of monitor connected, or the video output mode selected. Similarly, the Harpo cannot modify the state of the Topanga converter (i.e., the Harpo cannot defeat convolution via software switch). This is considerably different than the Macintosh II environment which is generally fully controlled from software, and has implications on the operation of the hardware to the Harpo user.

## Software Description

The software support for Harpo video output consists of two major groups of modifications. First, there are Harpo-specific modifications to QuickDraw and the start code that allow the screen size to be settable to one of the various resolutions. The second group of modifications update the Window and Menu managers on all CPUs to allow the curvature of the corners of the desktop to be settable. In addition, some amount of support for these modifications is required in the Harpo cdev.

On non-Color QuickDraw Macintoshes, the size and shape of the display is defined by a `bitmap` data structure, created in the QuickDraw A5 globals on each `_InitGraf` call. `InitGraf` constructs this bitmap, called `ScreenBits`, from a number of low-memory locations; `ScrnBase` provides the frame buffer base address, `ScreenRow` provides the rowbytes of the display, the screen height from `MaxY`, and the screen width from `MaxX`. QuickDraw can draw to a frame buffer described by any reasonable combination of these values. In the current start code, these low-memory globals are set up from constants. The Harpo start code is modified to maintain a value, `hcScrnCfg` (\$??), which selects the desired set of parameters for the globals. The specific values for the globals in each mode are listed below.

The Harpo frame buffer is designed to display 640 pixels per line with no unused space between scanlines. The Topanga converter always expects the video data to come out of a frame buffer of this format. All the video displays supported by the Topanga are different in size than the Harpo LCD. The Mac II monitor has the same scanline length, but is 80 scanlines taller. In this case, the Topanga adds additional black lines to the top and the bottom of the Harpo's display. Similarly, PAL composite displays have the same scanline length, but 86 extra scanlines which are also blacked out above and below the active display. The NTSC display has a smaller display area than the LCD, and require that a centered, reduced version of the desktop be drawn in the frame buffer. To generate such a display, `ScreenBits` is set up so that the bounds rectangle reflects the reduced screen size, but rowbytes is unchanged, and the base address is offset to the starting position of the reduced size display. In this case, the area surrounding the reduced display should be filled with solid black in order to darken the screen area out to the corners of the CRT. Here are the appropriate bitmap definitions for the four display modes:



For the Harpo Built-in LCD Display:

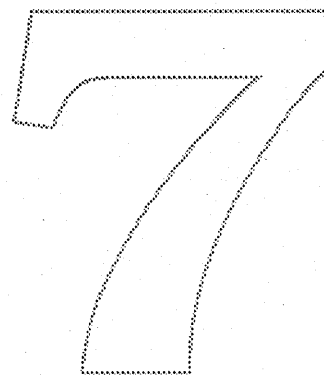
```
hcScrnCfg = 0
scrnBase = $FA0000
screenRow = 80
MaxX = 640
MaxY = 400
```

For the Macintosh II Hi-Res Display :

```
hcScrnCfg = 1
scrnBase = $FA0000
screenRow = 80
MaxX = 640
MaxY = 400 (80 lines are blacked)
```

For NTSC Display:

```
hcScrnCfg = 2
scrnBase = $FA0008
screenRow = 80
MaxX = 512
MaxY = 400
```



For PAL Display:

```
hcScrnCfg = 3
scrnBase = $FA0000
screenRow = 80
MaxX = 640
MaxY = 400 (86 lines are blacked)
```

Since all drawing is clipped to screenBits.bounds, in NTSC mode, it is sufficient to blank the screen edges once at startup. The screen parameters code only reads hcScrnCfg; it is set only by the Harpo cdev. The Mac Toolbox currently does not allow changes to the size of the desktop, and, as such, screen mode changes cannot take effect until the next reboot.

Rounded corners on the desktop are a fixture of the Macintosh user interface. The presentation of the corners is affected by write-to-black nature of the LCD. To improve the appearance of the display, the rounding of the desktop when displayed on the LCD should be done in white rather than black. When

Topanga output is selected, the Window Manager should revert to a black surround.

The code changes in this area are straightforward. The `_InitWindows` routine is modified to test the `hcScrnCfg` global. If `hcScrnCfg = 0`, the LCD display is selected, and the desktop is framed in white. If `hcScrnCfg` is non-zero, then a Topanga display is primary, and the framing is performed in black to mask the non-linearities in the edges of the CRTs.

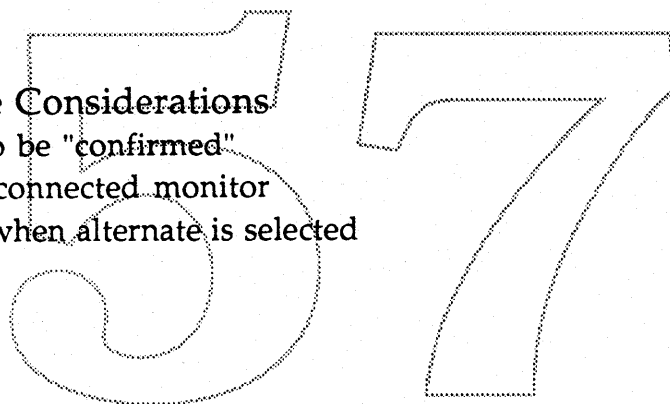
Finally, there are a number of changes to the drawing of the various graphic elements to accomodate the specifics of the display. In particular, the menubar is given and one-pixel frame to define it against the non-rounded background. This frame will be inset by one, which will have a minor effect on menu title positioning.

### Human Interface Considerations

Switch select has to be "confirmed"

No recognition of connected monitor

Display looks bad when alternate is selected



**The Jackson Pollock**

**Product Plan**

-or-

**Why 32-Bit QuickDraw is Not Just for the Low End**

57

## I. Introduction

"Jackson Pollock" is the codename of the project to extend QuickDraw to support up to 24 bits of color. In addition to the 24 bits, it will support the movement of an additional byte of information that is typically used by applications developers as an "alpha channel" or a transparency mask. While the 32-Bit QuickDraw code is 32 bit clean, it is NOT necessary to have a 32 bit clean system in order to run it. This code will be supported on all machines that have Color QuickDraw, specifically 020 and 030 class Macintoshes.

## II. Objectives

The objectives of the Jackson Pollock project are:

- 1) to substantially raise the competitive stakes of color in the personal computer market;
- 2) to make extensive color capability a standard on all color Macintosh computers.

QuickDraw is currently under a great deal of scrutiny in the personal computer market. In the area of basic primitives, drawing commands such as lines and curves, QuickDraw is significantly behind PostScript running on Unix-based machines and behind Presentation Manager in the IBM-compatible market. At present, the Macintosh is behind in outline font capability. Other than the color changes made to the Macintosh in 1987, little has been shown to customers and developers to change the perception that the Macintosh is standing still in graphics.

The area in which the Macintosh is significantly ahead of the competition, however, is in that of color and bitmapped graphics. Both Presentation Manager and Display PostScript have significant problems in their implementation of color. While they claim to have 24 bit support, it has yet to be demonstrated. In addition, starting with a device-independent model has caused them to basically skim over the ability to manipulate image or "bitmapped" graphics. Thus, the move to support 32-Bit QuickDraw is an attempt to push our competitive position to the limits. By being the only systems company to thoroughly support this level of color, we have positioned the Macintosh as the machine for which to do color development, unlike the PS/2, Sun or NeXT systems.

At the time this project began, it appeared that additional changes to the graphics system would not be achieved until at least the beginning of 1990. Thus, the project was begun as an interim measure to produce features that developers were demanding. Over forty people from 28 companies attended the original developer meeting held under nondisclosure during MacWorld--January, 1988.

In order to maintain a uniform platform for development, a goal which reduces Apple's maintenance costs, the intention is to fold this code into Color QuickDraw. Customers using color systems and color applications require more than 1 megabyte of memory. As the cost of video cards supporting 24 bits of color drops due to competition and the declining price of RAM, and as 24 bit scanners and color printers become available, it is believed that this will be the most desirable configuration. In fact, with improved printer drivers, it will be possible to get incredible color output on printers that cost as little as \$1000.

In addition, supporting one Color QuickDraw rather than two separate bodies of code is seen as critical to the long term success of the project. Apple has rarely had more than one person working full-time on QuickDraw, and there have been times when there were no engineers responsible for Apple's core graphics system.

### III. Benefits of 32-Bit QuickDraw

The chief benefit of 32-Bit QuickDraw to Apple's customers is that it takes the lid off the number of colors a customer can use. Smooth shading or transitions are possible in the simplest graphs and slides without dithering. Finally, lifelike images can be displayed. Even customers with eight bit video cards can take advantage of the functionality and manipulate 24 bit images with standard color paint packages, scanners and printers that have been modified to support 32-Bit QuickDraw.

It also alleviates some of the demand for higher resolution devices. Perceived image resolution is a function of two variables, the dots per inch and the color depth. While the Macintosh is currently being assailed for its lack of resolution independence, the level of depth Apple provides on the Macintosh produces startlingly high quality images without significantly changing the dots per inch on the screen. In addition, as with LaserPrinters, there is a cost/quality tradeoff in getting more dots onto the screen. While Apple has not yet pushed the dots per inch technology to its limit, with up to 24 bits of color, Apple has pushed the color depth. In fact, the value of extending to 32-Bit QuickDraw is in part that we have pushed that envelope to its limit. We will be able to add additional functionality to the other eight bits at a later date.

The most obvious existing markets for 32-Bit QuickDraw are Desktop Publishing, Desktop Presentations, Video and Film Production, and Scientific Visualization. In Publishing and Video, full color is useful for showing realistic images from natural sources. For Presentations, it is helpful for producing continuous tone "ramps" from one color to another. Finally, 24 bits of color makes continuous data easier to visualize for many scientific applications. As an enabling technology, image visualization can be expected to open many other new markets.

### IV. 32-Bit QuickDraw Implementation and Other System Software Implications

32-Bit QuickDraw will support 16 (1-5-5-5) and 24 bits (8-8-8-8) per pixel at its first release. The additional eight bits will be moved around without adding additional functionality. The PICT file format will be extended to support the 32 bit data. Three compression types have been defined.

In addition to drawing with more colors, 32-Bit QuickDraw adds the following features:

- Support for very large frame buffers  
(devices requiring more than 1 meg of video RAM)
- Support for direct devices  
(devices that do not look up colors with an index into a table)

This implementation is targeted at all CPU's with a 68020 or greater processor. It assumes that such CPU's have more than 1 meg of memory. QuickerDraw enhancements will be spread across all cases (bits per pixel) to achieve optimal performance across 4, 16 and 24 bits per pixel. It is expected to run under A/UX v1.1 with little or few changes.

To see 24 bits of color requires a change to the Monitors CDEV to support video cards of up to 32 bits per pixel. In addition, the General file must be changed so that the pattern editor continues to work. Finally, some minor changes will be made to the Palette Manager to keep it from doing anything wrong in 24 bit mode. The three files, "Monitors," "General," and "32-Bit QuickDraw," must be dragged into a System 6.0.3 folder to be installed. When 32-Bit QuickDraw is successfully installed at Startup, the icon will become a color icon.

32-Bit QuickDraw does not require a 32 bit clean system to run. However, it does run in 32 bit mode. In each QuickDraw call, it always switches the machine into 32 bit addressing mode before drawing and restores the system to the previous mode afterward. All pointers passed to 32-Bit QuickDraw are assumed to be valid 24 bit addresses—they are translated to valid 32 bit addresses via a new trap (`_Translate24to32 = $AB03`). The presence of this trap is sufficient evidence that Jackson Pollack is available. On future machines which natively run in 32 bit mode, the address translation trap and `_SwapMMUMode` should be replaced with an RTS.

Additional parts of the system that have been affected by the Jackson Pollock project include the Slot Manager, AppleShare, and the SwapMMUMode trap. Slot Manager extensions are necessary to allow the use of a 32-bit addressed video card as the boot (or only) screen. It may also be possible to support Rev-A Mac II roms. 32-Bit QuickDraw installs a patch to `_Open` (when necessary) to suppress the AppleShare activity arrows on 32-Bit Addressed screens. The SwapMMUMode trap is patched to prevent the loss of ASC interrupts on MacII's eith HMMUs. The color PostScript LaserPrinter driver designed by the Print Shop will support printing of 32-Bit QuickDraw images.

In addition, it would be desirable to have some minor acceleration hooks put into QuickDraw that would allow QuickDraw to achieve the best performance possible on standard industry coprocessors. This is considered a minor objective of the changes to QuickDraw and may not make it into the code depending on the schedule.

## V. Known Problems/Issues

- Size

The anticipated size of the extensions to QuickDraw is 32k but to facilitate rapid development of a stand-alone patch the shipping patch size may be as large as 90k. Speed vs Space tradeoffs have been approached with a bias towards speed.

- Speed

The code runs somewhat slower than 8 bit code, which is to be expected. However, drawing a 24 bit deep image will be faster on a 24 bit screen. While depth conversion will always be slower than native blits, optimizations for common cases will be made.

- Color WYSIWYG.

While it will be possible to print 24 bit color images with the Color PostScript Printer Driver, nothing will be done in the short term to ensure that the colors on the screen are well-represented by the colors on the printer.

- Testing

Testing does not require a 24 bit video card but does benefit from it. At present, there is a very low supply of these cards, both within Apple and in the developer community. Should testing appear to be incomplete, the project's shipment will be held up.

- All of QuickDraw in Patches.

To achieve the implementation, all of QuickDraw must be patched and put in a file that is easily transportable. The issue as to how to keep the code from being immediately disassembled is being investigated.

## VI. Apple Projects Affected

Apple currently has one project under development that depends on this code. A video card in design in SEG and planned for shipment in fall of 1989 is depending on the widespread release of 32-Bit QuickDraw prior to its introduction to drive the card's success. Expected price of the card is \$1300; it will support 24 bit color on standard Apple color monitors. Originally intended as the video card to drive the Testarossa monitor project, it will also support 8 bits per pixel on large screen, color monitors.

In addition, the Mirus film recorder, a QuickDraw film printer, is supporting slides that can display 24 bit color. In fact, slides with only 256 colors do not give the perception of high quality that this low cost film recorder needs to be successful.

Color scanners and frame grabbers are additional types of hardware that are currently on the market from third parties. Apple itself has plans to do projects such as these that will be introduced after 1989.

## VII. Product Schedule

This project was announced publicly at the Spring Developers conference in 1989 as it was one of very few definite features intended for the September system disk. Due to the slipping of System 6.0, the schedule on the project slipped. It is currently intended to be finished as of March 15, 1989 (see attached schedule). However, complete testing will have a substantial impact on its release date.

The last seeding occurred January 11th. This version resembles the final version in that 6.0.3 supports the installation of 32-Bit QuickDraw by the dragging of icons into the System Folder. The patch icon will turn into color when it is actually installed.

Testing will proceed from December through February. Aside from having Tester Doug Rosenberg and two testers from SIAC in SQA working on it full-time, the Product Manager has arranged to have it tested by key third parties.

## VII. Announcement and Introduction

The product will be reannounced at NCGA in March. There will be a developer meeting off the floor at MacWorld to discuss the number of bugs reported or not reported. Third parties will be showing 32-Bit QuickDraw compatible products at MacWorld. 32-Bit QuickDraw will be introduced/announced with the Color PostScript Printer Driver.

## VII. Distribution

The code will be released to developers by distributing it to all interested third party developers through APDA. Licensing of the three files will be readily available to developers interested in redistributing it with a copy of System 6.0.3. The patch files will also be distributed to all significant user groups and all dealers, and they will be posted to all bulletin boards currently supporting Apple software licensing. An additional promotional flier on the benefits of 32-Bit QuickDraw will be made available to dealers and user groups. Documentation and instructions on installation will be included in a "Read Me" file on the disk for customers.

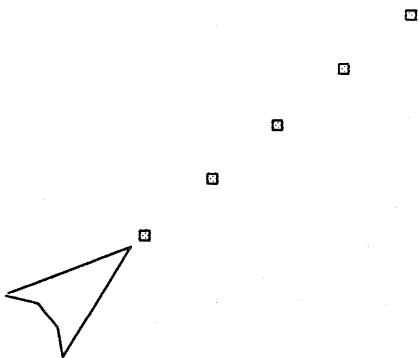
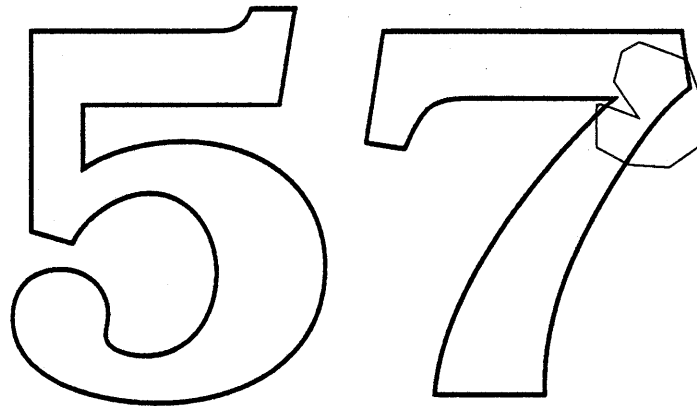
57



# Skia

ABOUT SKIA.....	1
Read these two pages to find out all about Skia.	
THE MATHEMATICAL FOUNDATION OF SKIA.....	3
The Coordinate Plane.....	3
Skia Geometric Structures.....	3
GRAPHIC ENTITIES.....	6
Bit Images.....	7
Regions and Skia Shapes.....	8
THE DRAWING ENVIRONMENT.....	10
TRANSFORMS.....	12
Using Transforms.....	14
Transforms and Bitmaps.....	16
Devices and gDevices.....	17
STYLES .....	18
Algorithmic Text Faces.....	21
Patterns.....	22
GENERAL DISCUSSION OF DRAWING.....	22
Lines.....	23
Filled and Framed Shapes.....	24
Transfer Modes.....	25
Drawing in Color .....	26
Pictures.....	27
SKIA WITHIN AN EXISTING QUICKDRAW APPLICATION.....	28
SKIA ROUTINES.....	30
Error Routines.....	31
Initialization.....	33
Shape Routines.....	33
Lines.....	35
Rectangles.....	37
Ovals and Rounded-Corner Rectangles .....	37
Arcs, Curves, and Wedges.....	38
Polygons and Paths.....	40
Bitmaps.....	41
Text.....	42
Pictures .....	43
Shape Operations.....	44
Calculations with Rectangles .....	49
Operations on Geometry.....	49
Operations on Contours, Vectors and Control Points.....	51
Shape Utilities.....	53
Transforms.....	55
Operations on Shapes and Transforms.....	57
Ports and the Transform's Port List .....	60

Styles.....	61
Operations on Styles.....	63
Color Tables.....	69
Converting Between Local and Global Coordinates .....	73
Devices.....	73
Resources.....	76
CUSTOMIZING SKIA OPERATIONS.....	76
Skia's Interpretation of QuickDraw Routines.....	80
SKIA INTERFACE SUMMARY.....	90
Constants.....	90
Data Types .....	93
Routines.....	95



This document describes, as the third-party developer would see them, Skia and the differences between QuickDraw and Skia. It documents the new features that Skia provides. Reading this assumes a good working knowledge of QuickDraw; words and diagrams in Inside Macintosh, volumes I, II and IV are not duplicated here.

Please forward all comments to: Cary Clark x43887, 27-AJ, or AppleLink FAIRMAN.M.

Skia is being developed in C, so this documentation presents Skia and QuickDraw from a C point of view. New routines are given with C prototypes, and new structures are given in their C type format. Pascal exceptions are noted as appropriate.

---

## ABOUT SKIA

---

Skia is a QuickDraw-call compatible graphics package that provides an advanced feature set to new Macintosh applications. Skia is accessed through an augmented QuickDraw interface, and requires an existing application to be re-compiled or re-assembled. Skia is intended to run on any Macintosh (Plus, SE, II, portable, or future 68030) with a hard disk. Since Skia is written in C, it can be ported to the AMD 29000 graphics accelerator, as well as the Apple II GS, Newton, Draco, or any future Apple hardware with a C compiler.

Skia does not replace QuickDraw. QuickDraw continues to service existing applications, and will be upgraded by 32-Bit QuickDraw (Jackson Pollock), Bass, and accelerated by PQD (Rhoda).

The Skia feature set includes.

- fixed point coordinate interface, fractionally positioned shapes
- device independent, centered geometry
- uniform line and curve thickness
- new capabilities for regions: resolution independence, rotation, etc.
- regions can capture all drawing functions, including bitmaps
- new primitives: paths (including Bass outlines), curves
- off-screen bitmap support
- drawing can be clipped to characters, bitmaps, paths
- PostScript-like features such as line joins, endcaps and dashing
- pattern, dashing, join and endcap capabilities beyond PostScript
- path following (e.g. text drawn around a circle)
- custom text styles, character placement and drawing modes
- length and area primitives
- faster algorithms for hairline drawing and repetitive drawing
- rotation, scaling, skewing, perspective -- full 3x3 transformations
- device independent clipping, framing, path following, hit-testing
- device independent patterns, region operations
- uniform interface available for all drawing primitives
- error detection and routing
- editable, parse-able pictures
- graphics accelerator support
- advanced window support, including panning, split panes, and zooming
- additional arithmetic modes and user definable drawing modes

- bitmap output (for screens and QuickDraw-style printers)
- Skia primitive output (for Skia remote imaging and Skia printers)
- line and cubic Bézier curve output (for PostScript printers)

Skia is staffed by Cary Clark, Michael Fairman and David Van Brink. Skia's product manager is Laurie Girand. The Skia project is in the Graphics Software Group, headed by Jim Batson. Skia will be shipped to a select group of developers along with the "Big Bang" Alpha System 7.0. Skia will be functional by the "Big Bang" release time frame.

Skia is estimated to be around 300K of object code on any given implementation. To accommodate small memory machines, Skia can be segmented to allow a small, single segment, memory resident footprint; this implementation detail is not visible to the user, and is not required on larger memory machines. Skia also supports disk-based data such as pictures, regions and bitmaps, to accommodate limited memory applications.

Skia does not require hardware floating point support or virtual memory; Skia can, however, benefit from both. Skia does not require the Macintosh memory model, although it is compatible with it. Skia is compatible with Bass outline fonts, and can accept Bass outlines as an internal data type.

Because it reuses the QuickDraw interface, existing applications can be revised to use Skia by including a different interface file when the application is re-compiled or re-assembled. For the application to take advantage of new features, such as rotation, clipping to paths or specifying fixed point co-ordinates, it must make new calls, or pass additional or different parameters to existing calls.

Skia does not attempt to be "pixel-perfect" with QuickDraw. Thus, a thick framed rectangle, polygon and region with the same input co-ordinates affect the same pixels under Skia (where as they may affect 3 different sets of pixels under QuickDraw).

Skia compatible applications:

- interface with Skia through its procedural interfaces only (unlike QuickDraw, which permits the application to interface through changing data structures). This will affect practically all applications.
- anticipate that a few QuickDraw calls are ignored, or their meaning is changed slightly. The application may be affected by Skia's interpretation of a shape's geometry. See the routines section for full details.
- written in Pascal require references to a few data types to be changed.

To make the QuickDraw to Skia transition as painless as possible, an MPW tool will be provided to do as most of the manual editing changes, and to identify parts of the program that are incompatible with Skia.

Note: This document does not include the necessary changes to Toolbox Utilities (fix math), the matrix math package, the Font Manager, the Color Manager, the Palette Manager, the Device Manager or the Window Manager.

---

# THE MATHEMATICAL FOUNDATION OF SKIA

---

## The Coordinate Plane

Skia expands QuickDraw's coordinate plane by expanding the grid coordinates to long integers (in the range of  $\pm 2540699648$ ). Grid coordinates default to fixed point numbers (16.16), defining a range of  $\pm 32768$ .

To help specify fixed-point constants, a simple macro is introduced:

```
#define f(a,b) (((long) (a) << 16) + (b))
```

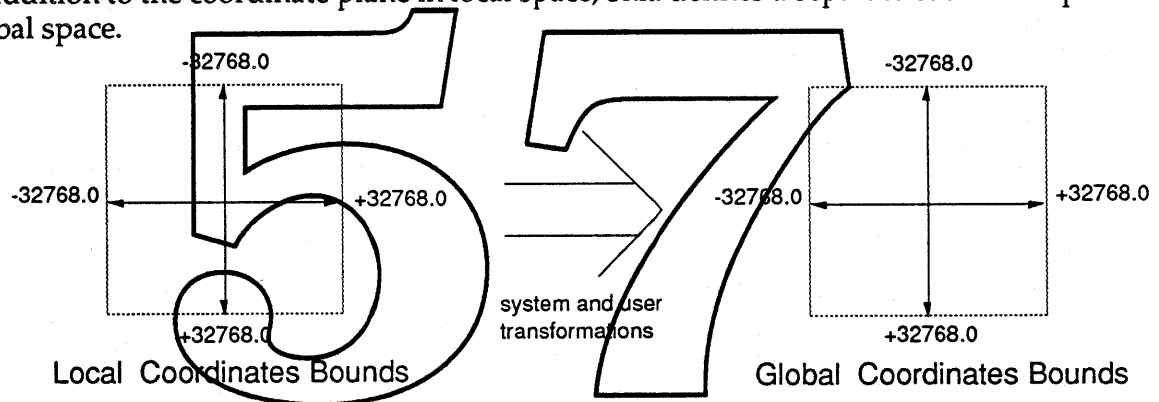
This allows  $f(1,0)$  to mean 1.0.

---

**Pascal note:** there is no equivalent to this mechanism in Pascal. For small integers hexadecimal strings can do the job. For instance, 1.0 and 2.0 are equivalent to \$10000 and \$20000.

---

In addition to the coordinate plane in local space, Skia defines a separate coordinate plane in global space.



Global space is also fixed point; each device visible to Skia occupies space in this fixed point plane. Global space is like the view of the devices presented by "Monitors"; the device's rectangle describes how big a picture or page the device can display, but says nothing of the resolution of the picture. Global space is mapped to device space by a matrix supplied by the device; in device space, 1.0 is normally equal to the size of one pixel. Applications normally work in local space, but will want to know about the properties of global space, and less often, device space.

## Skia Geometric Structures

All geometry is specified to Skia in terms of one or more x-y coordinate pairs, called **control points**. For instance, a line has two control points that define the line segment, and a rectangle has two control points that define opposing corners of the rectangles. Control points define the geometry relative to the other points within the geometric structure.

Skia defines points, rectangles and regions as **geometric structures**, just like QuickDraw. All of Skia's structures are described by fixed-point coordinates instead of integers. Additionally, Skia provides public geometric structures for lines, curves, ovals, polygons and paths. These structures contain only geometry, and say nothing of the position, resolution, color and clipping relative to the structure is drawn.

The geometry engine within Skia allows the user to manipulate and inquire about the geometry of a shape. Skia needs no more information about the shape than the values of the control points to perform these manipulations and inquiries. Skia maintains the highest precision possible when performing geometric manipulations.

For instance, it is possible to define a polygon and then clip it to some other shape, rotate it, and get the rectangle that defines the resulting bounds without any reference to the device world.

## Points

Skia supports QuickDraw points, which have two 16 bit integers, as well a new type, which contains two 32 bit long integers. Skia maintains the QuickDraw definition of the coordinate origin and direction. Like QuickDraw, Skia defines a pixel to be surrounded by grid lines, and for the point to be located on the upper left boundary of a pixel. (See [Inside Macintosh](#), I-140, figure 3.) The format of a Skia point is:

```
typedef struct {
    fixed    x;
    fixed    y;
} point;
```

Note that the order is different from QuickDraw. All new data structures introduced by Skia list the x element first, then the y element.

---

**Pascal note:** this data type is called `skiaPoint` in Pascal.

---

## Rectangles

Skia supports QuickDraw rectangles, which have four 16 bit integers, as well as Skia's fixed point version, which have four 32 bit long integers. The Skia rectangle structure looks like:

```
typedef struct {
    fixed    left;
    fixed    top;
    fixed    right;
    fixed    bottom;
} rectangle;
```

Note that the order of the elements is different from QuickDraw.

## Lines, Curves and Ovals

```
typedef struct {
    point  start;
    point  end;
} line;
```

Note that lines do have direction; that is, reversing the start and end of a line may cause the line to behave differently when drawn. (For instance, dashing begins at the start of a line.)

```
typedef struct {
    point  start;
    point  control;
    point  end;
} curve;
```

The curve described by these three points is a quadratic Bézier. (More about that later.)

```
typedef struct {
    fixed  left;
    fixed  top;
    fixed  right;
    fixed  bottom;
} oval;
```

Note that both the rectangle and oval definitions are identical to a two point left-top right-bottom definition.

## Polygons and Paths

The polygon data structure has been simplified, removing the size and bounding box from the data structure:

```
typedef struct {
    long    vectors;
    point   vector[];
} polygon;
```

---

**Pascal note:** this data type is called `skiaPolygon` in Pascal.

---

This, like the QuickDraw polygon, defines a single contour. Additionally, a new data structure allows more than one contour to be specified within a polygon:

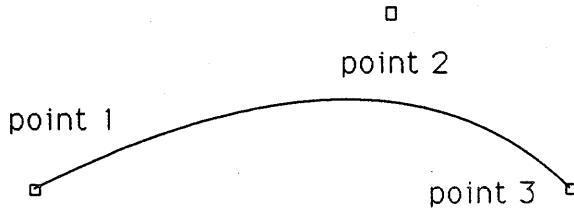
```
typedef struct {
    long    contours;
    polygon contour[];
} polygons;
```

This describes the result of the union of two polygons, for instance. Both polygon and polygons types are provided as a convenience; the polygon type implicitly specifies a single contour.

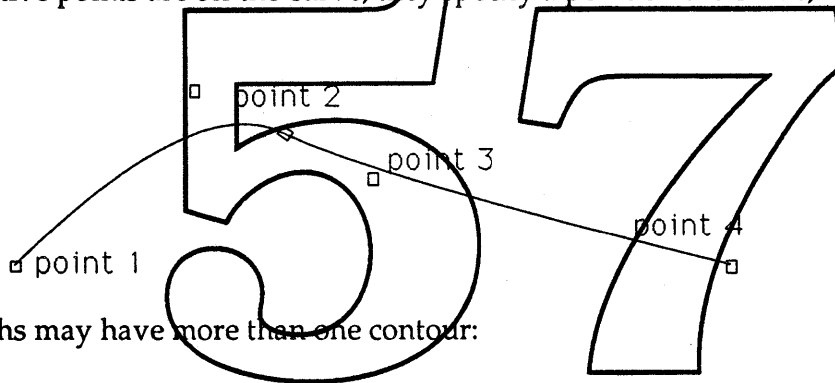
Polygons can have curved segments; when they do, they're called paths.

```
typedef struct {
    long    vectors;
    long    controlBits[];
    point   vector[];
} path;
```

A path is identical to a polygon, except it has an array of bits that specify when a vector point is on the curve or off the curve. Two consecutive points on the curve form a straight line. A point off the curve causes the path to diverge toward that point, and then back to the next point on the curve, in a smooth, continuous sweep, as described above under "Curves".



When two consecutive points are off the curve, they specify a point on the curve, halfway in between them.



Like polygons, paths may have more than one contour:

```
typedef struct {
    long    contours;
    path    contour[];
} paths;
```

For example, some outline characters, such as the letter  $\text{\textcircled{O}}$ , have two contours.

Polygons and paths may be defined by creating a new shape, and adding other shapes to it. Polygons and paths may also be defined by a static array of control bits and vectors.

---

## GRAPHIC ENTITIES

---

Skia introduces a few new terms to the bitmap fray, so here's a summary to help sort things out:

- A **bitmap structure** is a small record which the programmer can directly manipulate.
- A **bit image** is the actual bits pointed to by a bitmap structure.



- A **bitmap shape** is a container passed as to all shape operations (Draw, Rotate, etc.)

## Bit Images

Skia maintains the QuickDraw definition of a bit image.

The values returned by the Toolbox Utility ScreenRes are obsoleted by the multiple device environment. See the section on "Devices".

## Bit Maps

Skia changes the QuickDraw definition of a bitmap, rendering the old bitmap structure obsolete. To make full use of bitmaps, new applications should convert over to the new bitmap type.

Since Skia allows local coordinates to be specified by a transform, Skia introduces a new bitmap type that omits the integer bounding rectangle. (The rectangle is replaced by a clipping shape located in the transform.) Additionally, this bitmap includes the pixel size, that is, the physical bits per pixel.

```
typedef struct {
    char    *baseAddr;    /* pointer to pixels */
    long    rowWidth;     /* width in bytes */
    short   width;        /* width in pixels */
    short   height;       /* height in pixels */
    short   pixelSize;    /* physical bits per pixel */
} bitmap;
```

---

**Pascal note:** this data type is called skiaBitMap in Pascal.

---

Note: For optimal performance, the rowWidth field should be a long multiple, although Skia will work with row widths that are odd word multiples.

The offset allowed by the rectangle associated with a QuickDraw bitmap is kept in the default transform, or the transform associated with the Skia bitmap shape. The Skia bitmap can also specify, through the transform, a rotation, scaling and non-rectangular clipping.

The color table which maps the bitmap's pixel values to RGB colors is kept in the style. See the discussion on Styles, below.

The QuickDraw global, screenBits, should not be accessed since it can not describe a multiple device environment. See the section on "Devices".

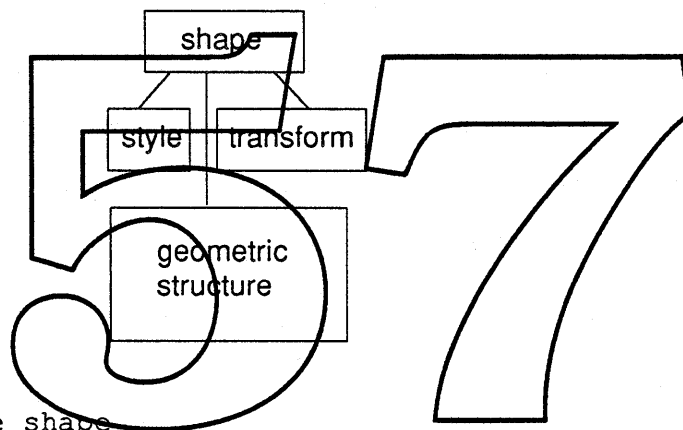
Skia allows bitmaps to be transformed, either as a geometric operation, which produces a new bitmap, or as a drawing operation, which causes the transformed bitmap to be displayed. The section on "Transformations", below, discuss the possible effects on bitmaps.

Skia allows bitmaps to describe the current clipping region. See the section on "Transformations".

## Regions and Skia Shapes

Skia greatly expands the functionality of regions. Skia regions, in addition to be composed of lines, polygons, ovals, roundRects and other regions, can be composed of text, bitmaps, arcs, curves, paths and even pictures. In addition to creating regions by calling routines that draw, Skia regions can be defined by data structures. Skia regions may be parsed and edited. Skia regions can be scaled, rotated and otherwise transformed without loss of information. Skia regions can control the direction of pen movement while drawing to allow shapes to follow a path. Skia regions are generally more compact than QuickDraw regions. Skia regions are unlimited in size, and maybe disk based.

To avoid confusion between the properties of Skia regions and QuickDraw regions, a Skia region is referred to as a **shape**. Shape better describes what a region is; an encapsulation of geometry. A shape contains a geometry structure, and a pointer to a transform and a style.



Shapes are defined by:

```
#define RgnHandle shape
typedef struct {
    long    dummy;
} **shape;
```

Shapes are not necessarily pointed to by handles. A RgnHandle may be a reference value for a shape located in the memory managed by a graphics accelerator. Skia shapes have no public data structure (not even a length or bounding box), so the definition look like:

Note: The internal format for shapes, like all Skia data structures, is not accessible to the application. The data can be parsed and edited procedurally, however.

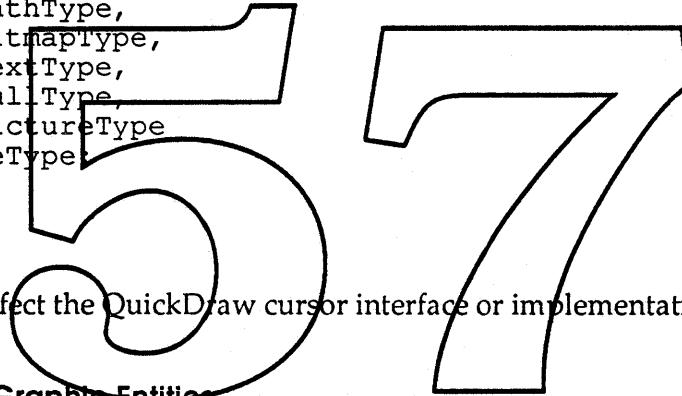
Note: the minimum shape size is not 10 bytes, like QuickDraw regions. Macintosh Memory Manager calls should not be used to allocate, re-size or get the size of Skia data structures.

## Geometric Structures and Graphic Entities as Shapes

Any graphic entity or geometric structure can be encapsulated in a shape, a one-size-fits-all container. To specify that a dots are to be used to dash a line, for instance, the dots must be contained in a shape. A QuickDraw region is exactly the same as a shape; that is, a region contains a series of geometries specified by drawing commands, and the region allows the geometries to be further manipulated or drawn. Shape is used here so that the restrictions contained within old QuickDraw regions do not confuse the Skia user.

Each shape has a type. The permissible types are:

```
typedef enum {
    noType,
    emptyType,
    pointType,
    lineType,
    arcType,
    curveType,
    rectangleType,
    ovalType,
    polygonType,
    pathType,
    bitmapType,
    textType,
    fullType,
    pictureType
} shapeType
```



### Cursors

Skia does not affect the QuickDraw cursor interface or implementation.

### New Types of Graphic Entities

QuickDraw uses special values of rectangles and regions to describe empty areas, or areas that fill the entire global coordinate space. Skia provides the specific shape entities **empty** and **full** to describe these instead. This provides a more compact form for these common constructs, and avoids problems described in Inside Macintosh caused by wide open regions. Skia also provides an inverse operator that allows the space outside a geometry or graphic entity to be operated on, rather than the inside.

### Graphic Entities as Resources

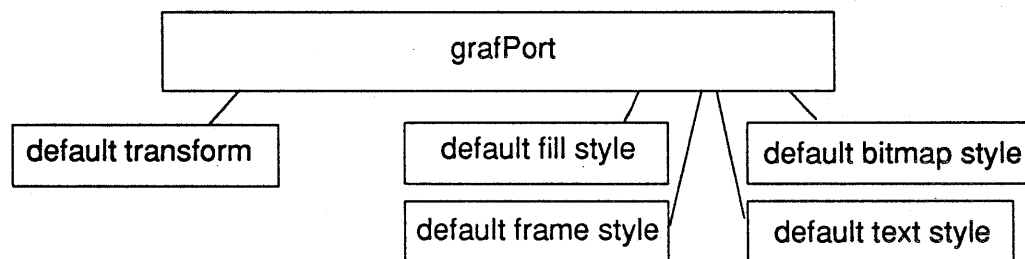
Skia supports drawing and manipulating all graphics types in memory and on disk; this is described in more detail later in the sections on pictures and shapes.

---

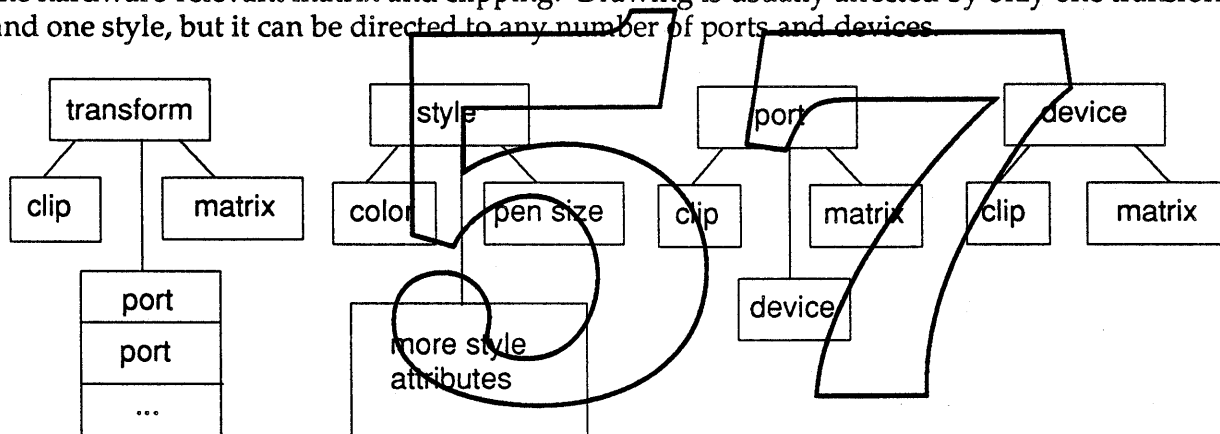
## THE DRAWING ENVIRONMENT

---

A **grafPort** is a collection containing the default transform, the default styles and some other global state.



Skia fragments the QuickDraw **grafPort** into several pieces to allow drawing state to be set with more modularity. These pieces are the transform, which contains a 3 x 3 matrix and a clipping shape, the style, which contains the pen's color, dash pattern, pen thickness and the like; the port, which contains the window-relevant matrix and clipping, and the device, which contains the hardware-relevant matrix and clipping. Drawing is usually affected by only one transform and one style, but it can be directed to any number of ports and devices.



Simplified GrafPort Components

Any shape, when drawn, is first clipped to the transform's clipping shape; the output is mapped according to the transform's matrix; it is then mapped by the port's matrix, then clipped to the port's clipping shape. This is equivalent to the operation of QuickDraw's `clipRgn` and `visRgn`. The drawing can be further transformed and clipped by the device. See "Transformations" for more information.

GrafPorts are defined as:

```
typedef struct {
    long dummy;
} **grafPort;
```

Note that **grafPorts** can no longer be explicitly declared as part of the global or local stack frame in an application. Although the **grafPort** definition is an indirect pointer structure, it is

not necessarily a handle. To create, manipulate, query or destroy a grafPort, a Skia (or QuickDraw) call must be made. Since grafPorts are normally allocated by the Window Manager, this should not affect most applications. *How it affects the Window Manager is the subject of another document.*

The important components of a grafPort, the transform and styles do not have public data structures. Transforms and styles look like:

```
typedef struct {
    long    dummy;
} **transform;

typedef struct {
    long    dummy;
} **style;
```

---

**Pascal note:** the Style data type in QuickDraw is the set of text face variations, so there is a name collision with the Skia style type. The Skia interface for Pascal therefore changes Style to oldStyle; any references to Style in your existing Pascal source will have to be changed for the source to be compiled under Skia.

---

Ports and devices have no public data structure at all. They are referred to by a number, called the port order or device order. This number indicates the order in which Skia draws to successive ports or devices.

A grafPort is a complete drawing environment. It specifies the appropriate default styles and transforms that accommodate QuickDraw compatible drawing. Skia follows the Color QuickDraw model and moves off-screen drawing from grafPorts to devices. For more information, see the section on "Devices".

To explain the transition from Skia to QuickDraw, here is more detail on what has happened to the QuickDraw grafPort's fields:

grafPort field	explanation
device	no equivalent in Skia. This has historically been used only by the printing model, which is changed for Skia. See the section on "Printing".
portBits	equivalent to the device's bitmap.
portRect	roughly equivalent to the bounding rectangle of the clip in the transform when it is thought of as defining the drawable area of a bitmap; it is roughly equivalent to the port's clipping bounds when it is thought of as defining the writable area of a window. Skia's design simplifies the number of clipping bounds QuickDraw provides; for that reason, there is no direct equivalent of a portRect.
visRgn	equivalent to the port's clipping shape.

- clipRgn equivalent to the transform's clipping shape.
- bkPat equivalent to the pattern contained by the default fill style
- fillPat equivalent to the pattern contained by the default frame style
- fgColor, bkColor equivalent to the colors contained by the default styles. All styles contain one or more colors in a color table. QuickDraw drawing modes select how the colors are interpreted.
- colrBit no equivalent. Skia does not support a multiple plane model.
- patStretch no direct equivalent. Skia does support scaling all drawing, including patterns. See "Transforms".
- picSave, rgnSave, have global equivalents, accessible procedurally, that specify polySave whether a drawing operation draws, or is added to a shape, or both.
- grafProcs Skia has nine steps to every drawing operation, and any of the nine can be augmented or replaced by the user. See "Customizing Skia" for more details.

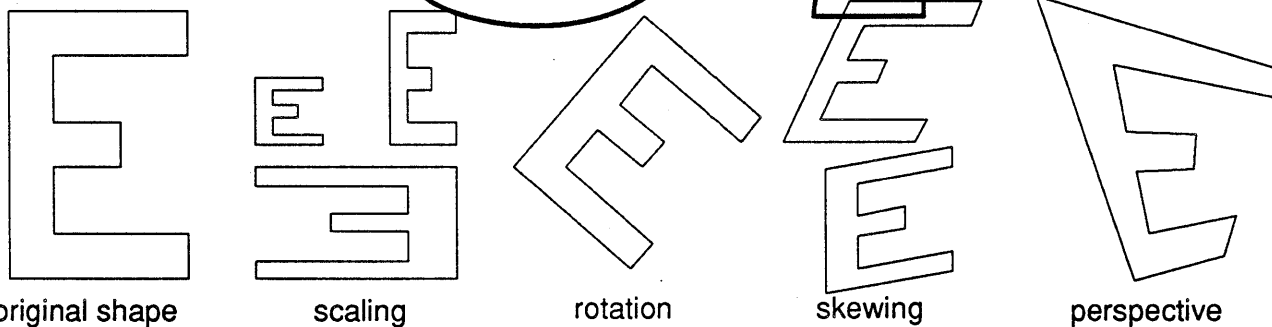
---

## TRANSFORMS

---

A **transform** is a description of the user clipping and matrix transformation to be applied to a shape when it is drawn.

Skia introduces a 3 x 3 matrix that allows, in addition to QuickDraw translation, scaling, skewing, rotation, perspective or any combination.

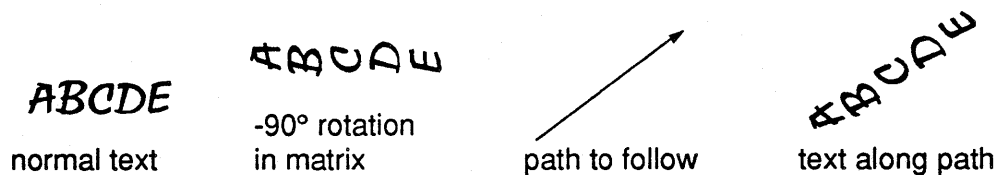


Note that scaling also allows mirroring; all shapes drawn, including bitmaps, can be distorted to be upside-down, right-to-left, or both. This is accomplished by passing negative values for scale factors.

The matrix is a part of the transform. Many transforms can exist at one time, although Skia creates only one transform for all standard drawing to use. The application may create additional transforms, and assign them to particular shapes or groups of shapes.

When text is drawn, the matrix rotates and translates the characters in the text string. Since follow paths also can rotate and translate text, it is important to understand how the two work together.

First, the clipping specified by the transform is applied to the shape. Next, the coordinates within the shape are transformed as specified by the matrix. The shape is then translated to the beginning of the follow path, so that the (0,0) coordinate corresponds to the beginning of the shape. Finally, each point within the shape is fit to the path -- the horizontal value of the point determines how far the shape travels down the path, and the vertical value determines how far the point is positioned perpendicular to the path.



Skia maintains QuickDraw's notion of the local and global coordinate systems. Shapes, style geometries and the transform's clip are specified in the local coordinate system. Ports and devices are specified in the global coordinate system. The transform and view's matrices specify the mapping from the local coordinate system to the global coordinate system.

QuickDraw uses rectangles, such as the portBits.bounds, to specify local coordinates and global coordinates. The introduction of multiple devices make this analogy difficult to support. Skia uses a transformation matrix for the same purpose. In QuickDraw, the upper left corner of the portBits.bounds is the same as the upper left corner of the bit image; in Skia, an identity matrix maps the local coordinate (0,0) to the global coordinate (0,0).

The port's bounds, transformed through the matrix from global coordinates into local coordinates is the equivalent of the grafPort's portRect. The device's bounds, transformed the same way, is the equivalent of the grafPort's portBits.bounds. Since the matrix may specify rotation, skew or perspective, either transformed rectangle is not necessarily rectangular.

The example in Inside Macintosh involving SetOrigin is best stated under Skia by changing the transform associated with the drawing instead. (You'll need to look at the original example to follow this.)

Given that:

```
SetOrigin(90,80)
```

in the QuickDraw chapter (page I-154) is the same as:

```
SetOrigin(gamePort^.portRect.left - 10, gamePort^.portRect.top - 20);
```

then the equivalent Skia call would be:

```
Offset(nil, f(10,0), f(20,0));
```

where the nil parameter causes the default transformation matrix to be affected, and

$f(10,0)$ ,  $f(20,0)$

define the fixed point numbers 10.0 and 20.0.

This would have the effect of changing the matrix from:

1.0	0	0		1.0	0	0
0	1.0	0	to:	0	1.0	0
0	0	1.0		10.0	20.0	1.0

Unlike QuickDraw, changing the local coordinates meaning this way does not change the values inside any shape or rectangle. Although at first this may seem confusing, the intent of Skia is to simplify QuickDraw's coordinate systems and allow more powerful methods of translating one coordinate system to another.

Analogous to QuickDraw, the shapes and clips defined by the user can be thought of to stick to the local coordinate system. The port and device's clips can be thought of as sticking to the global coordinate system, frequently the screen.

QuickDraw's LocalToGlobal and GlobalToLocal convert a point from one coordinate system to the other. Skia's shapes often have many points, so, rather than calling LocalToGlobal for each, more powerful calls are provided to map shapes from local to global and back.

### Using Transforms

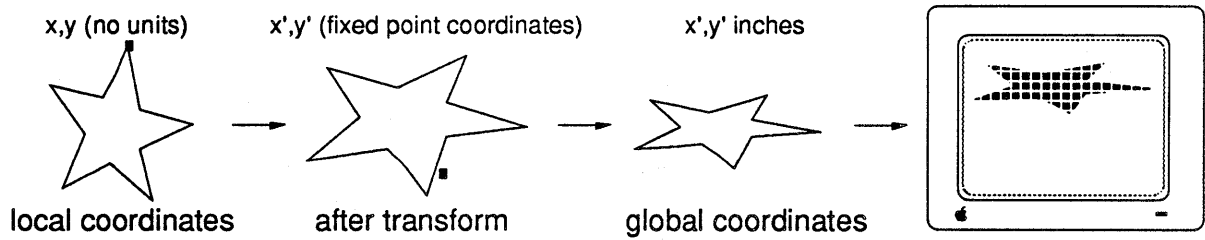
A transform can affect a shape in two different ways: first, the transform can distort the shape's geometry itself. That is, a shape can be rotated to produce a different shape. This class of operation is done by Skia's geometry engine. Second, the transform can affect the way a shape is drawn; that is, the shape can be rotated differently in global coordinates than it appears in local coordinates. This operation is done by Skia's rendering engine. These two capabilities are available in parallel, but Skia does not require the application use either or both. Rather, the interfaces allow the application to choose which manipulation best suits their needs.

### **The Transform's Port List**

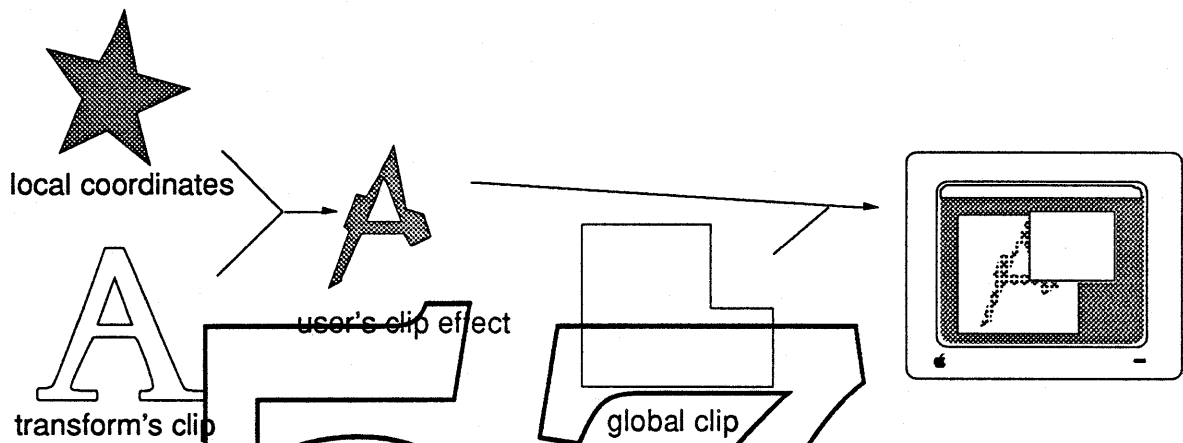
A **port** is an association of the mapping and clipping from local coordinate space to global coordinate space.

A shape is defined in local coordinate space. Local coordinate space is unbounded, and the values of the coordinate serve only to define the shape with respect to itself. A transform gives different shapes meaning with respect to each other. All shapes sharing the same transform are drawn to the same scale. Inside the transform is a structure called the port list. Each port contained in the port list is a mapping of the local coordinate space that shapes are defined in to the global coordinate space that devices are defined in.





The transform, in addition to determining the mapping of local coordinates to global coordinates, specifies the local clip and through the port list, the global clip.



Each transform contains one or more ports in the port list. A port contains:

- a 3 x 3 matrix that defines translation, scaling, skewing, rotation and perspective
- a clipping region that defines the global clip maintained by the Window Manager
- the device that the port causes drawing to go to
- a color table that defines the set of colors that drawing allows (a color filter)
- the port order, that is, when drawing in this port takes place in relation to other ports

A transform typically contains only one port; this makes the drawing most like the classic black and white QuickDraw. Color QuickDraw can draw on more than one device at a time; this is analogous to a transform having several ports, one for each device the window crosses onto.

Skia allows ports to be very flexible.

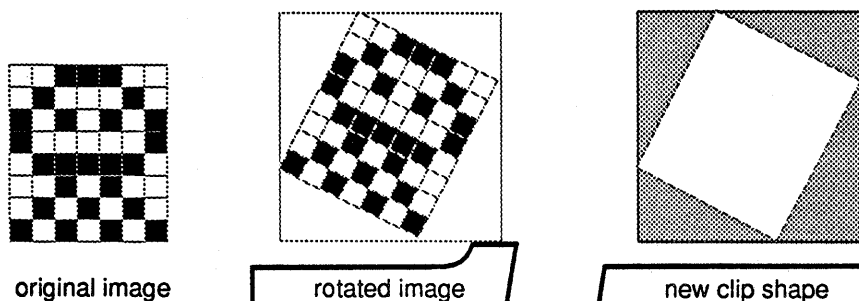
- Any number of ports can be associated by the application (or by the Window Manager) with a transform.
- Ports can be assigned to one or more transforms; each transform maintains its own unique port list.
- Ports may overlap.
- The same device can have more than one port.

This flexibility makes it easy to implement split screens or drawing to simultaneous full page views and editing views.

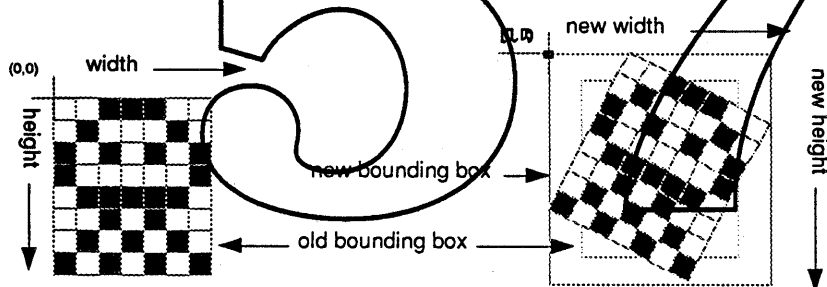
## Transforms and Bitmaps

Whenever a bitmap is transformed, it may be necessary to combine old pixels into new ones. Skia uses a point sampling filter to create a new transformed bit image. The application may specify the filter themselves. See "Customizing Skia" for more details.

The geometry engine allows clipping and matrix operations to be applied to shapes. This has some special implications for bitmaps. For instance, if a bit image is scaled, the block of memory containing the image will grow or shrink (possibly moving) to accommodate the new size. Other direct transformations will additionally cause a new clip shape to be created (or combined with the existing one) which masks out the unused area surrounding the image. The following diagrams show the components generated:



In this case and most others, the bounding box of the new image is not the same size as the old one, and the following size change (expansion in this case) results in something like this:



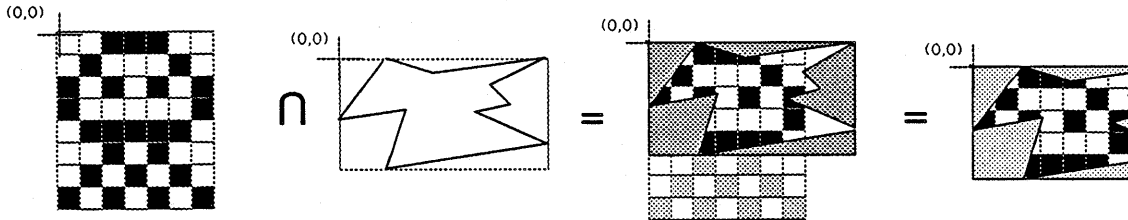
Note: Skia can only re-size the bit images that it allocates. Bit images allocated by the application must be re-sized by the application.

## Operations on Bitmaps

Skia supports performing the union, intersect, difference and xor operation on shapes, including bitmaps. Each operation takes two operands and returns a destination shape. When one of the source operands is a bitmap then the destination shape is always a bitmap. The difference operator ( $A - B$ ) is affected by the order of the operands, and does not allow  $A$  to be a geometry and  $B$  to be a bitmap. All other operations are commutative and, given that one of the operands is a bitmap, always return a bitmap shape.

## Clipping a Bitmap and Bitmap Clipping

Combining a bitmap and a geometry is implemented by combining the bitmap's clip with the geometry after finding the minimum enclosing rectangle of the two. If the bitmap was allocated by Skia, the size of the bitmap may be changed by the operation. Clipping a bitmap creates a irregularly-bound subset of the original bit image:



If the bitmap is clipped directly, then the bit image is modified so that the bounding box of the bit image is the size of the resulting clipping shape. The transform associated with the bitmap then points to the shape that describes how the bit image is clipped. The clip's bounds always fit inside the bit image's bounds. The hidden pixels are still in the bit image, untouched; the clip in the transform pointed to by the bit image masks them out.

The advantage of directly manipulating the bitmap data is that re-drawing the transformed image is much faster than transforming it as it is drawn. On the other hand, successive transformations are best done at drawing time, so that each time the image is drawn the original source image is used, minimizing the information loss. Transforms applied directly to the bit image are not necessarily reversible; some information may be lost.

When two bitmaps are combined, only their clips are combined. These operations only affect the bitmap's clip and do not change the values of the pixels.

When a transform's clip is set to a bitmap, it affects all shapes drawn through it by masking them with a black-and-white version of the image, where black bits let shapes drawn pass through and white bits obscure them. This monochrome mask is maintained as a separately in the transform.

Skia does not support clipping a shape containing a geometric structure to a bitmap, returning a new shape. Skia also does not support turning a bitmap into a geometric structure, a path made up of many small rectangles.

## Devices and gDevices

A device is a software description of all or part of global coordinate space.

Skia integrates the Color QuickDraw device model into the overall coordinate transformation. Like all Skia data structures, the device structure is not directly accessible, but can be examined and changed through a procedural interface.

Each device contains:

- a clipping shape in global coordinate space (which is normally rectangular)
- a 3x3 matrix which specifies the resolution and position relative to other devices
- a color table which describes the hardware characteristics as currently configured
- a bitmap

Skia extends the device model. Skia devices:

- exist on all machines supporting Skia (not just under Color QuickDraw).
- may allow support for a secondary screen, for flip-frame animation.
- can be used as off-screen bitmaps.

## Off-screen Bitmaps

Since transforming at draw time can be slow, Skia provides a mechanism for recording the intermediate image to assist in routine tasks such as updating a window. A new device can be created that includes a frame buffer that is not connected to any hardware display; this frame buffer is an off-screen bitmap. Off-screen bitmaps typically mirror actual hardware devices, but are not required to do so.

To draw into an off-screen bitmap, the user associates a port to map local coordinates into the newly created device's global coordinates. Typically, the application would specify that the global coordinates do not overlap with any hardware device's coordinates, so that drawing to the off-screen bitmap has no visible effect. To copy the off-screen bitmap to a visible screen, the user creates a bitmap shape from the new device and draws it.

In addition to using an off-screen bitmap for double buffering, Skia supports using off-screen bitmaps to backup completely or partially obscured windows. To backup the portion of a window obscured by a menu, for instance, the application (or the Menu Manager, if extended) would create a new port that directs the drawing intended for the obscured portion to an off-screen bitmap. (In this example, how the menu obscures the window beneath it is left up to the Window Manager.) When the menu is released, the area under the menu can be restored from the off-screen bitmap.

---

## SKIA STYLES

---

A style is the collection of state that applies color, pen thickness, dashing, patterns and so on when a geometry is drawn. Four default styles are created when Skia is initialized, and new styles can be created explicitly or by applying style operations to shapes. For instance,

```
shape myRectangle = NewRectangle(boxData);
SetColor(myRectangle, red, nil);
```

causes myRectangle, initially assigned the default fill style, to be assigned a newly created style that differs from the default fill style in that the first entry in its color table is red.

```
SetStyle(myOval, CurStyle(myRectangle));
```

causes myOval to share the same style as myRectangle.

```
SetPen (CurStyle (myRectangle), f(2,0));
```

sets the pen thickness for the style shared by myRectangle and myOval to 2.0.

```
SetPen (myRectangle, f(3,0));
```

sets the pen thickness for the style pointed to by myRectangle to 3.0. It will duplicate the style since the style was shared by between myRectangle and myOval.

```
Dispose (myOval);
```

```
Dispose (myRectangle);
```

in addition to throwing away the oval shape and rectangle shape throws away the red style with a pen thickness of 2.0 and the red style with a pen thickness of 3.0, since both style owner counts went to zero. Every style, including the default styles, have owner counts, that is, how many shapes and global references point to the style.

### Pen Characteristics

Skia's style specifies the graphics "pen". Many styles can exist at one time; normally one each exists for all frame and fill calls, as well as one each for bitmap and text drawing calls. The user may create additional styles, and associate them with a shape or a group of shapes.

Skia's pen is not rectangular, and is centered on the shape being drawn, instead of hanging below and to the right. Skia's rounding rule is down and to the right, so for halfway cases, Skia will choose the same pixel as QuickDraw.

Skia's pen can have a location, as set by the Move and MoveTo calls; normally, the pen follows the shape, and does not need a separate location. A style can also have a path to follow; in this case, the shapes are advanced along the path as they are drawn.

The pen size is one-dimensional, and specifies the thickness of the line or curve drawn. The drawing mode is treated the same as in QuickDraw. The pen's pattern is treated similarly to QuickDraw; additionally, Skia provides a one dimensional dashing shape. A dash shape, unlike a pattern, is rotated to be parallel to the pen as it moves down the line or curve and are scaled to the line thickness.

Skia provides control for what is drawn when the pen turns a corner. The drawing at the corners are called joins. Skia also provides control for what is drawn at the beginning or ending of a line segment that does not close a shape; these are the start cap and end cap.

As in QuickDraw, the style mode determines how the shape affects the pixels already in the bit image, and the style visibility determines if the shape is drawn or hidden.

## Text Characteristics

Skia uses Bass outline scaling to generate shapes to define characters. If the Bass font is not available, the bitmap font is used instead. Skia takes the character glyph index and horizontal and vertical offsets determined by the Layout Manager, and draws the character that has been grid-fitted to the device by Bass.

Skia provides for clipping to text as well as filling and framing it. Skia supports the QuickDraw faces Bold, Italic (Oblique), Underline, Shadow, Outline, Extend and Condense as well as providing an extensible text face mechanism.

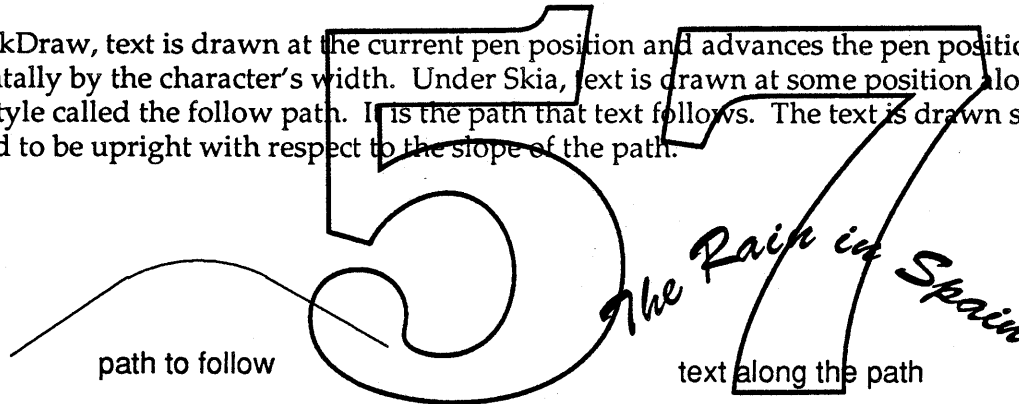
Since text are shapes, text can be dashed, arbitrarily transformed, filled with a pattern, or algorithmically distorted in any way that a shape can be affected.

Characters and strings may be drawn fit to a bounding box, or follow a path.

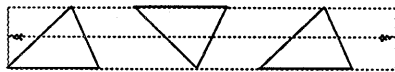
Skia defers the script and font naming and interface issues to Bass and the Layout Manager.

## Follow Path

In QuickDraw, text is drawn at the current pen position and advances the pen position horizontally by the character's width. Under Skia, text is drawn at some position along a path in the style called the follow path. It is the path that text follows. The text is drawn so that it is oriented to be upright with respect to the slope of the path.



(Jumping the gun a little bit, since dashing is discussed below:) The follow path works pretty much like dashing in reverse; rather than fitting the dashing shape to the path to be drawn, the follow path treats the shape to be drawn as the dashing pattern and fits it to the path. Follow paths are different from dashing in that the shape does not repeat, it is not scaled by the pen thickness, and it is not clipped to the thickness of the path. Follow paths also keep track of the last shape drawn, so that successive shapes may advance down the same path if they share the same style.



several shapes drawn



pattern to follow



shapes along path

## Algorithmic Text Faces

Skia text faces are algorithms that, given a shape (usually a path) and xy advance, can produce a new shape and a new xy advance. Applications can add new faces, but they are only available system wide if they are installed as an executable resource in the system file. Pictures can contain these code resources as well, so that the face can be ported to other systems capable of executing it. If the face is not available, the system defaults to plain text.

Text faces can apply to any geometry, but only fonts can be affected by Bass to take advantage of geometric distortions to make a face look best. Faces may not be applicable to bitmaps (depending on the face implementation).

Text faces can have any number of parameters, and can specify their ranges and the default setting. The parameters' settings can be overridden by the font and by the application.

The current QuickDraw faces are interpreted as follows:

Face	Meaning	Parameter	Default Setting
Condense	horizontal compression	reduced width	1/12 * point size
Extend	horizontal expansion	added width	1/12 * point size
Bold	fattens the outline	outset	1/12 * point size
Italic	horizontal skew (oblique)	angle of skewing	10.0°
Underline	line(s) under the character	pen thickness	1/12 * point size
		number of lines	1
		spacing between lines	1/12 * point size
		position of line	1/2 baseline + descender
Outline	path is framed	overwrite character?	false
Shadow	character is drawn twice	pen thickness	1/12 * point size
		character offset	(-1/12, -1/12) * point size
		color of shadow	entry 1 of the color table

The application of these algorithms for old TextFace calls are in the order shown above. Skia calls allow the algorithms to be applied in any order, and for the same algorithm to be applied more than once. A structure in the style, a face list, carries the face order and parameters.

Text faces are implemented as named resources. New faces can be added by adding new resources to the system folder or the picture, and can be selected by a menu through the Resource Manager call AddResMenu.

The resource contains a list of machine types, and an executable code block for each machine type. The code includes a data table of preferences, including whether the style operates on single characters or entire strings. The style takes a shape (which may be a list of shapes), a destination shape, the advance width/height table, and a variable length parameter list (which may be 0). See "Customizing Skia", below.

## Text Size and Space Extra

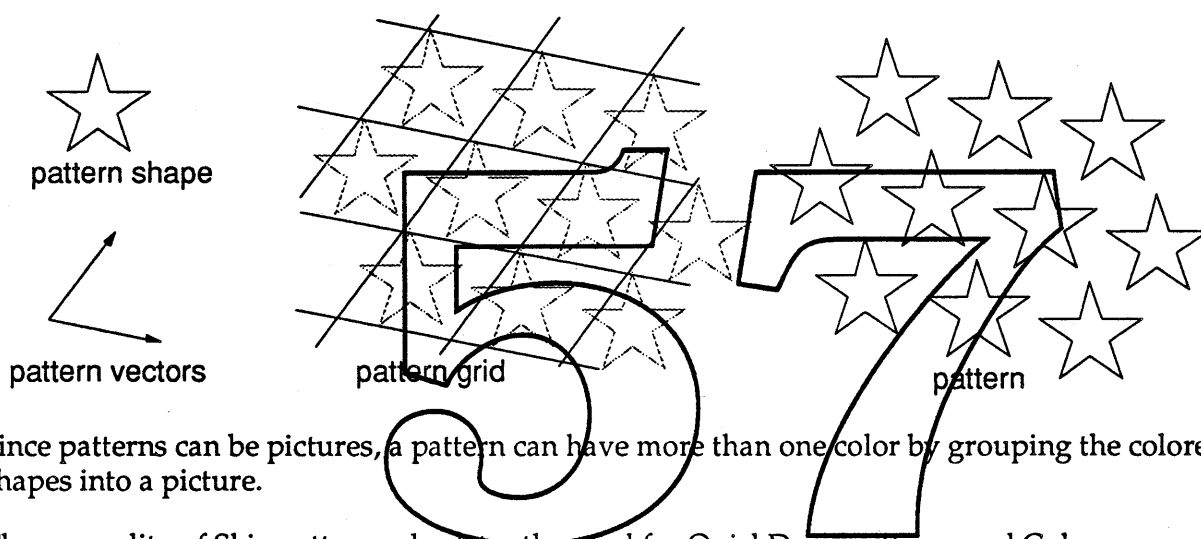
As in QuickDraw, the size in the text style is specified in typographic points, approximately 1/72 of an inch.

*Space extra and character extra are under the jurisdiction of the layout manager?*

## Patterns

Patterns, as defined under Classic QuickDraw, approximate color, such as gray, halftones and repeating bitmaps of a set size. Under Color QuickDraw, patterns can be more flexibly sized and colored, but are still bitmaps. Patterns under Skia refer only to arbitrary repeating shapes, where the shape can be a bitmap.

A Skia pattern is not limited to a particular size, as in QuickDraw, or a particular multiple, as in Color QuickDraw. A pattern can be defined to be resolution independent. A pattern consists of a shape to be repeated, and two vectors, described by a two points, that describe how the patterns are repeated. The vectors can be thought of as describing a grid of parallelograms, where the pattern is drawn at every intersection.



Since patterns can be pictures, a pattern can have more than one color by grouping the colored shapes into a picture.

The generality of Skia patterns obsoletes the need for QuickDraw patterns and Color QuickDraw pixel patterns.

---

## GENERAL DISCUSSION OF DRAWING

---

Drawing occurs:

- once per port defined by the current transform
- inside the bit images defined by the current set of devices, in the coordinate system defined by the concatenation of matrices
- always within the intersection of the device and port clip shapes, also intersected by the transform's clip shape mapped into global coordinate space



- at the coordinates defined by the shape, or, in the case of some line and curve calls, between the former pen position and the new pen position; in either case, the coordinates are transformed by the concatenation of the transform, port and device matrices.
- always with the current style's color and mode, and usually with the style's pattern, dashing, pen size, pattern, caps, and joins.

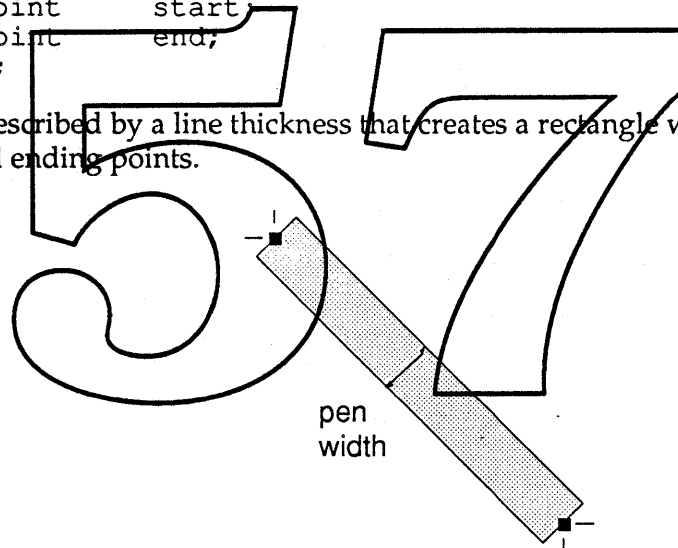
Skia enhances polygons to allow the specification of more than one contour; each contour can be drawn in a different color. Skia also enhances bitmaps and text, allowing them to specify clipping areas. In addition to QuickDraw's lines, shapes and text, Skia adds curves (defined as quadratic Béziers) and paths (polygons with curved segments).

## Lines

Lines are defined by two points, in two different ways: as always, they can be defined by the current pen location and the destination location. Or, like rectangles, they can be specified by the two points that make up the line structure:

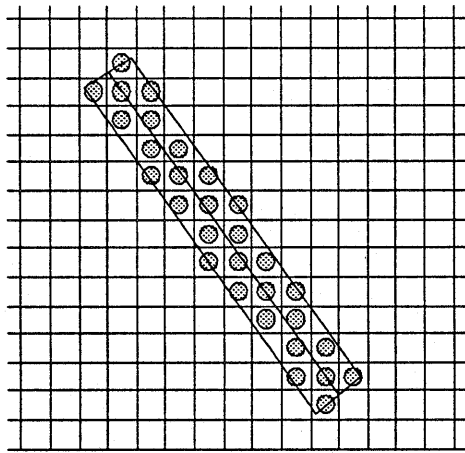
```
typedef struct {
    point    start;
    point    end;
} line;
```

Skia lines are described by a line thickness that creates a rectangle whose sides are centered on the starting and ending points.



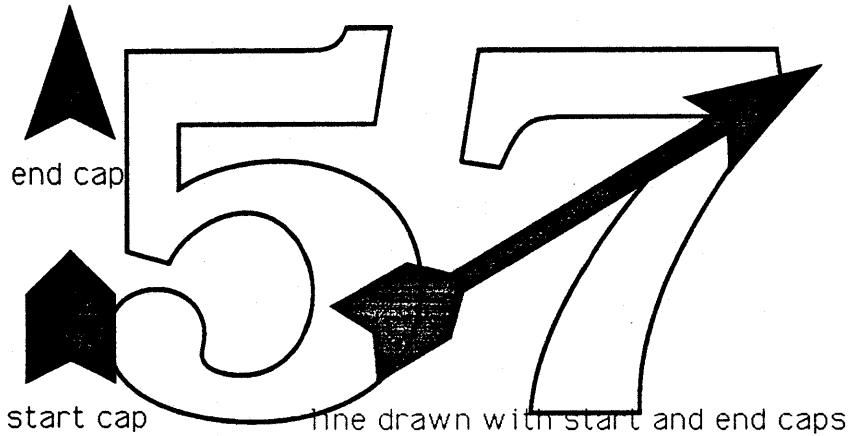
Unlike QuickDraw, Skia does not cause anything to draw if a line's starting and ending points are the same. Skia only turns on the pixels whose centers are surrounded by the rectangle.

57

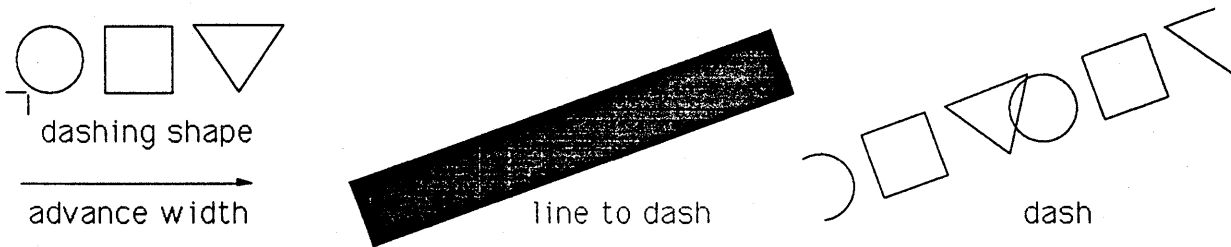


Pixels affected by a line draw

Lines can have caps; a line cap is a shape drawn at each end of the line. The shape is scaled by the pen thickness, and rotated by the angle at which the line is drawn.

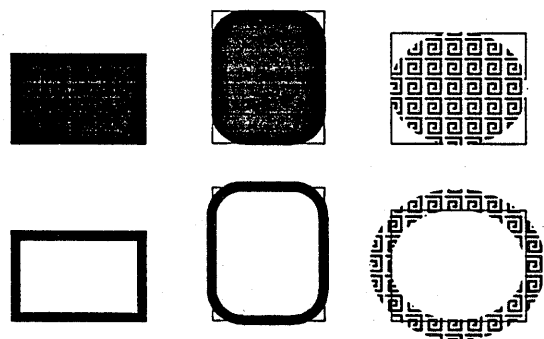


Lines can be dashed; a dashing shape is drawn, clipped to the line, advanced by a specified width.



**Filled and Framed Shapes**

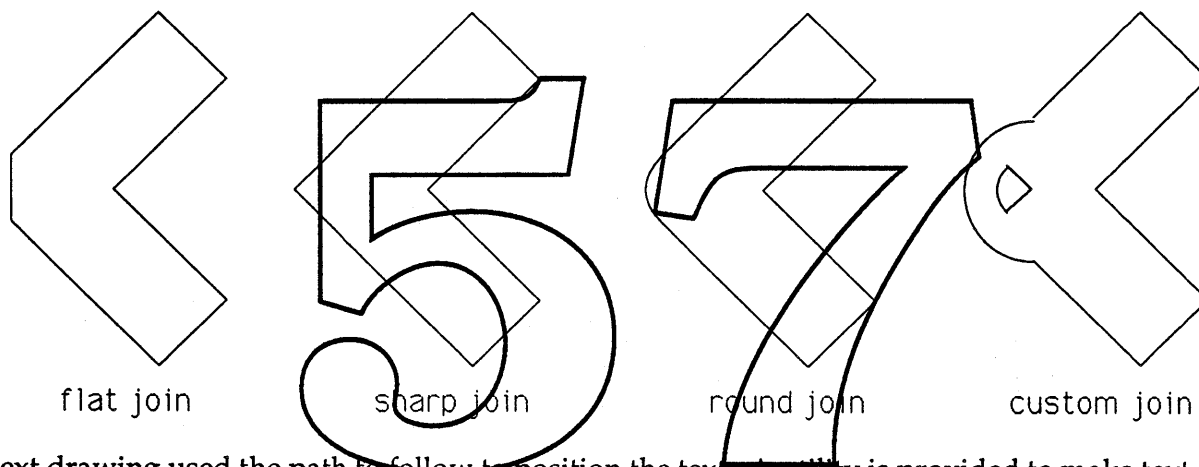
Skia allows the frame of any shape to be inside, centered on, or outside the bounding rectangle:



Solid Shapes and Framed Shapes

Skia causes the outline of every shape to be drawn within the shape's bounds (if desired) without exception.

A corner of a shape, can be drawn using a standard join, or a join shape.



Text drawing used the path to follow to position the text. A utility is provided to make text fit a bounding box. Unlike QuickDraw, any drawing in Skia can be specified to advance the pen location. The pen follows a path, pointed to by the current style, that describes the current pen position and direction. The distance the pen is moved down the path is called the advance width. Fonts include default advance widths for characters. Shapes add the spacing information in the style to the horizontal extent of their bounding box to determine their advance width. The application may intercept and tailor advance widths as text and shapes are drawn.

**Transfer Modes**

Each style, whether the style is used to draw lines and shapes, text, or bitmaps, has a mode. The eight QuickDraw pat modes are identical to their source counterparts. Six of the eight QuickDraw source modes turn into copy, while the two xor modes turn into the Color QuickDraw hilite mode. The copy, or, bic and not modes affect how the color table assigns colors to indices.

**Transfer mode Action**

copy	use color table to color destination
or	use only contours with a color index = 0 to color destination (with color 0)
xor	exchange all occurrences of color index = 0 with color index = 1 (hilite)
bic	use only contours with a color index = 0 to color destination (with color 1)
notCopy	use color table to color destination (invert color table entries 0 and 1)
notOr	use only contours with a color index = 1 to color destination (with color 0)
notXor	exchange all occurrences of color index = 0 with color index = 1 (hilite)
notBic	use only contours with a color index = 1 to color destination (with color 1)

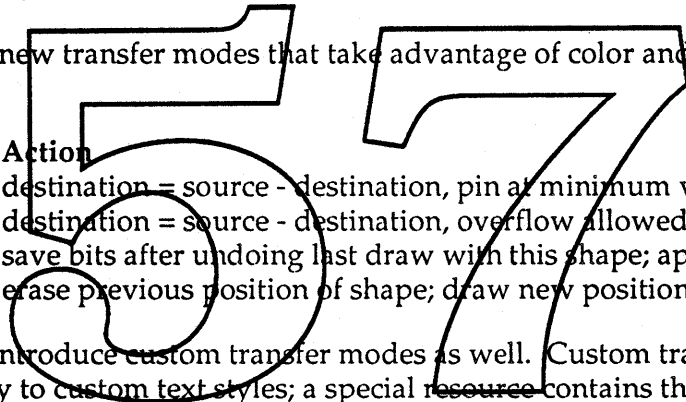
Normally, an application would use only copy, and would change the color table in the style to change the color of the shapes drawn.

Xor is special because it does not necessarily provide the expected results when used on a destination bit image which is more than 1 bit deep; for that reason, the hilite mode is used instead. Hilite is implemented with the xor instruction in 1 bit, providing the same performance as the QuickDraw srcXor mode.

Skia supports the Color QuickDraw arithmetic modes addOver, addPin, subOver, subPin, max, min, transparency and blend.

Skia introduces some new transfer modes that take advantage of color and reduce the need for off-screen bitmaps.

<b>Transfer mode</b>	<b>Action</b>
reverseSubPin	destination = source - destination, pin at minimum value
reverseSubOver	destination = source - destination, overflow allowed
saveAndCopy	save bits after undoing last draw with this shape; apply normal copy
eraseAndCopy	erase previous position of shape; draw new position in copy



The application may introduce custom transfer modes as well. Custom transfer modes are implemented similarly to custom text styles; a special resource contains the transfer routine, it is found by name, and is called by Skia directly. The transfer routine is passed a horizontal slab of pixels to draw, and has access to the device's bit image. This is discussed in greater detail under "Customizing QuickDraw under Skia".

**Drawing in Color**

Skia maintains the Color QuickDraw RGB color model. Like all other Skia data structures, the color table has no public fields:

```
typedef struct {
    long dummy;
} **colorTable;
```

Color tables contain one or more colors, defined by either an index or a 48 bit RGB specification, exactly the same as QuickDraw:

```

typedef struct {
    unsigned short red;
    unsigned short green;
    unsigned short blue;
} rgbColor;

typedef struct {
    unsigned short index;
    rgbColor      rgb;
} colorSpec;

```

Color tables for styles specify color selection. Color tables for ports specify color filtering. Filtering specifies the portion of the RGB cube that is allowed for drawing, and whether drawing outside of the filter is ignored or mapped within the filter. Color tables for devices specify hardware color availability. Color tables contain:

- an optional color table type (direct, fixed or indexed; point, line, plane or volume)
- one or more RGB or index (or both) specifications
- optional color attributes per color (courteous, dithered, tolerance, explicit, animated)

Color tables for styles are interpreted as desired (or actual) points within the RGB color cube. Color tables for ports are interpreted as either desired points or desired color lines, planes or volumes described by their vertices.

### Color in Bitmaps

A bitmap's pixel values are mapped to RGB colors by the color table in the default style. The color table represents the exact set of colors for the image and are matched to each device's actual color capabilities through index tables, one for each device the shape intersects. (For bitmaps with 16 or 32 bits per pixel, the color table is optional.)

The style also contains:

- the transparent pixel value
- whether the bitmap is indexed or direct

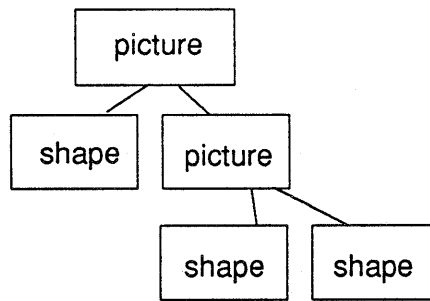
Note: Bitmaps of 16 and 32 bits per pixel are assumed to always be direct; bitmaps of 1 to 8 bits per pixel are assumed to always be indexed.

---

## PICTURES

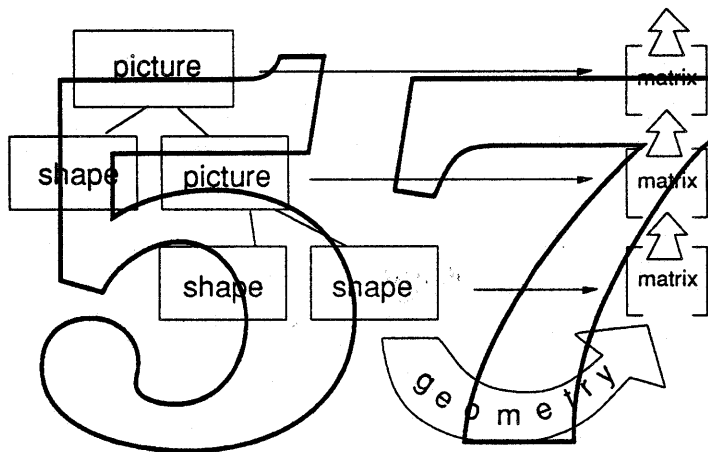
---

Skia enhances pictures by making them editable and parse-able. The picture can be thought of as a list of entries, where any entry may be a shape, or another list. In this way a hierarchy of drawing may be built up. Like QuickDraw pictures, Skia pictures allow one program to draw something defined in another program without having to know about what's being drawn.



In addition to the QuickDraw method of picture definition, Skia allows shapes to be added to a picture, inserted into or extracted out of a picture. Skia allows pictures to be added to pictures, and for each picture's transformation to be respected, no matter how deep the nesting goes.

Every element in a picture can have a transform and a style associated with it. The transform is concatenated with the picture's transform; the style overrides the picture's style. The same shape can be added to the same picture more than once, and each instance can have a unique or shared transform or style, or none at all.



The picture can use the transform's matrix to specify the translation and scaling as an alternative to using the picture's frame and the rectangle passed to DrawPicture.

Pictures have no size limit, and may be resident on disk when they are edited or drawn. Procedural parsing eliminates the need for making the internal format of a picture public.

Comments are special entries in a picture; they are enhanced in that they may include code that is executed when the picture is drawn.

---

## SKIA WITHIN AN EXISTING QUICKDRAW APPLICATION

---

Including "skia.h" causes the application to use Skia when it is compiled ("skia.p" if Pascal, or "skia.a" if assembly). An application may have portions that call QuickDraw directly, and portions that only use Skia, as long as those sections are in different source files. The data created by one portion won't be exchangeable with the other, however.

For example, an application only interested in adding rotation to drawing can do so by:

- Including "skia" where appropriate in the source header files
- Running the appropriate MPW tools to replace accesses to QuickDraw structures with QuickDraw and Skia routine calls
- Adding a new call to rotate the default transform

This application would be interested in reading the section on "Operations on Shapes and Transforms", below, but wouldn't have to read or know anything else.

*QuickDraw compatibility will be expanded in this section as the Skia project develops.*

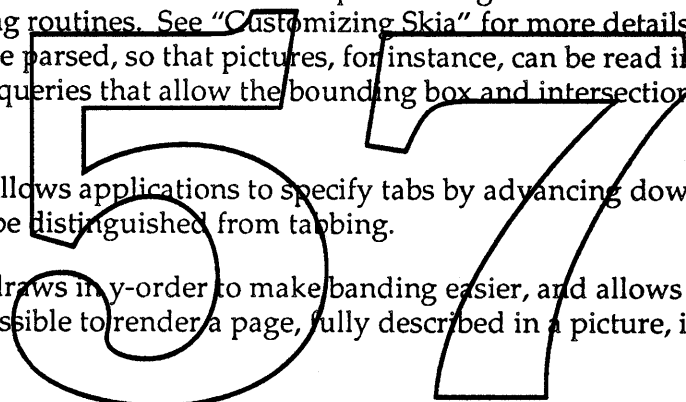
## Printing

Skia maintains device independent descriptions of all geometries and colors that can be drawn; this can make the Printing Manager's job easier by reducing the amount of guessing necessary to produce the best looking output.

Like QuickDraw, Skia allows all calls to be intercepted through a mechanism similar to the QuickDraw bottlenecking routines. See "Customizing Skia" for more details. Further, Skia allows its structures to be parsed, so that pictures, for instance, can be read in any order. Skia also provides geometry queries that allow the bounding box and intersection of objects to be determined.

The style's follow path allows applications to specify tabs by advancing down the path; thus, Move and MoveTo can be distinguished from tabbing.

Internally, Skia always draws in y-order to make banding easier, and allows combining operations to make it possible to render a page, fully described in a picture, in a single pass, from top to bottom.





---

## SKIA ROUTINES

---

### About the Skia Routine Calling Convention

Skia is different from QuickDraw in that it does not allow the user to read or write the internal data structures directly. Thus all Skia calls have routines that set the data structures, as well as return their current values. These Set and Cur calls have a convention by which the parameters are passed, and how Skia interprets nil or 0 passed in the place of an expected parameter.

Many calls that change the contents of a Skia data structure are of the form:

```
void SetThing(thing affected, params...); /* this is not a real call */
```

If the thing affected is intended to be a transform:

- If the thing affected is nil, then SetThing affects the default transform.
- If the thing affected is a shape, then SetThing affects the transform associated with the shape.
- If the parameters are nil or zero, then that part of the transform is reset to its initial condition.

The routine SetTransformClip, below, has examples of these rules.

If the thing affected is intended to be a style:

- If the thing affected is nil, then SetThing affects all relevant default styles.
- If the thing affected is a shape, then SetThing affects the style associated with the shape.
- If the parameters are nil or zero, then that part of the style is reset to its initial condition.

If the thing affected is intended to be a shape:

- If the thing affected is nil, no action occurs and a warning is posted.
- If the thing affected is a style or transform, an error is posted.
- If the parameters are nil (where nil is inappropriate), a warning is posted.

If the thing affected is intended to be a port or a device:

- If the thing affected is 0, negative, or out of range, no action occurs and a warning is posted.
- If the parameters are nil, then that part of the port or device is set to its initial condition.

Calls that return the current contents of a Skia data structure are of the form:

```
thing CurThing(thing queried, params *...); /* not a real call */
```

These calls fill in one or more parameters from thing queried. The information stored in the pointer passed after the thing queried is also the function result.

- If the thing queried is intended to be a style or a transform, then nil queries the logical default, and passing a shape queries the style or transform associated with the shape.

- Passing nil for a parameter defeats filling in the parameter pointer. This will return nil and post a warning if the first parameter is nil and the routine returns a shape, transform or style.
- If the parameter is a shape pointer and non-nil, the indicated shape is reused; if nil, the shape is allocated.

The routine CurClip, below, has examples of these rules.

## Error Routines

Skia provides warning and error routines for all QuickDraw calls. A warning is generated when Skia detects a nonsense or do-nothing call. An error is generated when Skia detects an unexpected or out of range value within a data structure, or when a system call like memory allocation fails. Skia detects implementation restrictions and out of range or nil parameters, but can be misled by faulty data caused by illegally accessed or damaged internal structures.

Most Skia routines can post warnings; fewer post errors. A call to a routine will generate only one error or warning, even if more than one error could have occurred. Warnings can be safely ignored by an application, although warnings can help debug applications. Errors are less easy to ignore, and some errors are unrecoverable.

Warnings indicate that although the routine did complete successfully, the data present suggests that the call should not have been made at all, or that the desired result may not have been produced. For instance, drawing an empty shape posts a warning. Calling a routine with a nil pointer when it is not expected posts a warning.

```
error Error(shape *guiltyParty);
```

Error returns the error code of the last routine to post an error. Errors are sticky; that is, the error code is not cleared by the successful completion of a call. Only InitSkia and Error clear the error code. If a non-nil pointer is passed, Error also returns a pointer to the shape was passed to the call that generated the error, or nil if none.

```
warning Warning(shape *guiltyParty);
```

Warning works the same way as Error. Error and Warning differ only in the severity of the problem. Warnings can be ignored, but errors are less likely to be recoverable.

```
void IgnoreWarning(warning warningNo);
```

IgnoreWarning specifies that if the warning indicated is encountered, no warning is posted. The main use for IgnoreWarning is to ignore intentional errors while debugging. IgnoreWarning is also used internally to prevent warnings internal to Skia from interfering with user warnings. IgnoreWarning saves warning numbers in a limited size queue, so that up

to 16 warnings can be ignored at one time. Overflowing the queue does not itself produce a warning; it merely overwrites the oldest warning to ignore.

```
void ErrorProc(void (*userFunction)());
```

The routine passed to ErrorProc is called if an error is encountered. The error routine is passed the offending shape, if any, and the error code. Passing nil to ErrorProc removes the error routine.

The error routine passed must be of the form:

```
void MyErrorProc(shape guiltyParty, error errNo);
```

Note: without an error routine, Skia will cause the application to quit, via a call to ExitToShell, when an error is encountered. Even with an error routine, the caller will not always continue to execute. Most callers will return to their caller (generally the application), but some callers, such as attempts for memory allocation, will cause the application to quit even if an error routine is in place. It is up to the error routine to clean up the stack, correct the error condition and restart the call if it is desirable to do so.

```
void WarningProc(shape (*userFunction)());
```

The routine passed to WarningProc is called if a warning is encountered. The warning routine is passed the offending shape, if any, the warning code, and the default shape or value that would have been returned. Passing nil to WarningProc removes the warning routine.

The warning routine passed must be of the form:

```
warning MyWarningProc(shape bad, warning, shape default);
```

Note that the warning routine must return a function result. A valid warning routine might look like:

```
shape MyWarning(shape, warning, shape);
static shape MyWarning(aShape, errCode, defaultReturn)
shape aShape, defaultReturn;
warning errCode;
{
    /* ... your warning code here */
    return defaultReturn;
}
```

The default shape will be nil if the calling function expects to return an integer value.

Like errors, warnings cause the call which generated the warning to exit to its caller. Unlike errors, warnings return valid data for the thread of execution to continue.

## Initialization

```
void InitSkia(void);
```

InitSkia allocates and initializes all Skia data structures the first time called. It re-initializes all Skia data structures each additional time called. InitSkia additionally notes on the Macintosh II if the machine is in 32 bit mode; if so, Skia assumes that all future Skia calls within that application will be made in 32 bit mode.

```
void ExitSkia(void);
```

ExitSkia de-allocates all Skia data structures allocated by InitGraf and, if appropriate, switches back to the addressing mode present when InitGraf was called.

## Skia Shape Routines

```
shape New(shapeType);
```

New creates a new shape of the specified type, and assigns it the default transform and the default fill style. The shape defaults to filled.

Note: Since no geometry parameters are passed, New make sense only for geometry-less shapes of type emptyType and fullType. Other types will post a warning, since their geometry is undefined.

```
shape New2(shapeType, fixed, fixed);
```

New2 creates a new shape of the specified type, and assigns it the default transform and the default frame style. The shape defaults to framed.

Note: This is intended for points; other types will post a warning, since there is either too little or too much information to define their geometry.

```
shape New4(shapeType, fixed, fixed, fixed, fixed);
```

New4 creates a new shape of the specified type, and assigns it the default transform and the default frame style. The shape defaults to framed.

Note: This is intended for lines, rectangles and ovals; other types will post a warning, since their is either too little or too much information to define their geometry.

```
shape NewMany(shapeType, long count, ...);
```

NewMany creates a new shape of the specified type, and assigns it the default transform and a default style. The default style for points, lines, arcs and curves is the frame style; for

rectangles, ovals, polygons, paths and text the default style is fill style; for pictures, the default style is set to nil; and for bitmaps the default style is the bitmap style.

Note: Any type that specified with too many or too few parameters will post a warning. A picture may have only shapes as valid parameters; other parameters will post an error. Care must be exercised when passing parameters in this way to paths or polygons to ensure that the contour count and the vector count match the number of parameters passed.

```
void Dispose(shape);
```

Dispose throws away the indicated shape, and decrements the owner counts of the shape's style and transform.

```
void DisposeAt(shape *);
```

DisposeAt works just like Dispose; it allows the shape variable declared to be assigned nil. As an example:

```
shape myShape = nil; /* the declaration of myShape */
...
/* code that may use myShape (or may not) */
DisposeAt(&myShape); /* If it was allocated, throw it away.
                      If not, do nothing. */
```

```
void Set2(shape, fixed, fixed);
```

Set2 is intended to change the fields of a point shape's geometry. Any other type will post a warning, since this call supplies too much or too little data.

```
void Set4(shape, fixed, fixed, fixed, fixed);
```

Set4 changes the geometry of a shape of type lineType, rectangleType or ovalType. A polygon or path shape will scale their bounds to equal the indicated rectangle. If the type is pointType, the first two parameters will change the point geometry and a warning will be posted. Any other type will post a warning without affecting the shape.

```
void SetShape(shape destination, shape source);
```

SetShape copies the geometry one shape into another, changing the type of the destination shape to the type of the first. It does not affect the transform, style or fill attributes of either shape.

```
void CopyTo(shape destination, shape source);
```

CopyTo creates a new shape which has the same geometry, style and transform of the original. For bitmaps, CopyTo does not copy the bit image, only the fields of the bitmap.

Note: CopyTo, like all Skia calls, affects the first shape parameter passed. The QuickDraw CopyRgn call affects the second parameter passed.

```
void CopyDeepTo(shape destination, shape source);
```

CopyDeepTo works the same way as CopyTo, except in the case of bitmaps or pictures that contain bitmaps; in these cases the bit image associated with the bitmap is copied as well. Skia allocates the space for the bit image copy.

```
void Draw(shape);
```

Draw draws the indicated shape with its style and transform. All of the possible graphic types that Skia knows how to draw can be drawn with Draw. The shape specifies the type of data to be drawn. The type can be any of the graphic types: emptyType, pointType, lineType, arcType, curveType, rectangleType, polygonType, pathType, textType, pictureType or fullType.

Most shapes are set to permit one or more of the following variations: filled or framed, inverted, winding number fill or xor fill. The transform contains the clip that the shape geometry is limited to, and the matrix that specifies how the geometry is interpreted. The style contains variations typically common to many shapes, such as color, line thickness, point size and so on. The transform also indicates which ports the drawing takes place in.

Another way to draw is to pass the geometry to be drawn to one of the following routines: DrawPt, DrawLine, DrawArc, DrawCurve, DrawRectangle, DrawPolygon, DrawPolygons, DrawPath or DrawPaths. See these routines for more information.

## Lines

```
shape NewLine(line *linePts);
```

NewLine creates a new shape of type lineType. It is assigned the default transform and the default frame style. The shape defaults to frame. Note that filled lines would not draw anything, since the line is infinitely thin. Filled lines are not allowed.

```
void SetLine(shape, line *);
```

SetLine changes the shape's type to lineType, and copies the line structure into it.

```
void DrawLine(line *);
```

DrawLine draws a line using the current default transform and style. See the Draw routine for more details.

```
void FixLineTo(fixed, fixed);  
void FixLine(fixed, fixed);
```

Line and LineTo draw a line and move the path to follow to the pen position. The follow path is reset.

These work the same as the QuickDraw equivalents; they take 32 bit long integer arguments instead of 16 bit. The arguments are fixed point numbers in the standard 16.16 format.

See also CurLine, listed under Polygons and Paths.

### Alternatives to "New"

Given:

```
shape myShape;  
fixed startX, startY, endX, endY
```

These calls also create a new line:

```
myShape = New(lineType); /* the line data itself is undefined */  
myShape = New4(lineType, startX, startY, endX, endY);
```

And given:

```
line myLine = {startX, startY, endX, endY};
```

These routines cast an existing shape into a line:

```
Type(myShape, lineType); /* the data may be a nonsense line */
```

or:

```
SetLine(myShape, &myLine);
```

or:

```
Set4(myShape, startX, startY, endX, endY); /* if myShape is a line */
```

See the individual routines for how they work and their restrictions.

## Rectangles

```
shape NewRectangle(rectangle *);
```

NewRectangle encapsulates the rectangle geometry into a shape, and assigns it the default transform and the default fill style. The shape defaults to fill, so that it represents a solid rectangle.

```
void SetRectangle(shape, rectangle *);
```

SetRectangle changes the specified shape into a rectangle type and sets the geometry as specified.

```
void DrawRectangle(rectangle *, shapeGeometry);
```

DrawRectangle draws a rectangle using the current default transform. The shapeGeometry parameter determines whether to use the default frame style or the default fill style. See the Draw routine for more details.

## Ovals and Rounded-Corner Rectangles

Ovals are approximated when drawn with quadratic Beziers; please read about curve error in the Style routines, above.

```
shape NewOval(oval *);
```

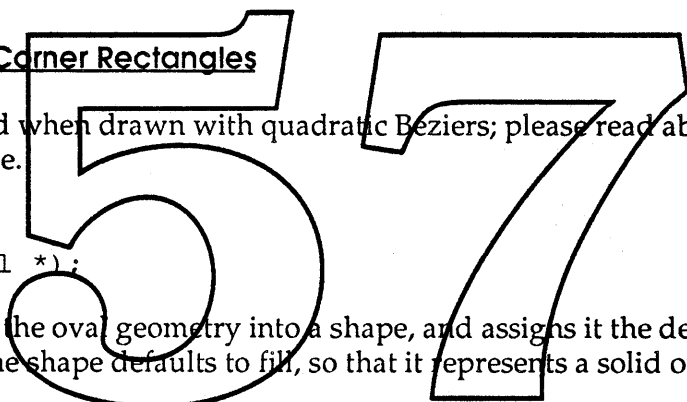
NewOval encapsulates the oval geometry into a shape, and assigns it the default transform and the default fill style. The shape defaults to fill, so that it represents a solid oval.

```
void SetOval(shape, oval *);
```

SetOval changes the shape into an oval type and sets the geometry accordingly.

```
void DrawOval(oval *, shapeGeometry);
```

DrawOval draws an oval using the current default transform. The shapeGeometry parameter determines whether to use the default frame style or the default fill style. See the Draw routine for more details.





```
shape NewRoundRect (rectangle *, point *ovalSize);
```

NewRoundRect encapsulates the rounded-corner rectangle geometry into a shape, and assigns it the default transform and the default fill style. The shape defaults to fill, so that it represents a solid rounded-corner rectangle. The shape is of type pathType.

```
void SetRoundRect (shape, rectangle *, point *ovalSize);
```

SetRoundRect changes the shape to a path type and sets the geometry to a rounded-corner rectangle.

```
void DrawRoundRect (rectangle *, point *ovalSize, shapeGeometry);
```

DrawRoundRect draws a rounded-corner rectangle using the current default transform. The shapeGeometry parameter determines whether to use the default frame style or the default fill style. See the Draw routine for more details.

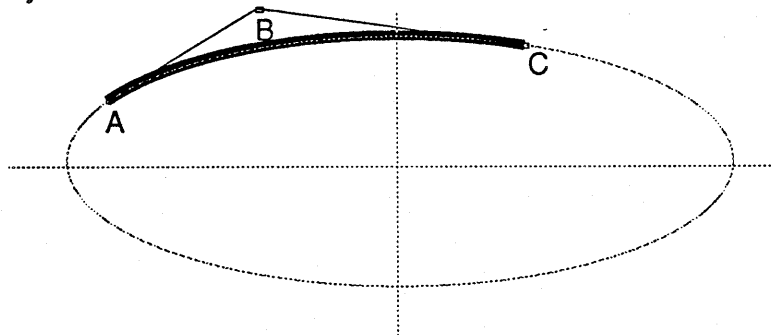
### Arcs, Curves, and Wedges

```
FrameArc, PaintArc, EraseArc, InvertArc, FillArc
```

These routines use the current transform and the current frame, paint, erase, invert or fill style to draw an arc described by the integer rectangle and the integer start and arc angles.

```
shape NewArc (curve *);
```

NewArc encapsulates the arc geometry into a shape, and assigns it the default transform and the default frame style.



An arc is defined by an upright ellipse segment that passes through the curve's end points and is tangent to the lines formed by the endpoints and the control point.

```
void SetArc (shape, curve *);
```

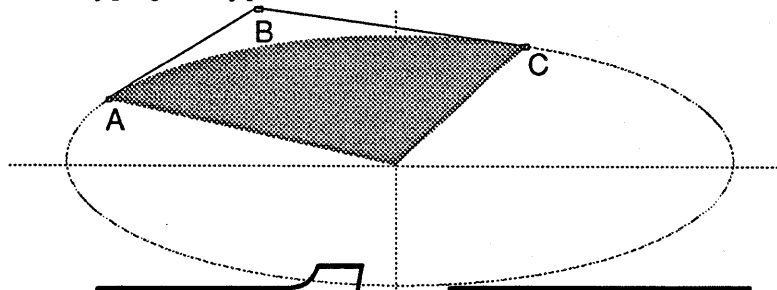
SetArc sets the shape to an arc type and sets the geometry accordingly.

```
void DrawArc (curve *);
```

DrawArc draws an arc using the current default transform and frame style. See the Draw routine for more details.

```
shape NewWedge (curve *);
```

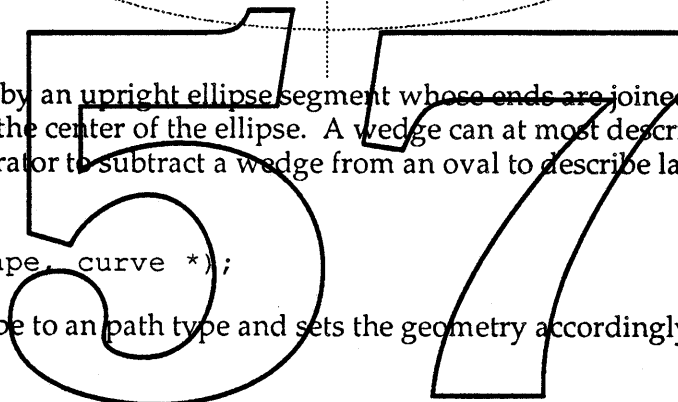
NewWedge encapsulates the pie-slice shaped geometry into a shape, and assigns it the default transform and the default fill style. The shape defaults to fill, so that it represents a solid wedge. The shape is of type pathType.



A wedge is defined as by an upright ellipse segment whose ends are joined by two lines segments that meet at the center of the ellipse. A wedge can at most describe nearly 180° of an oval. Use the Diff operator to subtract a wedge from an oval to describe larger wedges.

```
void SetWedge (shape, curve *);
```

SetWedge sets the shape to an path type and sets the geometry accordingly.

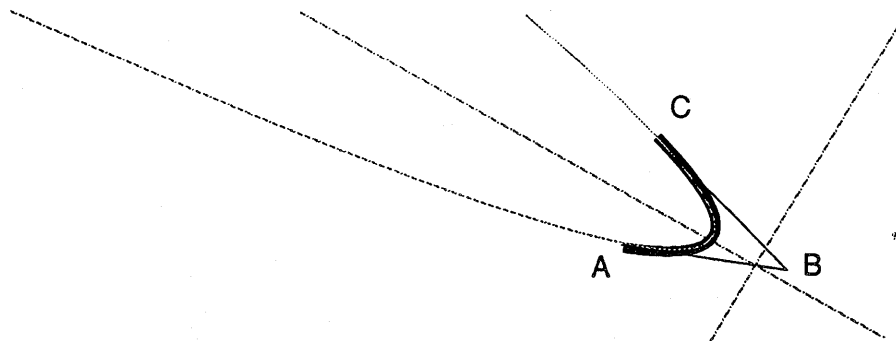


```
void DrawWedge (curve *, shapeGeometry);
```

DrawWedge draws a pie-slice shaped wedge using the current default transform and frame style. See the Draw routine for more details.

```
shape NewCurve (curve *);
```

NewCurve encapsulates the quadratic Bézier geometry into a shape, and assigns it the default transform and the default frame style.



A curve is defined as a parabolic segment that passes through the curve's end points and is tangent to the lines formed by the endpoints and the control point.

```
void SetCurve(shape, curve *);
```

SetCurve sets the shape to a curve type and sets the geometry accordingly.

```
void DrawCurve(curve *);
```

DrawCurve draws a quadratic Bézier using the current default transform and frame style. See the Draw routine for more details.

### Polygons and Paths

```
shape NewPolygon(polygon *);
```

NewPolygon encapsulates the polygon geometry into a shape, and assigns it the default transform and the default filling style. The shape defaults to fill, so that it represents a solid polygon. NewPolygon assumes that the newly created shape will have only one contour. For example:

```
/* outside of a routine ... */
long myTriangle[] = {4 /* number of points */,
    {0, 0}, {f(10,0), f(20,0)}, {f(20,0), f(10,0)}, {0,0}};

/* ... inside a routine */
shape myPoly = NewPolygon((polygon *) myTriangle);
```

```
shape NewPolygons(polygons *);
```

NewPolygons encapsulates the polygon geometries into a shape, and assigns it the default transform and the default filling style. The shape defaults to fill, so that it represents several solid polygons. NewPolygons is useful when the data describes more than one contour. Both NewPolygon and NewPolygons produce a shape of polygonType.

```
shape NewPath(path *);
```

NewPath encapsulates the path geometry into a shape, and assigns it the default transform and the default filling style. The shape defaults to fill, so that it represents a solid path. NewPath assumes the newly created shape will have only one contour.

```
shape NewPaths(paths *);
```

NewPaths encapsulates the path geometries into a shape, and assigns it the default transform and the default filling style. The shape defaults to fill, so that it represents several solid paths. NewPaths is useful when the data describes more than one contour. Both NewPath and NewPaths produce a shape of pathType.

```
void SetPolygon(shape, polygon *);  
void SetPolygons(shape, polygons *);
```

These routines change the specified shape into a polygon type and sets the geometry as specified.

```
void SetPath(shape, path *);  
void SetPaths(shape, paths *);
```

These routines change the specified shape into a path type and sets the geometry as specified.

```
void DrawPolygon(polygon *, shapeGeometry);  
void DrawPolygons(polygons *, shapeGeometry);  
void DrawPath(path *, shapeGeometry);  
void DrawPaths(paths *, shapeGeometry);
```

These routines draw polygons and paths using the current default transform. The default frame or fill style is chosen according to the shape geometry. See the Draw routine for more details.

## **Bitmaps**

```
shape NewBitmap(bitmap *);
```

NewBitmap creates a bitmap. The parameter passed is a pointer to a bitmap structure whose width, height, and pixel size fields should be correctly initialized for the dimensions of the image. The base address field can either be a pointer to a pre-allocated block of memory or nil, in which case Skia will attempt to create storage for the image based on its dimensions and also calculate the necessary rowWidth field rounded up to a long word boundary. The newly allocated pixels are not initialized. Skia keeps track of how the base address was allocated so that Skia will only dispose of the bit image if it allocated memory for it. If the rowWidth is set to zero, then it is calculated correctly to accommodate the width and pixel size fields.

Note: A bitmap with a rowWidth smaller than the bitmap width has an undefined drawing behavior.

```
void SetBitmap(shape, bitmap *);
```

SetBitmap changes the bitmap fields after the shape has been created. If Skia allocated the bitmap, the bit image can be re-sized by changing the rowWidth and height fields. This does not scale the image; the area the image can occupy is changed (like changing the number of pages in an existing document). The new pixels allocated, if any, are not initialized. This is useful when the system is keeping track of the image's memory since it will automatically move it around as needed. Finally, changing the depth of a bitmap by changing its pixelSize field and calling SetBitmap() will expand or compress the image. This also involves redistributing the colors in its style if the image's depth is decreased.

Note: Changing only the base address of the bitmap so that it points to another bit image does not invalidate any internal state kept from the last Draw call of that bitmap. This supports the rapid display of bitmaps sometimes called for in animation.

Note: If you change the width field and if Skia allocated the bit image, then you may want to set the rowWidth field to zero to cause it to be recalculated.

```
void CurBitmap(shape, bitmap *);
```

CurBitmap copies the bitmap structure from the shape into the indicated bitmap structure. To determine how much memory an image occupies, simply multiply the rowWidth field times the height field. The bit image may be manipulated directly by using the base address and rowWidth values. The byte address of a given pixel is determined by:

$$\text{baseAddr} + (\text{y position} * \text{rowWidth}) + (\text{x position} * \text{pixelSize} / 8)$$

```
void DrawBitmap(bitmap *);
```

DrawBitmap draws a bitmap using the current default transform and style. See the Draw routine for more details.

Reset and CopyTo also affects bitmap shapes in special ways; see "Skia Shape Routines".

## Text

```
shape NewText(char *text, short length);
```

NewText creates a new shape of type textType. It is assigned the default transform and the default text style. The shape defaults to fill, so that the text outline is filled when it is drawn.

```
void SetText(shape, char *text, short length);
```

SetText changes the shape type to textType, and copies the indicated text into the shape.

```
short CurText(shape, char *text);
```

CurText returns, for shapes of type textType, the length of the text. If the text parameter is non-nil, the text from the shape is copied into the storage pointed to by the parameter. It is up to the user to allocate the correct size storage.

```
shape NewLayout(layout *text);
```

```
void SetLayout(layout *text);
```

```
long CurLayout(layout *text);
```

```
void DrawLayout(layout *text);
```

*The Layout Manager defines the layout structure, that defines a 16 bit glyph code, a horizontal and vertical character placement, and a glyph font face/algorithmic face selector per character. Skia will support this format as soon as it firms up.*

## Pictures

Routines that work for shapes work for pictures, too. Here are some more shape routines that make sense only for pictures:

```
void Load(shape picture, short ID, char *name);
```

Load looks for a resource of with a matching resource ID of type "pict" and loads it. If the ID parameter is zero, then the resource name is used instead.

```
short Save(shape picture, short ID, char *name);
```

Save saves the picture to disk and returns the resource ID. If the resource ID is zero, the name is used instead. If the name parameter is nil, then a unique resource ID is generated.

```
long AddToPicture(shape picture, shape toAdd, transform, style);
```

AddToPicture adds a shape to a picture with a special transform or style. The index of the new element is returned.

AddTo also adds shapes to a picture;

```
AddToPicture(myPict, myShape, nil, nil);
```

is the same as:

```
AddTo(myPict, myShape);
```

```
void InsertPicture(shape picture, long index, shape toAdd, transform,  
style);
```

InsertPicture allows shapes to be added anywhere inside a picture. Insert performs the same function, without specifying a special transform and style.

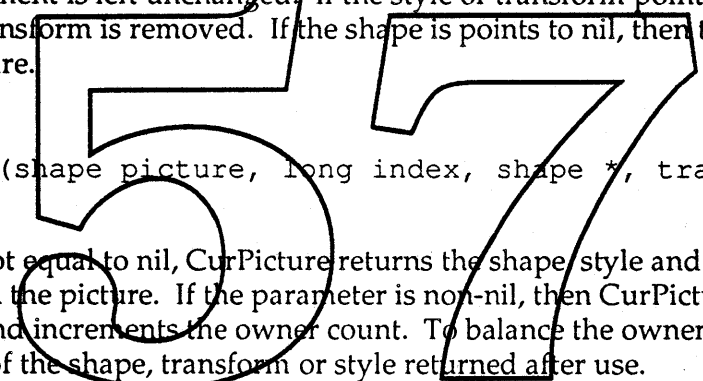
Extract allows shapes to be removed from a picture. See "Polygons and Paths" above for Insert and Extract.

```
void SetPicture(shape picture, long index, shape *, transform *,  
style *);
```

SetPicture changes the shape, style or transform (if the parameters are not nil) at the given index within the picture. If the shape, style or transform parameter passed is nil, then that parameter for this element is left unchanged. If the style or transform point to nil, then the overriding style or transform is removed. If the shape is points to nil, then the element is deleted from the picture.

```
shape CurPicture(shape picture, long index, shape *, transform *,  
style *);
```

For each parameter not equal to nil, CurPicture returns the shape, style and transform at the specified index within the picture. If the parameter is non-nil, then CurPicture causes the updates the pointer and increments the owner count. To balance the owner count, it is necessary to dispose of the shape, transform or style returned after use.



## Shape Operations

```
void SetType(shape, shapeType);
```

SetType changes the shape passed to the indicated type. SetType seldom affects the values of the control points, except in the case of rectangles and ovals, which are set to the bounding boxes of other shapes. SetType adapts the existing geometry to the new type, if possible.

old type	new type	action
any type	same type	warning: type_already_set
emptyType	not fullType	warning: new_shape_contains_invalid_data
line ... pathType	pointType	set to upper left corner of bounding box
pointType	line ... rectangle	set both points to point value
rectangle ... path	lineType	set to line from upper left to lower right of bounds
point ... line	arc ... curve	duplicate missing end points

arc, curve	curve, arc	change type only; control points remain unchanged
polygoneType	arc ... curve	keep the first 3 points as control points
line ... pathType	rectangleType	set to shape bounds
point ... rectangle	polygon ... path	set to 1 contour == shape
polygoneType	pathType	set to path with all control points on the curve
pathType	polygoneType	set to polygon that connects all control points
bitmapType	point ... path	warning: new_shape_contains_invalid_data
textType	point ... polygon	warning: new_shape_contains_invalid_data
textType	pathType	outline of text characters
pictureType	point ... path	warning: new_shape_contains_invalid_data
point ... path	bitmap ... picture	warning: new_shape_contains_invalid_data
fullType	not emptyType	warning: new_shape_contains_invalid_data

```
shapeType CurType (shape);
```

CurType returns the shape type contained within a given shape.

```
void SetAttributes (shape, shapeAttributes);
```

SetAttributes sets the shape state which determines special attributes, like locking and caching.

```
typedef enum {
    noAttributes,
    cache = 1,
    lock = 2,
    keepDirect = 4,
    keepRemote = 8
} shapeAttributes;
```



Here are the attributes' meaning:

attribute	meaning
lock	Determines whether the geometric or bitmap structure can be changed by a shape operation. If the geometry is locked, the transform may be affected instead.
cache	Determines whether the geometric or bitmap structure uses additional memory to draw, to speed subsequent drawing.
keepDirect	Specifies that the structure and associated state should be maintained by the main processor. This is particularly important to ensure that the bit image allocated by Skia is accessible by the application.
keepRemote	Specifies that the structure and associate state should be maintained by a graphics accelerator or other remote device, if possible. This allows placing a priority on which shapes the accelerator memory maintains locally.



The default attributes value is noAttributes.

```
shapeAttributes CurAttributes(shape);
```

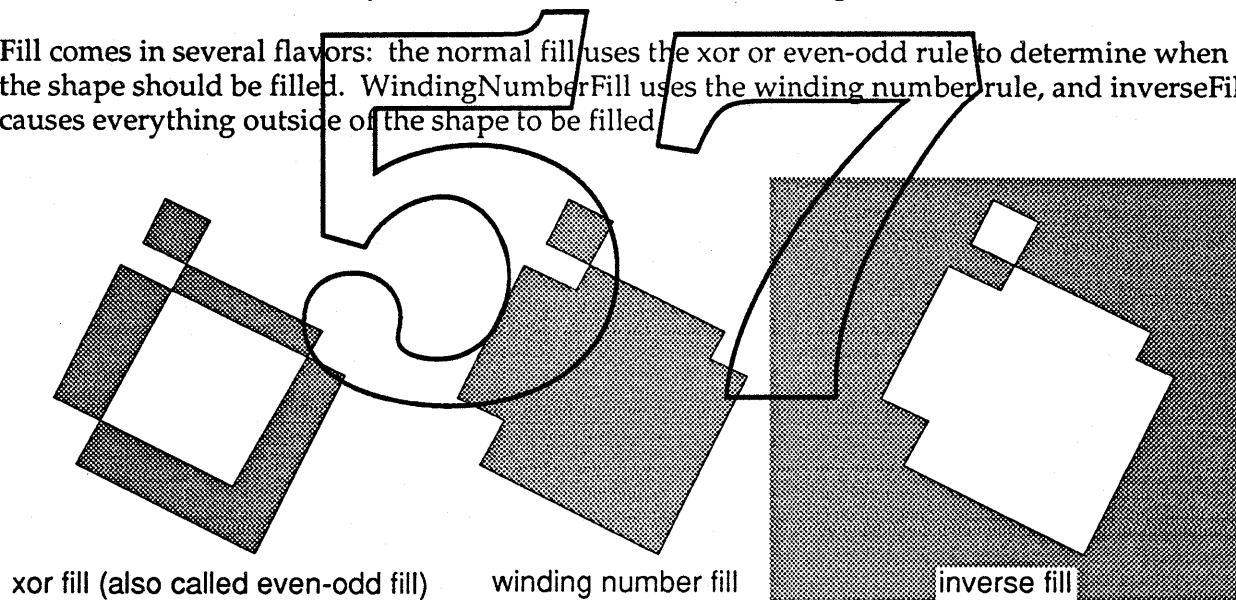
CurAttributes returns the current attributes for the specified shape.

```
void SetGeometry(shape, shapeGeometry);
```

SetGeometry changes the interpretation of the geometry inside a shape to frame (to a stroke with a given pen thickness), or to fill (filling the area bounded by the pen) as the pen moves from one control point to the next. Framing makes sense for lines, arcs, curves, and so on through paths; filling makes sense for only closed shapes; rectangles and ovals, polygons and paths only if the contour starts and ends at the same point. Frame and fill do not mean anything to bitmaps, text or pictures.

Framing comes in several flavors: the normal frame centers the frame about the geometry so that the pen width extends to either side; insideFrame makes the frame work like QuickDraw rectangles and regions, so the frame never extends the size of the bounding box; outsideFrame does the opposite, and always increases the size of the bounding box.

Fill comes in several flavors: the normal fill uses the xor or even-odd rule to determine when the shape should be filled. WindingNumberFill uses the winding number rule, and inverseFill causes everything outside of the shape to be filled.



The shapeGeometry enumerated type looks like:

```
typedef enum {
    noGeometry,
    insideFrame,
    frame,           /* centered frame */
    outsideFrame,
    fill,           /* xor fill */
    windingNumberFill,
    inverseFill
} shapeGeometry;
```

Note: Text can be stroked by applying an outline style; pictures can be stroked by affecting the entries within the picture.

Note: SetGeometry does not affect the style of the shape.

```
shapeGeometry CurGeometry(shape);
```

CurGeometry returns the current geometry for a shape.

```
void SetUser(shape, char *data, long length);
```

SetUser copies the specified data to the user field within the shape.

```
long CurUser(shape, char *data);
```

CurUser returns the length of the user data contained within the shape. If the data parameter is not nil, the user data is copied to the memory pointed to by the data parameter.

Note: it is up to the caller to allocate adequate memory to receive the user data.

```
void Reset(shape or TransformStyle);
```

Reset re-initializes the shape to its default condition. Reset does not affect most shapes. For a bitmap, Reset restores the original base address pointer as initially allocated by the system (if the user allocated it, Reset has no effect).

If a transform cast into a shape is passed, Reset sets the transform's matrix to identity and its clip to wide open.

If a style cast into a shape is passed, Reset sets the pen size to 1.0, the pen mode to copy, deletes the pattern, join, dash and cap shapes, sets the frame position to center, and sets the curve error to 1/16th. Reset does not affect the color, luminance, dither size, pen position, follow path or other style attributes.

If nil is passed, Reset affects the default transform and all four default styles.

```
void Changed(shape);
```

Changed lets Skia know that some data structure within the shape has been changed by the application directly. The only time an application is required to make this call is after writing to a bit image contained within a bitmap shape.

```
void Simplify(shape);
```

Simplify converts a shape into its simplest and smallest description. Simplify removes duplicate points from polygons or paths, connects successive contours that end and begin in the same place, and re-types shapes from paths to polygons, rectangles, ovals, curves, lines, points and so on as appropriate.

Simplify does not affect pictures, text or bitmaps.

```
void Primitive(shape);
```

Primitive converts a shape into a primitive description that describes how the shape will be drawn.

All drawing in Skia is implemented by a few simple primitives: hairlines, hair-curves, filled shapes, and bitmaps. Hairlines and hair-curves are the thinnest perceivable line or curve. All shapes are drawn by reducing them to one of these primitives. See also UserPrimitive, under "Customizing Skia", below.

```
extern void Cache(shape);
```

Cache is an expert-user call that computes as much as possible to ready drawing the shape. A cached shape generally draws much faster than an ordinary shape. The cache will be retained until the shape is changed, or until the style, transform, port or device is changed to make the shape draw differently.

Calling Cache on a picture or a series of shapes allows the shapes to be drawn successively very quickly, but the cache structures can take up a large amount of space. Accordingly, Skia installs itself in the Memory Manager growZone proc to delete cache structures when the application is low on space.

Use the SetAttributes call to cause a shape to be always cached.

```
void DisposeCache(shape);
```

DisposeCache deletes any temporary memory or cached data associated with a shape.

Use the SetAttributes call to clear the cache bit to avoid building a cache when the shape is drawn the next time.

Note: all shapes require temporary memory to draw. The memory is allocated and then associated with the shape to avoid a re-allocation the next time the shape is drawn. Cached shapes allocate even more memory to speed drawing; bitmaps that are transformed cache their intermediate transformed bits, and filled and framed shapes cache a compressed bitmap description. These intermediate caches can slow down drawing if the shape is drawn only once (since it takes time to build the intermediate cache), but can speed operations like updating windows where the same information is drawn more than once.

## Calculations with Rectangles

```
boolean SectR2 (rectangle *source1, rectangle *source2);
```

SectR2 returns true if rectangles source1 and source2 intersect.

```
boolean SectR3 (rectangle *source1, rectangle *source2,  
               rectangle *destination);
```

SectR3 works the same as SectR2; additionally, it returns the intersection the the rectangle destination.

```
boolean SectRP (rectangle *, point *);
```

SectRP returns true if the point intersects the rectangle.

```
boolean ContainsR2 (rectangle *container, rectangle *source);
```

ContainsR2 returns true if the container rectangle surrounds the source rectangle.

```
void UnionR3 (rectangle *destination, rectangle *source1, rectangle  
             *source2);
```

UnionR3 combines source1 and source2 and stores the resulting rectangle in destination.

```
void UnionR2 (rectangle *destination, rectangle *source);
```

UnionR2 combines destination and source and stores the resulting rectangle in destination.

## Operations on Geometry

```
void AddTo (shape dest, shape add);
```

AddTo allows the geometry to be added to the end of any existing region. If the first control point added matches the last control point in the shape, the last contour is extended; otherwise, a new contour is added to the shape. AddTo may promote the shape to a polygonType or a pathType.

See Insert to add new control points in the middle of a shape.

```
void Inset(shape, long h, long v);
```

Inset creates a new shape with all of its control points inside the old points by the given distances. For shapes with multiple contours, each contour is sized so that the bounding box of the contour is smaller.

```
void Sect(shape destination, shape source);
```

Sect intersects the source shape with the destination shape, and stores the resulting shape in the destination.

```
void Union(shape destination, shape source);
```

Union combines the source shape with the destination shape, and stores the resulting shape in the destination shape.

```
void Diff(shape destination, shape source);
```

Diff subtracts the source shape from the destination shape, and stores the resulting shape in the destination shape.

```
void ReverseDiff(shape destination, shape source);
```

ReverseDiff subtracts the destination shape from the source shape, and stores the resulting shape in the destination shape.

```
void Xor(shape destination, shape source);
```

Xor exclusive-ors the source and destination shapes and stores the result in the destination shape.

```
void Invert(shape destination);
```

Invert changes the shape to describe the coordinate space currently unaffected by it. See also "Shape Attributes", above.

```
boolean SectPoint(shape, point *);
```

SectPoint returns true if the point intersects the shape. See also MatchPoint, below.

```
boolean SectRectangle(shape, rectangle *);
```

SectRectangle returns true if the rectangle intersects the shape.

```
boolean RectangleContains(rectangle *container, shape source);
```

RectangleContains returns true if the shape is contained by the rectangle.

### Operations on Contours, Vectors and Control Points

```
void SetCVPoint(shape, long contour, long vector, point *);  
void SetIndexPoint(shape, long index, point *);
```

These two routines allow an individual control point within the shape to be changed. SetCVPoint takes the contour number and vector number, and SetIndexPoint takes a point index. Contour numbers may make changing polygons and paths easier.

```
void Insert(shape, long index, shape toAdd);
```

Insert works similar to AddTo, described above; it allows the geometry to be added anywhere inside the shape. Insert does not replace any of the current geometry; it expands the shape to make room for the new information, and moves the geometry at the point index forward, to make space for the new geometry. If the toAdd shape is nil or of emptyType, then the contour is broken and a new contour begins after the specified index.

Note: Insert will create new contours unless the first and last points in the added shape match the current point exactly.

```
shape Extract(shape source, long firstPoint, long numPoints,  
             shape *destination);
```

Extract creates a new shape or replaces the contents of an existing shape with one or more contours extracted from an existing shape. The specified geometry is removed from the source shape. The destination parameter may be nil, or may point to nil; in the latter case, the shape will be allocated as needed. The firstPoint parameter determines where to begin in the shape; the numPoints determines the number of points to copy. The shape is returned if the operation completed without error. Extract may reduce the number of contours if the points on either side of the extraction match exactly.

```
long Contours(shape);
```

Contours returns the number of contours in a shape.

```
long Points(shape, long contour);
```

Points returns the number of points in a contour in a shape. If zero is passed for the contour parameter, it returns the total number of points in a shape.

```
long Index(shape, long contour, long vector);
```

Index returns the index within a shape that corresponds to the start of the specified vector within the specified contour.

```
boolean CurControl(shape, long index);
```

CurControl returns whether the specified point within a shape is on or off the curve.

```
void CurPoint(shape, long index, point *);
```

CurPoint returns the point corresponding to the specified index within the shape.

```
void CurLine(shape, long index, line *);
```

CurLine returns a line beginning at the specified index within a shape; the line may degrade to a point (both points equal) if a true line is not present in the contour at the requested index.

Note: If the shape is of type lineType, the index parameter should be 1. For arcType and curveType, the index parameter may be 1 or 2; this selects the tangent line segments at the beginning or end of the curves. For rectangleType and ovalType, the index parameter can be 1 to 4; this returns the appropriate tangent line, clockwise starting with the left-top to right-top.

Warnings: bad\_parameter\_passed\_to\_CurLine

```
void CurCurve(shape, long index, curve *);
```

CurCurve returns a curve beginning at the specified index within a shape; the curve may degrade to a point (all three points equal) or a line (the last two points equal) if a true curve is not present in the contour at the requested index.

```
void CurRectangle(shape, long startIndex, long numberOfPoints, rectangle *);
```

CurRectangle returns the bounding rectangle for the subset of points within a shape as specified.

```
void CurPolygon(shape, long startIndex, long numberOfPoints, polygon *);
```

CurPolygon returns the subset of control points as specified. If the points requested exceed the number of points available in the contour, then a warning is posted.

```
void CurPolygons(shape, long startIndex, long numberOfPoints,
                 polygons *);
```

CurPolygons works the same as CurPolygon, returning multiple contours if appropriate.

```
void CurPath(shape, long startIndex, long numberOfPoints, path *);
```

CurPath works like CurPolygon, except it returns an array of control bits as well.

```
void CurPaths(shape, long startIndex, long numberOfPoints, paths *);
```

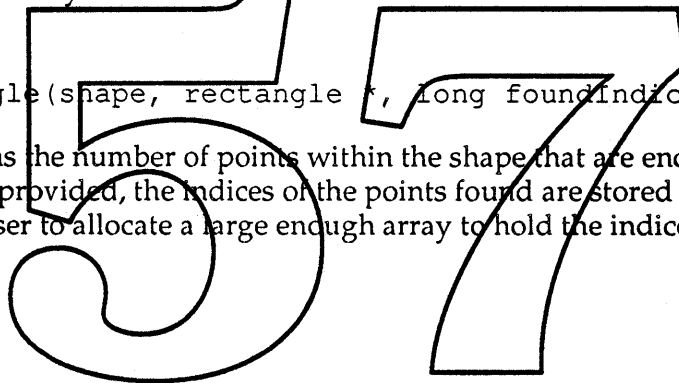
CurPaths works like CurPath, returning multiple contours if appropriate.

```
long MatchPoint(shape, point *, long foundIndices[]);
```

MatchPoint returns the number of points within the shape that match the specified point. If provided, the indices of the points found are stored in the supplied array. It is up to the user to allocate a large enough array to hold the indices.

```
long MatchRectangle(shape, rectangle *, long foundIndices[]);
```

MatchRectangle returns the number of points within the shape that are enclosed by the specified rectangle. If provided, the indices of the points found are stored in the supplied array. It is up to the user to allocate a large enough array to hold the indices.



### Shape Utilities

```
long Size(shape);
```

Size returns the memory occupied by a shape.

```
long SizeCache(shape);
```

SizeCache returns the memory occupied by a shape and its cache state. The difference between Size and SizeCache can be immediately recovered by the NoCache call. Skia treats cache memory as purgeable; cache is always disposed when additional memory is required.

```
fixed Time(shape);
```

Time returns the estimated time required to draw the specified shape, in seconds. Time returns a number from 0 seconds to 9 hours, in 15 microsecond intervals.



```
boolean Validate(shape);
```

Validate checks the integrity of a shape and of all of the data structures pointed to by the shape. Validate is intended primarily for debugging Skia applications, and may be called by the application's error and warning routines.

```
boolean Equal(shape, shape);
```

Equal returns true if the geometry of two shapes is the same.

```
fixed Length(shape);
```

Length returns the length of the perimeter of a shape. The length of a point is always zero. The length is pinned to the largest representable fixed point value (about +32768).

```
float FloatLength(shape);
```

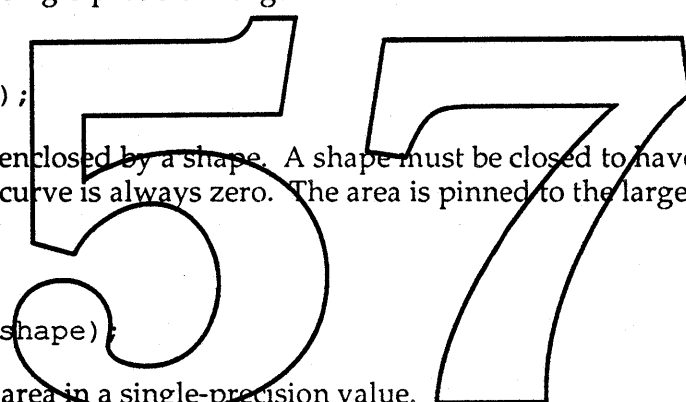
FloatLength returns a single-precision length.

```
fixed Area(shape);
```

Area returns the area enclosed by a shape. A shape must be closed to have an area. The area of a point, line, arc or curve is always zero. The area is pinned to the largest representable fixed point value.

```
float FloatArea(shape);
```

FloatArea returns the area in a single-precision value.



```
long GlobalArea(shape, long portOrder);
```

GlobalArea returns the number of pixels that a shape, drawn through a particular port, touches.

```
void Bounds(shape, rectangle *);
```

Bounds returns the bounding rectangle of the shape.

```
void LocalBounds(shape, rectangle *);
```

LocalBounds returns the bounding rectangle of the shape as drawn in local coordinates; that is, after all of the style information such as joins, caps, dashing, follow paths and so on has been applied.

```
void GlobalBounds(shape, long portOrder, rectangle *);
```

GlobalBounds returns the bounding rectangle of the shape, in the specified port, as drawn in global coordinates. The bounds correspond to the extent of the pixels affected by drawing this shape through the specified port.

## **Transforms**

```
transform NewTransform(fixed [3][3], shape clip);
```

NewTransform creates a transform, which encapsulates the indicated matrix and clip shape. The parameters passed may be nil; this defaults to the identity matrix, and a wide open clip. The clip shape is copied into the transform, so that the original clip shape can be changed or thrown away without affecting the transform. The transform's port list is initially empty. The owner count is set to 1.

```
void DisposeTransform(transform *);
```

DisposeTransform decrements the owner count, and if it is zero, throws away the indicated transform. Disposing a transform does not affect the ports in its port list.

```
void DisposeTransformAt(transform *);
```

DisposeTransformAt works just like DisposeTransform; it allows the shape variable declared to be assigned nil. See the example under DisposeAt.

```
transform CopyToTransform(transform destination, transform source);
```

CopyToTransform creates a copy of source and deposits it in destination. The copy is also a function result. If destination is nil, then a new transform is created, and its owner count is set to 1. If the destination already exists, the source transform's matrix, clip and view list is copied into the destination transform, but the destination owner count is not affected.

warnings: transform\_passed\_equals\_nil

errors: illegal\_type\_for\_transform

```
void SetTransform(shape, transform);
```

SetTransform removes the transform already associated with the shape, and the indicated transform is attached to the shape instead, and the transform's owner count is incremented. If this is the shape is the only owner of the transform, the transform is deleted.

```
transform CurTransform(shape);
```

CurTransform returns the transform associated with a shape.

```
void SetDefaultTransform(transform);
```

SetDefaultTransform changes the default transform to the specified transform.

```
transform CurDefaultTransform(void);
```

CurDefaultTransform returns the current default transform.

```
void SetTransformClip(shape orTransform, shape clip);
```

SetTransformClip sets the transform associated with the shape to use a copy of the indicated clip.

```
SetTransformClip((shape) myTransform, myClip);
```

If a transform is passed instead of the affected shape, all shapes sharing that transform will be clipped to the same shape.

```
SetTransformClip(nil, myClip);
```

If nil is passed for the affected shape, the default transform is affected instead.

```
SetTransformClip(myShape, nil);
```

If nil is passed for the clip shape, the affected shape's clip is set to be wide open. See also the example below.

```
shape CurClip(shape orTransform, shape *clip);
```

CurClip copies the clip owned by the shape's transform into the clip shape.

```
CurClip((shape) myTransform, &myClip);
```

A transform can be passed instead of a shape.

```
CurClip(nil, &myClip);
```

If nil is passed for the shape, the default transform's clip is returned instead.

```
shape myClip = nil;
```

A pointer to the clip is passed, so that the clip shape does not have to be created before the CurClip call is made; the clip shape call should be initialized to nil when declared.

```
if (CurClip(myShape, &myClip)) ...
```

CurClip returns the value of the copy to allow quick tests of whether the clip exists or not.

```
void SetMatrix(shape orTransform, fixed[3][3]);
```

SetMatrix sets the transformation's matrix to the indicated values. If a shape is passed as a parameter, and its transform is shared, then the transform is duplicated before the matrix is affected.

```
void CurMatrix(shape orTransform, fixed[3][3]);
```

CurMatrix returns the values contained in the transform's matrix.

See the math chapter for the set of matrix routines available.

### Operations on Shapes and Transforms

```
void Offset(shape orTransform, fixed h, fixed v);
```

Offset moves the shape or the transform by the specified offsets horizontally and vertically. If the resulting shape would fall outside of the coordinate system bounds, then the shape or transform is left unmodified and a warning is posted. If the shape is a bitmap, then the shape's transform is modified instead, creating a new transform if it is shared.

```
void Rotate(shape orTransform, fixed degrees);
```

Rotate rotates the shape about its center, or rotates the transform about (0,0). Note that the angle is passed in degrees, not radians. This convention was chosen to allow the exact specification of 180° rotation, without requiring an approximation of  $\pi$ . See Offset.

```
void RotateAbout(shape orTransform, fixed degrees, fixed h, fixed v);
```

RotateAbout rotates shapes or transforms about the specified point. See Offset.

```
void Skew(shape orTransform, fixed xSkew, fixed ySkew);
```

Skew is also called oblique, an algorithmic method of creating an Italic text face. Skew skews the shape about its center, or skews the transform about (0,0). See Offset.

```
void SkewAbout(shape orTransform, fixed xSkew, fixed ySkew, fixed h,  
fixed v);
```

SkewAbout skews shapes or transforms about the specified point. See Offset.

```
void Scale(shape orTransform, fixed hScale, fixed vScale);
```

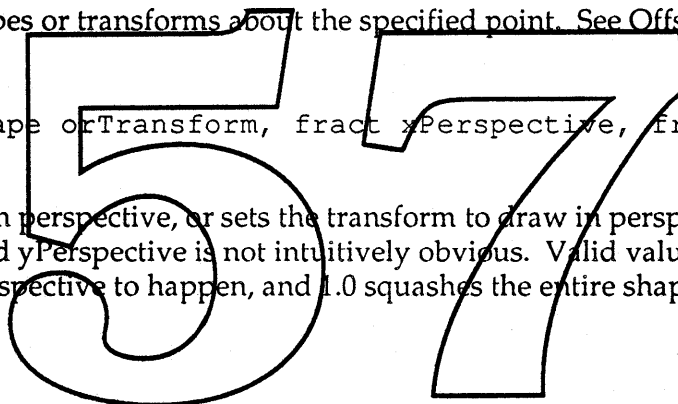
Scale scales the shape about its center, or rotates the transform about (0,0). Note that negative values for hScale and vScale are perfectly legal; they specify mirroring horizontally or vertically. See Offset.

```
void ScaleAbout(shape orTransform, fixed hScale, fixed vScale, fixed h,  
fixed v);
```

ScaleAbout scales shapes or transforms about the specified point. See Offset.

```
void Perspect(shape orTransform, fract xPerspective, fract  
yPerspective);
```

Perspect puts shapes in perspective, or sets the transform to draw in perspective. The meaning of the xPerspective and yPerspective is not intuitively obvious. Valid values range from -1.0 to 1.0; zero causes no perspective to happen, and 1.0 squashes the entire shape into a line. See Offset.



```
void Vanish(shape orTransform, fixed horizon, fixed azimuth);
```

Vanish puts shapes in perspective, or sets the transform to draw in perspective, by specifying the position of the vanishing point on the horizon, and on the azimuth. Vanish takes the current matrix transformation into account to compute the position of the vanishing planes regardless of the current transformation and scaling.

*One day an illustration will help show what is going on here.*

```
void Box(shape, rectangle *);
```

Box modifies the transform associated with the shape so that the shape's bounds fit into the box. For text, Box looks at the text's point size to determine the text extent.

## Associated Transform Routines and Examples

Here's an example to help explain what owners and owner counts are all about.

```
shape bowlingBall = NewPaths(&bowlingBallData);
transform scale2x = NewTransform(scaleBy2Matrix, nil);
```

At this point, the default transform has many owners, including Skia and bowlingBall. The variable scale2x is the only owner of the newly created transform.

```
SetTransform(bowlingBall, scale2x);
```

Now, the default transform has one less owner, bowlingBall. The new transform is owned by both scale2x and bowlingBall; its owner count equals 2.

```
DisposeTransformAt (&scale2x);
```

This decrements the owner count of the new transform back to 1; only bowlingBall owns it. DisposeTransformAt has set scale2x equal to nil.

```
DisposeAt (&bowlingBall);
```

This throws away the shape and decrements the owner counts of the style and transform that it points to. Since the owner count of the new transform is now zero, it is thrown away. The variable bowlingBall is set to nil.

What's wrong with this call?

```
SetTransform(bowlingBall, NewTransform(scale2x, nil));
```

The call will work just fine. The transform created has two owners: bowlingBall and thin air. When bowlingBall is deleted, the transform won't be thrown away. It would be better to write:

```
{ transform temp;
  SetTransform(bowlingBall, temp = NewTransform(scale2x, nil));
  DisposeTransform(temp);
}
```

This balances the owner count so that the transform will be disposed of when bowlingBall is thrown away.

Here are some examples of setting the transform's clip.

Example 1:

```
SetTransformClip(myShape, newClip);
```

replaces the existing clip in the transform pointed to by myShape. It creates a new transform if the transform was shared.

Example 2:

```
SetTransformClip((shape) Transform(myShape), newClip);
```

replaces the existing clip in the transform. The transform may be shared by other shapes.

Example 3:

```
{ shape temp = nil;  
  SetTransformClip(nil, Sect(CurClip(nil, &temp), newClip));  
  Dispose(temp);  
}
```

intersects the new clip with the current clip in the default transform, and uses the intersection to set the default transform's clip.

See also Reset under "Skia Shape Routines".

### Ports and the Transform's Port List

```
long NewPort(fixed [3][3], shape clip, colorTable);
```

NewPort creates a new port and adds it to the master list of ports maintained by Skia. NewPort returns the port order; when a transform contains more than one port, the ports are drawn to in ascending port order. The drawing is transformed according to the port's matrix, clipped to the port's clip, and filtered to allow only the colors present in the color table. Passing nil for the matrix causes an identity matrix to be used. Passing nil for the clip causes the port to cross all devices. The clip is specified in global coordinate space. Passing nil for the color table causes no color filtering to occur.

When a port crosses multiple devices, Skia actually creates more than one port, and returns the port order only for the first port. Additional ports may be associated with this first port, so that it is only necessary to refer to the first port when adding a port to a transform.

```
void DisposePort(long order);
```

DisposePort disposes of the indicated port, and also disposes of any associated ports created by the original port crossing multiple devices.

Drawing to a transform whose ports have been disposed of causes no drawing to occur, and posts a warning.

```
long CurPort(long order, fixed [3][3], shape *clip, colorTable *,
             long *device);
```

CurPort returns the port order, the current matrix, clipping, color table and device order associated with a port. Passing 0 for the port order returns the information about the first port in the master port list.

```
void UpdatePort(long order, fixed [3][3], shape clip, colorTable);
```

UpdatePort (which I'd rather name SetPort, but that's already used) updates the port's matrix, clip and color table.

```
void SetTransformPort(transform, long order);
```

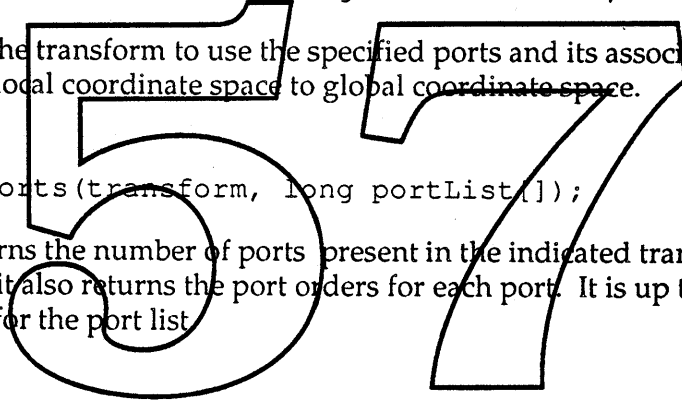
SetTransformPort sets the transform to use the port and its associates to associate local coordinate space to global coordinate space.

```
void SetTransformPorts(transform, long numberOfPorts, long portList[]);
```

SetTransformPorts sets the transform to use the specified ports and its associates contained in the port list to associate local coordinate space to global coordinate space.

```
long CurTransformPorts(transform, long portList[]);
```

CurTransformPorts returns the number of ports present in the indicated transform. If the port list parameter is not nil, it also returns the port orders for each port. It is up to the user to allocate adequate space for the port list



## Styles

```
style NewStyle(colorTable);
```

NewStyle creates a style, which points to the indicated color table. The color table passed may be nil; this defaults to a table with a single entry of black. The owner count is set to 1. All other elements of a style are set to their default values as shown:

style element	default value	interpretation
pen thickness	0	thinnest perceivable line or curve
dash	nil	no dashing
visibility	1	shape is visible, drawable
mode	copy	replace destination with source
pattern	nil	no pattern shape
start cap, end cap	nil	blunt cap at start of lines, polygons or paths
join	sharpJoin	
follow	point at (0,0)	pen position is at (0,0)



attributes	noAttributes	no grid fitting, pattern aligned to source
dither	0	no dithering
face	0	plain

The default styles created by InitSkia are initially customized to differ from these defaults as shown:

<b>default text style</b>	<b>default value</b>	<b>interpretation</b>
color table	white	fill defaults to erase
<b>default text style</b>	<b>default value</b>	<b>interpretation</b>
pen thickness	12.0	text point size
<b>default bitmap style</b>	<b>default value</b>	<b>interpretation</b>
color table	white, black	default colors for 1 bit black and white bitmap

```
void DisposeStyle(style);
```

DisposeStyle decrements the owner count, and if it is zero, throws away the indicated style.

```
void DisposeStyleAt(style *);
```

DisposeStyleAt works just like DisposeStyle; it allows the shape variable declared to be assigned nil. See the example under DisposeAt.

```
style CopyToStyle(style destination, style source);
```

CopyToStyle creates a copy of the source and deposits it in the destination. The copy is also a function result. If the destination is nil, then a new style is created, and its owner count is set to 1. If the destination already exists, the source style's joins, caps, dashing, patterns and a pointer to its color table is copied into the destination style, but the destination owner count is not affected.

```
void SetStyle(shape, style);
```

SetStyle removes the style already associated with the shape, and the indicated style is attached to the shape instead, and the style's owner count is incremented. If this is the shape is the only owner of the style, the style is deleted.

```
style CurStyle(shape);
```

CurStyle returns the style associated with a shape.

```
void SetDefaultStyle(shapeType, style);
```

SetDefaultStyle changes the default style for the specified shape type. The shape type determines which default style is replaced, as shown in the table below:

shape type	style affected
emptyType	---
pointType, arcType, curveType, lineType	default frame style
rectangleType, ovalType, polygonType, pathType	default fill style
bitmapType	default bitmap style
textType	default text style
pictureType	---
fullType	default fill style

```
style CurDefaultStyle(shapeType);
```

CurDefaultStyle returns the default style for the specified shape type.

### Operations on Styles

Skia creates four default styles to house the style state associated with different types of shapes. Those four defaults are the frame style, the fill style, the bitmap style and the text style. Style operators allow nil to be passed in place of the shape or style to affect the default styles. Some calls, such as Show, Hide and SetMode affect all four default styles. Other calls, such as SetPen, SetDash and SetFollow only affect the default frame style. Since the style calls are not orthogonal in this way, be careful when changing the default styles. Better yet, pass a shape or a private style so that the style operation works explicitly.

```
short Show(shape orStyle);
```

Show increments the pen visibility in the indicated style and returns the new visibility level. A shape is visible if the visibility is 0 or greater. If a style is passed, all shapes sharing the style are affected. If nil is passed, all four default styles are affected instead. If the shape is already visible, a warning is posted, and Show has no effect.

To make an invisible shape visible:

```
while (Show(myShape) < 0);
```

```
short Hide(shape orStyle);
```

Hide decrements the pen visibility in the indicated style and returns the new visibility level. A shape is invisible if the visibility is less than zero. If a style is passed, all shapes sharing the style are affected. If nil is passed, all four default styles are affected instead.

```
void SetPen(shape orStyle, fixed diameter);
```

SetPen sets the affected style to draw framed shapes with a thickness equal to the indicated diameter. If a style is passed instead of a shape, all shapes sharing that style are affected. If nil is passed, the default frame style is affected instead. When text is drawn, the pen is interpreted as the point size.

```
SetPen(myShape, 0);
```

Passing a thickness of zero causes the frame to be drawn as thin as is perceivable.

```
SetPen(myShape, f(0,001));
```

Very small diameters may cause all drawing to disappear, since the shape may fall in between pixels.

Warnings: style\_frame\_modified\_for\_filled\_shape  
pen\_size\_already\_set  
graphics\_type\_passed\_equals\_nil  
negative\_pen\_size\_ignored

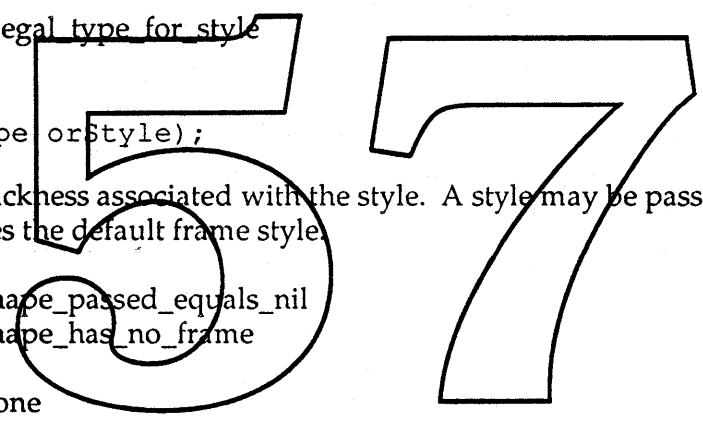
Errors: illegal type for style

```
fixed CurPen(shape orStyle);
```

CurPen returns the thickness associated with the style. A style may be passed in place of a shape. Passing nil uses the default frame style.

Warnings: shape\_passed\_equals\_nil  
shape\_has\_no\_frame

Errors: none



```
void SetStyleAttributes(shape orStyle, styleAttributes);
```

```
typedef enum {  
    sourceGrid = 1,  
    deviceGrid = 2,  
    devicePattern = 4  
} styleAttributes;
```

SetStyleAttributes applies one or more of the possible style attributes to the style. A value of zero eliminates any optional style attributes. The grid attributes cause the control points within a shape to be aligned to integral values by truncation before they are drawn. If the sourceGrid bit is set, the integral values are truncated in local coordinates before being transformed; if the deviceGrid bit is set, the integral values are truncated in global coordinates after being transformed. Normally patterns are aligned to the 0,0 origin within a shape. The devicePattern

bit, if set causes the pattern to be aligned to the 0,0 origin in global coordinates, typically the top left corner of the menu bar.

```
styleAttributes CurStyleAttributes(shape orStyle);
```

CurStyleAttributes returns the current attributes associated with the indicated style.

```
void SetCurveError(shape orStyle, fixed);
```

SetCurveError sets the maximum allowable deviance between the curves as specified and the curves as drawn. This is particularly important when drawing arcs, wedges and ovals; these elliptical segments are approximated with quadratic Bézier segments. The error is measured in local coordinates prior to any transformations.

```
fixed CurCurveError(shape orStyle);
```

CurCurveError returns the curve approximation allowable deviance in local coordinate units.

```
void SetMode(shape orStyle, mode, char *namedMode);
```

SetMode changes the affected style to the indicated mode. If the mode passed is zero, then the named mode is used. A style may be passed in place of a shape. If nil is passed for the shape, all four default styles modes are affected.

Some modes require additional parameters, like the blend ratio, hilite color or transparency. These parameters are specified as colors in the color table. Special attributes associate the entry with the hilite color, and so on.

```
mode CurMode(shape, mode *, char *);
```

CurMode returns the mode of the shape's style. If the mode is named, then the enumerated mode parameter is set to zero. A style may be passed in place of a shape. Passing nil uses the default frame style; in this case if the default frame style mode is different from any of the other three default styles, a warning is posted.

```
void SetDither(shape orStyle, long ditherLevel);
```

SetDither sets maximum permissible cell allowed for dithers or halftones. A value of zero sets up a 1 x 1 pixel cell.

```
long CurDither(shape orStyle);
```

CurDither returns the cell size allowed for dithers and halftones.

```
void SetPattern(shape orStyle, shape pattern, point *grid);
```

SetPattern copies the pattern and grid vectors into the style. A style may be passed in place of a shape. If nil is passed, both the fill and frame default styles are affected. See also style attributes, above.

```
shape CurPattern(shape orStyle, shape *pattern, point *grid);
```

CurPattern returns the current pattern, if any. A style may be passed in place of a shape. Passing nil uses the default frame style; in this case if the default frame style pattern is different from the default fill style pattern, a warning is posted.

```
void SetStartCap(shape orStyle, shape cap);  
void SetEndCap(shape orStyle, shape cap);
```

SetStartCap and SetEndCap copy the cap shape into the style. A style may be passed in place of a shape. If nil is passed, the default frame style is affected.

The start cap is used at the beginning of a line, or at the initial point in a contour that is not open. The end cap is used at the end of a line, or at the final point of a contour that is not open. (An open contour is a series of vectors within a polygon or path that ends some place different from where it begins.)

Caps are positioned so that the (0,0) point on the cap shape is on top of the end of the line segment. When drawn, the cap is scaled by the current pen thickness, and rotated to the angle at which the line is drawn. Straight up (12 o'clock) causes a 0° rotation. Clockwise rotation of the line causes a positive rotation of the cap. The cap is not clipped by the line.

Caps do not affect rectangle, oval or bitmap drawing. A point, when drawn, uses only the start cap.

```
shape CurStartCap(shape orStyle, shape *cap);  
shape CurEndCap(shape orStyle, shape *cap);
```

CurStartCap and CurEndCap return the current line caps, if any. A style may be passed in place of a shape. Passing nil uses the default frame style.

```
void SetJoin(shape orStyle, jointType, shape join);
```

SetJoin copies the join shape into the style. If no join shape is specified, the jointType sets the style to use a standard join. A style may be passed in place of the initial shape. If nil is passed for the initial parameter, the default frame style is affected.

Standard joins are defined by the following enumerated type:

```

typedef enum {
    shapeJoin,
    flatJoin,
    sharpJoin,
    curveJoin,
    arcJoin
} jointType;

```

Joins are placed at every point along a contour where the slope abruptly changes. Thus, a rectangle has four joins, but an oval or a path constructed out of a rounded rectangle has no joins. If a join shape is specified, the shape is scaled by the current pen thickness and placed, non-rotated, so that the (0,0) point on the join shape coincides with the line intersection defined by the join.

Joins do not affect point, line, arc, curve, oval or bitmap drawing.

```

shape CurJoin(shape orStyle, jointType *, shape *join);

```

CurJoin returns a copy of the join shape, and fills in the pointers to the current shape or the current join type. CurJoin fills in 0 for the join type if the shape exists; otherwise, it returns nil and fills in nil for the join shape pointer. A style may be passed in place of a shape. Passing nil uses the default frame style.

```

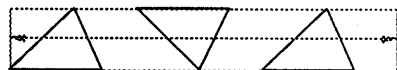
void SetDash(shape orStyle, shape dash, fixed advance);

```

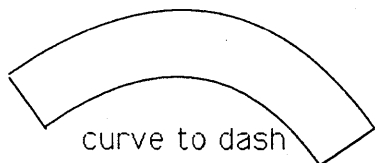
SetDash copies the dash shape into the style. A style may be passed in place of a shape. If nil is passed, the default frame style is affected.

The dash is initially positioned so that the (0,0) point on the dash shape is on the corner of the thick line or curve segment. The dash is scaled vertically to the pen thickness, and it is clipped to the thickness of the line or curve. Thus, parts of the dash which have a y-coordinate greater than 1.0 or less than 0 will always be clipped off. The dash may overlap horizontally if the advance width is smaller than the width of the dash.

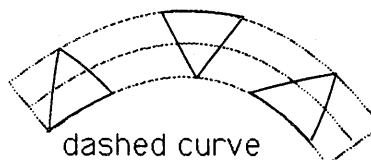
Dashing around curves is done by projecting the control points of the polygon or path of the dashing pattern so that the length of the pattern is preserved, and the distance from the line center to the control point is preserved.



dashing pattern



curve to dash



dashed curve

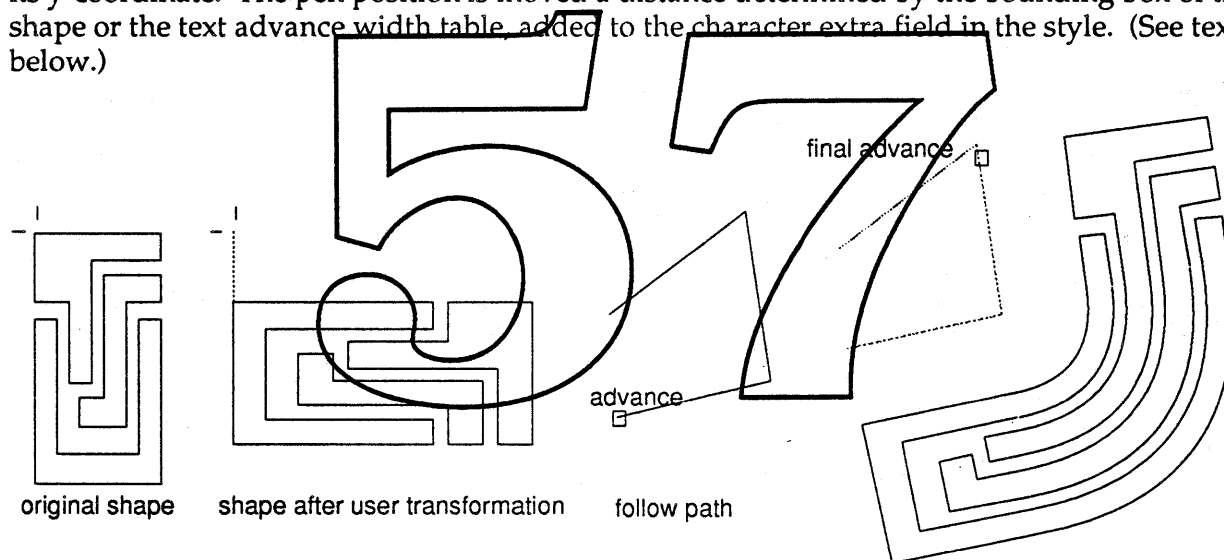
```
shape CurDash(shape orStyle, shape *dash, fixed *advance);
```

CurDash returns the current dashing shape and advance width, if any. A style may be passed in place of a shape. Passing nil uses the default frame style.

```
void SetFollow(shape orStyle, shape followPath, fixed advance,  
joinType);
```

SetFollow copies the follow path into the affected style. The advance field sets the pen position the specified distance down the path. The join type determines the standard join used to connect the shape as it turns a corner. Passing 0 or shapeJoin for the join type causes the shape to be disjointed around corners. Passing negative values for either the advance or joinType causes these parameters to remain unchanged. A style may be passed in place of a shape. If nil is passed, the default text style is affected.

The first shape drawn with a style containing a follow path is positioned as follows; first, the shape's points are clipped and transformed. Next, the shape is translated so that the (0,0) point is on top of the starting point of the path. Finally, each point is moved down the path a distance equal to its x-coordinate, and is moved perpendicular to the path a distance equal to its y-coordinate. The pen position is moved a distance determined by the bounding box of the shape or the text advance width table, added to the character extra field in the style. (See text, below.)



The pen position alone can be moved by:

```
SetFollow(myShape, CurFollow(myShape, nil, nil, nil), newPosition,  
-1);
```

Specifying a non-zero pen position when the follow path is non-existent (equal to nil) posts a warning.

```
shape CurFollow(shape orStyle, shape *followPath, fixed *advance,
                jointType *);
```

CurFollow returns the follow path in the style, if any. A style may be passed in place of a shape. Passing nil uses the default text style.

```
void FixMoveTo(fixed, fixed);
void FixMove(fixed, fixed);
```

If the follow path exists, FixMove and FixMoveTo translate the path to follow in the current style to the indicated pen position. The path's advance is reset to zero, and the join type is unchanged. If no follow path exists, then the pen position is remembered by the style to implement a FixLine or FixLineTo call.

## Color Tables

Styles, ports and devices point to color tables. *Integration with the Palette Manager has not been fully worked out here.*

```
colorTable NewColorTable(colorSpec [], long numberOfEntries);
```

NewColorTable creates a color table with the specified number of indicated colors. The color spec parameter may be nil or the number of entries may be zero, in either case an empty color table is created. The owner count is set to 1.

```
void DisposeColorTable(colorTable);
```

DisposeColorTable decrements the owner count, and if it is zero, throws away the indicated color table.

```
void DisposeColorTableAt(colorTable *);
```

DisposeColorTableAt works just like DisposeColorTable; it allows the shape variable declared to be assigned nil. See the example under DisposeAt.

```
void SetColorTable(colorTable, colorSpec [], long numberOfEntries);
```

SetColorTable replaces the entries in the color table with the specified number of indicated colors. As in NewColorTable, either the color spec parameter may be nil, or the number of entries may be zero.

```
long CurColorTable(colorTable, colorSpec []);
```

CurColorTable returns the number of colors in the color table. The color specification array may be nil; if not, it is up to the user to allocate the proper-sized array to receive the colors.



```
colorTable LoadColorTable(short ID, char *);
```

LoadColorTable returns a color table which is associated with a numbered or named resource. If the ID parameter is 0, the name is used instead.

```
short SaveColorTable(colorTable, short ID, char *name);
```

SaveColorTable returns an ID which associates the color table with a numbered or named resource. If the ID parameter is zero the resource is named; if the name parameter is nil, a new free number is assigned and returned.

### Operations on Color Tables

```
void SetColor(shape, colors, colorSpec *);
```

SetColor changes the color of the first entry in the color table to either the enumerated color or the color specification. The color specification is used if the enumerated color is zero.

```
void SetColorIndex(shape, long index, colors, colorSpec *);
```

SetColorIndex changes the color of any entry in the color table, growing the color table if needed.

```
colors CurColor(shape, colorSpec *);
```

CurColor returns the enumerated color of the first entry in the color table, if possible; if the color specification parameter is not nil, the color specification is returned.

```
colors CurColorIndex(shape, long index, colorSpec *);
```

CurColorIndex works like CurColor, allowing any entry in the color table to be returned.

```
void SetColorAttributes(colorTable, long index, colorAttributes);
```

SetColorAttributes associates the specified color attributes with the specified color table entry.

```
typedef enum {  
    tolerantColor = 1,  
    animatedColor = 2,  
    opColor = 4,  
    hiliteColor = 8,  
    transparentColor = 16  
} colorAttributes;
```

For a tolerant color, the index field of the color specification indicates the tolerance. For an animated color, the index field of the color specification indicates the desired animated index.

```
colorAttributes CurColorAttributes(colorTable, long index);
```

CurColorAttributes returns the color attributes associated with the specified color table entry.

```
colorTable GlobalColors(shape, long portOrder, colorTable *);
```

GlobalColors returns the device colors resulting from drawing the specified shape in the specified port.

### Text Style Routines

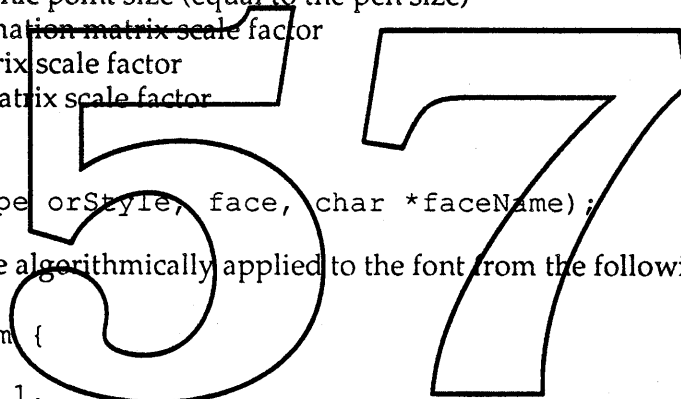
Note: the actual size of the text to be drawn is determined by the concatenation of:

- the typographic point size (equal to the pen size)
- the transformation matrix scale factor
- the port matrix scale factor
- the device matrix scale factor

```
void SetFace(shape orStyle, face, char *faceName);
```

SetFace selects the face algorithmically applied to the font from the following enumerated type:

```
typedef enum {  
    plain,  
    bold = 1,  
    italic = 2,  
    underline = 4,  
    outline = 8,  
    shadow = 0x10,  
    condense = 0x20,  
    extend = 0x40  
} face;
```



If the face passed is negative, then the named face is selected instead. In either case, the face uses the default settings specified by the font or face definition to implement the face algorithm.

```
void SetFaces(shape orStyle, faceList *faces);
```

SetFaces replaces the current face list in the style with the indicated face list. If the faces parameter equals nil, the face list is set to contain only the plain face.

The face list created looks like:

```
typedef struct {
    long numberOfEntries;
    struct {
        long    numberOfParameters;
        long    parameters[];
        face    builtIn;
        char    faceName[]; /* optional, used if face is
negative */
        /* long alignment, if necessary */
    } faceInfo[];
} faceList;
```

For instance, to set the face algorithm to first apply the built-in bold algorithm with its defaults, and then the user-supplied antique algorithm with a parameter equalling 2, an array should be set up as follows:

```
long boldItalic[] = { 2, /* number of entries */
                    0, /* number of bold parameters */
                    bold, /* the built-in algorithm selection */
                    1, /* number of antique algorithms */
                    f(2,0), /* fixed point 2.0 */
                    1, /* denotes a named face */
                    'anti',
                    'que\0' /* antique name as a 'C' string */
};
```

---

**Pascal and other languages note:** the SetFaces and CurFaces calls are implemented in a C library that expects C formatted strings; equivalent calls expecting Pascal formatted strings or unformatted strings (passed with an explicit word or long length) are provided as appropriate language library calls.

---

```
long CurFaces(shape orStyle, faceList *);
```

CurTextFace returns the size of the current face list. The current face list is copied into the passed parameter, if not nil. The handle will be allocated and re-sized as necessary. The copy can be defeated by passing nil for the handle parameter.

Note: It is up to the user to allocate enough memory to store a copy of the face list.

```
fixed FixCharWidth(char ch);
fixed FixStringWidth(char *ch);
fixed FixTextWidth(char *, short length);
```

These routines work the same as their QuickDraw counterparts; they return fixed-point widths instead of integral widths.

```
void CurFontInfo(shape orStyle, fontInfo *);
```

*to be determined by Bass 2.0 and the Layout Manager.*

### Converting Between Local and Global Coordinates

```
void GlobalPoint(point *, long portOrder);
```

GlobalPoint converts the point into global coordinates, using the transformation specified by the port parameter.

```
void LocalPoint(point *, long portOrder);
```

LocalPoint converts the point into local coordinates, using the transformation specified by the port parameter.

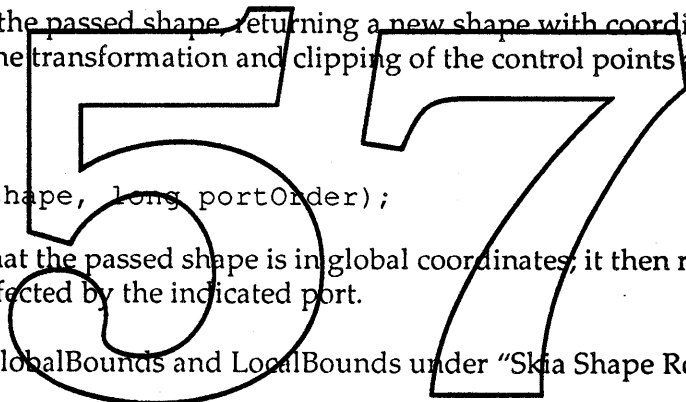
```
void GlobalShape(shape, long portOrder);
```

GlobalShape modifies the passed shape, returning a new shape with coordinates in global space that represents the transformation and clipping of the control points affected by drawing the shape.

```
void LocalShape(shape, long portOrder);
```

LocalShape assumes that the passed shape is in global coordinates, it then returns a new shape, in local coordinates, affected by the indicated port.

See also GlobalArea, GlobalBounds and LocalBounds under "Skia Shape Routines", above.



### Devices

```
long NewDevice(long associate, fixed [3][3], shape *clip, colorTable *,  
bitmap *);
```

NewDevice creates a new device. The associate parameter, if non-zero, associates this new device with an existing device. This is necessary to specify that a new off-screen bitmap will be used to double buffer an existing hardware device. All other parameters may be passed as nil; in this case, the associate device is used to generate in the missing parameters, as described below.

The matrix specifies the mapping between global space and the device's bitmap unit space. Pixels in the device bitmap are addressed from (0.0, 0.0) to (width, height). Passing nil for the matrix copies the associate device's matrix, offset to cause the resulting matrix to place the device in global coordinates so that the device does not overlap any other device.

The clip shape can further restrict the addressability of the bits within the bitmap. Normally, the clip shape is not necessary, since all drawing is restricted to the rectangle describing the bitmap's extent. If the clip parameter is nil, then the clip will equal the associate device's clip. If the clip parameter points to nil, then the device will be unclipped.

The color table specifies the hardware RGB value associated with a pixel index. The bitmap determines the physical address of the bit image, its dimensions, padding and pixel size. If nil is passed for the bitmap parameter, a new bitmap identical to the associate device will be created, except it will have its own bit image, allocated by Skia. The device order returned is the order in which Skia will draw to this device relative to other devices.

Note: Bitmaps with 16 or 32 bits per pixel are treated as direct devices, and do not require a color table. Note also that this call specifies only new software devices, usually off-screen bitmaps. Hardware devices continue to be specified through the Device Manager.

```
void DisposeDevice(long deviceOrder);
```

DisposeDevice disposes of the device and all associated memory structures.

Note: DisposeDevice disposes all ports associated with the device.

```
void SetDevice(long deviceOrder, long *associate, fixed [3][3], shape clip, colorTable *, bitmap *);
```

SetDevice changes the metrics passed in each non-nil parameter as specified.

Note: Changes to devices not allocated by the application may affect all applications currently running. It is up to the application to restore the device when quitting or switching to a different application. A hardware device may have many restrictions, e.g. it may not allow changing the bitmap's base address, or may have a fixed color table. Be sure to check for warnings to see if the call actually took affect.

```
long CurDevice(long deviceOrder, long *associate, fixed [3][3], shape *clip, colorTable *, bitmap *);
```

CurDevice returns the metrics to the structures pointed to by each non-nil parameter associated with the specified device.

```
long CurDevices(shape, long deviceList[]);
```

CurDevices returns the number of devices that the indicated shape intersects when drawn. If nil is passed, CurDevices returns the number of devices available. If the deviceList parameter is non-nil, CurDevices returns a list of the devices. It is up to the user to allocate enough memory to receive a copy of the devices. A Macintosh II can have as many as 6 built-in screen devices.

An example:

```
    if (CurDevices(nil, nil) <= 8)      /* the most devices this snippet can
handle */
    {
        shape deviceClip = nil;
        long myList[8];

        CurDevices(nil, myList);        /* Put the device values in my list */
        CurDevice(myList[0], nil, nil, &deviceClip, nil, nil); /* get the clip
*/
        if (deviceClip) ...            /* do something different if device is clipped
*/
    }
```

```
void SetDeviceAttributes(long deviceOrder, deviceAttributes);
```

SetDeviceAttributes changes the device to be:

- an indexed device that requires a color table
- a fixed device that requires a color table but cannot be changed
- a direct device without a color table

```
typedef enum {
    indexed,
    direct,
    fixed
} deviceAttributes;
```

```
deviceAttributes CurDeviceAttributes(long deviceOrder);
```

CurDeviceAttributes returns the attributes of the device.

```
void SetDeviceSpot(long deviceOrder, fixed spotSize);
```

SetDeviceSpot specifies the size of the pixel as a spot that may overlap other pixels. A setting of 1.0 sets the spot to a circle that completely encloses one pixel grid unit.

```
fixed CurDeviceSpot(long deviceOrder);
```

CurDeviceSpot returns the size of the pixel for the specified device.

```
GDHandle CurGDevice(long deviceOrder);
```

CurGDevice returns the QuickDraw hardware graphics device associated with a Skia device, or nil, if none exists.

## Resources

Like all other Skia data structures, Skia resources are not public. They can be accessed through the Load and Save routines, and differ from more general resources in that they may be loaded or unloaded, in whole or in part, without any effort on the part of the application.

Shapes (including bit images), styles, transforms and color tables may be disk resident. Ports and devices (except for the bit image pointed to by the device) are always memory resident.

Set the direct shape attribute to force a shape to be memory resident.

---

## CUSTOMIZING SKIA OPERATIONS

---

Skia drawing can be monitored and modified by specifying custom operations. When a Skia shape is drawn, it goes through the following steps:

if something about the transform has changed,

- update the clipping description
- update the sum of the point transformation matrices

if something about the shape's geometry has changed (as drawn),

- describe the shape to be drawn in terms of filled geometry
- determine the shape's bounding box

for each port that the shape intersects,

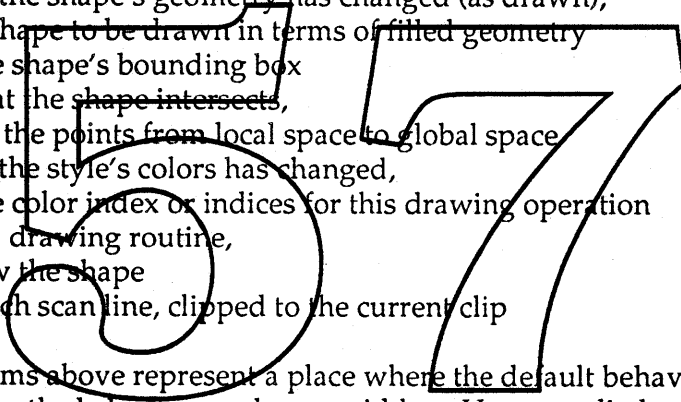
- transform the points from local space to global space

if something about the style's colors has changed,

- determine the color index or indices for this drawing operation

for each port with a drawing routine,

- set up to draw the shape
- copy or fill each scan line, clipped to the current clip



Each of the bulleted items above represent a place where the default behavior is selected from a list of routines, or where the behavior can be overridden. User supplied routines for any of these steps may call the system routine before or after the user routine, or, in some cases, not at all.

Customizing a shape, style or transform causes the drawing affected to perform somewhat slower, because the custom routines, unlike the internal routines, do not have access to internal data structures. Installing a custom routine will not guarantee that it is called each time a shape is drawn:

- the custom routine is specific to only the shape, style, device or transform specified.
- the custom routine will only be called if the appropriate drawing state has changed.
- the custom routine is called in remote imaging situations when appropriate.

For instance, in the case where a graphics accelerator provides access to the only device present in the system, all of the execution may take place on the accelerator card.

At worst, a drawing operation takes all nine steps to complete, but typically, Skia avoids one or more of the nine by keeping state about the drawing environment around, and by aborting calls as soon as Skia can detect that the call will have no effect.

Skia shapes, at any point in the drawing process can be parsed; pictures can be parsed as well. See "Skia Shape Routines" about parsing Skia shapes, and "Pictures" about parsing pictures.

### Custom Routines

```
void UserClip(shape orTransform, warning (*userFunction)(), long length);
```

UserClip installs a user supplied function in the indicated transform that in turn is called during a Draw call when the local or global coordinates clips have changed. The called function is given the opportunity to change the local transform clip, the global port clip, or the global device clip. The function must be of the form:

```
warning MyUserClip(shape *transformClip, shape *portClip, shape *deviceClip);
```

Skia always passes valid pointers, but any or all of the pointers may point to nil. The function may use DisposeAt to eliminate the clips, or if passed at pointer to nil, may allocate new clips and store them in the pointers. The clips may be modified with any of the Skia routines. The function returns a warning to abort the operation.

```
void UserMatrix(shape orTransform, warning (*userFunction)(), long length);
```

UserMatrix installs a user supplied function in the indicated transform that in turn is called during a Draw call when one of the transformation matrices have changed. The called function is given the opportunity to change the local transformation matrix, the port matrix, the device matrix, or determine the resulting matrix that is used to represent the concatenation of the individual matrices. The function must be of the form:

```
warning MyUserMatrix(long portOrder, fixed [3][3]local, fixed [3][3]port, fixed [3][3]device, fixed [3][3]output);
```

The user matrix call is made once per port contained within the transform. All matrices are pre-allocated and passed to the call; the output matrix is not set to any meaningful values. The user matrix may calculate the output matrix, or call SystemMatrix, below, to do the work.

Note: the matrix math package includes routines for concatenating matrices.

```
void UserPrimitive(shape, warning (*userFunction)(), long length);
```

UserPrimitive installs a user supplied function that is called each time the shape is drawn, when something about the geometry or style of the shape has changed. The called function is



given the opportunity to change the shape and style before Skia applies the style information to the shape and produces a drawable primitive. The user function must be of the form:

```
warning MyUserPrimitive(shape);
```

Note: Call the Primitive routine to break down a shape into the simple primitives that are used to draw it.

```
void UserSect(shape, warning (*userFunction)(), long length);
```

UserSect installs a user supplied function that is called once for each port that the shape may intersect. The called function should affect the intersects boolean to determine whether the shape intersects or not. If the intersects value is left unchanged, the intersection will be calculated by Skia.

```
warning MyUserSect(long portOrder, shape, boolean *intersects);
```

Note: Use GlobalShape to determine the device coordinates of a shape on a device.

```
void UserTransform(shape, warning (*userFunction)(), long length);
```

UserTransform installs a user supplied function that is called once for each port that the shape intersects. The called function should modify the shape as clipped to the local clip, transformed by the matrix, then clipped by the global clip.

```
warning MyUserTransform(long portOrder, shape, shape *localClip,  
    fixed [3][3], shape *globalClip);
```

```
void UserColors(shape, warning (*userFunction)(), long length);
```

UserColors installs a user supplied function that is called once per port that the shape intersects; given the shape and number of colors, color indices corresponding to entries in the device color table, or direct device values, should be returned in the pre-allocated array.

```
warning MyUserColors(long portOrder, shape, long numberColors,  
    long []colorIndices);
```

```
void UserSetup(shape, warning (*userFunction)(), long length);
```

UserSetup installs a user supplied function that is called once per port that the shape intersects. The function should initialize any state necessary to support the user mode routine defined below.

```
warning MyUserSetup(long device, shape);
```

```
void UserMode(long device, char *modeName, warning (*function)(), long
length, boolean longs);
```

UserMode makes the user defined drawing mode available under the indicated name, selected with the SetMode routine. If the longs parameter is true, the user function must look like:

```
warning MyUserMode(long *dest, long *color, long count, long
edge1, long edge2);
```

In this format, the user function should apply the indicated color or colors, starting at dest. Note that for flat filled drawing, a pointer to a single color will be passed. For bitmap drawing, the color array points to the source bitmap bit image. It is up to the user to keep track of what kind of drawing is going on. The count indicates the number of longs to apply the color. Edge1 and edge2 are masks indicating which pixels in the initial and final long are to be affected. If the count equals 1, only edge1 should be used.

If the longs parameter is false, the user function should look like:

```
warning MyUserMode(long *dest, long *color, long bitOffset,
long pixelCount);
```

In this format, the user function should apply the indicated color starting at the bit offset from dest, until the pixel count is exhausted.

In either case, the function should return 0 or a warning if it was unable to complete the operation. To disable the user function, pass nil for the user function parameter.

```
void UserWidths(shape orStyle, warning (*userFunction)(), long length);
```

UserWidths allows the application to intercept and modify the widths (*and styles?*) of the characters to be drawn. The user function called must look like

```
warning MyUserWidths(char *string, short length, layout *);
```

The function may modify the layout record, which corresponds to the horizontal and vertical displacements for the string of the indicated length. The function should return 0 or a warning if it was unable to complete the operation. To disable the user function, pass nil for the user function parameter.

*The format of the layout structure is to be agreed upon with the Layout Manager group, as well as when the call is made.*

Note to C programmers: the string may not be null terminated.

```
void UserFace(char *faceName, warning (*function)(), long length);
```

UserFace makes the user defined text face available under the indicated name, selected with the SetFace routine. The user function must look like:

```
warning MyUserFace(char *string, short length, layout *, shape *,
    short numberOfParameters, ...);
```

Like the user widths procedure, the user face receives the entire string to be modified, and may modify the widths contained in the layout structure. Additionally, it is passed a shape which is the composite of all shapes necessary to describe the string, positioned in ideal space by the Layout Manager. If the layout structure is nil, then the caller does not require the modified widths. If the shape passed is nil, the caller (usually the Layout Manager) is requesting that only the custom widths be returned. The function should modify the passed shape stylistically as defined by the shape. The function will be passed the number of parameters specified by the face list, which may be zero. The function should return 0 or a warning if it was unable to complete the operation.

To disable the user function, pass nil for the user function parameter in the UserFace call.

---

## SKIA'S INTERPRETATION OF QUICKDRAW ROUTINES

---

Here's how Skia uses QuickDraw calls to draw using the transform and style, and modify a shape's geometry. A QuickDraw routine called within the scope of Skia may:

- call the original ROM routine unchanged.
- be changed via the macro mechanism into an equivalent Skia routine.
- call "glue" that implements the functionality of the routine primarily through Skia calls.
- have no effect.

The calls that fit into the first category are untouched; the original QuickDraw routines are called just as if Skia did not exist. These calls do not access internal data structures, and do not impact drawing, so no modification is necessary.

Most calls fit into the second and third categories. Only a couple of QuickDraw calls have no equivalent in the Skia architecture.

---

**Pascal note:** There is no equivalent to the C language macro mechanism in Pascal. An MPW tool will be provided that translates as many of the direct map calls from QuickDraw to Skia as possible; the remainder will be supported by library glue.

---

### GrafPort Routines

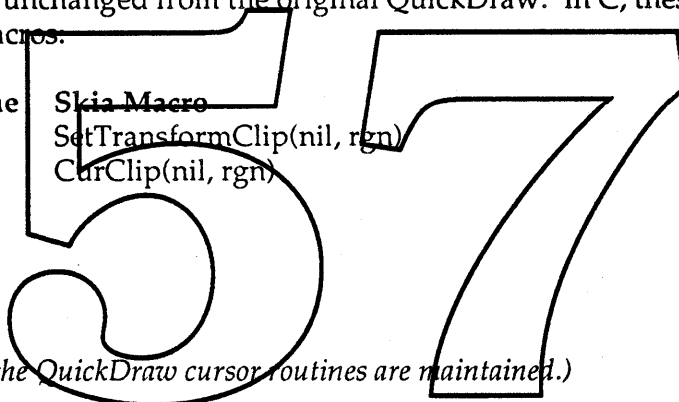
The Window Manager is the logical access point for managing the grafPorts. A Skia-Window Manager-wise application would never call these routines.

QuickDraw Routine	"Glue" Functionality
InitGraf	Initialize Skia.
OpenPort, OpenCPort	Create a default style and transform, and point the grafPort to them.
InitPort, InitCPort	Initialize the current default style and transform.
ClosePort, CloseCPort	Delete the default styles and transform pointed to by the grafPort.

SetPort	Make the default styles and transform the current default.
GetPort	Return the current grafPort.
GrafDevice	Has no effect.
SetPortBits <sup>1</sup>	Direct all drawing to the indicated bitmap by replacing the default transforms' port list with a device specifically allocated for off-screen drawing.
PortSize, MovePortTo	Has no effect. This functionality is taken care of by the Window Manager.
SetOrigin <sup>2</sup>	Changes the x and y translation vectors of the current transformation matrix to the difference between the bounding box of the port's clip projected into source space, and the upper left corner the parameters passed in SetOrigin.
BackPat	BackPat sets the second color table entry in the default fill style to a gray level corresponding to the QuickDraw patterns white, ltGray, gray, dkGray or black, if appropriate. BackPat does not affect the default style's pattern.
ClipRect(r)	Sets the default transform's clip to the integer rectangle.

The following calls are unchanged from the original QuickDraw. In C, these calls are implemented as the macros.

QuickDraw Routine	Skia Macro
SetClip(rgn)	SetTransformClip(nil, rgn)
GetClip(rgn)	CurClip(nil, rgn)



## Cursors

*(For now, I assume that the QuickDraw cursor routines are maintained.)*

## Regions

Many QuickDraw region calculation procedures have exact Skia analogies:

QuickDraw Routine	Skia Macro
NewRgn()	New(emptyType)
DisposeRgn(rgn)	Dispose(rgn)
CopyRgn(a, b)	CopyTo(b, a)
SetEmptyRgn(rgn)	Type(rgn, emptyType)
SetRectRgn(a, b, c, d)	Set4(rectangleType, f(a,0), f(b,0), f(c,0), f(d,0));
RectRgn(r)	Set4(rectangleType, f(r->left,0), ...);

<sup>1</sup> To draw to an off-screen bitmap, allocate the bitmap with a NewDevice call. See "Devices", below.

<sup>2</sup> This call has a new meaning; existing programs that make this call may get the wrong result. Because there is no Skia equivalent to a portRect, the origin offset may not be calculated correctly. Instead, the new application should change the transformation matrix with the Offset, Move or MoveTo calls.

OffsetRgn(rgn, dh, dv) Offset(rgn, f(dh, 0), f(dv, 0));  
 InsetRgn(rgn, dh, dv) Inset(rgn, f(dh, 0), f(dv, 0));  
 EqualRgn(a, b) Equal(a, b)  
 EmptyRgn(a) (CurType(a) == emptyType)

The remaining QuickDraw region operations are:

**QuickDraw Routine** "Glue" Functionality  
**OpenRgn** Initializes rgnSave, a global swapped by SetPort, to a shape of emptyType. All drawing calls then call AddTo to add the shape's geometry to rgnSave.  
**CloseRgn** Copies the shape collected into rgnSave into the specified region. The drawing calls are directed to quit saving the geometry into rgnSave.  
**SectRgn, UnionRgn, DiffRgn, XorRgn** Work as defined by QuickDraw. Rather than setting the destination to the empty rectangle, the destination shape is set to be of type emptyType.  
**PtInRgn, RectInRgn** Work as defined by QuickDraw. RectInRgn returns true only if the rect intersects the specified region.

Note: The shape routine AddTo provides an alternative approach to collecting a region through drawing operations. See "Skia Shape Routines", above, for a description.

shape RgnSave(void);

RgnSave returns the save region. Since all Skia data structures are private, this provides the equivalent functionality as "reaching" into the port to access the rgnSave field.

### Graphic Operations on Regions

These routines use the current transform and the current frame or fill styles to draw the shape.

**QuickDraw Routine** "Glue" Functionality  
**FrameRgn** Use the default frame style.  
**PaintRgn** Use the current mode to select the color in the default fill style.  
**EraseRgn** Fill in copy mode, using color 2, using the default fill style.  
**InvertRgn** Fill in hilite mode using the default fill style.  
**FillRgn** Select the correct gray to match the pattern, then use the default fill style.  
**FillCRgn** Use the default fill style with the selected pattern.

### Pen Styles

**QuickDraw Routine** Skia Macro  
**HidePen()** Hide(nil)  
**ShowPen()** Show(nil)

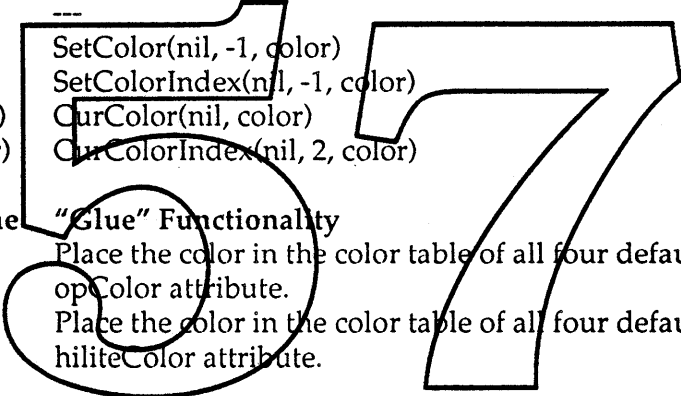
PenSize(width, high)	SetPen(nil, f(width,0))
GetPenState(pnState)	CopyToStyle(pnState, nil)
SetPenState(pnState)	CopyToStyle(nil, pnState)
MoveTo(h, v)	FixMoveTo(f(h, 0), f(v,0))
Move(h, v)	FixMove(f(h,0), f(v,0))
PenNormal()	Reset(nil)
PenPat(pat)	SetColor(nil, pat, nil)

<b>QuickDraw Routine</b>	<b>"Glue" Functionality</b>
GetPen	Return the integral advance on the follow path, or the pen position.
PenMode	Change the default frame and fill styles to the mode specified.

## Colors

<b>QuickDraw Routine</b>	<b>Skia Macro</b>
GetCTable(ctID)	LoadColorTable(ctID, nil)
DisposeCTable(ctTab)	DisposeColorTable(ctTab)
ForeColor(color)	SetColor(nil, color, nil)
BackColor	SetColorIndex(nil, color, 2, nil)
ColorBit(whichBit) <sup>1</sup>	---
RGBForeColor	SetColor(nil, -1, color)
RGBBackColor	SetColorIndex(nil, -1, color)
GetForeColor(color)	CurColor(nil, color)
GetBackColor(color)	CurColorIndex(nil, 2, color)

<b>QuickDraw Routine</b>	<b>"Glue" Functionality</b>
OpColor(color)	Place the color in the color table of all four default styles with the opColor attribute.
HiliteColor(color)	Place the color in the color table of all four default styles with the hiliteColor attribute.



## Lines

<b>QuickDraw Routine</b>	<b>Skia Macro</b>
LineTo(h, v)	FixLineTo(f(h, 0), f(v,0))
Line(h, v)	FixLine(f(h,0), f(v,0))

## Text

<b>QuickDraw Routine</b>	<b>"Glue" Functionality</b>
TextFont	set the default text style to use the indicated font family.
TextMode(mode)	SetMode(CurDefaultStyle(textType), mode)

<sup>1</sup> ColorBit is not supported in Skia; Skia supports a chunky color model. Skia does support the multiple pass application of color that is required for multicolor ribbon printing, through the creation of multiple ports and color filters. See the section on "Ports", above.

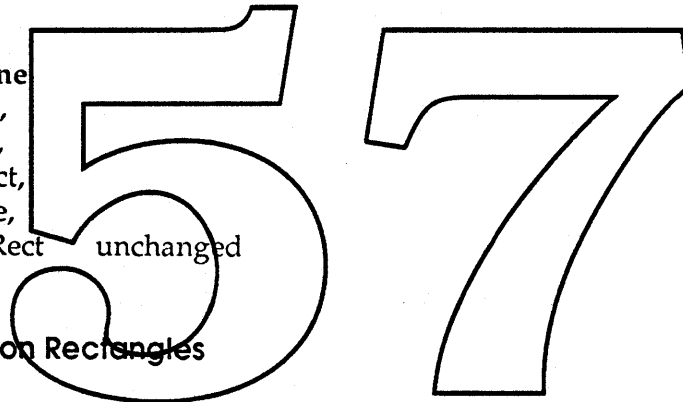
TextFace	set default text style family's face, or algorithmically-derived face
TextSize(size)	SetPen(CurDefaultStyle(textType), size)
DrawChar, DrawString, DrawText	Draw the text using the default transform and default text style.
CharWidth, StringWidth, TextWidth	Return the integral width (no change).
SpaceExtra	Specify inter-word spacing for follow path.
CharExtra	Specify inter-character (or inter-shape) spacing for follow path.
GetFontInfo	Returns the integral font coordinates (no change).

See "Customizing Skia Operations" about a user procedure that enables individual character widths to be changed when drawn.

*I assume that the interface for selecting the script, font family and face, which currently exists in both the Font Manager (by name) and in QuickDraw (by number) will be resolved between the Layout Manager group and the BASS group. If that is not the case, please let me know.*

### Rectangles

**QuickDraw Routine**  
SetRect, OffsetRect,  
InsetRect, SectRect,  
UnionRect, PtInRect,  
Pt2Rect, PtToAngle,  
EqualRect, EmptyRect



### **Graphic Operations on Rectangles**

These routines use the current transform and the current frame or fill styles to draw the integer rectangle.

<b>QuickDraw Routine</b>	<b>"Glue" Functionality</b>
FrameRect	Use the default frame style
PaintRect	Use the current mode to select the color in the default fill style.
EraseRect	Fill in copy mode, using color 2, using the default fill style.
InvertRect	Fill in hilite mode using the default fill style.
FillRect	Select the correct gray to match the pattern, then use the default fill style.
FillCRect	Use the default fill style with the selected pattern.

### Ovals

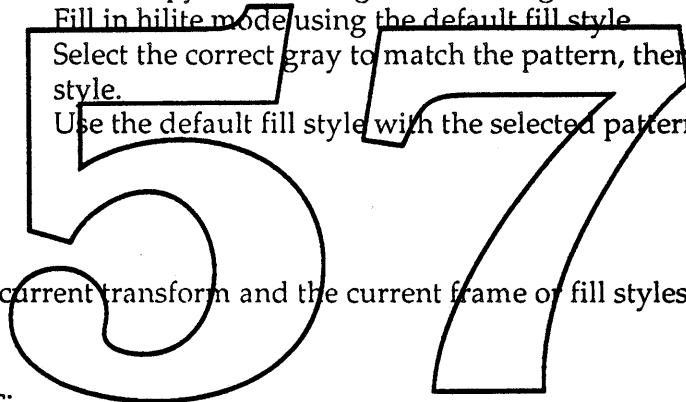
These routines use the current transform and the current frame or fill styles to draw the integer oval.

<b>QuickDraw Routine</b>	<b>"Glue" Functionality</b>
FrameOval	Use the default frame style
PaintOval	Use the current mode to select the color in the default fill style.
EraseOval	Fill in copy mode, using color 2, using the default fill style.
InvertOval	Fill in hilite mode using the default fill style.
FillOval	Select the correct gray to match the pattern, then use the default fill style.
FillCOval	Use the default fill style with the selected pattern.

### Rounded-Corner Rectangles

These routines use the current transform and the current frame or fill styles to draw the integer rounded-corner rectangle.

<b>QuickDraw Routine</b>	<b>"Glue" Functionality</b>
FrameRRect	Use the default frame style
PaintRRect	Use the current mode to select the color in the default fill style.
EraseRRect	Fill in copy mode, using color 2, using the default fill style.
InvertRRect	Fill in hilite mode using the default fill style.
FillRRect	Select the correct gray to match the pattern, then use the default fill style.
FillCRRect	Use the default fill style with the selected pattern.



### Arcs

These routines use the current transform and the current frame or fill styles to draw the integer arc or wedge.

FrameArc draws an arc:

<b>QuickDraw Routine</b>	<b>"Glue" Functionality</b>
FrameArc	Use the default frame style

The rest draw a wedge:

<b>QuickDraw Routine</b>	<b>"Glue" Functionality</b>
PaintArc	Use the current mode to select the color in the default fill style.
EraseArc	Fill in copy mode, using color 2, using the default fill style.
InvertArc	Fill in hilite mode using the default fill style.
FillArc	Select the correct gray to match the pattern, then use the default fill style.
FillCArc	Use the default fill style with the selected pattern.

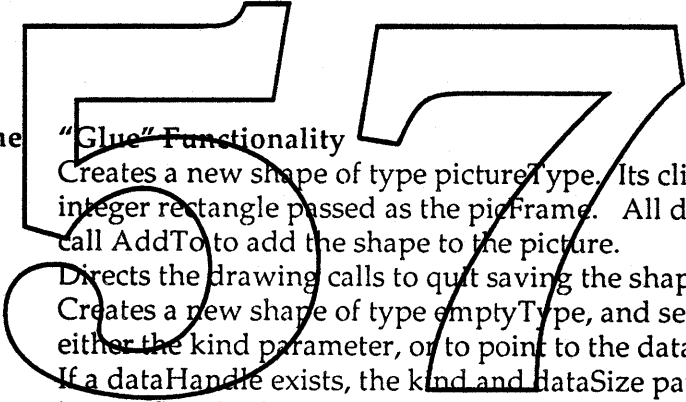


## Bitmaps

<b>QuickDraw Routine</b>	<b>"Glue" Functionality</b>
ScrollRect	Works as defined by QuickDraw. If the current transformation includes perspective, then ScrollRect won't actually scroll anything; instead, the updateRgn will be set to the entire scrolling rectangle.
CopyBits	Works a differently from how it is defined by QuickDraw, because the bitmap data structure is different. The srcRect and dstRect are used to determine the translation and scaling, in addition to the current transform. The mode overrides the transfer mode in the current style. If the dstBits bitmap is also a device bitmap, then the maskRgn is converted to local coordinates, and then is intersected with the transform's clip. Otherwise, the transform clip is overridden by the maskRgn, and the port list is ignored.

Because CopyBits is so different from the rest of the calls in Skia and QuickDraw, you may want to consider using Draw or DrawBitmap instead. Draw and DrawBitmap require that all destination bitmaps are pointed to by a device. The device can be either a hardware device or an off-screen bitmap. Draw and DrawBitmap can then replace CopyBits. See "Devices" about creating your own off-screen bitmap.

## Pictures



<b>QuickDraw Routine</b>	<b>"Glue" Functionality</b>
OpenPicture	Creates a new shape of type pictureType. Its clip is set to the integer rectangle passed as the picFrame. All drawing calls then call AddTo to add the shape to the picture.
ClosePicture	Directs the drawing calls to quit saving the shapes into the picture.
PicComment	Creates a new shape of type emptyType, and sets its user field either the kind parameter, or to point to the dataHandle parameter. If a dataHandle exists, the kind and dataSize parameters are inserted as the first long word of the data.
DrawPicture	Draws each shape contained in the shape list. DrawPicture sets the transform to scale and translate the picture so that the picture's clipped bounds fit into the destination rectangle.

Use Draw to draw a picture without specifying a frame.

<b>QuickDraw Routine</b>	<b>Skia Macro</b>
KillPicture(picHandle)	Dispose(picHandle)

## Polygons

Here's how Skia interprets the QuickDraw polygon calls. Note that OpenPoly and OpenRgn perform exactly the same function, only to different temporary shape globals.

**QuickDraw Routine** "Glue" Functionality  
**OpenPoly** Initializes polySave, a global swapped by SetPort, to a shape of emptyType. All drawing calls then call AddTo to add the shape's geometry to polySave.  
**ClosePoly** Copies the shape collected into polySave into the specified shape. The drawing calls are directed to quit saving the geometry into polySave.

**QuickDraw Routine** Skia Macro  
**KillPoly(poly)** Dispose(poly)  
**OffsetPoly(poly, dh, dv)** Offset(poly, f(dh, 0), f(dv, 0))

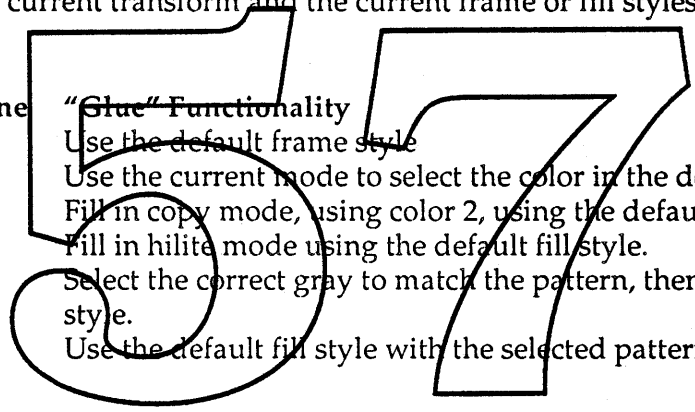
shape PolySave(void);

PolySave returns the save shape created by OpenPoly. See RgnSave, above.

**Graphic Operations on Polygons**

These routines use the current transform and the current frame or fill styles to draw the integer rectangle.

**QuickDraw Routine** "Glue" Functionality  
**FramePoly** Use the default frame style  
**PaintPoly** Use the current mode to select the color in the default fill style.  
**ErasePoly** Fill in copy mode, using color 2, using the default fill style.  
**InvertPoly** Fill in hilite mode using the default fill style.  
**FillPoly** Select the correct gray to match the pattern, then use the default fill style.  
**FillCPoly** Use the default fill style with the selected pattern.



**Coordinate Conversion**

The QuickDraw routines LocalToGlobal and GlobalToLocal routines are implemented as follows:

**QuickDraw Routine** "Glue" Functionality  
**LocalToGlobal,**  
**GlobalToLocal** return the point as specified by the first port in the default transform

**Calculations with Points**

**QuickDraw Routine**  
**AddPt, SubPt,**  
**SetPt, EqualPt** unchanged

Equivalent Skia routines for fixed-point points are not provided.

## Devices and gDevices

Note: The Graphics Devices calls are normally available only on the Macintosh II and newer 68020 and 68030 machines. Under Skia, the following Graphics Devices calls are available on all machines: NewGDevice, InitGDevice, GetGDevice, SetGDevice, DisposGDevice, GetDeviceList, GetMainDevice, GetNextDevice, SetDeviceAttribute, TestDeviceAttribute and GetMaxDevice. The calls' definition and implementation are are not affected by Skia.

QuickDraw Routine	Notes
NewGDevice, DisposGDevice, InitGDevice	no affect if made after boot time
GetGDevice, SetGDevice	unaffected
GetDeviceList, GetMainDevice, GetNextDevice	
SetDeviceAttribute, TestDeviceAttribute, TestDeviceAttribute, GetMaxDevice	intercepted by Skia to keep Skia's devices current unaffected

Use CurDevices to get a list of available Skia devices, and CurGDevice to map the Skia devices to gDevices.

## Miscellaneous Routines

QuickDraw Routine	
Random, StuffHex	
ScalePt	
MapPt, MapRect	unaffected

The map routines are similar to the Box utility:

QuickDraw Routine	"Glue" Functionality
MapRgn, MapPoly	Scale and translate the shape as indicated by the rectangles.

The note about detecting memory overflows in Inside Macintosh does not apply to Skia.

QuickDraw Routine	"Glue" Functionality
GetPixel, GetCPixel	Get the color mapped by the first port in the default transform.

SetCPixel

Set the pixel to the specified color, mapping from local coordinates to the global coordinates determined by the first port.

Note: GetPixel and GetCPixel present a problem, because Skia allows a single local coordinate to be mapped to multiply global coordinates. It's better to use GlobalColors, instead.

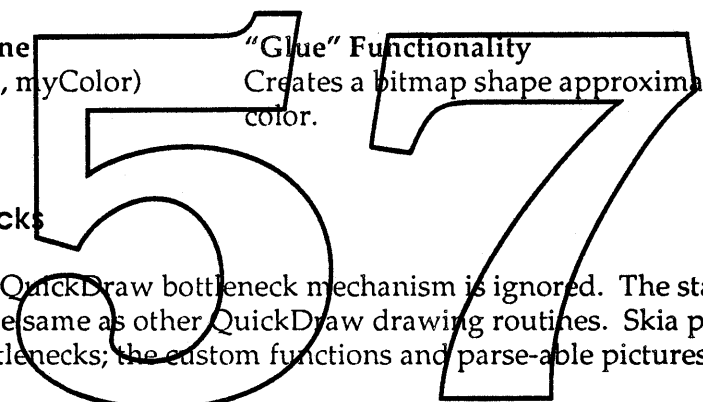
**Additional Color QuickDraw Routines**

Pixel patterns and pixel maps are treated as general shapes in Skia, not special structures.

**Operations on Pixel Patterns**

QuickDraw Routine	Skia Macro
NewPixPat(), NewPixMap()	New(emptyType)
DisposPixPat(a), DisposePixMap(a)	Dispose(a)
CopyPixPat(source, destination), CopyPixMap(source, destination)	CopyTo(destination, source)
GetPixPat(patID)	Load(patID, nil)

QuickDraw Routine	"Glue" Functionality
MakeRGBPat(ppat, myColor)	Creates a bitmap shape approximating the given color.



**QuickDraw Bottlenecks**

For the most part, the QuickDraw bottleneck mechanism is ignored. The standard drawing routines are treated the same as other QuickDraw drawing routines. Skia provides two alternatives to the bottlenecks; the custom functions and parse-able pictures.

QuickDraw Routine	"Glue" Functionality
SetStdProcs	Ignored. See the routines above.
StdText, StdLine, StdRect, StdRRect, StdOval, StdArc, StdPoly, StdRgn, StdBits	Work as in QuickDraw.
StdComment	Ignored.
StdTxMeas	works the same as TextWidth.
StdGetPic, StdPutPic	Ignored.

**QuickDraw Resources**

QuickDraw pictures, versions 1 and 2, are translated into Skia pictures on the fly. All other QuickDraw resources must be translated into Skia resources at compile time.

---

## SKIA INTERFACE SUMMARY

---

### Constants

```
typedef enum {
    noType,
    emptyType,
    pointType,
    lineType,
    arcType,
    curveType,
    rectangleType,
    ovalType,
    polygonType,
    pathType,
    bitmapType,
    textType,
    fullType,
    pictureType
} shapeType;
```

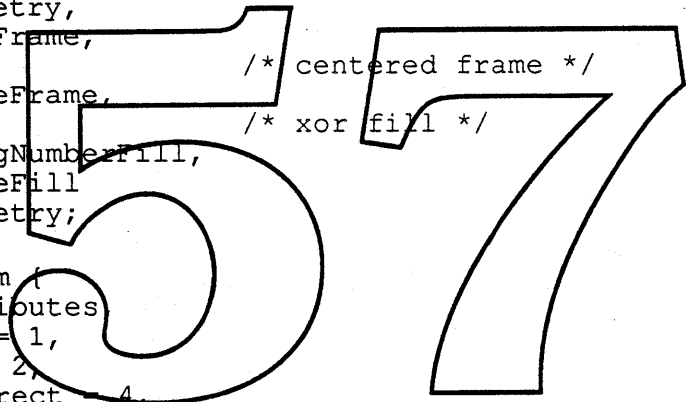
```
typedef enum {
    noGeometry,
    insideFrame,
    frame, /* centered frame */
    outsideFrame,
    fill, /* xor fill */
    windingNumberFill,
    inverseFill
} shapeGeometry;
```

```
typedef enum {
    noAttributes,
    cache = 1,
    lock = 2,
    keepDirect = 4,
    keepRemote = 8
} shapeAttributes;
```

```
typedef enum {
    sourceGrid = 1,
    deviceGrid = 2,
    devicePattern = 4
} styleAttributes;
```

```
typedef enum {
    tolerantColor = 1,
    animatedColor = 2,
    opColor = 4,
    hiliteColor = 8,
    transparentColor = 16
} colorAttributes;
```

```
typedef enum {
    indexed,
    fixed,
```



```

    direct
} deviceAttributes;

#define Pattern colors
#define ltGray light+gray
#define dkGray dark+gray

#define blackColor black
#define whiteColor white
#define redColor red
#define greenColor green
#define blueColor blue
#define cyanColor cyan
#define magentaColor magenta
#define yellowColor yellow

typedef enum {
    rgbSpec,
    black,
    white,
    red,
    green,
    blue,
    cyan,
    magenta,
    yellow,
    gray,
    grey = gray,
    light = 16,
    dark = 32
} colors;

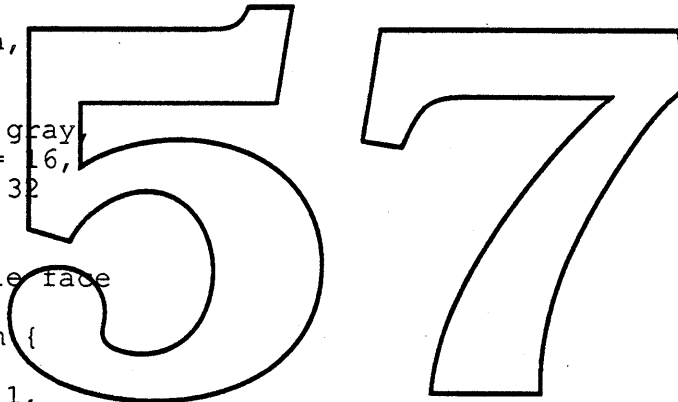
#define Style face

typedef enum {
    plain,
    bold = 1,
    italic = 2,
    underline = 4,
    outline = 8,
    shadow = 0x10,
    condense = 0x20,
    extend = 0x40
} face;

#define srcCopy copy
#define patCopy copy
#define srcXor hilite
#define patXor hilite

typedef enum {
    userMode,
    copy,
    addOver,
    addPin,
    subOver,
    subPin,

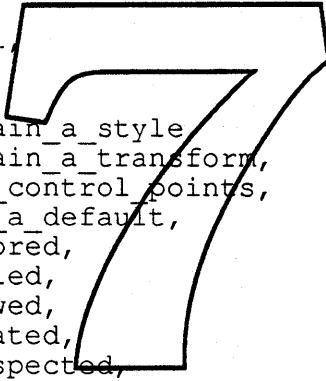
```



```
reverseSubPin,  
reverseSubOver,  
max,  
min,  
blend,  
hilite,  
transparent,  
saveAndCopy,  
eraseAndCopy  
} mode;
```

```
typedef enum {  
    shapeJoin,  
    flatJoin,  
    sharpJoin,  
    curveJoin,  
    arcJoin  
} joinType;
```

```
typedef enum {  
    noWarning,  
    firstWarningID,  
    graphic_type_passed_equals_nil,  
    shape_passed_equals_nil,  
    style_passed_equals_nil,  
    transform_passed_equals_nil,  
    port_passed_equals_nil,  
    shape_already_disposed,  
    graphic_type_does_not_contain_a_style,  
    graphic_type_does_not_contain_a_transform,  
    graphic_type_does_not_have_control_points,  
    graphic_type_does_not_have_a_default,  
    graphic_type_cannot_be_colored,  
    graphic_type_cannot_be_scaled,  
    graphic_type_cannot_be_skewed,  
    graphic_type_cannot_be_rotated,  
    graphic_type_cannot_be_perspected,  
    graphic_type_cannot_be_reset,  
    new_shape_contains_invalid_data,  
    style_frame_modified_for_filled_shape,  
    type_already_set,  
    pen_size_already_set,  
    join_type_already_set,  
    start_cap_already_set,  
    end_cap_already_set,  
    dash_already_set,  
    attributes_already_set,  
    cache_already_cleared,  
    color_index_requested_not_found,  
    port_passed_not_found_in_port_list,  
    contour_count_returned_for_first_list_element_only,  
    contour_request_exceeded_contours_in_shape,  
    point_request_exceeded_points_in_contour,  
    graphic_type_does_not_have_multiple_contours,  
    bad_parameter_passed_to_Line,  
    bad_parameter_passed_to_Point,  
    extra_data_passed_was_ignored,
```



```

    shape_passed_has_no_bounds,
    point_expected,
    onePastLastWarning,
    lastWarningID = onePastLastWarning - 2
} warning;

```

```

typedef enum {
    noError,
    firstErrorID,
    heap_allocator_failed,
    heap_reallocator_failed,
    translate_shape_out_of_range,
    scale_shape_out_of_range,
    rotate_shape_out_of_range,
    perspective_shape_out_of_range,
    illegal_type_for_shape,
    illegal_type_for_device,
    illegal_type_for_style,
    illegal_type_for_transform,
    illegal_type_for_add_argument,
    number_of_contours_exceeds_implementation_limit,
    number_of_points_exceeds_implementation_limit,
    size_of_polygon_exceeds_implementation_limit,
    functionality_unimplemented,
    onePastLastError,
    lastErrorID = onePastLastError - 2
} error;

```

**Data Types**

```

typedef struct {
    fixed x;
    fixed y;
} point;

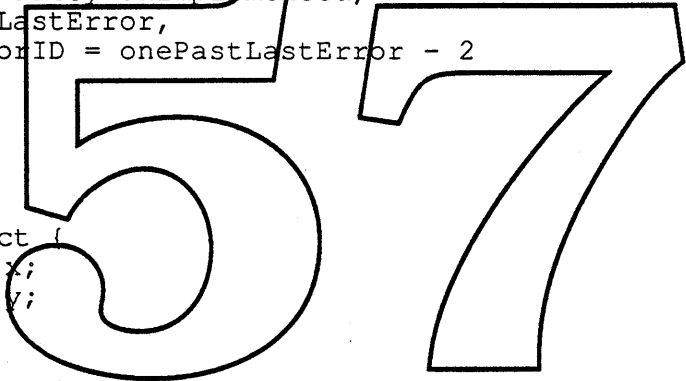
typedef struct {
    point start;
    point end;
} line;

typedef struct {
    point start;
    point control;
    point end;
} curve;

typedef struct {
    fixed left;
    fixed top;
    fixed right;
    fixed bottom;
} rectangle;

typedef struct {
    fixed left;

```





```

    fixed top;
    fixed right;
    fixed bottom;
} oval;

typedef struct {
    long vectors;
    point vector[];
} polygon;

typedef struct {
    long contours;
    polygon contour[];
} polygons;

typedef struct {
    long vectors;
    long controlBits[];
    point vector[];
} path;

typedef struct {
    long contours;
    path contour[];
} paths;

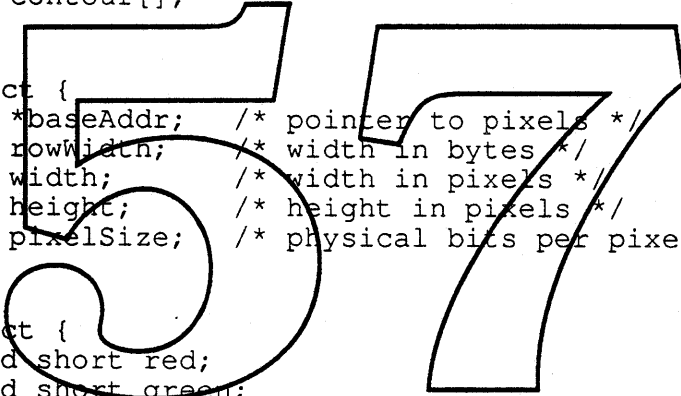
typedef struct {
    char *baseAddr; /* pointer to pixels */
    long rowWidth; /* width in bytes */
    short width; /* width in pixels */
    short height; /* height in pixels */
    short pixelSize; /* physical bits per pixel */
} bitmap;

typedef struct {
    unsigned short red;
    unsigned short green;
    unsigned short blue;
} rgbColor;

typedef struct {
    unsigned short index;
    rgbColor rgb;
} colorSpec;

typedef struct {
    long numberOfEntries;
    struct {
        long numberOfParameters;
        long parameters[];
        face builtIn;
        char faceName[]; /* optional, used if face is
negative */
        /* long alignment, if necessary */
    } faceInfo[];
} faceList;

```



```

#define RgnHandle shape

typedef struct {
    long    dummy;
} **shape;

typedef struct {
    long    dummy;
} **transform;

#define PenState style

typedef struct {
    long    dummy;
} **style;

#define GrafPort grafPort

typedef struct {
    long    dummy;
} **grafPort;

```

## **Routines**

### **Errors and Warnings**

```

error Error(shape *guiltyParty);
warning Warning(shape *guiltyParty);
void IgnoreWarning(warning warningNo);
void ErrorProc(void (*userFunction)());
void MyErrorProc(shape guiltyParty, error errNo);
void WarningProc(shape (*userFunction)());
warning MyWarningProc(shape bad, warning, shape default);

```

### **Initialization**

```

void InitSkia(void);
void ExitSkia(void);

```

### **Transforms**

```

transform NewTransform(fixed [3][3], shape clip);
void DisposeTransform(transform );
void DisposeTransformAt(transform *);
transform CopyToTransform(transform destination, transform source);
void SetTransform(shape, transform);
transform CurTransform(shape);
void SetDefaultTransform(transform);
transform CurDefaultTransform(void);
void SetTransformClip(shape orTransform, shape clip);
shape CurClip(shape orTransform, shape *clip);
void SetMatrix(shape orTransform, fixed[3][3]);

```

```
void CurMatrix(shape orTransform, fixed[3][3]);
```

## Operations on Shapes and Transforms

```
void Offset(shape orTransform, fixed h, fixed v);  
void Rotate(shape orTransform, fixed degrees);  
void RotateAbout(shape orTransform, fixed degrees, fixed h, fixed v);  
void Skew(shape orTransform, fixed xSkew, fixed ySkew);  
void SkewAbout(shape orTransform, fixed xSkew, fixed ySkew, fixed h,  
    fixed v);  
void Scale(shape orTransform, fixed hScale, fixed vScale);  
void ScaleAbout(shape orTransform, fixed hScale, fixed vScale, fixed h,  
    fixed v);  
void Perspective(shape orTransform, fract xPerspective, fract  
    yPerspective);  
void Vanish(shape orTransform, fixed horizon, fixed azimuth);  
void Box(shape, rectangle *);
```

## Ports and the Transform's Port List

```
long NewPort(fixed [3][3], shape clip, colorTable);  
void DisposePort(long order);  
long CurPort(long order, fixed [3][3], shape *clip, colorTable *,  
    long *device);  
void UpdatePort(long order, fixed [3][3], shape clip, colorTable);  
void SetTransformPort(transform, long portOrder);  
void SetTransformPorts(transform, long numberOfPorts, long portList[]);  
long CurTransformPorts(transform, long portList[]);
```

## Shapes

```
shape New(shapeType);  
shape New2(shapeType, fixed, fixed);  
shape New4(shapeType, fixed, fixed, fixed, fixed);  
shape NewMany(shapeType, long count, ...);  
void Dispose(shape);  
void DisposeAt(shape *);  
void Set2(shape, fixed, fixed);  
void Set4(shape, fixed, fixed, fixed, fixed);  
void SetShape(shape destination, shape source);  
void CopyTo(shape destination, shape source);  
void CopyDeepTo(shape destination, shape source);  
long Size(shape);  
long SizeCache(shape);  
fixed Time(shape);  
boolean Validate(shape);  
boolean Equal(shape, shape);  
fixed Length(shape);  
float FloatLength(shape);  
fixed Area(shape);  
float FloatArea(shape);  
long GlobalArea(long portOrder, shape);  
void Bounds(shape, rectangle *);
```

```

void LocalBounds(shape, rectangle *);
boolean GlobalBounds(shape, long portOrder, rectangle *);
void Draw(shape);

```

## Operations on Shapes

```

void SetType(shape, shapeType);
shapeType CurType(shape);
void SetAttributes(shape, shapeAttributes);
shapeAttributes CurAttributes(shape);
void SetGeometry(shape, shapeGeometry);
shapeGeometry CurGeometry(shape);
void SetUser(shape, char *, long length);
long CurUser(shape, char *);
void Reset(shape orTransformStyle);
void Changed(shape);
void Simplify(shape);
void Primitive(shape);
extern void Cache(shape);
void DisposeCache(shape);

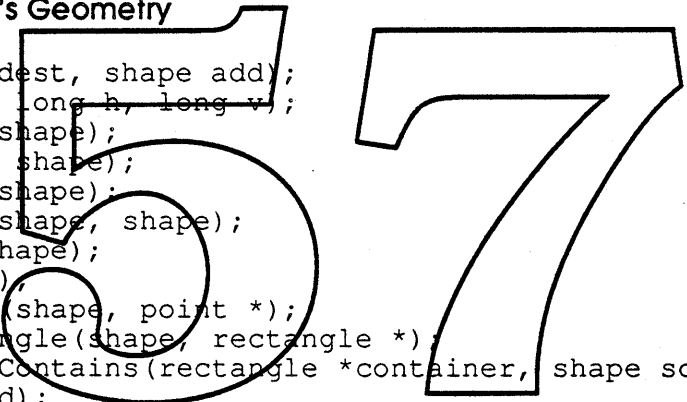
```

## Operations on Shape's Geometry

```

void AddTo(shape dest, shape add);
void Inset(shape, long h, long v);
void Sect(shape, shape);
void Union(shape, shape);
void Diff(shape, shape);
void ReverseDiff(shape, shape);
void Xor(shape, shape);
void Invert(shape);
boolean SectPoint(shape, point *);
boolean SectRectangle(shape, rectangle *);
boolean RectangleContains(rectangle *container, shape source);
shape RgnSave(void);

```



## Styles

```

style NewStyle(colorTable);
void DisposeStyle(style);
void DisposeStyleAt(style *);
style CopyToStyle(style destination, style source);
void SetStyle(shape, style);
style CurStyle(shape);
void SetDefaultStyle(shape, style);
style CurDefaultStyle(shapeType);

```

## Style Operations

```

short Show(shape orStyle);
short Hide(shape orStyle);
void SetPen(shape orStyle, fixed diameter);

```

```

fixed CurPen(shape orStyle);
void SetStyleAttributes(shape orStyle, styleAttributes);
styleAttributes CurStyleAttributes(shape orStyle);
void SetCurveError(shape orStyle, fixed);
fixed CurCurveError(shape orStyle);
void SetMode(shape orStyle, mode, char *);
mode CurMode(shape, mode *, char *);
void SetDither(shape orStyle, fixed ditherLevel);
fixed CurDither(shape orStyle);
void SetPattern(shape orStyle, shape pattern, point *grid);
shape CurPattern(shape orStyle, shape *pattern, point *grid);
void SetStartCap(shape orStyle, shape cap);
shape CurStartCap(shape orStyle, shape *cap);
void SetEndCap(shape orStyle, shape cap);
shape CurEndCap(shape orStyle, shape *cap);
void SetJoin(shape orStyle, jointType, shape join);
shape CurJoin(shape orStyle, jointType *, shape *join);
void SetDash(shape orStyle, shape dash, fixed advance);
shape CurDash(shape orStyle, shape *dash, fixed *advance);
void SetFollow(shape orStyle, shape followPath, fixed advance,
    jointType);
shape CurFollow(shape orStyle, shape *followPath, fixed *advance,
    jointType *);
void FixMoveTo(fixed, fixed);
void FixMove(fixed, fixed);

```

## Color and Color Tables

```

colorTable NewColorTable(colorSpec [], long length);
void DisposeColorTable(colorTable);
void DisposeColorTableAt(colorTable *);
void SetColorTable(colorTable, colorSpec [], long length);
long CurColorTable(colorTable, colorSpec []);
colorTable LoadColorTable(short ID, char *name);
short SaveColorTable(colorTable, short ID, char *name);
void SetColor(shape, colors, colorSpec *);
void SetColorIndex(shape, long index, colors, colorSpec *);
colors CurColor(shape, colorSpec *);
colors CurColorIndex(shape, long index, colorSpec *);
void SetColorAttributes(colorTable, long index, colorAttributes);
colorAttributes CurColorAttributes(colorTable, long index);
colorTable GlobalColors(shape, long deviceOrder, colorTable *);

```

## Lines

```

shape NewLine(line *linePts);
void SetLine(shape, line *);
void DrawLine(line *);
void FixLineTo(fixed, fixed);
void FixLine(fixed, fixed);

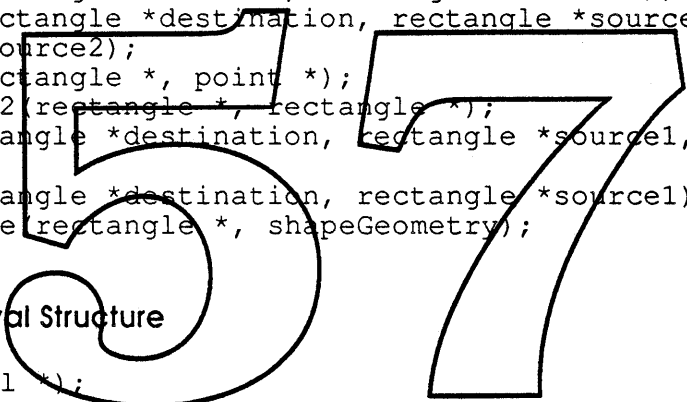
```

## Text and Text Styles

```
void SetFace(shape orStyle, face, char *faceName);
void SetFaces(shape orStyle, faceList *faces);
short CurFaces(shape orStyle, faceList *);
fixed FixCharWidth(char ch);
fixed FixStringWidth(char *ch);
fixed FixTextWidth(char *, short length);
void CurFontInfo(shape orStyle, fontInfo *);
shape NewText(char *text, short length);
void SetText(shape, char *text);
short CurText(shape, char *text);
shape NewLayout(layout *text);
void SetLayout(shape, layout *);
long CurLayout(layout *text);
void DrawLayout(layout *text);
```

## Operations on the Rectangle Structure

```
shape NewRectangle(rectangle *);
void SetRectangle(shape, rectangle *);
boolean SectR2(rectangle *source1, rectangle *source2);
boolean SectR3(rectangle *destination, rectangle *source1,
               rectangle *source2);
boolean SectRP(rectangle *, point *);
boolean ContainsR2(rectangle *, rectangle *);
void UnionR3(rectangle *destination, rectangle *source1, rectangle
             *source2);
void UnionR2(rectangle *destination, rectangle *source1);
void DrawRectangle(rectangle *, shapeGeometry);
```



## Operations on the Oval Structure

```
shape NewOval(oval *);
void SetOval(shape, oval *);
void DrawOval(oval *, shapeGeometry);
```

## Operations on Rounded-Corner Rectangles

```
shape NewRoundRect(rectangle *, point *ovalSize);
void SetRoundRect(shape, rectangle *, point *ovalSize);
void DrawRoundRect(rectangle *, point *ovalSize, shapeGeometry);
```

## Operations on the Curve Structure

```
shape NewArc(curve *);
void SetArc(shape, curve *);
void DrawArc(curve *);
shape NewWedge(curve *);
void SetWedge(shape, curve *);
void DrawWedge(curve *, shapeGeometry);
shape NewCurve(curve *);
```

```
void SetCurve(shape, curve *);
void DrawCurve(curve *);
```

### Operations on the Bitmap Structure

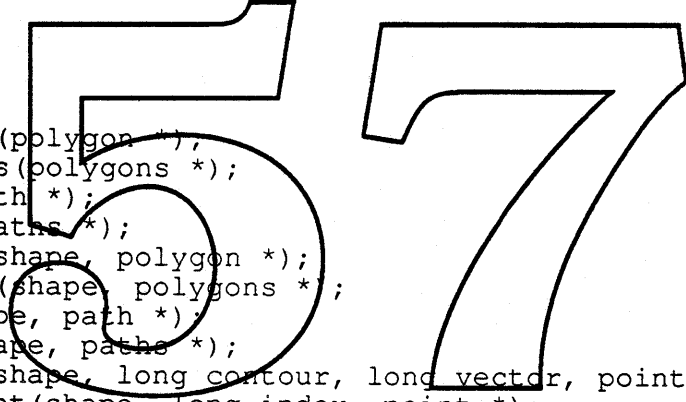
```
shape NewBitmap(bitmap *);
void SetBitmap(shape, bitmap *);
void CurBitmap(shape, bitmap *);
void DrawBitmap(bitmap *);
```

### Pictures

```
warning Load(shape, short ID, char *name);
warning Save(shape, short ID, char *name);
long AddToPicture(shape picture, shape toAdd, transform, style);
void InsertPicture(shape picture, long index, shape toAdd, transform,
style);
void SetPicture(shape picture, long index, shape *, transform *,
style *);
shape CurPicture(shape picture, long index, shape *, transform *,
style *);
```

### Polygons and Paths

```
shape NewPolygon(polygon *);
shape NewPolygons(polygons *);
shape NewPath(path *);
shape NewPaths(paths *);
void SetPolygon(shape, polygon *);
void SetPolygons(shape, polygons *);
void SetPath(shape, path *);
void SetPaths(shape, paths *);
void SetCVPoint(shape, long contour, long vector, point *);
void SetIndexPoint(shape, long index, point *);
void Insert(shape, long index, shape toAdd);
shape Extract(shape source, long firstPoint, long numPoints,
shape *destination);
long Contours(shape);
long Points(shape, long contour);
long Index(shape, long contour, long vector);
boolean CurControl(shape, long index);
void CurPoint(shape, long index, point *);
void CurLine(shape, long index, line *);
void CurCurve(shape, long index, curve *);
void CurRectangle(shape, long startIndex, long numberOfPoints,
rectangle *);
void CurPolygon(shape, long startIndex, long numberOfPoints, polygon *);
void CurPolygons(shape, long startIndex, long numberOfPoints,
polygons *);
void CurPath(shape, long startIndex, long numberOfPoints, path *);
void CurPaths(shape, long startIndex, long numberOfPoints, paths *);
long MatchPoint(shape, point *, long foundIndices[]);
long MatchRectangle(shape, rectangle *, long foundIndices[]);
```



```
shape PolySave(void);
```

## Drawing Polygons and Paths

```
void DrawPolygon(polygon *);  
void DrawPolygons(polygons *);  
void DrawPath(path *);  
void DrawPaths(paths *);
```

## Converting Between Local and Global Coordinates

```
void GlobalPoint(point *, long portOrder);  
void LocalPoint(point *, long portOrder);  
void GlobalShape(long portOrder, shape);  
void LocalShape(shape, long portOrder);
```

## Devices

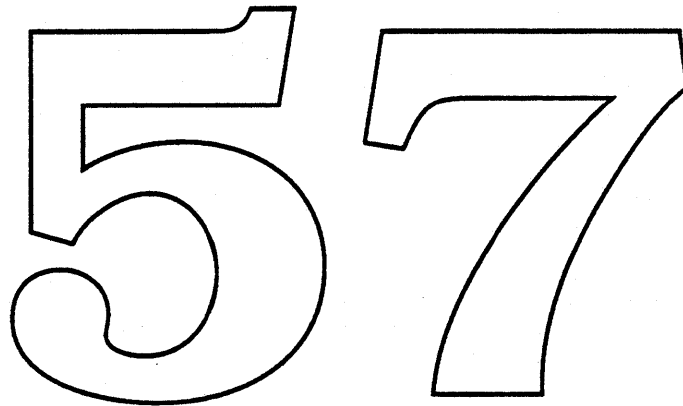
```
long NewDevice(long associate, fixed [3][3], shape clip, colorTable,  
  bitmap *);  
void DisposeDevice(long device);  
void SetDevice(long device, long associate, fixed [3][3], shape clip,  
  colorTable, bitmap *);  
long CurDevice(long device, long *associate, fixed [3][3], shape *clip,  
  colorTable *, bitmap *);  
long CurDevices(shape, long deviceList[]);  
void SetDeviceAttributes(long deviceOrder, deviceAttributes);  
deviceAttributes CurDeviceAttributes(long deviceOrder);  
void SetDeviceSpot(long device, fixed spotSize);  
fixed CurDeviceSpot(long device);  
GDHandle CurGDevice(long device);
```

## Custom Skia Operations

```
void UserClip(shape orTransform, warning (*userFunction)(),  
  long length);  
warning MyUserClip(shape *transformClip, shape *portClip,  
  shape *deviceClip);  
void UserMatrix(shape orTransform, warning (*userFunction)(),  
  long length);  
warning MyUserMatrix(long port, fixed [3][3]local, fixed [3][3]port,  
  fixed [3][3]device, fixed [3][3]output);  
void UserPrimitive(shape, warning (*userFunction)(), long length);  
warning MyUserPrimitive(shape);  
void UserSect(shape, warning (*userFunction)(), long length);  
warning MyUserSect(long portOrder, shape, boolean *intersects);  
void UserTransform(shape, warning (*userFunction)(), long length);  
warning MyUserTransform(long portOrder, shape, shape *localClip,  
  fixed [3][3], shape *globalClip);  
void UserColors(shape, warning (*userFunction)(), long length);  
warning MyUserColors(long portOrder, shape, long numberColors,  
  long *colorIndices);
```



```
void UserSetup(shape, warning (*userFunction)(), long length);
warning MyUserSetup(long device, shape);
void UserMode(long device, char *modeName, warning (*function)(),
    long length, boolean longs);
warning MyUserMode(long *dest, long *color, long count, long edge1,
    long edge2);
warning MyUserMode(long *dest, long *color, long bitOffset,
    long pixelCount);
void UserWidths(shape orStyle, warning (*function)(), long length);
warning MyUserWidths(char *string, short length, layout *);
void UserFace(char *faceName, warning (*function)(), long length);
warning MyUserFace(char *string, short length, layout *, shape *,
    short numberOfParameters, ...);
```



57

**There is currently no ERS for the new  
Monitors CDev.**

57

57

**The Video Architecture Extensions documents will be distributed separately.**

57

57

# Video Configuration ROM Software Specification - Delta Guide

David Fung, FUNG1  
Product Development, Graphics Software Group  
24 March 1989

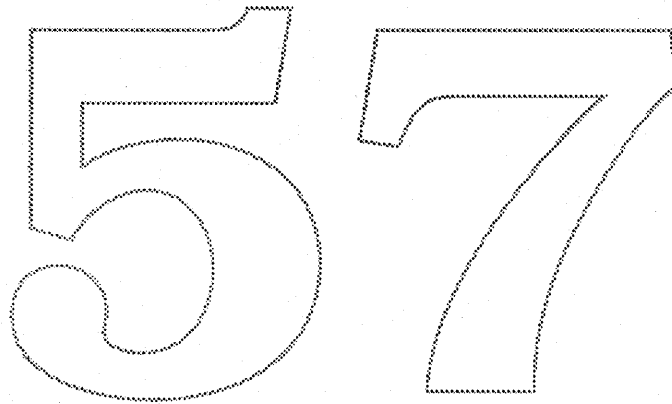
Of course, we couldn't leave well enough alone. This document is a quick delta guide to the changes between the Video ROM spec versions 3.0 and 3.1. The changes are fairly minor, and by and large should just be a matter of cut and paste implementation. Besides, the original document was way too serious. I mean, hey, if you can't tell Slot Manager jokes, what good is it being in the config ROM business anyway (did you hear the one that ends "Video sRsrc list? That was my Board ID!") (we've probably been working too hard).

You'll get the updated 3.1 version immediately after this note. Anyway, here's the half page summary of the changes:

- 1) Gamma Directories are in the video sRsrc list rather than the board sRsrc list. This way you don't see the gamma tables for other monitors.
- 2) Numbering in the gamma directories has changed: Rather than random, unordered gamma directories, the table entries are in monotonic ascending order starting at 128. This greatly simplifies searching. The table identified by spID=128 is the default gamma table.
- 3) All cString names should be end-padded to word alignment: Just a simple ALIGN 2 after names.
- 4) The names Image16 & Image32 are now obsolete: I liked 'em but they were crushed by bureaucracy.
- 5) Watch that GetGamma call!: Since GetGamma returns the driver's gamma correction table ptr, it's important that this be in a public form. The sample driver source that DTS distributes (and our current TFB driver) don't do this right.

You've probably noticed by now that the latest release of Monitors doesn't see a lot of this new stuff yet. The next one will, and I will probably link it out if it isn't in the mail before the middle of next week. Also, if you have gotten all the Primary and SecondaryInIt stuff working, you probably have noticed that Macsbug 6.0 is not your good buddy anymore. The next JP seeding should include a new, extended frame buffer-friendly MacsBug which will be released generally shortly thereafter.

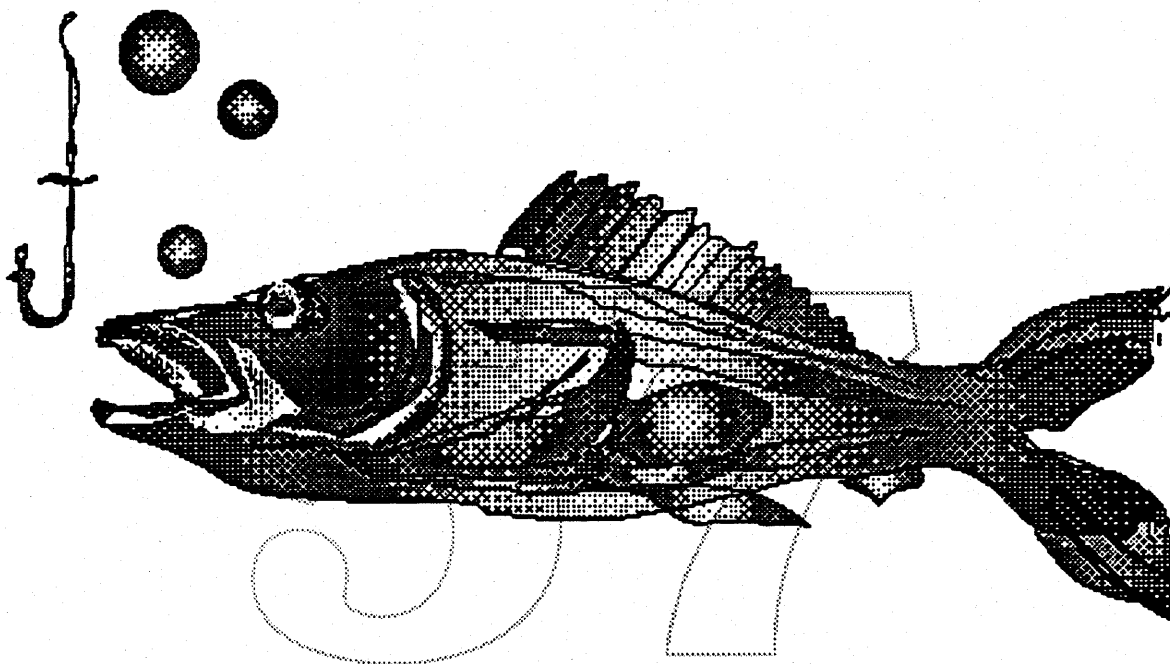
Good luck!

A large, stylized number '57' rendered in a dotted outline format, centered on the page. The '5' has a thick, rounded base and a curved top, while the '7' is a simple, bold, sans-serif style.



Bass 1.0  
(Intelligent Font Scaling Technology for Macintosh)  
External Reference Specification

Version 1.2



The Bass Team<sup>1</sup>  
M/S 27AJ

February 20, 1988

Apple Computer Confidential

---

1. Please contact the specific team member directly whenever possible. Footnotes throughout the document will reference appropriate contacts. For general technical comments and concerns, please contact K. Weisberg, x2763.

57

## Table of Contents

<u>Section/Topic</u>	<u>Page No.</u>
1.0 About This Document	5
2.0 Why Bass?	5
2.1 Non Competitive Aspects	5
2.2 Competitive Aspects	6
3.0 Bass Design Objectives	6
4.0 Bass Design Principles	7
5.0 Major Milestones	8
6.0 Bass Feature Set	10
6.1 Fonts	10
6.2 Individual Character Caching	11
6.3 Proportional Scaling	12
6.4 Banding	12
6.5 Kerning	13
6.6 Engine Characteristic Compensation	13
6.7 Expandable Data Structures	13
6.8 Support of International	14
7.0 Functional Aspects	16
7.1 Fonts	16
7.1.1 Quadratic B-Splines	16
7.1.2 Outline Font Data Protection	16
7.2 Spline Font Package	17
7.2.1 Instructions	17
7.2.2 Interpreter	18
7.2.3 Scan Conversion	18
7.3 QuickDraw	19
7.3.1 QuickDraw Routines	19
7.3.2 Font Manager Routines	19
7.3.3 Glue	20
7.4 Printing	20
7.5 Font Tools	20
7.6 Documentation	21
7.6.1 For Font Manufacturers	21
7.6.2 The Font Tutor	21
7.7 User Interface	21
7.7.1 Font/DA Mover	21
7.8 Hardware Requirements	22
8.0 Implementation	23
8.1 Fonts	23
8.1.1 Sources	23
8.1.2 Times, Helvetica, Courier and Symbol	23
8.1.3 Lucida	23

# Apple Computer

## Graphics & Animation Software

## Font Development

8.1.4	Apple Exclusive Fonts	23
8.1.5	Screen Fonts	24
8.2	Spline Font Package	24
8.2.1	Instruction Application	24
8.2.2	Back-up VS Translator	24
8.3	QuickDraw	24
8.3.1	DrawText.a	24
8.3.2	Partial Resources	25
8.3.3	Glue	25
8.4	Printing	30
8.5	Font Tools	30
8.5.1	Font Editor	30
9.0	Future Enhancements	31
10.0	Documentation	32
10.1	Inside Macintosh	32
11.0	Testing	32
12.0	Developer Issues	32
12.1	Developer Survey	32
12.2	Menus	32
12.3	Font Manufacturers	32
12.4	Font Editors	33
12.5	Applications	33
13.0	Schedule	34
14.0	Risks and Open Issues	34
14.1	Risks	34
14.2	Open Issues	34
15.0	Internal Dependencies	34
15.1	Finder and Applications	34
15.2	Printing	34
15.3	Resource Manager	34
16.0	The Team	35
17.0	Appendix	36
17.1	Glossary	36
17.2	Sub ERS(s)	36
17.2.1	System Interface	
17.2.2	Font Instruction Set and Interpreter	
17.2.3	Scan Converter	
17.2.4	Bass Instruction Editor	
17.2.5	Outline Font Data	
17.2.6	Outline Font Data Protection	
17.2.7	VS Translator	

## 1.0 About This Document

This document has been distributed for the purpose of presenting the implementation details and functional aspects of the Bass Project. Please feel free to comment and return your feedback to the team.

Bass is a Software Engineering project to provide outline font technology in Macintosh and to eliminate the need for large, memory intensive bitmaps for each imaging device. Version 1.0 of this project is scheduled for completion in the Summer of '89 with functionality limited to the 2MB Macintosh environment. Nothing specifically prohibits functionality in a 1MB environment, however, performance may be an issue. The strategy includes the use of four new pieces of code (to be written in-house) and rework of some existing QuickDraw and Font Manager routines. Companion software dependencies include: revision of the Font D/A Mover to move Bass resources (Sec. 7.7.1), and a Postscript compatible interpreter/scan conversion solution (Sec. 15.2).

This project requires a concentrated Evangelism effort to ensure compatibility with major applications where Font Manager crimes are being committed, and to ensure a source of commercial fonts at introduction.

Version 1.0 of this project will include a set of Core fonts, a Font Instruction Set, Interpreter, Scan Converter, system interface (glue code) and an additional font resource designed to supply the actual spline data and font metrics. This resource will be designed to include extensible font data structures to support International. The Bass 1.0 feature set includes: individual character caching, proportional scaling, banding, engine spot-size compensation and extensible data structures for the support of International.

## 2.0 Why Bass?

### 2.1 Non-Competitive Aspects

There are three essential reasons why outline fonts in Macintosh are required

#### 1. *Huge Bitmap Fonts*

Each imaging and/or display device currently requires its own set of bitmaps targeted to the resolution of that device. For example; the LaserWriter II SC with an effective resolution of 288 dpi requires a 96 point bitmap font (4x of its 72 dpi equivalent) for printing at 24 point. This single bitmap occupies in excess of 156K of available RAM. This huge bitmap also resides in the System Heap in the form of a strike comprised of all the characters for that font. Four fonts for this device (Times, Helvetica, Symbol and Courier in plain, bold and italic) occupy 3.6MB when installed. Users are becoming increasingly confused by the specialized nature and use of customized bitmaps which are shipped with every output device and discouraged by the enormous amount of storage required to house and use these fonts.

#### 2. *Inadequate Scaling of Intermediate Sizes*

Current scaling algorithms perform a limited set of routines which find the closest size to the size requested and scale that size up or down. This gives different results depending on the number and sizes of bitmaps available. Neither the device driver nor

QD can cope well with the lack of intermediate sizes of bitmaps which simply do not exist.

### 3. *Inadequate Data Structures*

Existing data structures do not provide adequate fields or tables for the handling of non-Roman based character sets. Extensive reordering and displacement data structures for display of characters in alternative textual representations such as ligatures and writing from right to left or top to bottom are required. Actual features will be implemented in the Bass 2.0 timeframe through the new Layout Manager.

## 2.2 Competitive Aspects

Cricket Graph, Compugraphic, Bitstream, URW and Folio are all competing for control of an outline font format in Macintosh. If Apple does not take control of this environment, users may suffer the loss of cross functionality and ease-of-use. None of their approaches are fully open in format and some are tied to post-processors or buried in printer ROM and have sub-standard user interfaces and/or marginal performance. Furthermore, none offer a full range of instruction based routines and none successfully handle diagonal strokes. It will be important to evangelize Bass to these third party font manufacturers and to printer manufacturers who are contemplating the use of one or more of the formats mentioned above. <sup>2</sup>

## 3.0 Bass Design Objectives

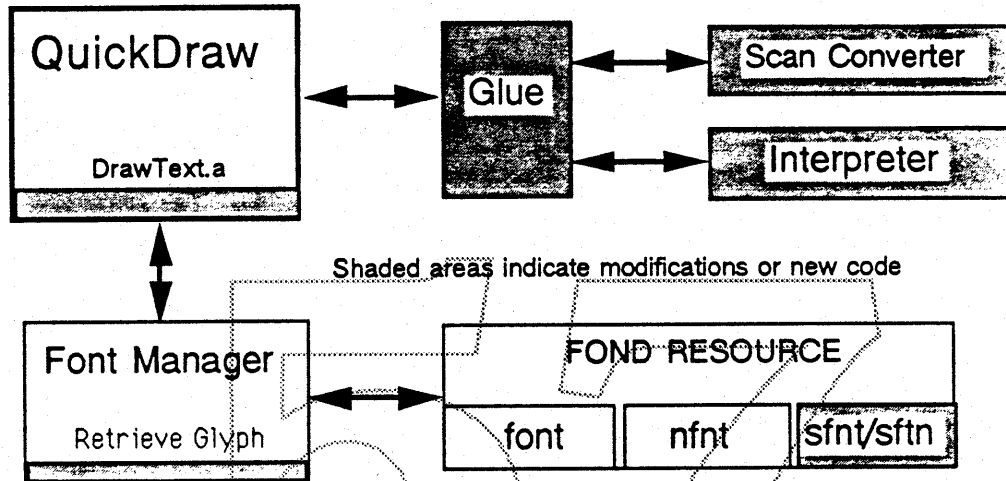
1. Create an environment where bitmap and outline font descriptions can co-exist with the ultimate goal to eliminate the need for bitmap fonts.
2. Define and develop an individual character caching scheme and hashing mechanism which allows large characters and fonts with large character sets to be drawn to the screen without the need for a strike format.
3. Provide a data structure which is expandable for future architectures and support of International.
4. Provide a set of fonts which is universal to all devices.
5. Provide a spline font description (Quadratic) which is compatible with future Apple graphic architectures and provides a minimum of first degree continuity.
6. Develop an advanced and robust instruction set which allows for greater flexibility in outline font scaling and graphical transformations.
7. Adapt current QuickDraw and Font Manager routines with separate code modules for the Spline Font Package and system interface "glue code".

The objective of the Bass Project is to provide an intelligent scaling routine in Macintosh. The format for this technology will be published to allow third parties to provide outline fonts for the Macintosh on a commercial scale. The Spline Font Package containing the Interpreter and Scan Conversion Routines will be a separate code segment accessed by the glue code.

- 
2. Extreme caution should be exercised in revealing the existence or scope of this project to Adobe without a very well thought-out strategy. Product Management and Evangelism are working on announcement strategies and timing.

### 4.0 Bass Design Principles

Bass 1.0 will be designed assuming a minimum operating environment of 2MB.<sup>3</sup> For backward compatibility and speed of implementation, all modifications will be made to current QD and Font Manager routines. The current FOND resource will be expanded to include a new resource "sfnt" which incorporates the necessary spline font metrics and the splines themselves. This expanded data structure is designed to support the needs of International.



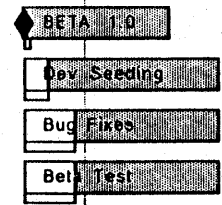
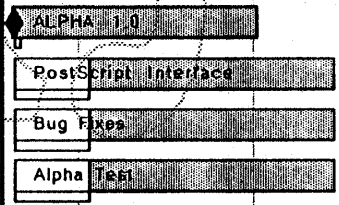
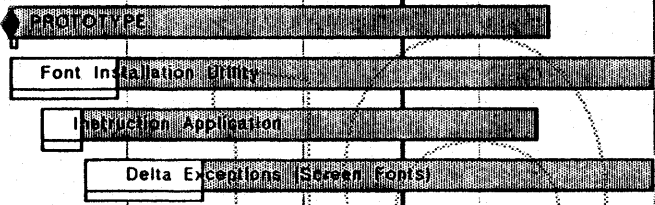
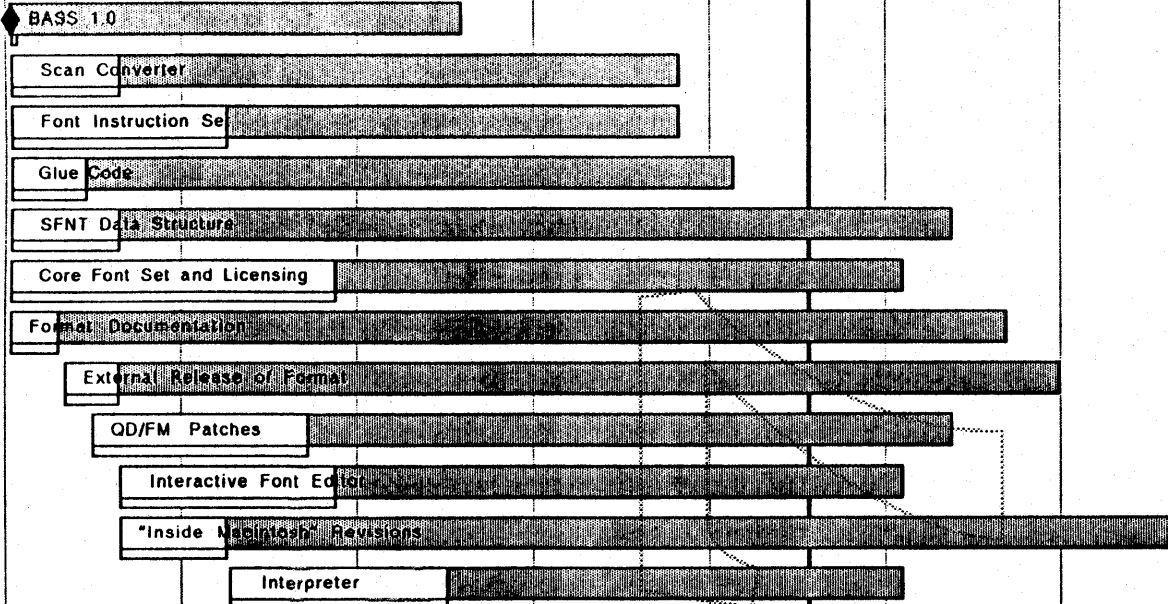
The system will continue to recognize bitmaps as well as the new spline resources to avoid reformatting of existing documents, however "instructions editing" will assume linear scaling to 90° at 72 dpi. This means that should the user wish to obtain WYSIWYG and is not concerned with reformatting, he/she may do so by simply removing the bitmap fonts from his/her system.

The user interface is intended to be transparent to applications and users, however, some indicator will be required to differentiate splines from bitmaps at installation time.

### 5.0 Major Milestones - 1988-89 timeframe

(Next Insert - Fold out)

3. It is not true that this technology necessarily eliminates the 1MB environment. Bass 1.0 will install and function in a 1MB environment. However, performance under low memory runtime conditions may prove undesirable without segment swapping.



**FINAL**



## 6.0 Bass Feature Set

### 6.1 Fonts

There will be 25 fonts under development in the Bass 1.0 timeframe. All 25 fonts will not be complete at the time of release. The Apple QuickDraw Character Set will include the Adobe Extension for backward and forward compatibility. Bass will include a core set of outlines specifically tuned to take advantage of Apple's Instruction Set (grid-fitting routines).

#### Bass 1.0 Outline Font Core Set

Times New Roman (Roman, Bold, Italic and Bold Italic)

Helvetica (Roman, Bold, Italic and Bold Italic)

Courier (Roman, Bold, Italic and Bold Italic)

Symbol (Roman only)

Lucida Bright (Roman, Bold, Italic and Bold Italic)

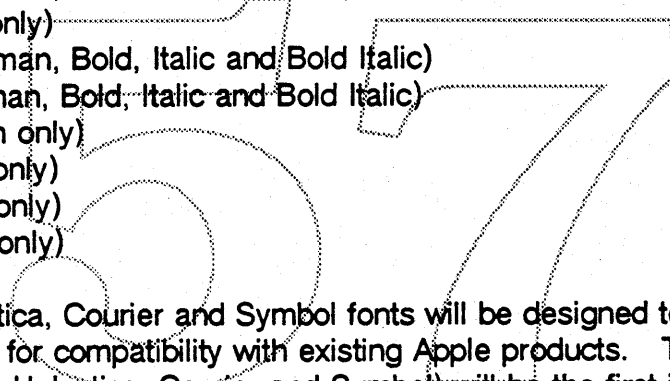
Lucida Sans (Roman, Bold, Italic and Bold Italic)

New York (Roman only)

Geneva (Roman only)

Monaco (Roman only)

Chicago (Roman only)



The Times, Helvetica, Courier and Symbol fonts will be designed to match the Postscript metrics for compatibility with existing Apple products. The four standard core fonts (Times, Helvetica, Courier and Symbol) will be the first implemented with the bold and italic versions to follow. A set of Apple exclusive outlines are in the design phase for New York, Geneva, Monaco and Chicago. These fonts are being designed by the studio of Bigelow & Holmes who will also provide Lucida Bright and Lucida Sans. It is unlikely that all these typefaces will be complete in the Bass 1.0 timeframe. They will be included in the core set as completed.

## Apple QuickDraw Character Set - Extended

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0				0	@	P	`	p	Ä	ê	†	∞	¿	-	‡	🍏
1			!	1	A	Q	a	q	Å	ë	°	±	ı	—	.	Ò
2			"	2	B	R	b	r	Ç	í	¢	≤	¬	“	,	Ú
3			#	3	C	S	c	s	É	ì	£	≥	√	”	„	Û
4			\$	4	D	T	d	t	Ñ	î	§	¥	f	‘	%	Ü
5			%	5	E	U	e	u	Ö	ï	•	µ	≈	’	Â	ı
6			&	6	F	V	f	v	Ü	ñ	¶	∂	Δ	+	Ê	^
7			’	7	G	W	g	w	á	ó	ß	§	«	◊	À	~
8			(	8	H	X	h	x	à	ò	®	Π	»	ÿ	Ë	-
9			)	9	I	Y	i	y	â	ô	©	π	...	ÿ	È	˘
10			*	:	J	Z	j	z	ä	ö	™	∫		/	í	·
11			+	;	K	[	k	{	ã	õ	’	ª	Á	◻	Î	°
12			,	<	L	\	l		å	ú	”	º	Ã	<	ï	
13			-	=	M	]	m	}	ç	ù	≠	Ω	Ö	>	ì	
14			.	>	N	^	n	˘	é	û	Æ	æ	Œ	fi	Ó	
15			/	?	O	_	o		è	ü	Ø	ø	œ	fl	Ô	

### 6.2 Individual Character Caching

Characters will be cached on a per glyph basis in a bitmap hash buffer rather than stored on disk in the familiar strike format.<sup>4</sup>

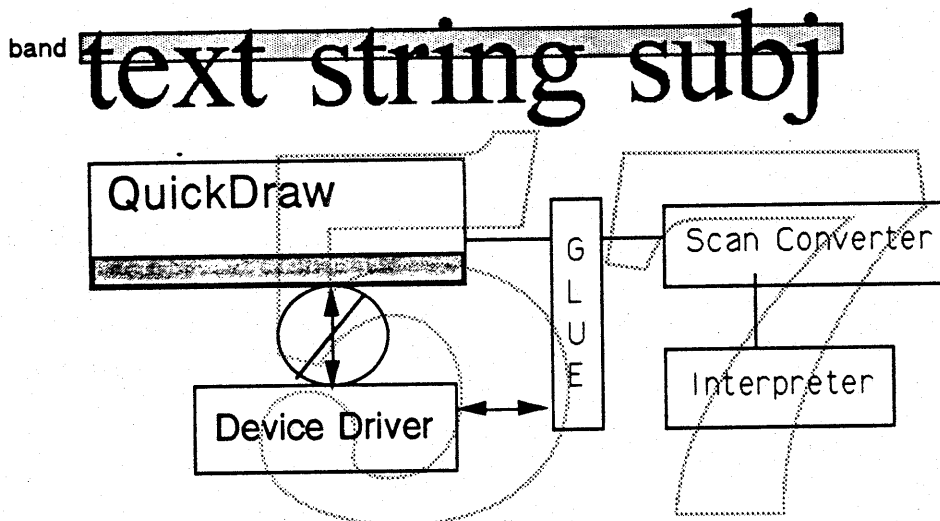
### 6.3 Proportional Scaling

Fonts which replace existing bitmap versions (Times, Helvetica, etc.) will be provided in Adobe character widths and match the hypothetical 1 pt. font description. This description will be scaled by a factor of the point size requested.

If 1 pt. = .0100 then, 12 pt. = .1200, and 36 pt. = .3600

### 6.4 Banding

Bass will support banding for printing of very large characters through QuickDraw, new routines will need to be generated which break up a string of characters into bands.



### 6.5 Kerning

Minimum kerning routines and mapping tables of kerned pair values will NOT be provided in Bass 1.0. International will develop the tables and routines for this function. This table contains information on kerning n-tuples (generally pairs, although triplets and even larger forms are allowed). For each n-tuple, this table identifies which character codes participate in the kerning, as well as minimum, optimal and maximum permitted kerning amounts. <sup>4</sup> Availability for kerning will be dependent on the Layout Manager.

To Kern or not To Kern

### 6.6 Engine Characteristic Compensation

Instructions are implemented to allow the interpreter to manipulate stem weights when scaling to compensate for variable spot sizes in printing engines. The instructions allow distances to be broken into 4 categories. Distances in Category 0 (gray) will always be rounded in the normal way, but for the other categories, a

4. For more information, reference the Boffin ERS.

compensation term will be added before rounding. For example, assume that the engine makes black distances 0.8 pixels too heavy and white distances 0.8 pixels too narrow. In this case the black compensation would be -0.8 pixels and the white compensation would be +0.8 pixels.

### 6.7 Expandable Data Structures

A new font resource has been developed for Bass. This new resource (sfnt) is called by the Font Manager and provide offset pointers to many new tables. <sup>5</sup> These resources do not replace any of the existing font resources (font or nfont) and are access through the FOND. The spline font data structure has specific fields reserved for expansion in the areas of International positioning, vertical metrics and non-proportional (optical) scaling.

### 6.8 Support of International

Routines: IsChar, GetChar, GetMetrics, AddChar, DeleteChar, SetChar

The new data structure has been designed to support the requirements of International character sets. International requires four tables (Re-ordering, Forms, Ligatures and Positioning) and each are included in the sfnt resource description. In non-Roman character sets there are a large amount of glyphs which will usually not be constructed from the whole character set but only a percentage. There will be a sparse and non-sparse table look-up provided for both cases. The speed of the non-sparse array will still exist while the sparse array will maintain the more compact storage space. <sup>6</sup>

Tables for the following planned character transformations will be included in the sfnt data structure.

Positioning Table: This table identifies floating anchor point metrics for characters, which allow the on-the-fly creation of composite (e.g. accented) characters that do not themselves appear in the font. This table also contains information on (x,y) micropositioning of characters for better visual appearance of certain characters

(such as centering the minus sign for upper-case letters).

“a” “u” “i” “o”

5. For a description of the data structures, see the attached System Interface Sub-ERS (Appendix 17.2.1) or contact Charlton Lui.

6. For further details, see the Boffin and Script Manager ERS(s).

**Ligature Table:** Many languages (including English) allow some cases of m-on-n (usually m-to-1, although more is allowed) mappings. For example, in English it is permitted to replace an occurrence of an "f" followed by an "i" with a "fi" ligature. Ligatures are very common in Arabic as well. This table identifies which characters are ligatures, and which sequence of other characters there are ligatures for. Example:

fi fl Œ Æ vs. fi fl OE AE

**Reordering Table:** In certain languages (such as Hindi), certain letters do not follow the language's normal layout order (whether left-to-right or right-to-left). This table will contain information identifying which characters (if any) participate in this

behavior, and provides information on the rules that control when and where such reorderings take place. Example:

hindi is *ordered* ihndi in Devanagari

**Forms Table:** Arabic Naskh is an example of a font that will require this table. It provides information that identifies initial, medial and final forms that appear for a given letter in different parts of a word. It also identifies so-called "special-pairs", which are letters that alter their appearance when they appear next to each other.

## 7.0 Functional Aspects

### 7.1 Fonts

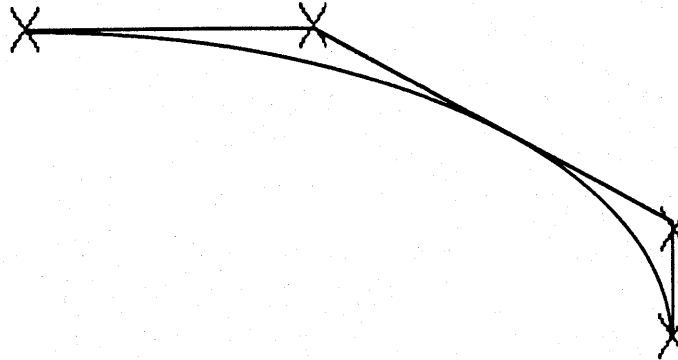
Fonts in IK format (original digitized data format) will be delivered to URW by the font manufacturer for conversion to Quadratic B-Splines prior to shipment to Apple.<sup>7</sup>

#### 7.1.1 Quadratic B-Splines

Parametric B-Splines offer the advantage of controlling the degree of continuity between adjacent curve segments. Quadratic B-Splines guarantee first degree continuity with the least possible number of control points. The characters will be described by a 2nd order B-spline format. In this format there are only two kinds of points; those on the curve and those off the curve. A straight line segment is simply described by two consecutive points on the curve. A non-straight B-spline

7. Sources and other information regarding the outlines are covered in the Implementation Section (8.0).

can be broken down to a quadratic B-Spline format where a curve segment is described by three points and new points can be created on the curve exactly in between each two consecutive points laying outside the curve.<sup>8</sup>



### 7.1.2 Outline Font Data Protection

Digitized and edited outline fonts with grid-fitting instructions can be considered source code and require some level of protection from vandalism and piracy. To avoid the administrative hassles, it may be desirable to use some sort of Public Key encipherment for decoding in Macintosh. This would apply to fonts manufactured by major commercial third party sources such as Bitstream, Compugraphics, etc. and offered for sale in a Macintosh environment.

This procedure would involve a public encryption mapping that would be used by font manufacturers to encrypt the random block encoding key that they chose for a font. The rendering mechanism would then use a public-key mapping to recover the block encoding key and use the standard block decoding to obtain the outline data. The important characteristic of a public key encryption scheme is that knowledge of the encryption mapping does not allow calculation of the decryption mapping.

Because the 64 bit block decoding key is in memory, the public key deciphering algorithm is also in memory, available to any competent hacker. Some anti-reverse engineering steps may be taken by obfuscating the code to make it more difficult to disassemble. It may also be possible to detect attempts at stepping through the code and force wrong things to happen. However, font manufacturers will have to accept that this method of encryption will not be unbreakable nor guaranteed and will not be policed in any way by Apple.

Font Manufacturers will also be warned that encryption of the spline itself will negate any use of that font in Apple's future graphics environment (SKIA, ALBERT). Graphical transformations cannot be performed on encrypted outlines.<sup>9</sup>

8. For an example and description of the parametric equation, see attached Sub-ERS for the Scan Converter (Appendix 17.2.3, last page) or contact Sampo Kaasila, x5505.

9. See the attached Sub-ERS on Outline Font Data Protection

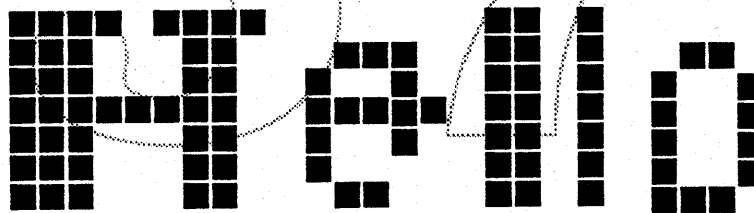
## 7.2 Spline Font Package

### 7.2.1 Instructions

A program is attached to every character. This program contains Instructions which are required to move points within an outline (character) in relation to other points within that outline. These instructions contain primitives for changing the status of control points within the character, like shifting points, interpolating points, grid-aligning points, etc. The instructions will be byte encoded and stack based. The Instruction Set also contains primitives for manipulating elements on the stack, and reading/writing X and Y coordinates. A delta exception instruction allows for the tuning of individual point sizes between 9° and 24°<sup>10</sup>

### 7.2.2 Interpreter

The interpreter is an internal module and not directly callable by an application. The interpreter's function is to eliminate various artifacts appearing on raster devices when scan converting outline based characters. These artifacts include pixel dropouts, unpleasing curves (widowed pixels/flat curves), uneven stem weights, bad weight progressions, etc. Control of stem weights and other critical character elements during rotation has been incorporated to support graphical transformations.

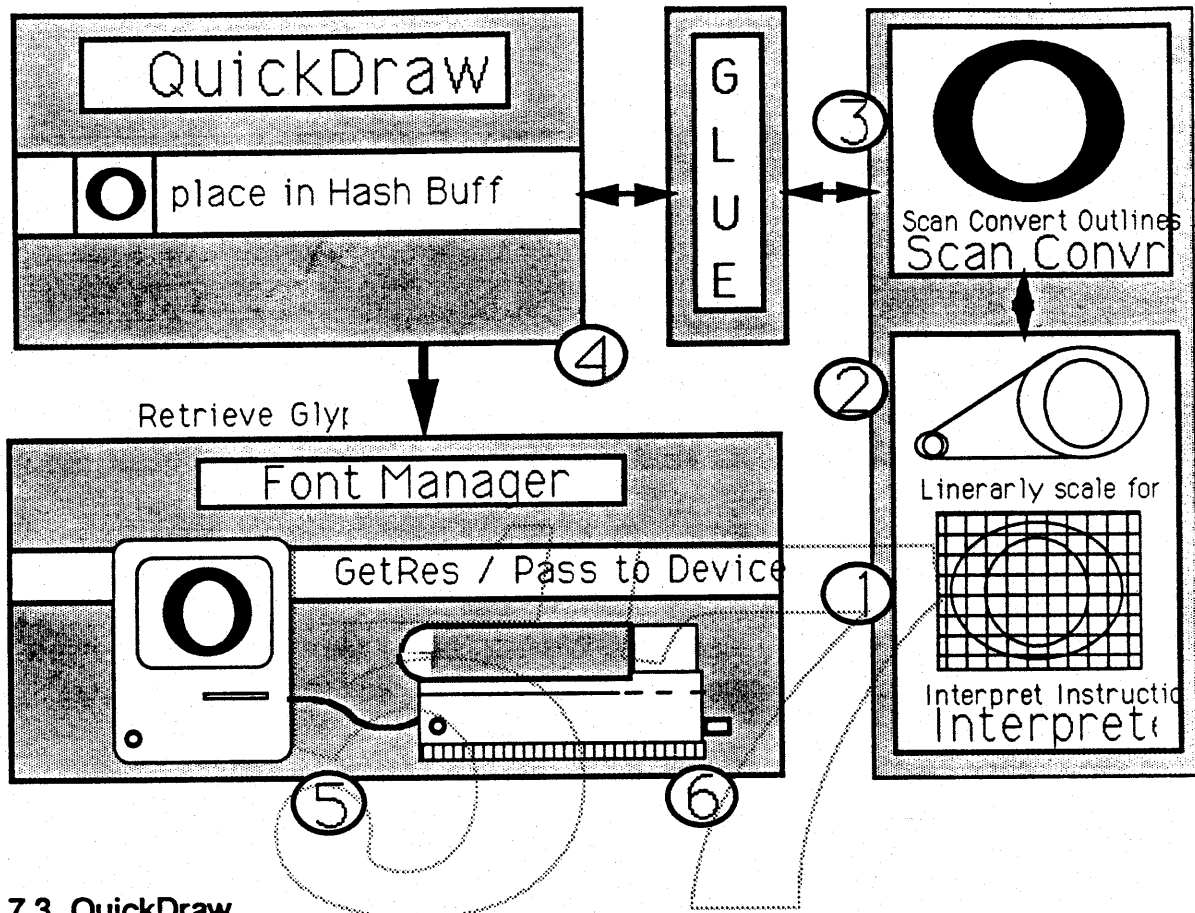


### 7.2.3 Scan Conversion

The scan conversion code is an internal module like the interpreter, and not directly callable by an application. Its function is to take an outline description of a shape and return a bitmap description. The algorithms will be designed to work optimally in terms of speed at small point sizes (i.e., 12° at 300 dpi). The scan conversion process consists of three steps; 1) Breaking up the curve into straight line vectors using forward differencing, 2) Finding the scanline intersections using a slightly modified Bresenham algorithm and , 3) Painting the bitmap using x-oring. When the bitmap is painted it will be handed to the Font Manager for caching.<sup>11</sup>

10. For an indepth look at the instruction set, see the attached SubERS (Appendix 17.2.2) or contact Sampo Kaasila.

11. For a full description of the mathematical foundation for the forward differencing, the modified Bresenham algorithm the painter and the memory usage, see attached SubERS (Appendix 17.2.3) for the Scan Converter or contact Sampo Kaasila, x5505.



### 7.3 QuickDraw

#### 7.3.1 QuickDraw Routines

Two existing modules (i.e., DrawText.a and FontMgr.a) will need modifications in the decision making and small routine feature enhancements.<sup>12</sup>

#### 7.3.2 Font Manager Routines

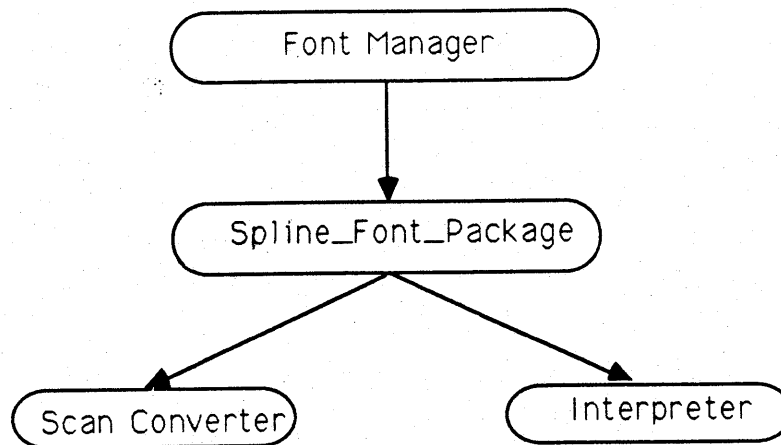
Six routines will be added or modified (GetFontName, GetFNum, RealFont, SetFontLock, FMswapFont, FontMetrics ).<sup>12</sup>

#### 7.3.3 Glue

The glue code provides a functional interface to the sfmt resource. It is used by the Scan Converter, Interpreter, and Font Manager. For the Scan Converter, the control points are unpacked along with their instructions. For the Font Manager, character requests are processed, computing the actual size needed for the output device, and then sent on to the Scan Converter.

12. See System Interface Sub-ERS (Appendix 17.2.1) for detailed description of changes or contact Charlton Lui.





#### 7.4 Printing

Device drivers will be requested to return the device resolution, engine characteristics and name of the font to QD. QD will return a font size cut-off. Point sizes below the cut-off will be supplied as bitmaps; point sizes at or above the cut-off will be supplied as splines. When requesting a bitmap character, the driver will supply QD with the font name, character, device resolution, engine characteristics and point size. When requesting a spline character, the driver will not need to supply the point size.

The above mechanism will allow device drivers to build partial fonts. The first request for a character in a non resident font will cause the driver to query QD for the cutoff point size. The existing Postscript interpreter will be augmented by the Bass Interpreter written in Postscript. Some modifications to the resulting bitmap will be required prior to rendering due to the differences between the Postscript and Bass Scan Conversion models. It is not clear that a solution will be possible in the Postscript device driver due to numerous limitations including available memory. Barring a reasonable solution, Bass outlines would be downloaded to the Postscript engine uninstru-cted (unhinted).

Ginsu drivers which call on Font Metrics to obtain widths from the Font Manager are assuming proportional scaling and will break where other forms of scaling are enabled by font manufacturers. These additional scaling methods i.e., Cubic or Quadratic, are available through an entry in the sfnt data structure. They were made available as part of the "expandable" nature of that resource for optically scaled font metrics. Font Manufacturers will be informed that these alternative forms of scaling are not currently compatible with the system or printing architectures.

#### 7.5 Font Tools

An interactive Font Editor will be designed as an internal development tool for the manipulation of outlines and application of instructions with multiple windowing for making modifications and visual comparisons between characters. The editor will display the outline with or without instructions applied, apply instructions, display the resulting bitmap, scale the font and display the font data and a text sample.<sup>13</sup>

13. See Bass Edit SubERS - Appendix 17.2.4

## 7.6 Documentation

### 7.6.1 For Font Manufacturers

Nearly every major manufacturer of outline fonts in the world uses a system based on the principles of IKARUS (IK). However, there are as many methods of grid-fitting as there are font manufacturers. To develop an open format (one that can be used by all font manufacturers), it will be necessary to base the origin on IK and then incorporate instructions which cover as many of the functional aspects of these different grid-fitting routines as is practical within the constraints of data storage, speed of rendering and quality of scaling. Each Font Manufacturer will be at liberty to employ any or all of the instruction set supplied. Further compatibility with the Apple format will require a translator of varying complexity depending on the deviation from standard grid-fitting techniques and conversion of their spline definition to quadratic B-Splines.<sup>14</sup>

### 7.6.2 The Digital Font Tutor

A special Apple publication will be developed in conjunction with the Apple Exclusive fonts<sup>15</sup> in cooperation with Bigelow & Holmes for the purpose of describing the typography of digital fonts. This document is intended to aid users/developers in development/editing of their own digital fonts. The nomenclature of type, letterforms and typographic measurement will be briefly explained. There will be a section on how the user can modify digital designs (using an editor like MacIkarus or Fontographer). How to make a true italic; how to embolden; how to change x-height; how to modify serif structure; etc.

---

14. Most of the major developers of fonts have met with Apple prior to the outset of this project to discuss the timing and technical aspects of this translator. Most manufacturers have begun development of those translators to make their existing libraries of spline fonts available on the Macintosh.

15. See Outline Font Data SubERS - Appendix 17.2.5

## 7.7 User Interface

Bass 1.0 should be functionally transparent to the user with the possible exception of installing font resources in a 6.0 and earlier Finder environment.

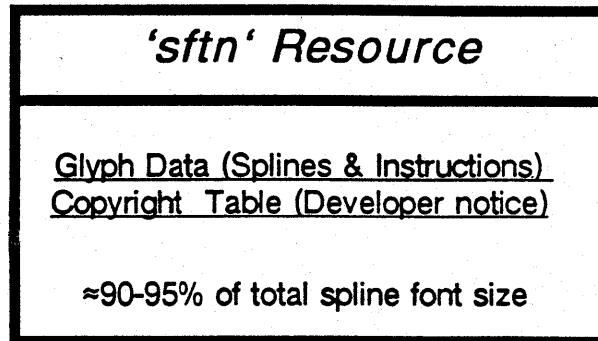
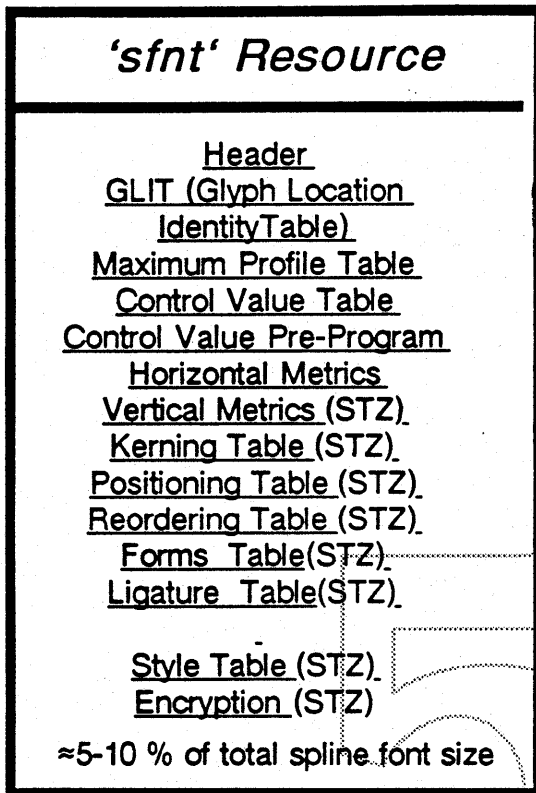
### 7.7.1 Font/DA Mover (or Equivalent)

Font/DA Mover or its functional equivalent will be required to recognize, update, copy, move and remove all font resources. Existing resources (font, FOND and nfnt) should behave exactly as they currently do while the new sfnt resource is integrated into the system. The user should be able to distinguish the difference between bitmaps and outlines. Help screen information should explain the concept of outline fonts for the first time user (i.e., that a single font description in outline form can be scaled to any size requested).

Several new offsets are incorporated in the sfnt to accommodate International data structure requirements. The font installation utility should be able to recognize these data structures as fonts are loaded into and removed from the system.

Where bold and italic fonts are available in a given family of fonts, they should not conflict with the existing nfnt data structures and the FOND resources should contain fractional width tables in the hypothetical 1 pt. size for compatibility with existing Postscript device drivers.

The following is a description of the sfnt data structure. For a detailed description See the Bass System Interface SubERS - Appendix 17.2.1.



The spline font format contains two resources. The first type is 'sfnt' (Spline Font) and the second 'sftn' (Spline Font next). The 'sfnt' portions contains the Header, GLIT (directory to the glyph data), Maximum Profile Table (memory requirements), Control Value Table, Control Value Preprogram, Metrics (Horizontal, Vertical), Kerning Table, Positioning Table, Reordering Table, Forms Table, Ligature Table, Style Table and Encryption.

The second 'sftn' contains the Glyph Data (Splines and Instructions) and the Copyright Table. **Developer Note:** This is the greater part of the spline font format. This will be kept on disk in low memory situations.

**Mover:** The Bass team needs the help of the mover to install two resources into the system file instead of the one previously required by 'FONT's and 'NFNT's. Both resources can be moved with the same methods as the font resources are moved. There FOND will indentify a spline font by a pt size of zero.

### 7.8 Hardware Requirement

Bass is expected to function in a standard Macintosh operating system environment with the exception that it will require at least 2MB of available RAM to operate efficiently.

The only other hardware dependency comes from the assumption that Macintosh screens using delta exceptions and/or optimized bitmaps will remain at the effective resolution of 72 dpi. As display resolution increases, this technology becomes even more economical. Delta outlines and/or bitmaps optimized for low res screens become less of a requirement for readability.

## 8.0 Implementation

### 8.1 Fonts

#### 8.1.1 Sources

The outlines will be purchased in Line and Arc IK format without grid-fitting instructions. The outlines will then be converted to Quadratic B-Splines and subjected to a semi-automatic program which will apply the basic set of Apple's grid-fitting instructions. The remaining instructions will be applied manually according to each font's style requirements. Any digital editing must take place in the Line and Arc IK format at production time.

Bass will not correct unfavorable artifacts in the outline data. The Font Instructions, Interpreter and Scan Converter are designed to "regularize" outline data for scaling to small sizes in low resolution and to convert outline data to bitmaps. Therefore, the quality of the Apple's Core Set is directly proportional to the quality of the original outline data used. The best quality outlines are available from three sources: Monotype, Corp. (Times New Roman), Linotype of Germany, LTD (Helvetica), and B & H (Lucida).

#### 8.1.2 Times, Helvetica, Courier and Symbol

The digital data for Times New Roman and Symbol is complete in Line and Arc IK format from URW as supplied by Monotype of England. The digital data for New Helvetica is available from URW as supplied by Linotype of Germany. Both require additional characters. Courier is available in IK format from URW. Required digital editing will take place prior to purchase of those designs. These outlines will then be converted to Quadratic B-Splines by URW prior to delivery to Apple.

#### 8.1.3 Lucida

Lucida Bright was redesigned from its original shapes to support very high resolution devices as well as those in the medium and low resolution range. Most of the digital data for the Lucida fonts is complete, some additional characters are required to complete the Apple character set. This work will be done at the studio of Bigelow & Holmes. These fonts will then be shipped to URW for conversion to Quadratic B-Spline format prior to delivery to Apple. B & H will be supplied with the semi-automatic instruction application programs for New Helvetica and Times New Roman. These programs should allow B&H to apply a partial set of instructions to Lucida Sans and Lucida Bright. B&H will apply the remaining instructions manually using Apple's Font Editor.

#### 8.1.4 Apple Exclusive Fonts

Original outline designs and digital data for Geneva, New York, Monaco and Chicago are in progress. These fonts will be jointly designed by Apple (K. Weisberg) and B&H. Design and development of the Apple exclusive font set will begin following the completion of the Lucida character sets. These fonts will not be encrypted.

### 8.1.5 Screen Fonts

Delta Exceptions (Hand-tuned outlines) will be developed for font sizes between 9 and 24 at 72 dpi. These outlines will be the result of manipulating control points on the final grid-fit outlines and storing the delta of those control points to the original outline. Each size will require a separate outline description, but the overall storage and retrieval will be more economical than hand-tuned bitmaps. These outline descriptions will also offer WYSIWYG when used in place of the existing bitmaps offered by Apple due to the linear metrics maintained throughout the size range.

### 8.1.6 Trademarks

The trademark of Times New Roman® is owned by Monotype of England. Helvetica® is owned by Linotype AG of West Germany. The digital data for these fonts are available from a number of digital font foundries which are not licensed to sell these fonts under those trademarked names. Lucida® is the registered trademark of B&H. Agreements are in progress for paid-up license of these trademarks on all Apple products.

## 8.2 Spline Font Package

### 8.2.1 Instruction Application

A program will be written which automatically applies as many instructions as possible on a per font basis to each font in the core set. Instructions will be applied to New Helvetica first, then Times New Roman, Courier and Symbol in that order. Basic instructions such as grid-aligning straight and curve extrema, positioning cap and horizontal references, identifying round and flat references will be done automatically. Remaining instructions required to identify peak and control of diagonal and additional fine-tuning of the outlines will be done manually using the instruction Editor. One of the most significant pieces of this technology will be missing when the product is announced. That piece is a truly automated instruction tool which is designed to eliminate the tedious and timeconsuming process of applying instructions to fonts. It is hoped that this piece will be under development shortly after release of Bass and after development should be licensed to all manufacturers of fonts for the Macintosh.

### 8.2.2 Back-up Translator

As a contingency plan a back-up translator is being developed to convert URW, VS formatted fonts into the Apple format. This would allow already instructed (hinted) outlines in URW's VS format to be translated to Bass Format automatically should we be unable to produce the necessary font set in time for introduction. See the VS Translator SubERS - Appendix 17.2.6.

## 8.3 QuickDraw

### 8.3.1 DrawText.a

**DRAWTEXT.A:** Responsible to call the Spline/BitMap Interface when the current font is of type "spline". DrawText must image the retrieved BitMaps to the screen. DrawText.a framework that exists today will be used. Some of the decision making will be altered to set the necessary variables when a spline font is the current type. Retrieval will be from the Hash/Buffer instead of the font or nfnt strike. Blitting routines will be duplicated as much as possible from the previous DrawText.a routines.

**TEXT.A:** Responsible to set up standard procs. Text.a will be called by DrawText.a when a string is needed to be measured. Text.a already has all the necessary capabilities to carry out these tasks. NeedBits flag needs to be set correctly.

### 8.3.2 Partial Resources

The International sfnt resource consumes large amounts of storage space. A utility is required that will break down the sfnt resource into smaller, more manageable and transferable pieces allowing for transfer by floppies. The Resource Manager will be modified to allow for partial resources. Therefore, only a Look-up Table will be needed in memory and not the entire sfnt. Two routines are needed by the Resource Manager to read and write partial resources. GetPartialResource will pass a buffer which reads or writes the resource Type, ID, Offset and Size. When correct parameters are provided, the Resource Manager will perform error checking and pass the partial resource back. This is expected to be a trivial modification and may require one or two new traps (\$A81D and \$A81E).<sup>16</sup> Support of this feature will be implemented in the Bass 1.0 timeframe.

### 8.3.3 Glue

The glue code provides a functional interface to the sfnt resource. The glue code routes high level calls from the Font Manager to the Scan Converter and Interpreter. This occurs, in general, in two different ways. When a new font or font size is requested, FillWidthTable is called, and a scaled fractional width table is filled out accounting for point size and device resolution. In addition, fields such as ascent, descent, leading and widMax are filled out for the width table and FMOutput record. This second structure maintains compatibility with earlier font metric models. When a specific character is requested, DoGlyph is called, and the outline and instructions are found and unpacked, and then sent on to the Scan Converter and Interpreter. The resulting bitmap is returned to the caller for caching or blitting to the screen. See the Section on Glue in the Bass System Interface Sub-ERS - Appendix 17.2.1.

16. ReadPartialResource (theResource: Handle; offset: LongInt; buffer: Ptr; count: LongInt);  
WritePartialResource (theResource: Handle; offset: LongInt; buffer: Ptr; count: LongInt);

## 8.4 Printing

A new routine: `boolean IsOutline( )`; will be added to support printing.

## 8.5 Font Tools

### 8.5.1 Font Editor

The editor's primary function, at this time, is to allow modification of the grid fitting hints for individual character outlines. A powerful, stack based set of instructions has been designed to improve readability of outline characters at small point sizes. Due to its compact nature, the font instructions are difficult at best to read or modify in their raw state. In conjunction with instruction editing, a useful display of the current character including outline, bits, and associated metrics is included.

The Editor offers a graphical interface to creating instructions, and displays the current graphic state available to the interpreter.

## 9.0 Future Enhancements

Several enhancements are planned for the 2.0 release of Bass. Some of those intended enhancements are listed here. Fundamental typographic considerations (i.e., accurate point sizing, integrity of design parameters, valid style properties and accurate metrics, to name a few). A proposal for these and other goals/enhancements will be described in a separate document.

Priority A:  
Automated Instruction Application  
Code Optimizations

Priority B:  
Gray Scale  
Multi-bit deep fonts  
Enhanced Core Set  
Vertical Kerning

Priority C:  
Intelligent Banding  
Optical Scaling  
Special Effects



## 10.0 Documentation

### 10.1 Inside Macintosh

The Font Instruction Set should be published in Inside Macintosh.

## 11.0 Testing

Prototype code will be given to SIAC for the purpose of testing 50 major applications. Alpha and Beta 1 testing will be contained within s/w engineering and SQA. Beta 2 testing will include a general release to the Software Engineering Library and major developers.

## 12.0 Developer Issues

### 12.1 Developer Survey

Developers will be queried to determine who does the following:

1. accesses the Font record directly
2. calls FMSetFont
3. changes information in the FMOut record
4. looks at the Font record via FMSetFont
5. looks at the Font record via FontMetrics
6. looks at or changes the width table (via FontMetrics or not)
7. jams low memory globals
8. references low memory globals
9. uses SetFractEnable
10. uses SetFScaleDisable
11. uses Kerning tables
12. generates PostScript
13. bypasses StdText
14. supports gray scale
15. supports color

### 12.2 Menus

When splines are present RealFont should always return TRUE to ensure that the Size menu displays all font sizes in style *outline* to indicate that all sizes are available. Well behaved applications already use this approach, however they may need to rev to offer infinite sizes to the user. The former limitation of 127 • does not exist in Bass and is a basic assumption by most applications developers.

### 12.3 Font Manufactures:

Font Manufactures have requested space in the sfnt resource to store their name and Copyright information. The sfnt will provide a Copyright Ascii Offset that can be set to any length. The Copyright table will be an area where the Font Manufacturers can place their credits. Any information that they wish to exist in the Font such as; Copyright dates and notices, Designer, version, etc. should be contained in this table.

An offset is planned in the sfnt resource to provide the ability to point to information the Font Manufacturer wishes to publish.

Font Manufacturers would also like to display their names in the Font Menu along with the font name. The menu entry is a pascal string and will provide a 255 character space. The only other restrictions are the screen size which will be clipped on the right.

**12.4 Font Editors:** Developers writing font editors in the Macintosh environment, such as Altsys, will be required to rev. to support the new data structures. They will also require a path to the outline making data encryption a difficult and unyielding issue.

**12.5 Applications:** Developers writing applications in the Macintosh environment will be expected to rev. where they are violating the guidelines of Inside Macintosh with respect to the Font Manager.

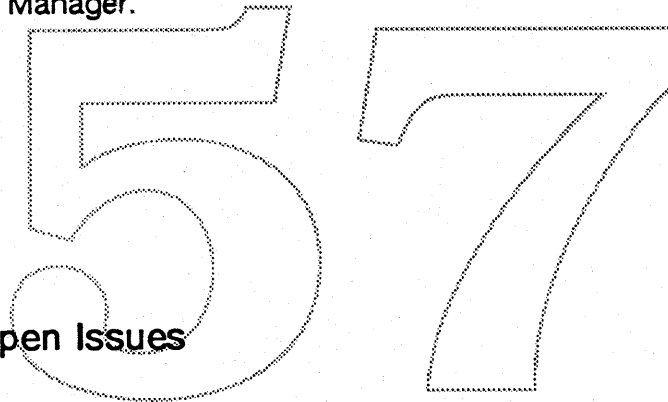
### 13.0 Schedule

Prototype 7/29/88

Alpha 2/27/89

Beta 5/30/89

Final 9/10/89



### 14.0 Risks and Open Issues

#### 14.1 Risks

- A. Ability to function in a 1MB Macintosh environment.
- B. Successful modification of QuickDraw and Font Manager routines with minimum effect on major applications.
- C. Memory management in a 2MB environment with Ginsu and Multifinder.
- D. Successful modification of Postscript drivers in time to seed developers.
- E. Implementation of a solid Test Plan from SQA prior to Alpha release.
- F. Successful completion of translation software and availability of production fonts from Font Manufacturers by introduction.
- G. Compatibility with NuFinder.
- H. Third party release of fonts relying on a specific set of instructions and scan conversion techniques.

#### 14.2 Open Issues

- A. Strategy for release of Bass implementation details to Adobe.
- B. Effect of QD/FM patches on major applications.
- C. Viable Postscript solution

## 15.0 Internal Dependencies

### 15.1 Finder & Applications

A compatible version of the Font /DA Mover or it's equivalent will be required prior to Beta release in May '89. An experimental version has been created which sucessfully moves sfnt resources into the System. This version may be released to developers at seed time.

### 15.2 Printing

A compatible version of the Postscript Driver will be required prior to Beta release in May '89. Announcement of Bass on May 8 and release of Bass to major developers in that timeframe neccessitates a Postscript compatible driver.

### 15.3 Resource Manager

The Resource Manager must be able to handle partial Resources prior to Beta release in May '89.

## 16.0 The Team

Kathryn Weisberg - Proj. Lead, Instruction Editing

Sampo Kaasila - Coding: Interpreter, Instructions, Scan Conversion, Glue Code

Mike Reed - Coding Font Editor, Glue Code

Charlton Lui - Coding Font Manager, QuickDraw Mods, Glue Code and Hash Buffer

Richard Becker (Contract) - Coding VS Translator, Outline Font Data Protection Scheme

Lee Collins and Dave Opstad - Coding International Data Structures

Jim Gable - Hardware Product Management

John Harvey - Developer Technical Support

Dianne Patterson - Inside Macintosh and Format Documentation Development

Andy Yarborough and Pam Martin - System and Printer Software Quality and Test

## 17.0 Appendix

### 17.1 Glossary

**Core Set** - the set of fonts shipped with a given device. Apple is not in the business of retailing or otherwise offering fonts commercially. These fonts are designed to support the standard needs of the average user at time of purchase. Additional fonts will be available from font suppliers such as Compugraphic, Adobe, Bitstream, URW, Linotype and Monotype.

**Protection Scheme**; provides protection to font developers against plagiarism and trademark infringement.

**Font Instruction Set** is a set of primitives for the control of points and typographic features when grid-fitting a digital outline. (Sometimes called "hints").

**GLIT** - Glyph Location Identity Table

**Glyph** - character

**Grid-fitting** - the alignment of control points in a digital outline description to a grid for the purpose of facilitating intelligent scaling routines

**Hints** - a set of primitives which the Interpreter can decode for the purpose of grid-fitting a digital outline, more correctly referred to as "instructions".

**Interpreter**; the program which interprets the instruction set and transforms the outlines accordingly.

**Linear Scaling** - the process of deriving all sizes of a shape from a single definition where the new size is an exact multiple of that basic definition.

**Optical Scaling** - in classic typography, changes are made to a design as the size gets smaller. These changes to the x-height, ascenders, descenders and counters are done to "optically" correct for the loss of integrity and legibility. This requires the use of either multiple masters or a very sophisticated algorithm, neither of which are planned for Bass 1.0.

**Quadratic B-Spline** format is described by two sets of points; those on the curve and those not on the curve. A straight line segment is described by 2 consecutive points on a curve. This B-Spline guarantees zero and first degree continuity.

**Scan Conversion** converts outline descriptions into bitmap format.

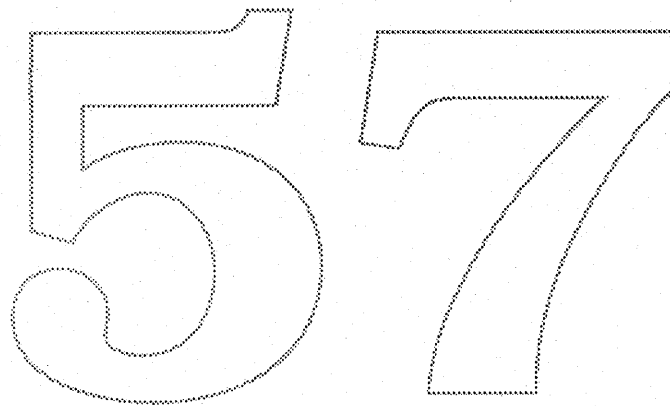
**sfnt** - Spline Font Resource. The data structure which houses the actual splines, tables and offset pointers. See System Interface Sub-ERS for a full description of the sfnt Resource.

**Strike** - a Macintosh convention for compressing all the characters in a given font side by side, with black bits touching. The strike is used by the Font Manager and Quickdraw to identify pen location, height and width of characters for drawing to the screen.

**Translator** is a piece of code designed to translate an external format for the core set of fonts to the Bass format. Commercial font suppliers will be made aware of the intended format in time to generate their own translators to supply fonts in a Macintosh environment.

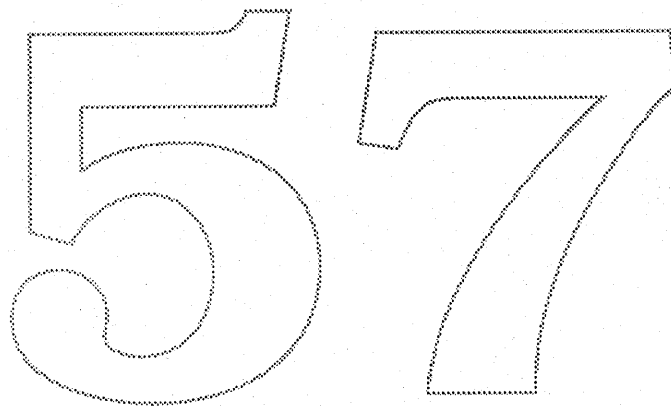
17.2 APPENDIX - Sub-ERS(s) THE REAL MEAT!

- 17.2.1 System Interface
- 17.2.2 Font Instruction Set and Interpreter
- 17.2.3 Scan Converter
- 17.2.4 Bass Instruction Editor
- 17.2.5 Outline Font Data Protection
- 17.2.6 VS Translator



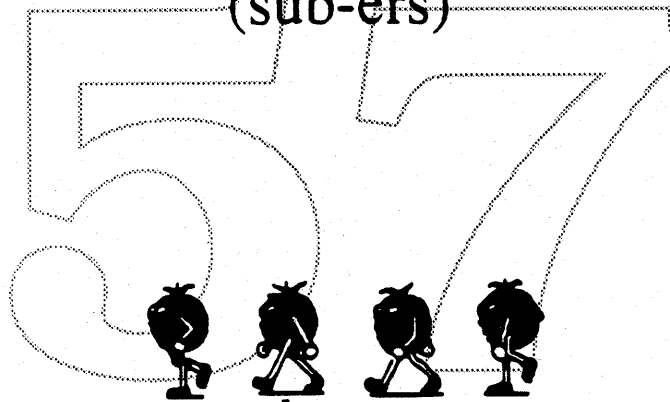
## 17.2.1 System Interface SubERS

This System Interface has many recent changes which have not been incorporated into this SubERS. See Charlton Lui for an updated release of this SubERS.



# “BASS”

## System Software Interface (sub-ers)



Charlton E. Lui  
MS: 27 AJ; Extension 4-2331  
Product Development  
Apple Computer, Inc.

Mon, Jan 23, 1989; 12:02 PM  
Version: 1.1

## TABLE OF CONTENTS

I.	<b>Introduction</b>	3
II.	<b>System Interface Overview</b>	4
	A. Simple Task Overview	5
	B. QuickDraw Interface	6
	C. Font Manager Interface	7
	D. Buffer Interface	10
	E. Glue Interface	17
	F. Client Interface	22
III.	<b>Appendices</b>	28
	A. Issues of Concern	28
	B. Size and Speed Considerations	30
	C. Questions for Developers	31
	D. Spline Font Resource Description	32
IV.	<b>Figures</b>	
	1. System Interface Overview	4
	2. QuickDraw Interaction	6
	3. Font Manager Interaction	7
	4. Font Association Table	8
	5. Buffer Interaction	10
	6. Buffer Diagram	12
	7. Glue Interaction	17
	8. Client Interface Interaction	22



## Introduction

*The primary objective of the system interface is to pinpoint relevant issues that impact and enhance the classic Macintosh system.* Currently, only bitmaps exist which may create poor quality output of intermediate- and large-scale fonts. To achieve a higher quality output, the user must maintain a diverse library of hand-tuned bitmaps. However, this would consume vast amounts of disk space and still not solve the problem of large scale fonts. It is important to mention that though a user may have an exact bitmap, it does not account for the printing engine characteristics (i.e., fatter size pixels). Therefore, to obtain precise output, a user needs two separate fonts; one for the printing device and one for the display device.

The primary goal of BASS is to produce intermediate- and large-scale fonts. This goal includes the ability to render larger than 127 pixels per em to output devices (e.g., produce fonts greater than 24-point screen resolution that will print on a 300 dpi laserwriter). By keeping only the spline font (SFNT) resource and small size bitmaps, a user may have many sizes without consuming large amounts of disk space relative to the hand-tuned bitmaps. This will afford intermediate and large size fonts.

Memory requirements is an extremely important topic that the system interface is slated to address. If the system interface loads the SFNT into memory and creates the whole character set for large size fonts, the requirements may become overwhelming and not feasible. However, it is possible to only render and cache on a per glyph basis. Individual glyph caching will provide many different fonts and sizes to reside in one area. This area is the bitmap hash/buffer. The buffer will contain all fonts, glyphs, sizes (not greater than some maximum), and engine characteristic variations. When switching between applications, a glyph may not need to be re-rendered if it still resides in the cache. Therefore, the system will have the blinding blitting bitmap speed of cached glyphs.

A concern of Bass is that a few applications are looking directly at the current FONT or NFNT resource. So far, the only known applications are two: AltSys Font Editor and MicroSoft Word. Microsoft has already fixed their bug in the 4.0 version of Word! Since Bass is introducing a completely new resource format, the applications will not be able to recognize it. It would not be appropriate to utilize the current FONT or NFNT resource due to the variations in format.

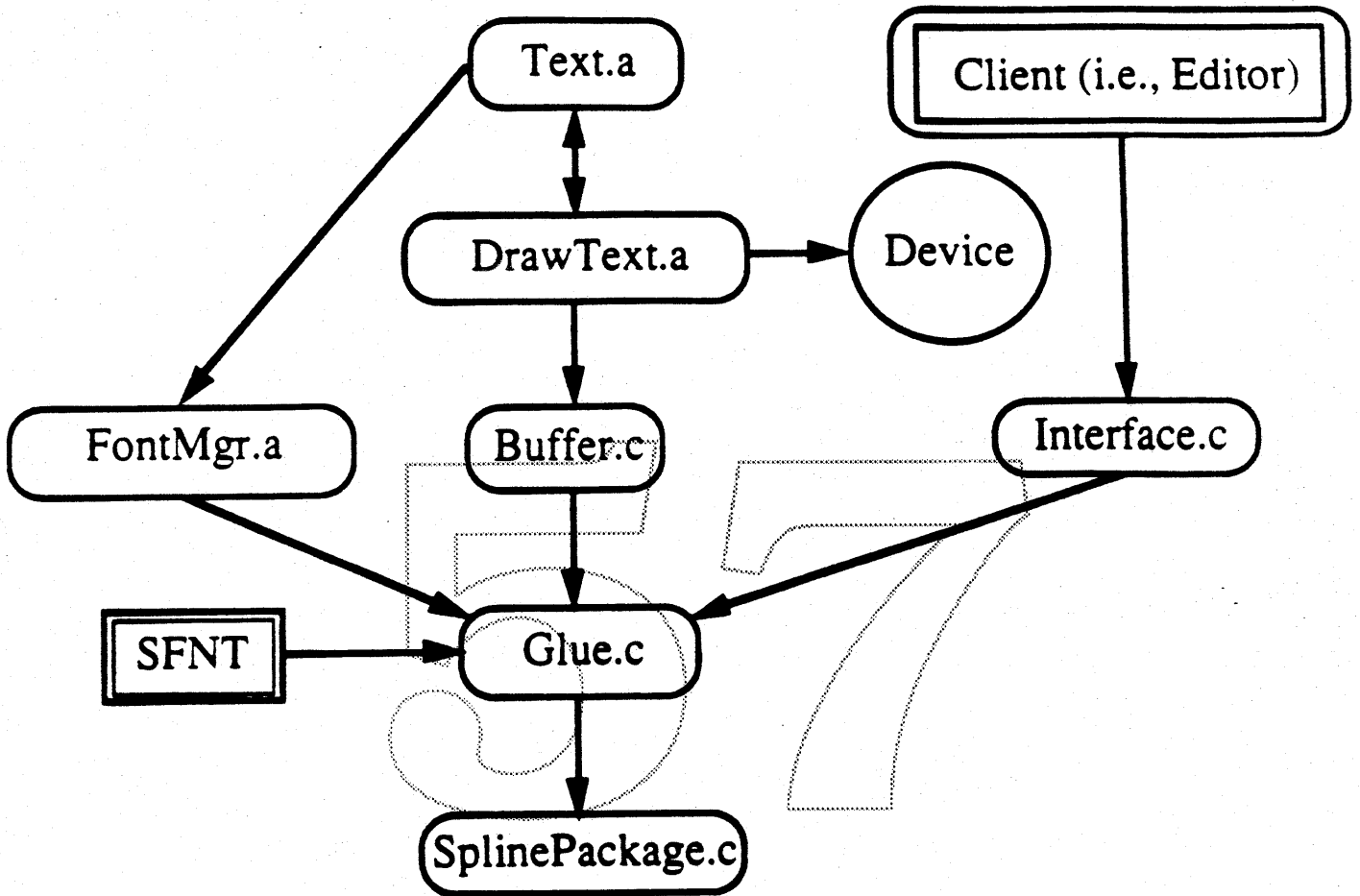
Many individuals have contributed their ideas to this System Interface Specification. From these ideas and suggestions, the system interface has been designed and will further evolve through continued discussions and feedbacks that is generated from this specification. Please feel free to offer any new perspective on this topic.

BASS' System Interface intends to implement a 1.0 version release due in the 89' summer time frame. The System Interface will integrate the Spline Font package into the Classic Macintosh system.

**Please note:** This is the system interface introduction, for more general information regarding the BASS project look at the general ERS overview.

*A user need not wait for years before getting great looking fonts at large- and intermediate-sizes at an affordable price.*

## System Interface Overview



**Figure 1**

The System Interface will be described by the six modules: QuickDraw, FontManager, Buffer, Glue, Spline Package and Editor Interface. Note that the Buffer, Glue, Spline Package, and Interface files are written in "C" and did not previously exist in the Macintosh system. Two existing pieces are QuickDraw (Text.a and DrawText.a) and the Font Manager which will need moderate modifications to the decision path and routine enhancements.

### Simple Task Overview

Please refer to *figure 1* for the descriptions below...

1. **QuickDraw:** Text.a and DrawText.a modifications will retrieve, clip and blit from an individual glyph strike. Quickdraw will use sfnt metrics and all existing blitting routines. The measurement of text will be enhanced to use fixed metrics (i.e., ascent, descent, leading and widMax) vs. the one byte (255) limitation. Therefore, text will be greater than the 255 ppm limitation.
2. **FontManager:** All present capabilities of Font Manager routines will remain the same. The Font Manager will be enhanced to recognize and load spline font resources when it is appropriate (e.g., if an exact size bitmap does not exist). When a spline font is chosen it will use an extended width table with sfnt metrics and private spline storage data. If a spline is current, the font chosen may be greater than 255 ppm.
3. **Buffer:** The buffer code is primarily responsible for the caching and retrieval of glyphs. If a glyph exist in the cache, it will return the rendered bitmap glyph data. Otherwise, it will call the glue code to render the glyph. The buffer code caches individual glyphs on an per need basis, searches via a hash function, and stores glyphs in dynamic cache chunks. It will cache either bitmaps XOR grid-fitted contours depending on memory and speed requirements. The buffer code will band on individual glyphs to conserve memory. The buffer code is the path in which a rendered glyph may be retrieved.
4. **Glue:** The glue code's tasks include creating an extended spline width table, and unpacking and scaling glyph data as well as allocating memory for the spline package.
5. **Spline Package:** This contains the Scan Converter and Font Interpreter and is only here to depict the interaction with the glue code. Once the glue code unpacks the spline data into digestible data for the spline package, the spline package routines will be called accordingly. For further details, see Sampo Kaasila's "Font Instruction Set and Interpreter" and "Scan Converter" sub-ers.
6. **Client Interface:** The Interface code is currently used with Font Editors when retrieving grid-fitted or non-grid fitted contours, querying for metrics and scan converting. This may be used by other clients such as Weitek or the skia group.

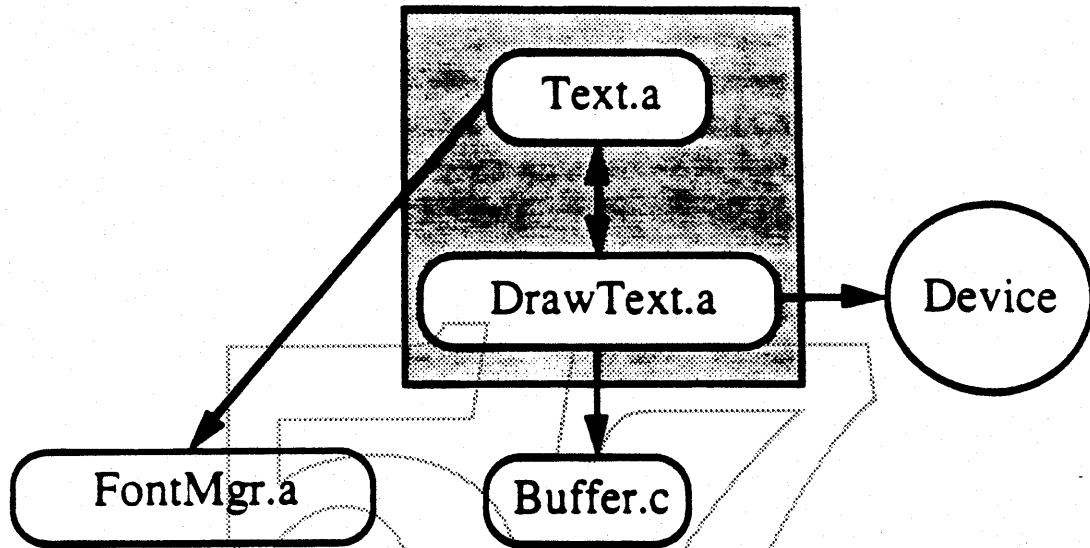
QuickDraw

Figure 2

QuickDraw interacts between the Font Manager, Buffer code and the Device. Quickdraw will call Font Manager and Buffer routines, and output to the Device.

- I. QuickDraw calling the Font Manager... QuickDraw calls the FmSwapFont from StdTextMeas to load the font. A font can be of type FONT, NFNT or sfnt. The FONT and NFNT font resource loads will carry out the present functionality. If a spline font is loaded, QuickDraw will use a modified version of DrawText.a. This version will use the sfnt metrics and call out to the buffer code for each individual glyph. DrawText.a will clip and blit from the individual character strike.
- II. QuickDraw calling the Buffer code... When a spline font is current, the Buffer code will be called to retrieve the individual glyph data. There is only one call to the buffer code which is documented in the Buffer section of the System Interface Overview.
- III. QuickDraw outputting to the Device... All blitting code is preserved so QuickDraw will output to devices as it does currently. Stretchbits will still be called when there is an algorithmic style change, non-black destination is needed, or the source and destination do not match.

Other Notes: Spline fonts introduce larger ppem size fonts which makes the algorithmic styles (i.e. bold, shadow and underline) work poorly. Enhancements will be necessary for these styles to appear appropriately. When stretching is invoked, the spline package will carry out this transformation, therefore stretchbits will not be called.

FontManager

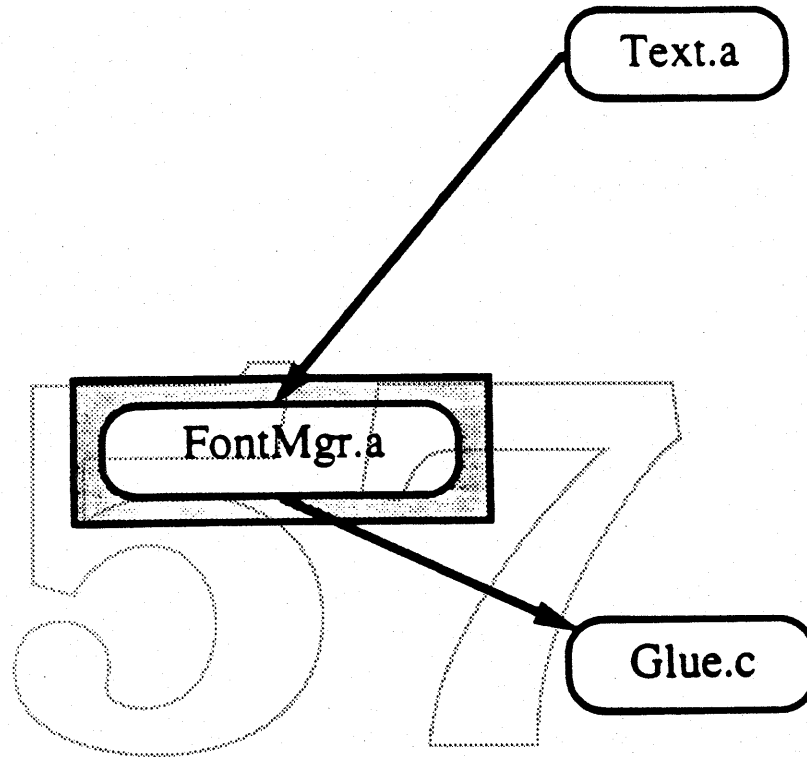


Figure 3

The Font Manager interacts with QuickDraw (Text.a) and the Glue Code. The Font Manager is called by Text.a and calls the Glue code.

- I. Font Manager called by QuickDraw (Text.a)... Text.a calls FmSwapFont to load the correct font. The font will be of type FONT, NFNT or sfnt. If an sfnt exists, it will be the first entry in the FOND resource (refer below to figure 4) and its size will be set to zero. FmSwapFont will continue to search for an exact size bitmap font. If an exact bitmap is found, it will use the font and carry out existing Font Manager functions when loading an FONT or NFNT. Otherwise, the intermediate- or large-size font requested will result in the use of the spline font. Upon usage of the spline font, the glue code will be called to create an extended spline width table.

FOND Associati		
Size	Style	ID
0	Plain	#ID (Spline)
12	Plain	#ID (BitMap)
14	Plain	#ID (BitMap)
18	Plain	#ID (BitMap)
24	Plain	#ID (BitMap)

Figure 4

- II. Font Manager calling the Glue code... FmSwapFont will call the glue code to fill out the extended width table when a spline font is current.
- II. Other Clients... Other clients may call the Font Manager routines:

**InitFont:** Initializes the spline workspace and buffer. If the memory allocation fails, spline fonts will not be utilized.

**GetFontName:** This routine will remain the same.

**RealFont:** If the current font is of type spline, RealFont will always return true. This is done since a spline font can be of any size.

**SetFontLock:** This routine will remain the same.

**FontMetrics:** The FmOutput record contains a one-byte (255) limitation for the ascent, descent, leading, and widmax. The sfnt resource will contain fixed precision metrics which will be stored in the extended spline width table. These metrics will be used in the font metrics routines instead of the FMOutput metrics. The horizontal and vertical scale will be added in as done currently. The shadowed adjustment will be made to the ascent and descent values. Extra style width will be added to the fixed widmax value.

**SetFScaleDisable:** This routine will remain the same. Take note that this will not effect the spline fonts since it will be able to scale to any size.

**SetFractEnable:** This routine will remain the same. Also note that the sfnt resources will always use fixed metrics and fractional widths. Thus, this routine will not affect spline

fonts.

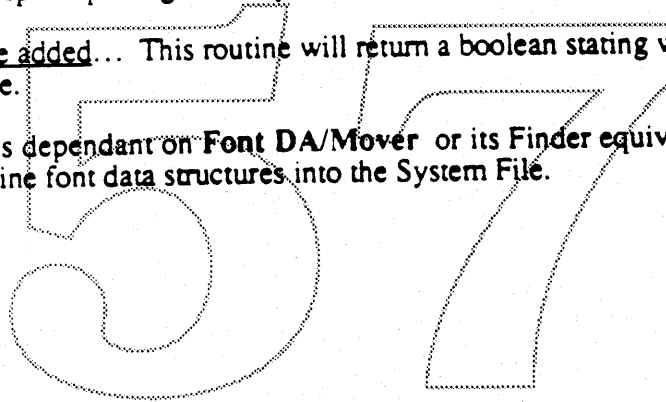
**FMSwapFont:** The *FindSize* routine has been modified to look for and select spline fonts when an exact size bitmap does not exist. Entries in the FOND Association table are ordered by size (see figure 4 above). If there is a spline entry, set a register to specify that a spline match has been found. FindSize will carry out its normal operation of trying to find the exact FONT- or NFNT-size match. If there is an exact FONT or NFNT match, carry out the current operation. If an exact match was not found, use the sfnt if one exists.

Once a spline font matches in FindSize, the *Style* font must be matched. Match the Style by circulating through the fonts which are indicated by the size zero.

*LoadNewStrike* must load the sfnt resource if a spline font is chosen. When loading the font it will also call the glue code to allocate necessary size enlargements of the widthtable and spline package workspace. If this fails, substitution will take place.

III. IsSpline routine added... This routine will return a boolean stating whether the current font is of type spline.

Other Notes: BASS is dependant on **Font DA/Mover** or its Finder equivalent to be able to add, remove and update spline font data structures into the System File.



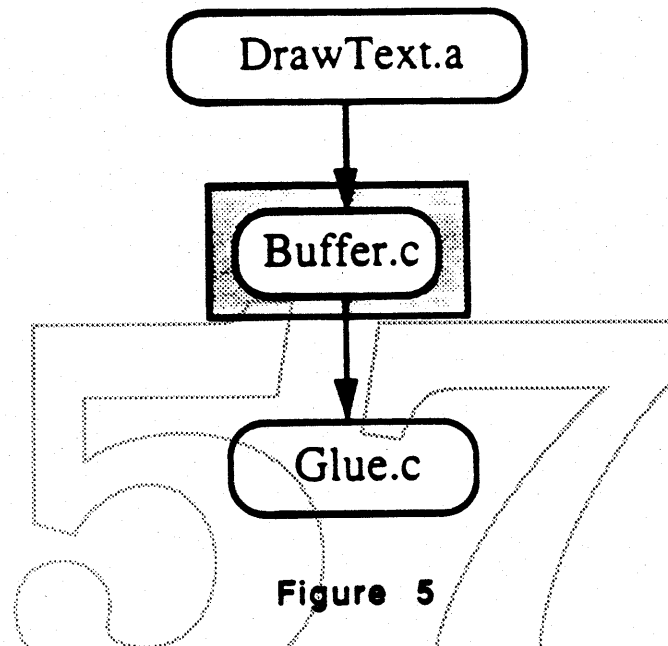
Buffer

Figure 5

The BitMap Hash/Buffer is introduced to overcome the limits of spline-to-bitmap rendering speed. By keeping the BitMap in memory, the high speed of BitMap blitting can offset the rendering speed to create useable interaction speed. These routines cache and de-cache on a per glyph basis.

The Buffer code interacts with QuickDraw (DrawText.a) and the Glue code. The Buffer code is called by DrawText.a and calls the glue code.

- I. Buffer code called by QuickDraw (DrawText.a)... The buffer code is called when DrawText.a wishes to retrieve the glyph data. The buffer code returns an individual glyph strike that is adjusted by the clipping region. In turn, QuickDraw will blit from the glyph strike.
- II. Buffer code calls the Glue Code... When a glyph is requested, the buffer code will search the cache. If the glyph is found, it will be clipped with the clipping region and returned with its metrics up to the caller. Otherwise, the glue code must be called. The glue code will return the individual character, and if below certain size restrictions, blit directly into the cache. The rest of the glyph information will be filled out into the cache and then the information is returned up to the QuickDraw level.

Other Notes: When a device is introduced, a buffer with its characteristics will be created. Further enhancements to keep in mind: if gray scale fonts are created, they could be cached; multi-depth fonts may be cached also; a new cache could be introduced for each depth of device. Banding will be provided for low memory consumption.



Interface routines to outside clients:

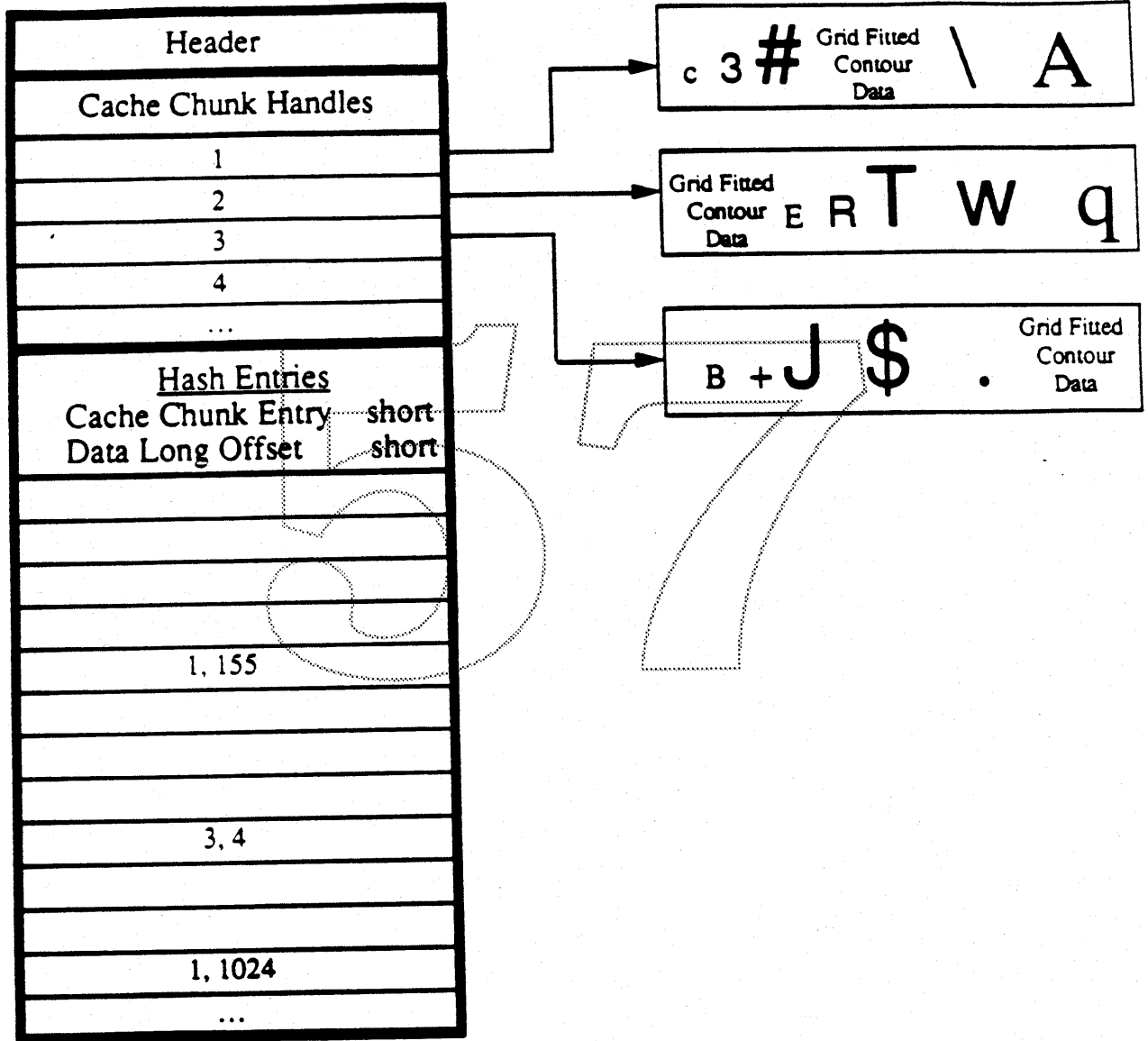
```

/*      sb_RetrieveGlyph
**
**      This routine will be called by QuickDraw from the DrawText.a module. The splineKey is for
**      the spline packages memory. The width table handle is an extended spline width table (see the
**      data structures below). The glyphPtr is the record which is filled out by the buffer code that will
**      be sent back up to QuickDraw. This is the individual glyph strike that DrawText.a will blit
**      from.
*/
void sb_RetrieveGlyph( splineKey, widTabHand, glyphPtr )
    sp_SplineKey          **splineKey;
    Handle                widTabHand;
    register sp_Glyph     *glyphPtr;

/*      sb_SetCache
**
**      This routine will be called at init time once for the screen and consecutive times for the
**      output devices from the glue code. The glue code will call this routine when the output
**      device has changed. The splineKey is the spline packages workspace, the cacheHintPtr
**      is used to give hints to the buffer code of what, when and how glyphs should be cached.
**      Please refer to the data structures for the breakdown of the fields.
*/
int sb_SetCache( splineKey, cacheHintPtr, deviceResolution, engine )
    Handle                splineKey;
    register sb_CacheHint *cacheHintPtr;
    long                  deviceResolution;
    F26Dot6               engine[];

```

**Buffer Diagram**



**Figure 6**

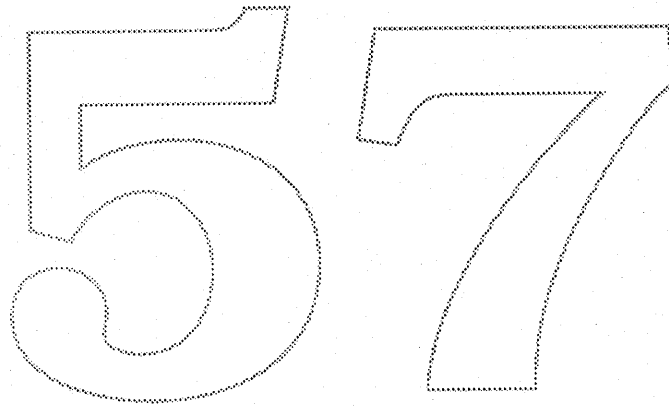
**Header:** The header contains current cache state and cache-hint values. The header is continually updated as the cache expands and shrinks.

**Hash Entries:** Each entry will contain a cache chunk number and an offset to the glyph data. The hashing routine will match to a specific cell in the Hash Table and then a check may be made to see

the glyph data matches. Each hash entry equals one long.

**BitMap Entry:** Each BitMap entry will include a header (see the buffer data structures). After the header, the bitmap will reside and be padded to be long-aligned.

**Grid Fitted Contour Entry:** Each Contour entry will contain a header (see the buffer data structures). After the header, the Contour Data will reside and be padded to be long-aligned.



**Buffer and Related Data Structures**

```

/*****/
/** SPLINE resource defines **/
/*****/

/* This resource type will be read to fill out the caching hints for the buffer mechanism
*/

#define RES_TYPE                'scsh' /* res type */
/*****/

/*****/
/** CACHE HINT STRUCTURE **/
/*****/

/* This structure will be used to tune the cache for the best possible performance.
*/

typedef struct {
    uint16    initialHashSpots; /* Start hash at this size */
    uint16    hashSpotGrowth; /* Grow the hash by this amount */
    uint16    maxHashSpots; /* Maximum Hash Size */
    uint16    hashFullPercent; /* A integer percent of hash fullness */
    uint16    initialChunkLongs; /* Initial chunk size */
    uint16    growChunkLongs; /* Growth of chunks */
    uint16    chunkLongLimit; /* If it grows pass this point use another chunk */
    uint16    maxPPEMCached; /* LargestCached ppem */
    uint16    contoursCacheOn; /* Cache contours */
    uint16    bitsCacheOn; /* Cache bitMaps */
} sb_CacheHint;
/*****/

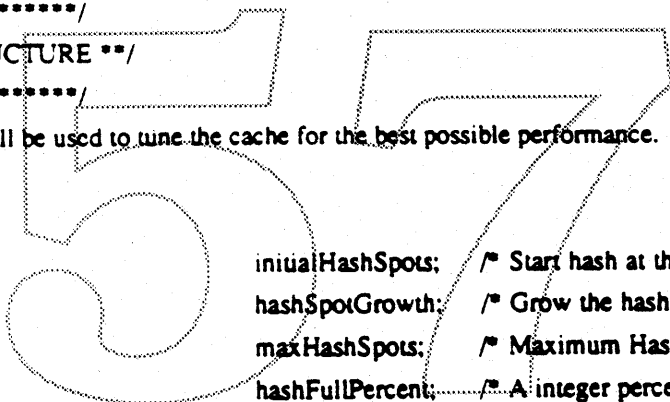
/*****/
/** CACHE HEADER **/
/*****/

/* The header information will be used to keep track of the caches state and also help decide when and how
** information will be stored.
*/

#define MAX_CACHE_CHUNKS                32 /* This may be tuned for the optimal number of chunks */

typedef struct {
    sb_CacheHint    cacheHint; /* What you ever wanted from a cache */

```



## Buffer Interface

## System Interface Overview

```

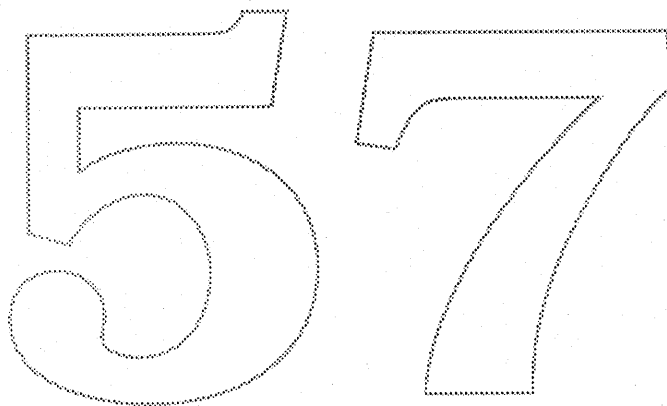
uint16
F26Dot6
uint16
uint16
uint16
uint16
uint16
sb_ChunkEntry
/** ALWAYS LAST!!!, HASH TABLE STARTS HERE **/
uint32
} sb_CacheHeader;
#define CACHE_STRUCT_LONGS ((sizeof( sb_CacheHeader ) + 3 - sizeof(uint32)) >> 2)
/*****/
/*****/
/** HASH ENTRY STRUCT **/
/*****/
/*    A quick directory to the cached data.
*/
typedef struct {
    int16
    uint16
} sb_HashEntry;
/*****/
/*****/
/** BitMap Header **/
/*****/
/*    Glyph data to be processed and passed up to QuickDraw.
*/
typedef struct {
    short
    short
    short
    short
    long
    short
    short
    devRes; /* Caches device resolution */
    engine[ENGINE_ARRAY_ELEMENTS]; /* Engine Characteristics */
    hashSpots; /* A integer percent of hash fullness */
    hashSpotsUsed; /* A integer percent of hash fullness */
    hashSpotLimit; /* Hash Spot Limit */
    currentChunkNum; /* current chunk in use */
    collisionCount; /* How many times was there a hash collision */
    cacheChunkEntry[MAX_CACHE_CHUNKS]; /* chunk entries */
    startHashTable; /* Start of Hash Table */
    cacheChunkNum; /* Cache chunk number data is in */
    dataLongOffset; /* The long offset to the data */
    yMin; /* Y min off baseline to place BitMap */
    yMax; /* Y max off baseline */
    byteWidth; /* number of byte width */
    bitWidth; /* Width of char bits */
    lsb; /* Left side bearing */
    rError; /* Error in case of huge size ll bad font */
    xMin; /* Save xMin and also use to calc xMax */

```

## Buffer Interface

## System Interface Overview

```
short      fontID: /* SFNT resource id */
short      glyphID: /* Glyph code ID */
short      ppem: /* Pixels per em */
short      yesCnrs: /* If Cntr then Scan it */
short      dataLongOffset: /* where it lies within the chunk */
/* DO NOT PLACE ANYTHING AT END, THIS IS DATA TERRITORY */
long      cacheData: /* Data starts here */
} sb_CacheEntry;
#define CACHE_HEAD_LONGS ((sizeof( sb_CacheEntry ) + 3 - sizeof(long)) >> 2)
/*****/
```



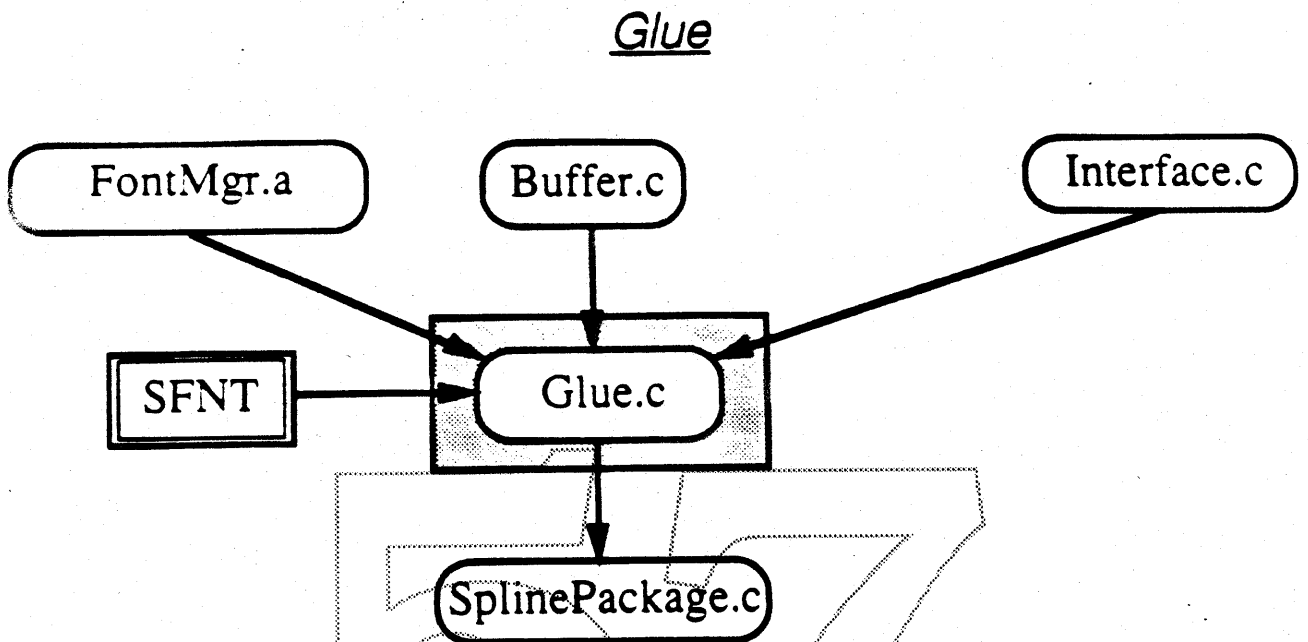


Figure 7

The Glue code interacts with the Font Manager, Buffer Code, sfnt resource, Client Interface, and Spline Package. The Glue Code is called by the Font Manager, Buffer Code and Client Interface. The Glue Code will in turn call the Spline Package when it is appropriate.

- I. Glue Code called by the Font Manager... The Font Manager calls the glue code when a width table and the spline workspace must be provided. Once the memory is allocated, it will call the glue code once more to fill out the width table.
- II. Glue Code called by the Buffer Code... The Buffer code calls the glue code when a glyph is not found in the cache. The Glue code will unpack the spline data for the spline package and call the appropriate spline package routines to grid fit the contours. The glue code will fill out all the necessary glyph metrics and data, and pass it up to the buffer code.
- III. Glue Code called by the Client Interface... The Glue code when called by the Client interface will unpack the spline data and call the spline package grid-fitting routines. The glue code will fill out all the necessary glyph metrics and data, and pass it up to the Client Interface.
- IV. Glue Code calls the Spline Package... The Glue code will call the grid-fitting routines once the spline data is properly unpacked. It will take the information passed up from the Spline Package and send back the appropriate information to its client.
- V. Glue Code using the sfnt... The Glue code will unpack and scale sfnt data for the Spline Package to digest. The unpacking algorithm is as follows:

**Constant:**

PIXELSIZE 16

**Byte Flags:**

ONCURVE 0x01  
 XSHORT 0x02  
 YSHORT 0x04  
 REPEAT\_Flags 0x08  
 NEXT\_X\_IS\_ZERO 0x10  
 NEXT\_Y\_IS\_ZERO 0x20  
 Reserved 0x40,0x80

**Packed Spline Format:**

ctrs 2 bytes /\* number of contours \*/  
 sp ctrs\*2 bytes /\* value of start point (e.g., value = 0) \*/  
 ep ctrs\*2 bytes /\* value of end point (e.g., value=17) \*/  
 sizeOfInstructions 2 bytes /\* length of the instruction list \*/  
 instructions sizeOfInstructions bytes /\* instruction array \*/  
 Byte Flags variable size  
 x array of bytes or shorts /\* flags state which one \*/  
 y array of bytes or shorts /\* flags state which one \*/

**UnPacked Spline Format - fnt\_ElementType:**

```
typedef struct (
    long *x; /* Modifiable interpreter points */
    long *y; /* Modifiable interpreter points */
    long *ox; /* Old Scaled points */
    long *oy; /* Old Scaled points */
    long *oox; /* Old unscaled points */
    long *ooy; /* Old unscaled points */
    long nScale /* numer required to scale points to size */
    long dScale; /* denom required to scale points to size */
    short *flags /* array of flags */
    short nc; /* number of contours */
    short *sp; /* Start points array */
    short *ep; /* End points array */
    char *f; /* Internal flags, one byte for every point */
) fnt_ElementType
```

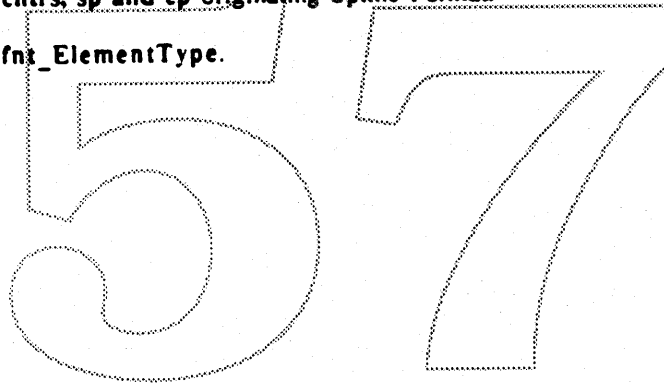
**Algorithm:**

- I. Interpret Byte Flags...
- A. Set the flags fnt\_ElementType to the onCurve bit value.
- B. If the XSHORT bit is set, expand next x value in Packed Format to a short and place contents in the oox fnt\_ElementType. Otherwise, move the next 2 bytes from the x value in the Packed Format to the oox fnt\_ElementType. Repeat the same algorithm for the YSHORT flag but place the results in the ooy fnt\_ElementType.
- C. If the REPEAT\_FLAGS is set check the next flag for a count and repeat the flags that many times. Do not repeat the NEXT\_X\_IS\_ZERO or NEXT\_Y\_IS\_ZERO unless those flags are set.



- D. The `NEXT_X_IS_ZERO` and `NEXT_Y_IS_ZERO` are used to re-use the x or y positions. If either of the flags are set re-use the last value and place them into the `oox fnt_ElementType` or `ooy fnt_ElementType`, respectively.
- II. Calculate the `nScale` and `dScale` values.  

$$\text{nScale and dScale equation: } (\text{point} * \text{nScale}) / \text{dScale} = (\text{point} * (\text{ppem} * \text{pixsize})) / (\text{units per em})$$
- III. Use the `nScale` and `dScale` equation to figure out each point for the `ox fnt_ElementType` and `oy fnt_ElementType`.
- IV. Copy the arrays `ox fnt_ElementType` and `oy fnt_ElementType` to the `x fnt_ElementType` and `y fnt_ElementType`.
- V. NOTE: `f fnt_ElementType` is not filled out, it is used in the Spline Package.
- VI. Loop through the number of contours and fill out the `sp` and `ep fnt_ElementType` with the Packed `cntrs`, `sp` and `ep` originating Spline Format.
- VII. Return `fnt_ElementType`.

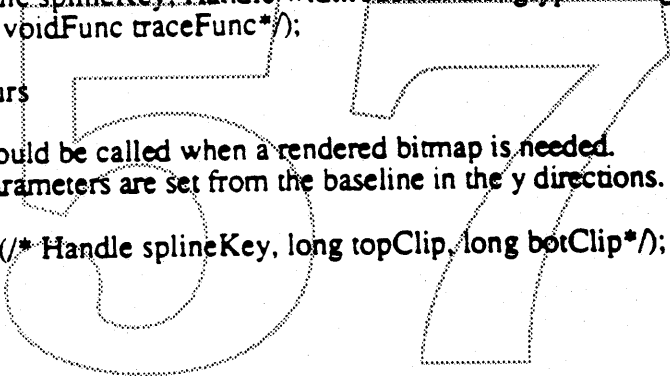


**Interface routines to outside clients:**

```
/*    sp_FillWidthTab
**
**    This routine should be called when a extended spline width table needs to be fill
**    out.
*/
void sp_FillWidthTab(/* Handle splineKey, Handle widthTabHand, voidFunc tracefunc*/);

/*    sp_GridFit
**
**    This routine should be called when a grid-fitted and non-grid-fitted contours are needed.
**    Set useHints to true for grid-fitted and useHints to false for non-grid-fitted contours.
*/
int sp_GridFit(/* Handle splineKey, Handle widthTabHandle, glyphRec *glyphPtr,
short useHints, voidFunc traceFunc*/);

/*    sp_ScanContours
**
**    This routine should be called when a rendered bitmap is needed.
**    The clipping parameters are set from the baseline in the y directions.
*/
void sp_ScanContours(/* Handle splineKey, long topClip, long botClip*/);
```

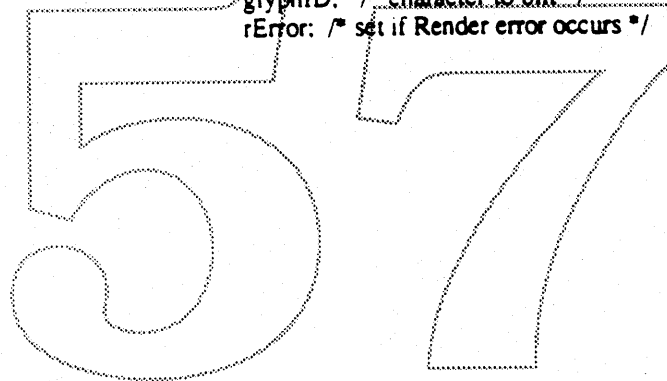


**Glue and Related Data Structures**

```

/*****/
/** For QuickDraw **/
/*****/
typedef struct (
    int16
    int16
    uint16
    uint16
    uint16
    Fixed
    uint32
    int16
    int16
    uint16
    uint16
)
sp_Glyph;
/*****/
yMin: /* Y min off baseline */
yMax: /* Y max off baseline */
scan: /* # of scan lines */
byteWidth: /* byte width of bitmap */
bitWidth: /* Word for char bit width */
lsb: /* Left side bearing */
*bitMapPtr: /* Points to the BitMap */
botClip: /* bottom clip off baseline */
topClip: /* top clip off baseline */
glyphID: /* character to blit */
rError: /* set if Render error occurs */

```



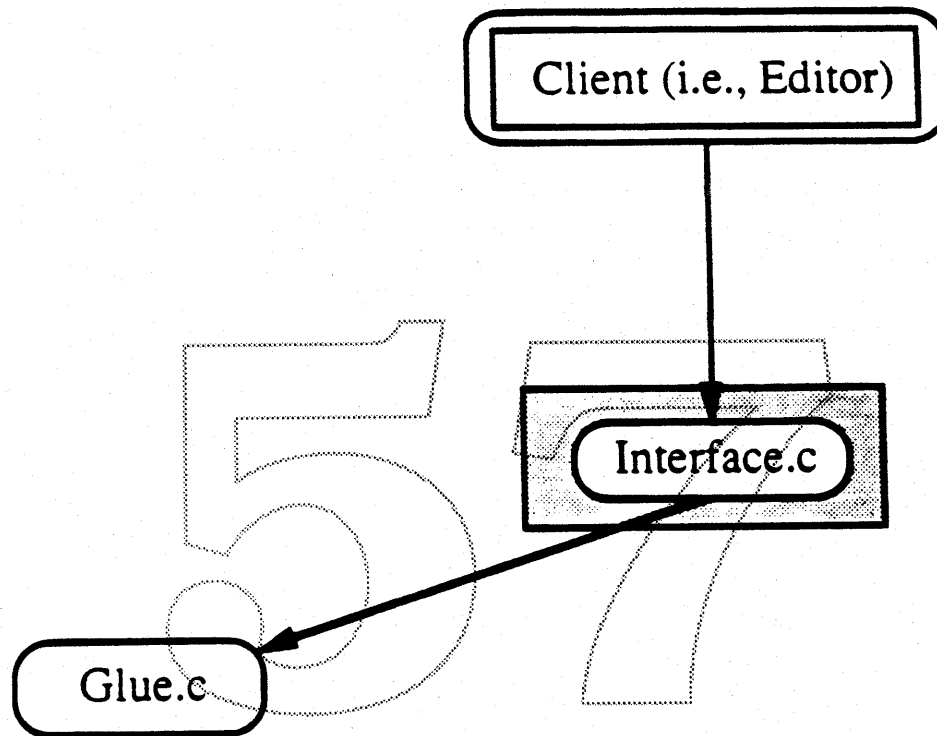
Client Interface

Figure 8

The Client Interface interacts with the Glue code and the Client (i.e., Editor). The Client Interface is called by the Client and calls the Glue code.

- I. Client Interface is called by the Client (i.e., Editor)... The Client calls the Client Interface when retrieval of grid-fitted and non-grid-fitted contours, or scan converting bitmaps are needed.
- II. Client Interface calls the Glue code... The Client Interface calls the glue code to unpack the *sfnt*, initialize the workspace, return the grid-fitted or non-grid-fitted contours, and scan convert outlines.

**Notes:** The Client Interface only recognizes a binary file format called *sfnt* (see the *sfnt* section). Warning, only use the Client Interface Scan Converting routine to guarantee the quality and integrity of the bitmap. By using the Client Interface Scan Converter, it will be possible to produce high quality fonts since all Font Manufacturers will produce their fonts knowing the results of the rendering model.

**Simple Calling Convention Definition:**

Calling Convention: (Further "C" definitions follow this brief description).

1. *its\_OpenSplinePackage*: Warning... call this routine first. No other routines may be called until the Spline Package is initialize.
2. *its\_InitPreProgram*: Call this routine when a new font, size, engine characteristic, or transformation is introduced. This routine must be called before *its\_ContourGridFitt()* or *its\_ContourNoGridFit()*.
3. *its\_ContourGridFitt* or *its\_ContourNoGridFitt*: To retrieve a grid-fitted contour or non-grid-fitted contour, call one of these routines respectively. These routine must be called before *its\_QBSplineTo3rdDegreeBezier()*.
4. *its\_QBSplineTo3rdDegreeBezierO*: Do not call this routine unless necessary. This only needs to be called when the Quadratic B-splines needs to be translated to 3rd Degree Beziers.
5. *its\_ContourScan*: Call this routine to render the bitmap . If there are no changes in font, size, engine characteristics, or transformation, then loop back to *its\_ContourGridFitt* or *its\_ContourNoGridFitt*. Otherwise, go back to *its\_InitPreProgram*. This routine may only be called after *its\_ContourGridFitt()* or *its\_ContourNoGridFit()*.
6. *its\_CloseSplinePackage*: When shutting down the spline package call this routine.

Interface routines to outside clients:

```

/*      its_OpenSplinePackage()
**
**      This should be called at boot time or before the first time the spline package is used.
*/
extern void its_OpenSplinePackage(/* SplineKey *key */);

/*      its_InitPreProgram()
**
**      This should be called whenever the typeface, size, engine characteristics, or
**      transformation change.
*/
extern void its_InitPreProgram( /* SplineKey key,
                               its_GlyphInfoType *inputPtr,
                               its_GlyphInfoType *glyphInfoPtr */);

/*      its_ContourGridFit()
**
**      To create the a grid fitted path for a glyph, call this routine.
*/
extern void its_ContourGridFit(/* SplineKey key,
                               its_GlyphInfoType *inputPtr,
                               its_GlyphInfoType *glyphInfoPtr */);

/*      its_ContourNoGridFit()
**
**      To create the a a non-grid fitted path for a glyph, call this routine.
**
**
*/
extern void its_ContourNoGridFit(/* SplineKey key,
                                 its_GlyphInfoType *inputPtr,
                                 its_GlyphInfoType *glyphInfoPtr */);

/*      its_QBSplineTo3rdDegreeBezier()
**
**      This routine is called when a Quadratic B-Spline definition must be translated to 3rd Degree
**      Bezier. Only call this routine after its_ContourGridFit or its_ContourNoGridFit. The
**      resulting data of one of these routines will be translated into the 3rd Degree Bezier data.
**
**      NOTE: Do not call this unless the path is absolutely required.
*/
extern void its_Get3rdDegreeBezier(/* SplineKey key,
                                   its_GlyphInfoType *inputPtr,
                                   its_GlyphInfoType *glyphInfoPtr,
                                   its_3BezierType *bezier3 */);

/*      its_FindBitMapSize
**
**      If the caller is providing the memory allocation (e.g., a caching strategy), this routine
**      must be called to find out how much memory is needed.
*/
extern void its_FindBitMapSize( /* SplineKey key, long *size */);

```

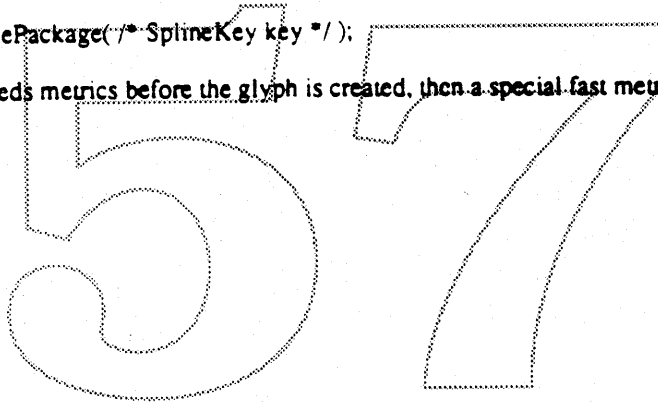
```

/*  its_ContourScan()
**
**  This routine will render the bits of the bitmap and should only be called after
**  its_ContourGridFit or its_ContourNoGridFit has been previously executed.
**  Set the glyphInfoPtr->bitMapInfo.baseAddr to null when the client wishes the Interface
**  package to allocate the bitmap memory.  Otherwise, if glyphInfoPtr->bitMapInfo.baseAddr
**  is non null, then it is assumed that the client has allocated the memory correctly.
**  we will allocate the memory
*/
extern void its_ContourScan( /* SplineKey key,
                           its_GlyphInfoType *glyphInfoPtr */);

/*  its_CloseSplinePackage()
**
**  The client should call this after the spline package is non longer in use.
**  This will deallocate its memory.
*/
extern void its_CloseSplinePackage( /* SplineKey key */);

/* NOTE: if the system needs metrics before the glyph is created, then a special fast metric case call may need to be
provided */

```



**Client Interface Data Structures:**

```
/* This defines the interface in C style */
#ifndef MACINTOSH
```

```
typedef void (*voidFunc)();
```

```
#define F26Dot6      long /* 26.6 */
#define Handle      long
#define Fixed       long /* 16.6 */
```

```
typedef struct {
    short           top, left;
    short           bottom, right;
} Rect;
```

```
typedef struct {
    unsigned char  rowBytes;
    short          Rect;
} BitMap;
```

```
#endif NO MAC
```

```
#define HEIGHT      3
#define WIDTH       3
```

```
/* Spline key is an internal data type and hidden from the external code */
#define SplineKey   Handle
```

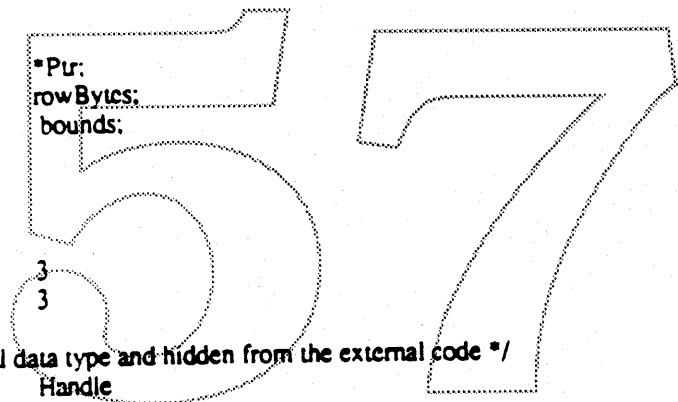
```
/* Do not use the translation part since we do not place characters */
typedef struct {
    Fixed          transform[HEIGHT][WIDTH];
} transMatrix;
```

```
typedef struct {
    short          glyphID; /* Apple character code */
    short          resolution; /* device resolution - dots per inch */
    Fixed          pointSize; /* requested size in points */
    transMatrix    transformMatrix; /* the transformation */
    Fixed          pixelDiameter; /* engine characteristics computed from this */
    Handle         sfntHandle; /* Handle to the binary sfnt data */
    voidFunc       traceFunc; /* Always set to NULL for PostScript !!! */
} its_GlyphInputType;
```

```
/* NOTE: pointsize 12 with a transformation of * 1 is not the same as
pointsize 6 with a transformation of * 2 !!!!. So we need pointSize */
/* Set pixelDiameter to this for perfect engine */
#define SQR2 0x00016A0A
```

```
typedef struct {
    Fixed          x, y;
} vectorType;
```

```
typedef struct {
```





## Client Interface

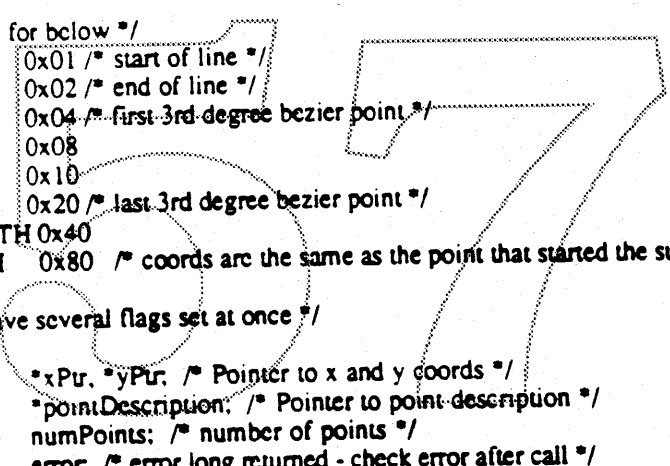
## System Interface Overview

```
vectorType advanceWidth, leftSideBearing, rightSideBearing; /* Horiz. metrics */
vectorType reserved1, reserved2, reserved3; /* reserved for later metrics */
vectorType reserved4, reserved5, reserved6; /* reserved for later metrics */
} metricsType;

typedef struct (
    metricsType metricInfo; /* Metrics of the glyph */
    F26Dot6 *xPtr, *yPtr; /* Pointers to the x and y points */
    short *startPointPtr, *endPointPtr; /* Pointer to start and end points */
    char *onCurveFlagPtr; /* Pointer to on Curve Flags */
    F26Dot6 *controlValuePtr; /* Pointer to the control value data */
    long error; /* Error long - always check this after each call */
    BitMap bitMapInfo; /* QuickDraw bitmap */
    short numberOfContours; /* total number of contours for this glyph */
) its_GlyphInfoType;

/* flags "pointDescription" for below */
#define LINE_0 0x01 /* start of line */
#define LINE_1 0x02 /* end of line */
#define BEZ3_0 0x04 /* first 3rd degree bezier point */
#define BEZ3_1 0x08
#define BEZ3_2 0x10
#define BEZ3_3 0x20 /* last 3rd degree bezier point */
#define START_SUB_PATH 0x40
#define END_SUB_PATH 0x80 /* coords are the same as the point that started the subpath */

/* note that a point may have several flags set at once */
typedef struct (
    F26Dot6 *xPtr, *yPtr; /* Pointer to x and y coords */
    char *pointDescription; /* Pointer to point description */
    long numPoints; /* number of points */
    long error; /* error long returned - check error after call */
) its_3BezierType;
```



## Appendix A

### Issues of Concern

#### Memory:

**Issue I:** Is it possible to introduce BASS' Spline FONT technology into the existing low memory conditions of a 1 megabyte system?

**Solution:** BASS is to be released on the during the summer system disk. This disk must work in a 1 megabyte system. If there is not enough memory for BASS, substitution will take place like the Font Manager does today. Therefore, a user needed worry about whether they can install BASS or not. It will be there if enough memory exists. When system segment loading is created BASS will have a much better time working in a 1 megabyte environment all the time.

**Issue II:** Rendering BitMaps from "Hinted Splines" is slower than merely blitting BitMaps.

**Solution:** The system interface has designed a font "BitMap Hash/Buffer" that will store rendered BitMaps. Thus, the blinding blitting speed of BitMaps will be averaged in with the rendering from "Hinted Splines".

#### General:

**Issue I:** When will BitMaps and Splines be utilized?

**Solution:** BASS' system interface will use exact BitMaps if they exist before using Splines. This is the 1.0 viewpoint and can change at any point in the development cycle.

**Issue II:** Will BASS provide the ability to use Multiple Masters (Spline Formats) per font face???

**Solution:** Most Manufactures did not seem very animate to obtain the ability to have more than one Master per FONT face. BASS does not incorporate Multiple Master and has not intended to do so.

**Issue III:** Microsoft Word and Alsys looks directly into the FONT or NFNT resources.

**Reply:** Microsoft Word 4.0 has already fixed this bug. Alsys being a font editor must rev there app if they wish to work with sfnt resources.

**Issue IV:** Font Manufactures would like to display their names in the Font menu along with the Font Name.

**Solution:** The menu entry is a pascal string and will provide a 255 character space. The only other restricuons are the screen size which will be clipped on the right.

**Issue VI:** Font Manufactures would like a place in the SFNT resource to store their name and Copyright information.

**Solution:** The SFNT has provide an Copyright Ascii Offsct that can be set to any length.

#### NON-Roman (e.g. Kanji, Hanzi, Arabic, etc...):

**Issue I:** Some Non-Roman character sets contain a huge number of glyphs (e.g. 7,000-Kanji at present, 13,000-Hanzi). It is extremely difficult to load all BitMaps into memory and time consuming to render.

**Solution:** A user normally uses 300-400 different glyphs during a session. BASS will render and cache each BitMap on a per glyph basis.

**Issue II:** International wishes the ability to provide non-Roman based ligatures & conjunct character replacement.

**Solution:** The BASS' Spline package will provide a Displacement Table where the glyph may be generated from two glyphs or be completely replaced by a pure glyph that exists in the FONT.

**Issue III:** Non-Roman characters may go from Left to Right or Top to Bottom or both.

**Solution:** BASS has intended to support International from the start. The new SFNT resource has been designed with the collaboration of the International group. The SFNT resource has all the necessary tables that International wishes in BASS' 1.0 release and will be flexible enough for any additions needed in the future.

**Issue IV:** Other character sets have different input devices (stroke, different keyboards, etc...).

**Solution:** In the past, International has dealt with input device issues. Further response by International is that

## Appendix A

### *Issues of Concern*

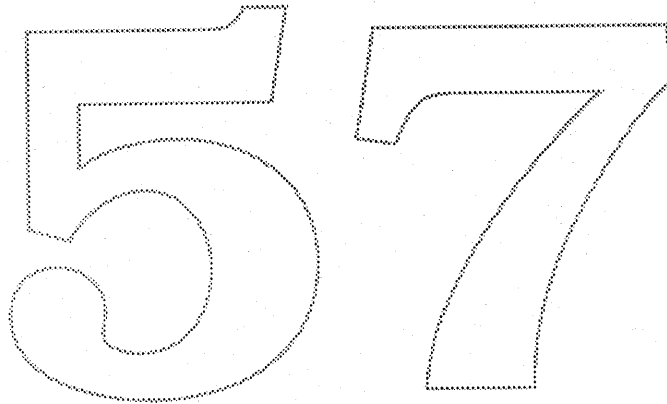
there would be no need for additional support within the BASS' project.

**Issue V:** Asian characters (i.e., Kanji and Hanzi) have a vast number of glyphs. It is close to impossible to construct all glyphs in memory at once due to the low memory constrictions.

**Solution:** Each Bitmap will be cached and de-cached on a per glyph basis (i.e., only BitMaps used will occupy memory).

**Issue VI:** International's SFNT resource consumes large amounts of storage space. How will an International SFNT be transferred from machine to machine and how will it be loaded into memory?

**Solution:** A utility should be written that will break down the SFNT resource into smaller, more manageable and transferable pieces allowing for transfer by floppies. The Resource Manager will be modified to allow for partial resources. Therefore, only the a Look-up Table need to be loaded into memory and not all of the SFNT.



## Appendix B Size and Speed Considerations

### BitMap Space (per char)

	Render(milli)	1/2 Width	Full Width	1 1/2 Width
6 point	5.00	2.25	4.50	6.75
12 point	5.40	9.00	18.00	27.00
24 point	6.25	36.00	72.00	108.00
36 point	7.00	81.00	162.00	243.00
48 point	7.21	144.00	288.00	432.00
96 point	11.00	576.00	1152.00	1728.00
128 point	13.12	1024.00	2048.00	3072.00
256 point	26.25	4096.00	8192.00	12,288.00

The above figures only represent relative comparison values and hold no bearing on actual final size and speed.

Generation: Spline Format -> Grid Fitted spline -> BitMap  
25% 75%

Size of BitMap Resources presently:

Courier (9,10,12,14,18,24) -> 26,322  
Geneva (9,10,12,14,18,24) -> 55,630  
Helvetica (9,10,12,14,18,20,24) -> 33,180  
New York (9,10,12,14,18,24) -> 27,298  
Zapf Chancery (10,12,14,28,24) -> 55,026  
10 -> 3,650  
Micro 36 -> 20,458  
Times 96 -> 154k  
Times 400 -> 3700k

Quantums: 30 mil seek, 8 mil rotational latency, read 750k/sec

Rodime: 85 mil seek, 16 mil rotational latency, read 640k/sec

## Appendix C *Questions For Developers*

**Effects:** The font record may be accessed by certain applications, e.g., Altsys' Font Editor and Microsoft's Word. The system interface has a completely different font resource (i.e., the SFNT resource) that describes the Splines Format. If applications that access the font record break, since there are no representation of the FONT or NFNT, then the system interfaces has two choices. The first is to check if a partial FONT or NFNT resource could be created to give the applications enough information to continue running (e.g., if the applications only need the offset/width table, the system interface could perhaps create a fake offset width table. This is an option, but may cause more damage later, if the application developers wish to rely on this. The second option would be for the application developers to revise their application with more intelligence on what kind of fonts exists. They could then take advantage of the new SFNT resource. The following three questions may be asked of the developers to see how many are doing such things (the system interface has only heard of two: Altsys' Font Editor and Microsoft's Word).

- I. Who accesses the font record directly?
  - II. Who looks at the font record via FMswapFont?
  - III. Who looks at the font record via FontMetrics?
- Note: If so, what do they look at???

**Effects:** The following are natural functions and should not be effected. Although, it would be nice if these were noted for later debugging purposes.

- IV. Who calls FMswapFont?
- V. Who changes information in the FMOut record?
- VI. Who looks at or changes the width table (via FontMetrics or not)?
- VII. Who uses SetFractEnable?
- VIII. Who uses SetFScaleDisable?
- IX. Who uses Kerning tables?
- X. Who generates PostScript?
- XII. Who supports gray scale?
- XIII. Who supports color?

**Effects:** Developers should not do these three things unless it is well documented. If so, they should probably proceed with caution. If certain cases are not documented, they should be forewarned and perhaps should revise their software immediately.

- XI. Who bypasses StdText?
- XIV. Who jams low memory globals?

***Normal Humans Do Not Call the Font Manager Directly... Jerome Coonen.***

Appendix D

sfnt Resource Description

Currently the Macintosh system contains two font resources, the 'FONT' and 'NFNT'. These resources were constructed to house bitmap fonts. A new resource is introduced for spline font glyphs. It is called the 'sfnt' resource. The structure is defined as follows...

Helpful Definitions:

- U: Unsigned    S: Signed
- UPEM: Units per Em
- STZ: Set Length and Offset to Zero
  - Long: 4 bytes
  - Short: 2 bytes
- ULong: Unsigned Long
- UShort: Unsigned Short

SENT Resource Tables:

Please refer to figure (1) for the following description. The first section of the 'sfnt' contains a directory to the various Spline font tables. The first entry is the exact number of tables currently in the 'sfnt'. Each table will contain 1 long for the offset and 1 long for the length of the table. If a table does not exist, both longs should be set to zero. Each offset is measured from the beginning of the 'sfnt' file. Some tables are currently set to zero and will be defined in future documentation. Make sure each table is long aligned.

Header. Maximum Profile:

Refer to the following pages...

Control Value Table:

The Control Value Table stores values that may be used to control stems, rounds, counters, serifs, etc., throughout the font. For further details see the "Font Interpreter and Scan Converter" specifications.

Pre-Program Table:

The Pre-Program is a base of interpreter instructions that modify the Control Value Table within the spline font. It may also change the global graphics state, and the storage area. Note: It may never access parts in character element 1 or higher since it does not exist at that time. The Pre-Program will be called every time there is a font or size change.

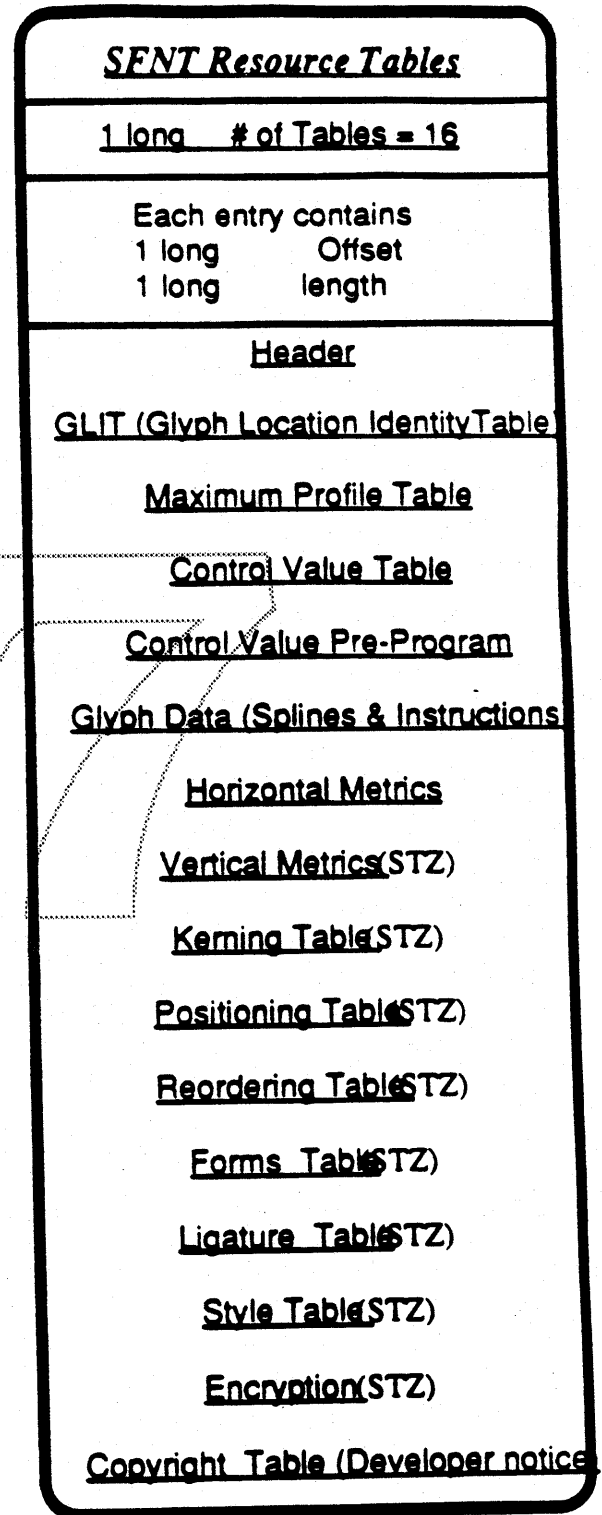


Figure (1)

**Glyph Data Table:**

This section contains the Quadratic B-Splines and Font Interpreter Instructions for each glyph. Note: The glyph data is in packed form. For further details see the section on unpacking and packing of glyph data.

**Horizontal Metrics Table:**

Refer to the following pages...

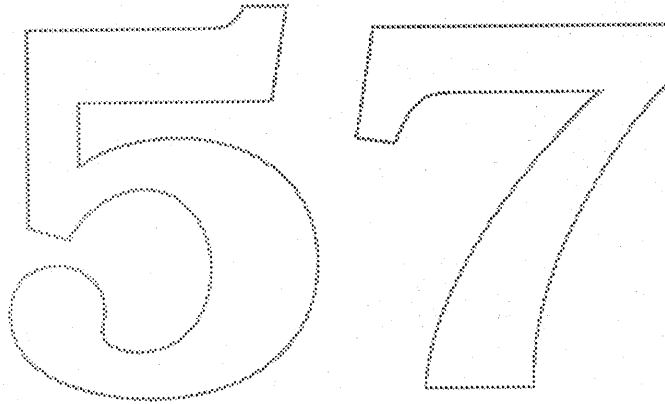
**Vertical Metrics, Kerning, Positioning, Reordering, Forms, Ligature, Style,**

**Encryption Tables:**

Currently these are not well defined although will be in future documentation. Set each tables to zero in both the offset and length fields.

**Copyright (Developer notice) Table:**

Font Manufacturers may place their credits and any other pertinent information they wish such as copyright and author. This is treated as Ascii data.



Header Table

'sfnt' version Major.Minor:

This is the current 'sfnt' data structure definition. Refer to the figure (2) for the correct version setting.

Font Revision # Major.Minor:

The Font Manufacturers version of the font.

Checksum Adjustment:

1) Make sure 'sfnt' is long aligned. 2) Add up all the longs in the sfnt. 3) Calculate a Checksum Adjustment value so that the addition of the unsigned longs plus itself equals hex # 0xB1B0AFBA.

First and Last Glyph:

The first and last glyph id.

Font style:

Use Macintosh style definitions.

Units Per Em:

The Font Manufactures typographical workspace size.

Maximum Advance Width:

The largest advance width metric.

Ascender and Descender:

Refer to the next pages...

Line Gap:

Line Gap = UPEM - (Acender - Descender).

Max and Min Y value:

Refer to the next pages in this document...

Min Left & Right Side Bearing:

Set this to the minimum value for both the Left and Right side Bearing...

<u>Header</u>	
1 Long	'sfnt' version Major.Minor 0.75
1 Long	Font Revision # Major.Minor (U).(U)
1 Long	Checksum Adjustment
1 UShort	FirstGlyph
1 UShort	LastGlyph
1 ULong	Font style (Macintosh Font Face)
1 UShort	Units Per Em
1 UShort	Max. Advance Width (Units)
1 Short	Ascender (Units)
1 Short	Descender (Units)
1 Short	Line Gap (Units)
1 Short	Max. Y value off baseline (Units)
1 Short	Min. Y value off baseline (Units)
1 Short	Min. Left Side Bearing (Units)
1 Short	Min. Right Side Bearing (Units)
1 UShort	Max. InterChar Expansion (Units)
1 UShort	Max. InterChar Compression (Units)
1 UShort	SpaceChar Max. (Units)
1 UShort	SpaceChar Min. (Units)
5 Longs	Metric Scaling - Quadratic
1 Fixed	each (a,b,c,d,limit) $x = \text{PointSize} (ax^3 + bx^2 + cx + d)$
1 Short	Flags
0 bit	off=Short GLIT, on=Long GLIT
1 bit	on=Horizontal Metrics Same
2 bit	on=Vertical Metrics Same
3 bit	on=Dont cache bitmaps

Max InterChar Expansion and Compression:

These are Font Manufactures recommendations greatest expansion and compression of the inter character spacing.

SpaceChar Max and Min:

The Font Manufactures Maximum and Minimum recommended space character.

Figure (2)

Metric Scaling:

Refer to the next pages in this document for description and figures...

Flags:

For a sparse GLIT set bit 0 on and use the long format. If the Horizontal or Vertical Metrics are the same, set bit 1 or 2 on, respectively. To turn off the bitmap cache set bit 3 on (e.g., Font Editors).



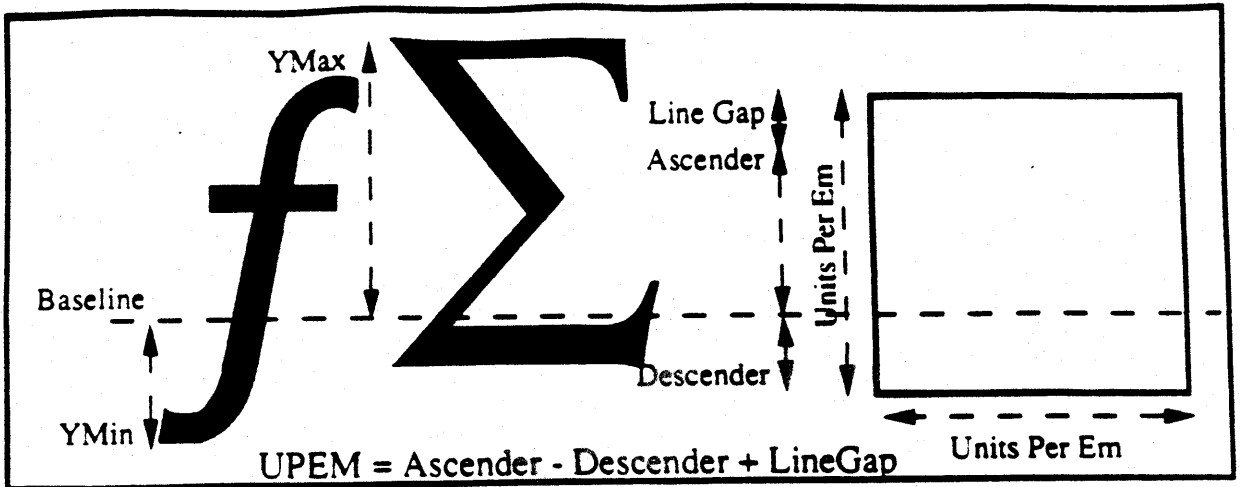


Figure (3)

Ascender, Descender, Line Gap, Max & Min Y

Ascender, Descender & Line Gap:

Refer to figure (3). Ascender and Descender are measured from the baseline.  $UPEM = (Ascender - Descender) + LineGap$ .

Max and Min Y value:

The highest and lowest outline values off the baseline. Refer to figure (3).

Metric Scaling Cubic Formula:

To achieve linear scaling set the a, b, c and d values that are depicted in figure (4). Non-Linear scaling will change one or more of the linear scaling values.

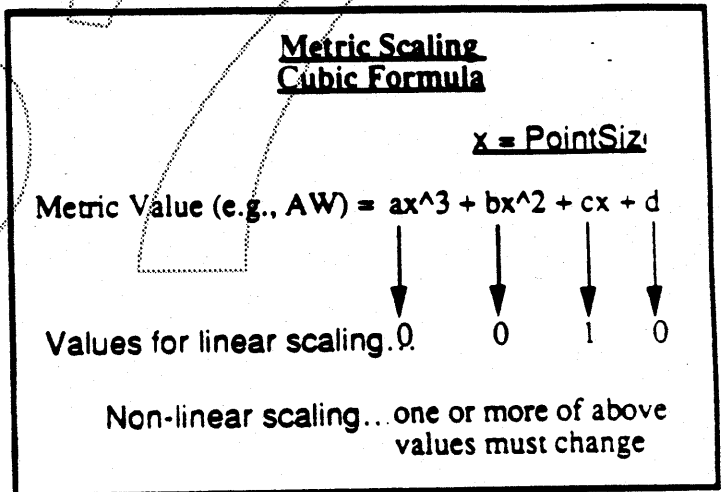
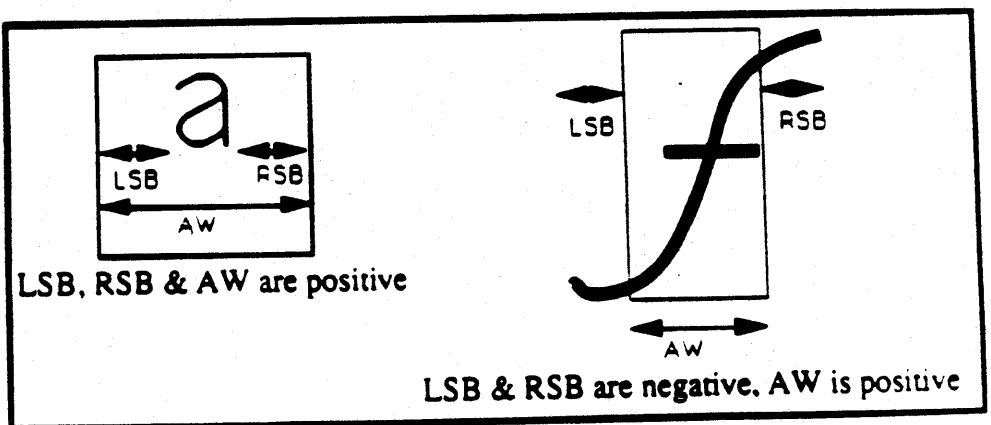


Figure (4)

Minimum Left and Right Side Bearing

Notice the left glyph contains a positive LSB & RSB, the right most glyph contains both negative values for the left and right side bearings.



Glyph Location Identity Table

The Glyph Location Identity Table is a directory to the glyph data. It comes in two flavors: The long format contains the glyph id, format length and a offset to the data. This is commonly used when glyphs are sparse. The short format, which contains only a spline offset, is used when the glyphs are not sparse. **Note:** When using the short format it is necessary to have one extra offset at the end of the GLIT entries. This enables the last entry length to be calculated.

<u>Glyph Location Identity Table</u>	
<u>Long Format</u>	
1 UShort	GlyphCode
1 UShort	Format Lengt
1 ULong	Spline Offset
<u>Short Format</u>	
1 ULong	Spline Offset
...repeat GLIT entries for each Glyph	

Figure (6)

Maximum Profile Table

The Maximum Profile Table aids the system to allocate memory efficiently.

Set the number of points, contours, elements, points in element zero, storage location, function definitions, stack elements and size of instructions to the maximum values the font will need.

<u>Maximum Profile Table</u>	
1 UShort	Maximum # of Points
1 UShort	Maximum # of Contours
1 UShort	Maximum # of Elements
1 UShort	Maximum # of Points in Element Zero
1 UShort	Maximum # of Storage Locations
1 UShort	Maximum # of Function Definitions
1 UShort	Maximum # of Stack Elements
1 UShort	Maximum Size (# of bytes) of Instructio

Figure (7)

Horizontal Metrics Table

The horizontal metrics contain an Advance Width, Left Side Bearing and Right Side Bearing. Refer to figure (5) for Left and Right Side Bearings. Notice that the Advance width will never be negative.

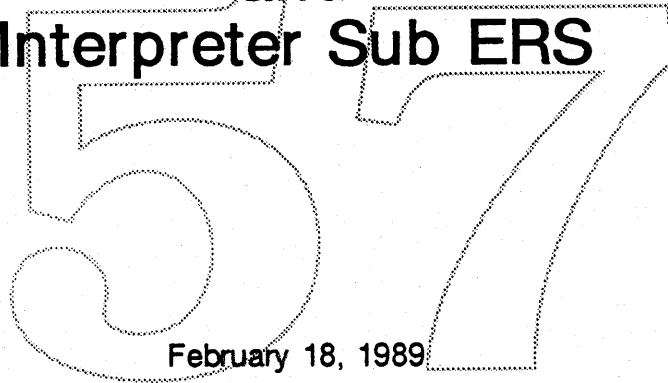
<u>Horizontal Metrics</u>	
1 UShort	Advance Width
1 Short	Left Side Bearing
1 Short	Right Side Bearing

Figure (8)

57

# The Bass Project

## Font Instruction Set and Interpreter Sub ERS



Sampo Kaasila  
MS: 27-AJ; Extension 45505  
Product Development  
Apple Computer, Inc.

# Contents

- 1.0 Introduction
- 2.0 Description
- 3.0 Graphics State
  - 3.1 Default Settings for the Graphics State
  - 3.2 Compensation for Engine Characteristics
- 4.0 Implementation Limitations
- 5.0 Nomenclature for the Instruction Description
  - 5.1 Instructions for Setting the Freedom and Projection Vectors
  - 5.2 Instructions for Setting Internal Points and Character Elements
  - 5.3 Instructions for Modifying Internal Settings
  - 5.4 Stack Manipulating Instructions
  - 5.5 Interpolation and Shift Instructions
  - 5.6 Instructions for Moving Points
  - 5.7 Instructions for Reading and Writing Data
  - 5.8 Relational Instructions and Instructions for Selection "IF-statements"
  - 5.9 Arithmetic and Math Instructions
  - 5.10 Short Push Instructions
  - 5.11 Function Calls
  - 5.12 Delta Exceptions
  - 5.13 Instructions for Reading and Writing Metrics
  - 5.14 Miscellaneous Instructions
- 6.0 Interface

# Font instructions and the Interpreter

## 1.0 Introduction

The interpreter is an internal module to the outline font package and not directly callable by an application. The outline font package will be an integral part of the Macintosh system software. The interpreter accomplishes the elimination of various artifacts appearing on raster devices when scan converting outline based characters. These artifacts include pixel dropouts, unpleasing curve shapes (widowed pixels/flat curves), uneven stem weights, bad weight progressions,.....etc. This is the most important and complex task in font scaling.

## 2.0 Description

The way this control is accomplished is through changing the shape of the character outlines in a size dependent way. This takes place before the scan conversion.

There is a little program attached to every character. The program consists of a sequence of font instructions. In these instructions there are primitives for doing various things like shifting points, interpolating points, grid-aligning points, etc. The interpreter will process these font instructions. The instruction based approach allows for an arbitrary sequencing of the primitives. This is the only reasonable approach since we want to allow translation into this format from various other vendor formats.

There is also space in the outline font data structures for a preprogram which is executed whenever a new font or a new pointsize is requested. This takes effect before any of the characters are requested. This preprogram will typically set up some values in a global storage area.

The instructions are byte encoded and stack based.

The input to the interpreter consists of the points that make up the character and information describing the start and end points of the contours. In the input there are also font instructions and a control value table. The control value table contains data that can be accessed by the font instructions. It is expected that it will typically contain values for different heights, stem-widths, serif-widths, sizes of rounds, etc. Simply stated the interpreter reads a stream of byte encoded instructions in the context of a graphics state while also modifying the graphics state.

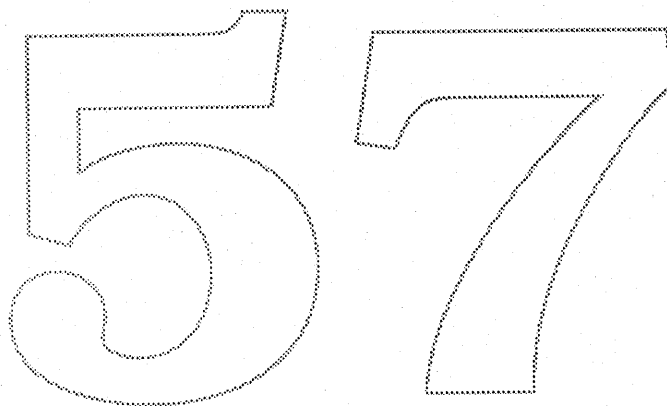
A point is always referenced with a point number and the setting of a character element pointer. The character being processed is normally expected to be character element 1. Character element 0 is reserved for situations when it is advantageous to move a point that does not exist on the character and later use this point as a reference when moving a point that does exist on the character. For instance a possible usage is for the uppercase "A" when determining the position of the top. It is normally positioned slightly higher than the top of the "B". So, a point in character element 0 could be

positioned as if were located on the same height as the top of the "B" and after this the top of the "A" could be shifted by the amount this point moved when forced to conform to the grid. Character element zero is simply a temporary point storage.

The output consists of all the points that describe the character.

One very important concept is the freedom and the projection vectors. Whenever a point has to move it only moves along the freedom vector. Also, all distances are measured along the projection vector. This allows for control of measurements not aligned with the x and y axis e.g. in diagonals and italic characters.

The interpreter is a a byte code interpreter. In the implementation all the instructions (bytes) go through a jump table when they are executed. This means that the instructions are patchable on an individual basis.



## 3.0 Graphics State

The Interpreter has a graphics state which defines the context the font instructions operate in. The graphics state can be divided into a local and a global graphics state. The local graphics state does not have any inter character memory, so it is fresh for every character. The global graphics state has inter character memory and also stays in effect between the preprogram and the character. Note that it is highly illegal to have a font program attached to a character that modifies the global graphics state in such a way that an other character in the same font is affected !

The local graphics state is composed of the following:

- 01) Three internal point numbers: Pt0, Pt1, Pt2.  
These are used by some instructions.
- 02) Three internal character element pointers: CE0, CE1, CE2.  
These are used by some instructions.
- 03) A state variable called: Round\_To\_Grid.  
This variable impacts: MDAP(), MIAP(), MDRP(), MIRP().  
In state 0 distances are rounded to the closest half ( $n + 1/2$ ) pixel.  
In state 1 distances are rounded to the closest integer ( $n$ ) pixel.  
In state 2 distances are rounded to the closest half or integer ( $n/2$ ) pixel.  
In state 3 distances are rounded down to the closest integer ( $n$ ) pixel.  
In state 4 distances are rounded up to the closest integer ( $n$ ) pixel.
- 04) A minimum distance variable: Minimum\_Distance.  
Some instructions will keep distances greater or equal to this value even if the distance naturally would round to something smaller. Typically this will be set to one pixel so that everything that would round to zero will be maintained at one pixel.
- 05) Two vectors: Projection\_Vector, Freedom\_Vector.  
Whenever a distance is measured it is first projected onto the projection vector and then measured. Similarly, whenever a point has to move it will only move along the freedom vector.
- 06) A stack pointer.
- 07) A loop variable: loop.  
Some instructions like the shift instructions can repeat themselves many times, all depending on the value of this variable.
- 08) An instruction pointer. It is used by the interpreter to walk through the instructions.
- 09) A pointer to the character element list. This is what we use to access the character.  
The points describing the character in each character element. Character element 1 normally contains the character and character element 0 is reserved for internal use.
- 10) A pointer to the global graphics state. This is how we find the global graphics state.

The global graphics state is composed of the following:

- 01) A storage area.
- 02) A function definition area.
- 03) A pointer to the preprogram.
- 04) The Control Value Table.  
This is used mainly by the Move Indirect Relative Point instruction (MIRP()).
- 05) A Control Value Table Cut In Value.  
This value determines when the interpreter should use the control value table and when it should use the value directly from the outline of the character. When the difference between the value in



the table and the measurement directly from the outline is greater than the cut in value then the outline will be used. This only applies to the MIRP() and MIAP() instructions (& only if they are rounding). The justification for this is that it allows for regularizing the font at smaller sizes but at larger sizes the true unregularized design will take over.

- 06) A Single Width Cut In Value  
This determines when the single width should be used instead of for instance the width table value. When the difference between the single width value and the real value is smaller than this cut in value then the single width will be used. This applies to the MIRP() and MDRP() instruction.
- 07) A Single Width value.  
This can be used at small sizes to force all distances to be the same, e.g. rounds and stems.
- 08) The Auto\_Flip Boolean. If this is set to true then the interpreter will change the sign of the read value from the Control Value Table if it is of a different sign than the actual measurement from the character. This only affects the MIRP() instruction.
- 09) The Delta Base value. It is used by the Delta instruction.
- 10) The Delta Shift value. It is used by the Delta instruction.
- 11) The Angle Weight. Is is used by the Adjust Angle "AA" instruction.

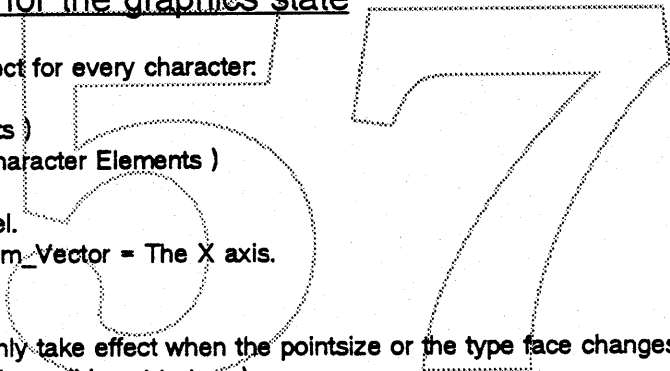
### 3.1 Default settings for the graphics state

These settings will take effect for every character:

Pt0 = Pt1 = Pt2 = 0. ( Points )  
CE0 = CE1 = CE2 = 1. ( Character Elements )  
Round\_To\_Grid = TRUE.  
Minimum\_Distance = 1 pixel.  
Projection\_Vector = Freedom\_Vector = The X axis.  
loop variable = 0.

These settings below will only take effect when the pointsize or the type face changes:  
(whenever "init" in the function call is set to true )

Control Value table cut in = 17/16 pixels.  
Single Width Cut in = 0 pixels = same as disabled  
Single value = undefined.  
Auto\_Flip = TRUE  
deltaBase = 9;  
deltaShift = 3;  
angleWeight = 128; /\* this is likely to change \*/



## 3.2 Compensation for Engine Characteristics

The instructions allow distances to be categorized into 4 categories. Distances in category 0 (gray) will always be rounded the normal way, but for the other categories we can add a compensation term before the rounding. For example assume that the engine makes black distances about 0.8 pixels too heavy and white distances about 0.8 pixels too narrow. In this case the black compensation would be -0.8 pixels and the white compensation would be +0.8 pixels.

category 0 = gray distances (00)  
category 1 = black distances (01)  
category 2 = white distances (10)  
category 3 = reserved for future use (11)

## 4.0 Implementation limitations

If the system running this code does not have enough memory then of course we can not process the font, but otherwise there should not be any serious limitations. The memory requirements are specified by the `sint` (font file) and it contains information like the amount of storage area, maximum size of stack, maximum number of points in a glyph, etc... .  
The internal data format is 26.6 bits. This means that the size of a pixel is 64 units.

## 5.0 Nomenclature for the Instruction Description

An instruction is described by a sequence of characters followed by an "(", followed by zero or more Boolean arguments, followed by ")". The `stack` field describes what happens to the stack. The left side of "`=>`" is before the operator takes effect and the right side is after. The `code` field defines the byte encoding of the instruction. In case there are arguments to the instruction within the "(" then the correct code is the lowest code shown in the range for the instruction plus the argument in binary form where the leftmost bit is the most significant.

example:

```
stack pt1, pt2 => -
```

In this case the instruction takes two arguments and `pt2` is at the top of the stack.

*Disclaimer: This is not the final instruction set. Instructions may be modified, deleted or added to the list.*

## 5.1 Instructions for Setting the Freedom and Projection Vectors.

### SVTCA(a)

Set Vectors To Coordinate Axis

a: X or Y Axis ?

stack: no change.

code: 0x00 - 0x01

### SPVTCA(a)

Set Projection Vector To Coordinate Axis

a: X or Y Axis ?

stack: no change.

code: 0x02 - 0x03

### SFVTCA(a)

Set Freedom Vector To Coordinate Axis

a: X or Y Axis ?

stack: no change.

code: 0x04 - 0x05

### SPVTL(a)

Set Projection Vector To Line

a: Rotated by 90 degrees clockwise?

stack: pt1, pt2 => -

code: 0x06 - 0x07

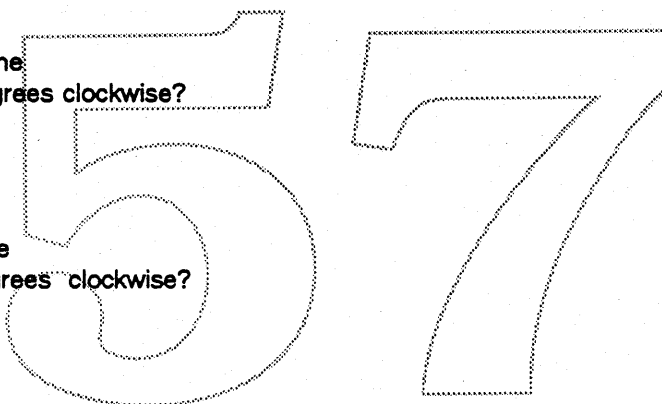
### SFVTL(a)

Set Freedom Vector To Line

a: Rotated by 90 degrees clockwise?

stack: pt1, pt2 => -

code: 0x08 - 0x09



### WPV()

Write Projection Vector ( y is calculated from x automatically )

stack: x => -

code: 0x0A

A length of one is represented by 0x4000 (hex).

### WFOV()

Write Freedom Vector ( y is calculated from x automatically )

stack: x => -

code: 0x0B

A length of one is represented by 0x4000 (hex).

### RPV()

Read Projection Vector

stack: - => x

code: 0x0C

A length of one is represented by 0x4000 (hex).

### RFV()

Read Freedom Vector

stack: - => x

code: 0x0D

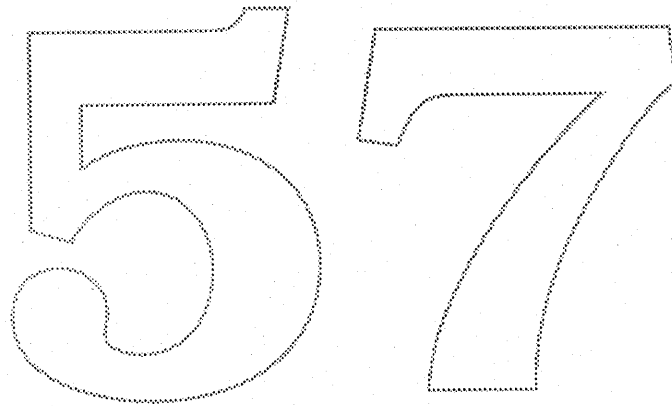
A length of one is represented by 0x4000 (hex).

SFVTPV()

Set Freedom Vector To Projection Vector

stack: no change.

code: 0x0E



## 5.2 Instructions for Setting Internal Points and Character Elements

SRP0()

Set Reference Point 0  
stack: n => -  
code: 0x10

SRP1()

Set Reference Point 1  
stack: n => -  
code: 0x11

SRP2()

Set Reference Point 2  
stack: n => -  
code: 0x12

SCE0()

Set Character Element 0  
stack: n => -  
code: 0x13

SCE1()

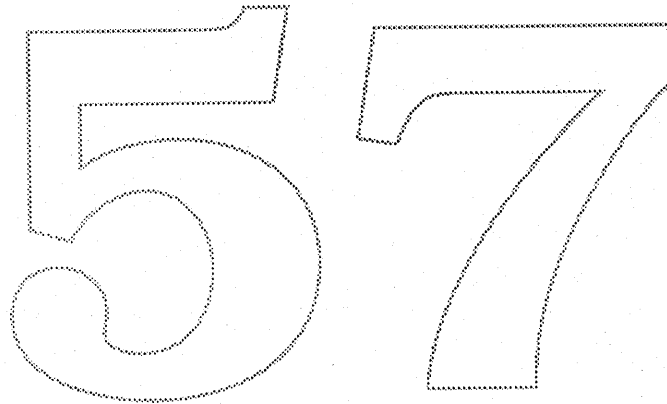
Set Character Element 1  
stack: n => -  
code: 0x14

SCE2()

Set Character Element 2  
stack: n => -  
code: 0x15

SCES()

Set Character Elements  
stack: n => -  
code: 0x16



## 5.3 Instructions for Modifying Internal Settings

### LLOOP()

Load LOOP variable

stack: n => -

code: 0x17

### RTG()

Round To Grid

stack: no change.

code: 0x18

Round\_To\_Grid = 1;

### RDTG()

Round Down To Grid

stack: no change.

code: 0x7d

Round\_To\_Grid = 3;

### RUTG()

Round Up To Grid

stack: no change.

code: 0x7c

Round\_To\_Grid = 4;

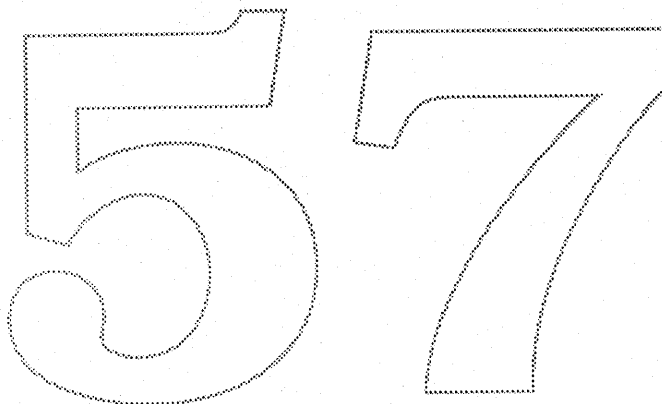
### RTHG()

Round To Half Grid

stack: no change.

code: 0x19

Round\_To\_Grid = 0;



### RTDG()

Round To Double Grid

stack: no change.

code: 0x3d

Round\_To\_Grid = 2;

### LMD()

Load Min Distance,

stack: distance (expressed in fractional pixels) => -

code: 0x1A

### LCVTCI()

Load Control ValueTable Cut In

stack: n => - ( n in fractional pixels )

code: 0x1D

### LSWCI()

Load Single Width Cut In

stack: n => - ( n in fractional pixels )

code: 0x1E

### LSW()

Load Single Width

stack: n => - ( n in the same units as the character )

code: 0x1F

FLIPON()

Set the Auto\_Flip Boolean to TRUE.

stack: no change

code: 0x4D

FLIPOFF()

Set the Auto\_Flip Boolean to FALSE.

stack: no change

code: 0x4E

SANGW()

Set Angle Weight

stack: weight => -

code: 0x7E

This instruction sets the Angle Weight in the global graphics state. The weight is used by the AA instruction.

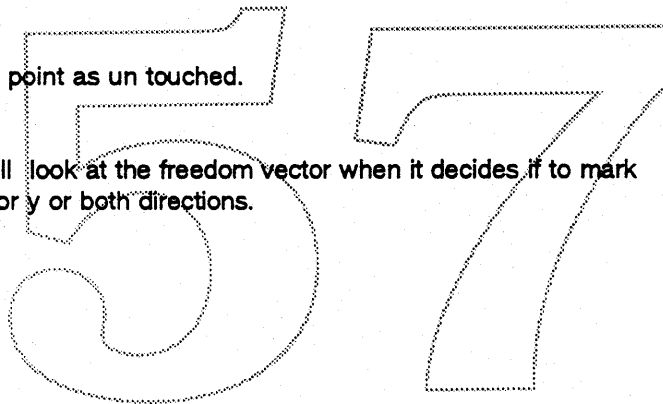
UTP()

Un Touch Point. Marks the point as un touched.

stack: pt # => -

code: 0x29

Note that this instruction will look at the freedom vector when it decides if to mark the point as untouced in x or y or both directions.



## 5.4 Stack Manipulating Instructions

### DUP()

DUPlicate top stack element

stack: n => n, n

code: 0x20

### POP()

POP top stack element

stack: n => -

code: 0x21

### CLEAR()

CLEAR the entire stack

stack: ... => empty stack

code: 0x22

### SWAP()

SWAP top two elements

stack: n1, n2 => n2, n1

code: 0x23

### DEPTH()

Returns the DEPTH of the stack

stack: - => n

code: 0x24

### CINDEX()

Copy the INDEXed element to the top of the stack

stack: nk, ..., n2, n1, k => nk, ..., n2, n1, nk

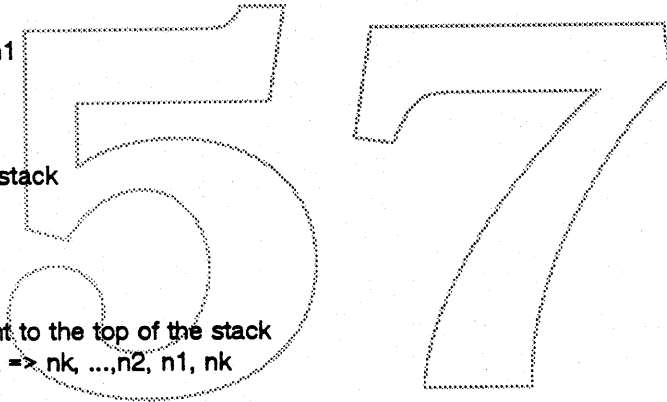
code: 0x25

### MINDEX()

Move the INDEXed element to the top of the stack

stack: nk, ..., n2, n1, k => -, ..., n2, n1, nk

code: 0x26





## 5.5 Interpolation and Shift Instructions.

### IUP(a)

Interpolate untouched points through the outline

a: X or Y  
stack: no change.  
code: 0x30 - 0x31

This instruction uses CE2.

The interpreter will look at the character and find all points that have not been touched and it will also find all point that have been touched. It will walk around every contour of the character and find a pair of consecutive touched points and then all the untouched points on the outline in between will be linearly interpolated. But, points outside the stretching range are shifted by the amount of the closest edge.

### SHP(a)

Shift point by the last pt.

a: point 1 or 2 ? (Determines if the point is to be shifted by the movement of internal point 1 or 2 )  
stack: pt# => -  
code: 0x32 - 0x33

This instruction uses CE0 with Pt1 or CE1 with pt2 and CE2 with pt#.

It uses the loop variable.

### SHC(a)

Shift contour by the last pt

a: point 1 or 2 ?  
stack: ctr# => -  
code: 0x34 - 0x35

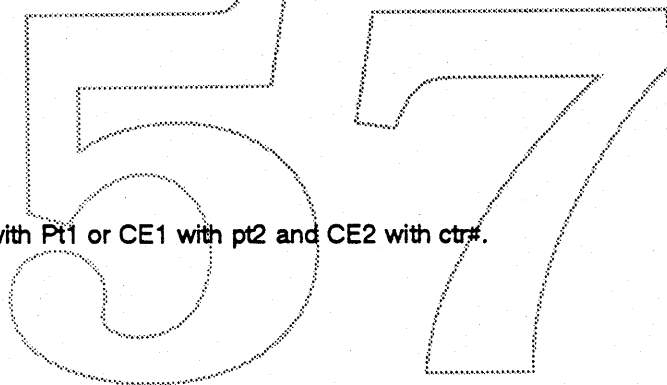
This instruction uses CE0 with Pt1 or CE1 with pt2 and CE2 with ctr#.

### SHE(a)

Shift element by the last pt

a: point 1 or 2 ?  
stack: el# => -  
code: 0x36 - 0x37

This instruction uses CE0 with Pt1 or CE1 with pt2 and CE2 with el#.



### SHPIX()

SHift point by a PIXEL amount

stack: pt, amount => -  
code: 0x38

This instruction uses CE2.

It uses the loop variable. In case the loop variable is used then the amount is popped only once, so say loop = 3, then the stack should be as follows: pt0, pt1, pt2, amount => -

### IP()

Interpolate point by the last relative stretch

stack: pt# => -  
code: 0x39

This instruction uses CE0 with Pt1 and CE1 with pt2 and CE2 with pt#.

No matter where the point is it is stretched according to Pt1 and Pt2.

It uses the loop variable.

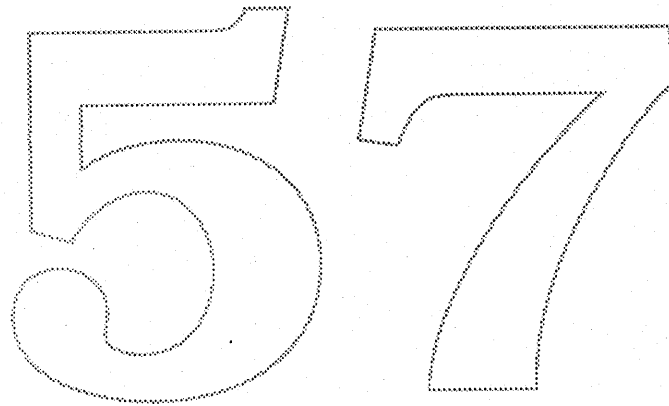
### ISECT()

InterSECT, puts a point at the intersection of two lines

stack: pt#, ptA0, ptA1, ptB0, ptB1 => -

code: 0x0f

line-A is defined by ptA0 and ptA1, similarly ptB0 and ptB1 defines line-B.  
This instruction uses CE2 with Pt# and CE1 with line-A and CE0 with line-B.



## 5.6 Instructions for Moving Points

### MSIRP(a)

Move Stack Indirect Relative Point

a: Move or don't move reference point to pt # ?

stack: pt#, distance => -

code: 0x3A - 0x3B

This instruction uses CE1 with pt# and CE0 with pt0.

The distance on the stack is in fractional pixels. The distance between pt0 and pt# will be forced to the value on the stack.

### ALIGNRP()

ALIGN Relative Point

stack: pt# => -

code: 0x3C

This instruction uses CE1 with pt# and CE0 with pt0.

This instruction will force the distance between pt0 and pt# to be zero.

### AA()

Adjust Angle

stack: pt# => -

code: 0x7F

This instruction uses CE1 with pt# and CE0 with pt0.

The instruction might move pt# along the freedom vector to a location that causes the line from pt0 to pt# to be at a good angle. A good angle is an angle with a small pixel pattern repetition period. The instruction will find an optimal angle, and if none is found then the point will not be moved, just touched. The instruction will consider 152 different distinct angles. These are all the angles that have a repetition period of less than 9 pixels. The instruction will not change an angle more than about +- 10 - 12 degrees. Within this range it will search for an angle that has the smallest penalty. The penalty also has to be less than 10 pixels. The penalty is defined as:

The repetition period + The line pt0 to pt# projected into a unit vector at an angle orthogonal to the angle being examined \* angleWeight / ( 64 = pixel size ) [pixels].

This means that at a small size the instruction is more likely to change the angle a lot and at a sufficiently high size the angle will not change at all.

High values for angleWeight => The angle will not change much.

Low values for angleWeight => The angle will change a lot.

### ALIGNPTS()

ALIGN Relative Point

stack: pt#1, pt#2 => -

code: 0x27

This instruction uses CE0 with pt#1 and CE1 with pt#2.

This instruction will force the distance between pt0 and pt# to be zero by moving both along the freedom vector to the average of both their projections in to the projection vector.

### MDAP(a)

Move Direct Absolute Point

a: Round or don't round the value ?

stack: pt# => -

code: 0x2E - 0x2F

This instruction uses CE0 and sets Pt0 = Pt1 = pt #.

This instruction will move the point to the closest grid point. The instruction looks at the boolean Round\_To\_Grid. If the boolean argument "a" is set to zero then the only effect will be that the point "pt#" )

is marked as a touched point.

#### MIAP(a)

Move Indirect Absolute Point

a: Round or don't round the value ?

stack: pt#, distance id# => -

code: 0x3E - 0x3F

This instruction uses CE0 and sets Pt0 = Pt1 = pt #.

The instruction looks at the boolean Round\_To\_Grid. The instruction will look up the value in the control value table indexed by "distance id#" and then move the point to that value.

#### MDRP(abcde)

Move Direct Relative Point

a: Move or don't move reference point to pt # ?

b: Keep distance >= min distance ?

c: Round or don't round the distance ?

d: distance type

e: distance type

stack: pt# => -

code: 0xC0 - 0xDF

This instruction uses CE0 with Pt0 and CE1 with pt#. After it has moved the point this instruction sets pt1 = pt0, pt2 = pt#, and maybe pt0 = pt#.

The instruction looks at the boolean Round\_To\_Grid.

This instruction will move "pt#" so that the distance between pt0 and pt# is what it originally was or that distance rounded.

#### MIRP(abcde)

Move Indirect Relative Point, pt #, distance id #

a: Move or don't move reference point to pt # ?

b: Keep distance >= min distance ?

c: Round or don't round the distance ?

d: distance type

e: distance type

stack: pt#, distance id# => -

code: 0xE0 - 0xFF

This instruction uses CE0 with Pt0 and CE1 with pt#. After it has moved the point this instruction sets pt1 = pt0, pt2 = pt#, and maybe pt0 = pt#.

The instruction looks at the boolean Round\_To\_Grid.

Same as the previous instruction except that the distance is not measured from the original outline. Instead it is taken from the control value table.

## 5.7 Instructions for Reading and Writing Data

NPUSHB(), n, b1, b2,...bn

PUSH N bytes

stack: - => b1,b2...bn

code: 0x40

NPUSHW(), n, w1, w2,...wn

PUSH N words

stack: - => w1,w2...wn

code: 0x41

WS()

Write Store

stack: address, value => -

code: 0x42

RS()

Read Store

stack: address => value

code: 0x43

WCVT()

Write Control Value Table

stack: address, value => -

code: 0x44

RCVT()

Read Control Value Table

stack: address => value

code: 0x45

RC(a)

Read Coordinate ( Reads coordinate projected into the projection vector)

a: old or new coordinates ?

stack: pt# => value

code: 0x46-0x47

The instruction uses CE2.

WC()

Write Coordinate (Writes coordinate using projection and freedom vectors)

stack: pt#, value => -

code: 0x48

The instruction uses CE2.

MD(a)

Measure Distance

a: old or new coordinates ?

stack: pt1, pt2 => dist

code: 0x49-0x4A

The instruction uses CE0 with pt1 and CE1 with pt2.

**MPPEM()**

**Measure Pixels Per EM**

stack: - => ppem

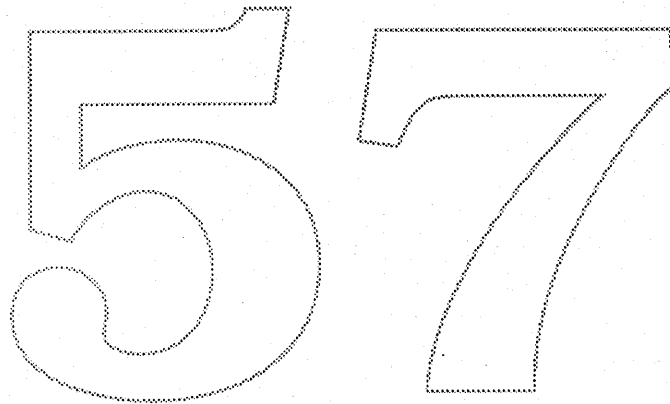
code: 0x4B

**MPS()**

**Measure Point Size**

stack: - => PointSize

code: 0x4C



## 5.8 Relational and Logical Instructions and Instructions for Selection "IF-statements"

LT()

Less Than,  $n1 < n2$

stack:  $n1, n2 \Rightarrow$  boolean  
code: 0x50

LTEQ()

Less Than or Equal,  $n1 \leq n2$

stack:  $n1, n2 \Rightarrow$  boolean  
code: 0x51

GT()

Greater Than,  $n1 > n2$

stack:  $n1, n2 \Rightarrow$  boolean  
code: 0x52

GTEQ()

Greater Than or Equal,  $n1 \geq n2$

stack:  $n1, n2 \Rightarrow$  boolean  
code: 0x53

EQ()

Equal,  $n1 == n2$

stack:  $n1, n2 \Rightarrow$  boolean  
code: 0x54

NEQ()

Not Equal,  $n1 != n2$

stack:  $n1, n2 \Rightarrow$  boolean  
code: 0x55

ODD()

ODD

stack:  $n1 \Rightarrow$  boolean  
code: 0x56

EVEN()

Even

stack:  $n1 \Rightarrow$  boolean  
code: 0x57

IF()

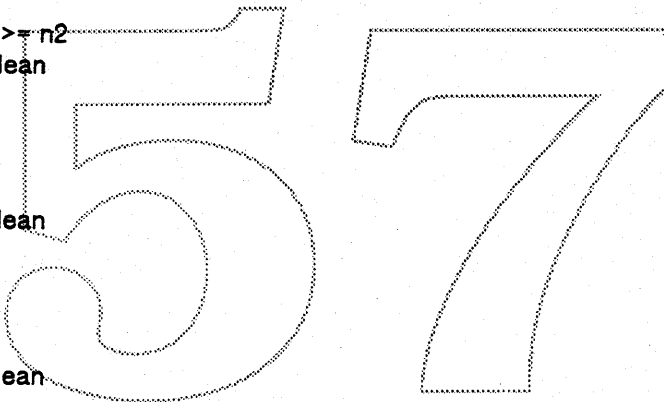
IF

stack: boolean  $\Rightarrow$  -  
code: 0x58

EIF()

End IF

stack: no change  
code: 0x59



AND()  
AND

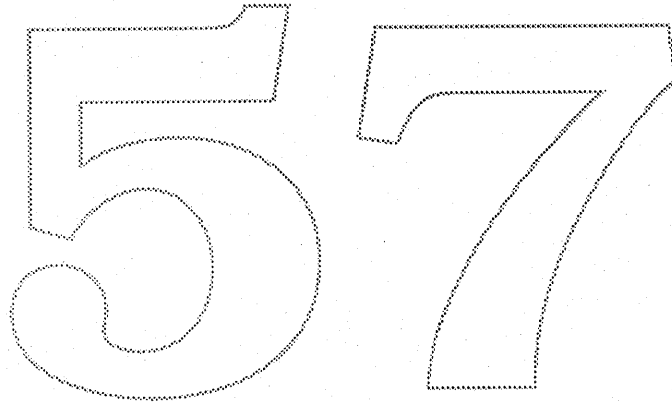
stack: n1, n2 => ( n1 and n2 )  
code: 0x5a

OR()  
OR

stack: n1, n2 => ( n1 or n2 )  
code: 0x5b

NOT()  
NOT

stack: n1 => ( not n1 )  
code: 0x5c





## 5.9 Arithmetic and Math Instructions

ADD()

ADD

stack: n1, n2 => (n1 + n2)  
code: 0x60

SUB()

SUBtract

stack: n1, n2 => (n1 - n2)  
code: 0x61

DIV()

DIVide

stack: n1, n2 => (n1 / n2)  
code: 0x62

MUL()

MULTIply

stack: n1, n2 => (n1 \* n2)  
code: 0x63

ABS()

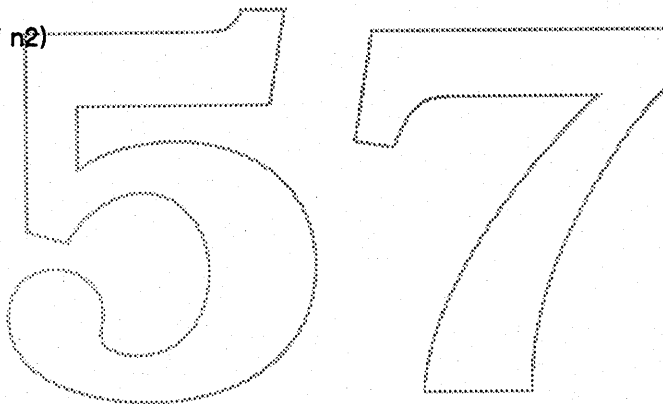
ABSolute value

stack: n1 => n  
code: 0x64

NEG()

NEGate

stack: n1 => -n1  
code: 0x65



FLOOR()

FLOOR

stack: n1 => n ( n is the greatest integer value less than or equal to n1 )  
code: 0x66

CEILING()

CEILING

stack: n1 => n ( n is the least integer value greater or equal to n1 )  
code: 0x67

ROUND(ab)

ROUND value

Rounds the distance to an integer number of pixels while compensating for the engine.

a: distance type

b: distance type

stack: n1 => n

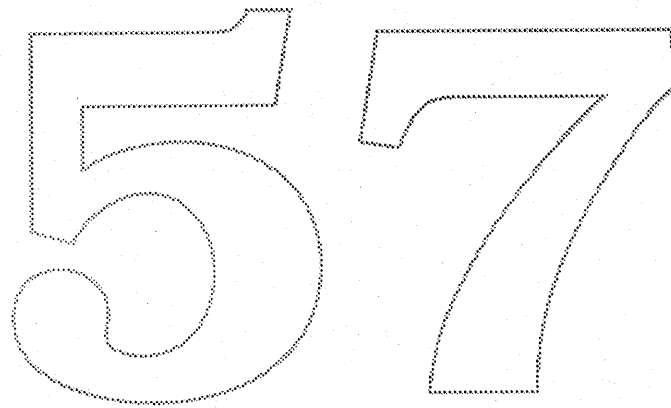
code: 0x68 - 0x6B

NROUND(ab)

No ROUNDing of value

Compensates for the engine.

a: distance type  
b: distance type  
stack: n1 => n  
code: 0x6C - 0x6F



## 5.10 Short Push Instructions

**PUSHB(abc), b0, b1,..bn**

**PUSH Bytes**

**a,b,c:** number of bytes to be pushed - 1.

**stack:** b0, b1, ...,bn

**code:** 0xB0 - 0xB7

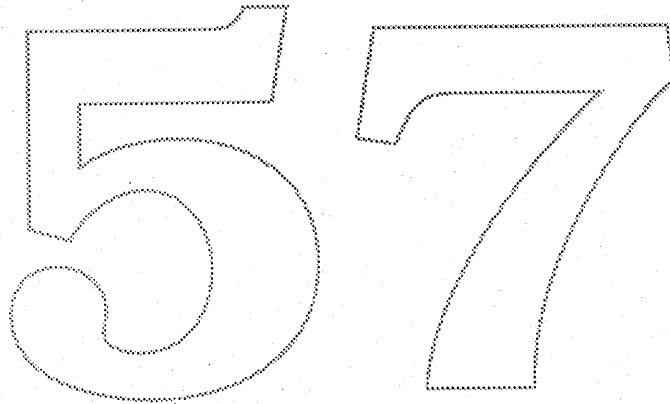
**PUSHW(abc), w0,w1,..wn**

**PUSH Words**

**a,b,c:** number of words to be pushed - 1.

**stack:** - => w0, w1, ...wn

**code:** 0xB8 - 0xBF



## 5.11 Function Calls

FDEF()

Function DEFinition, can only appear in the pre-program

stack: n => -, n is the function number

code: 0x2C

ENDF()

END Function definition, can only appear in the pre-program

stack: -

code: 0x2D

CALL()

CALL function

stack: n => -

code: 0x2B

LOOPCALL()

LOOP and CALL function (calls "function #" "count" times)

stack: count, function # => -

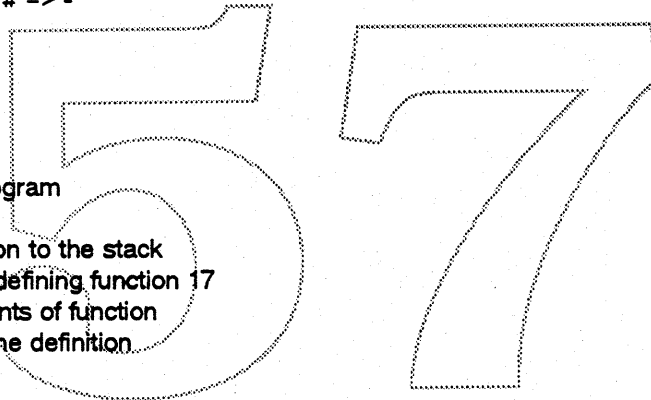
code: 0x2A

A Simple Example:

Assume this is in the pre-program

```
PUSHB(000), 17 ; push 17 on to the stack
FDEF()         ; start defining function 17
MDAP(1)       ; contents of function
ENDF()        ; end the definition
```

Now in a character if we call function 17, it will have the same effect as doing MDAP(1) instead [MDAP(1) = PUSHB(000),17 , CALL() ].



## 5.12 Delta Exceptions

DELTA()

DELTA exceptions

stack: [ arg1, pt# ]\*, n => -  
code: 0x5D

\* repeat n times

This instruction takes a variable number of arguments of the stack and the instruction allows the use of exception of the form: at size x apply the following movement d to point pt#. The instruction is capable of moving more than one point at any particular size.

arg1 is a byte and it is composed of two parts.  $arg1 = (ppem \ll 4) | exception$ .

The high 4 bits describe the pixels per em that will activate the exception. The pixels per em which will be used is  $ppem + deltaBase$  in the global graphics state. So if we want to specify an exception at 12 ppem and the deltaBase is set at 9 then we need to store 3 in the high nibble. The exception is a number between 0 and 15. Internally this is remapped as follows [ 0 -> -8, 1 -> -7,...7 -> -1, 8 -> 1, 9->2,...15->8 ]. Note that zero is lacking in the output range. Now we are going to move the point by the remapped amount in pixels shifted by the delta shift in the global graphics state.

Always observe that this instruction expects the argument list to be sorted according to ppem. The lowest ppem should be leftmost (deepest on the stack) and the highest ppem should be to the right (towards the top of the stack).

Example: assume that deltaBase is 9, and delta Shift is 3 (default), and the top of the stack is stack: 56, 15, 1.

Now if we execute

DELTA()

Then this instruction will move point 15 at 12 ppem  $1/8$  of a pixel along the projection vector.

$((3 \ll 4) | 8 = 56, 3 + 9 = 12 \text{ ppem}, 8 -> 1 => (1 \gg 3) = 1/8 \text{ pixels.})$

Also 56, 15 and 1 will be popped of the stack.

This instruction should be used sparingly since there is a relatively high storage overhead associated with it, but it can come in handy to solve "impossible" cases.

SDB()

Set Delta Base in the global graphics state.

stack: n => -  
code: 0x5E

SDS()

Set Delta Shift in the global graphics state.

stack: n => -  
code: 0x5F

## 5.13 Instructions for Reading and Writing Metrics

RLSB()

Read Left Side Bearing

stack: - => left side bearing  
code: 0x1b

WLSB()

Write Left Side Bearing

stack: left side bearing => -  
code: 0x1c

RAW()

Read Advance Width

stack: - => advance width  
code: 0x28

## 5.14 Miscellaneous Instructions

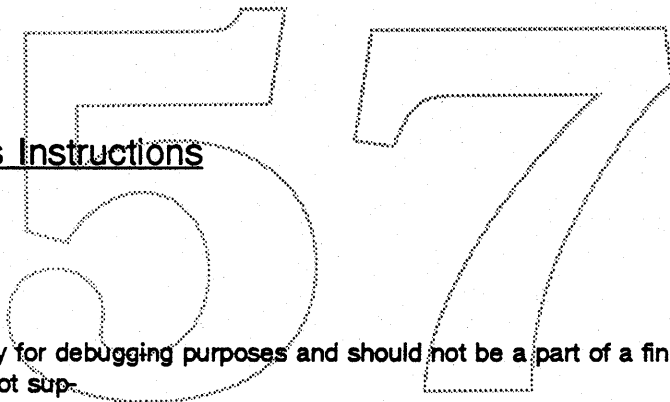
DEBUG()

DEBUG call

stack: number => -  
code: 0x4f

This instruction is here only for debugging purposes and should not be a part of a finished font. All implementations may not support this instruction.

On the current Macintosh this instruction will go to MacsBug and print out the number taken from the stack. Typing g will cause execution to resume.



## 6.0 Interface

```
/*
 * file: fnt.h
 *
 * © Apple Computer Inc. 1987, 1988, 1989.
 *
 */
#ifndef __SETJMP__
#include <SetJmp.h>
#endif

#define F26Dot6 long

typedef long (*longFunc) ();

typedef Fract (*FractFunc) ();

typedef void (*voidFunc) ();

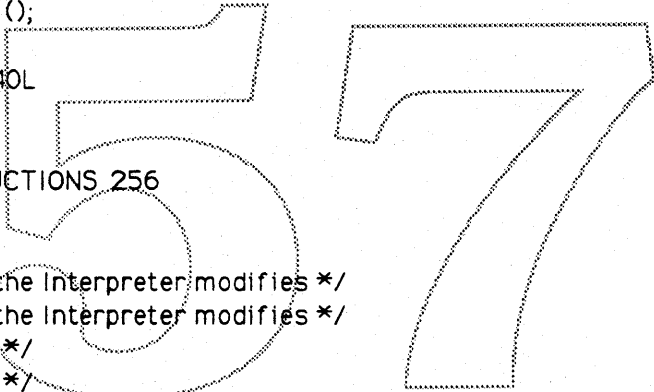
#define fnt_pixelSize 0x40L
#define fnt_pixelShift 6

#define MAXBYTE_INSTRUCTIONS 256

typedef struct {
    long *x; /* The Points the Interpreter modifies */
    long *y; /* The Points the Interpreter modifies */
    long *ox; /* Old Points */
    long *oy; /* Old Points */
    long *oxx; /* Old Unscaled Points */
    long *ooy; /* Old Unscaled Points */
    char *onCurve; /* indicates if a point is on or off the curve */
    short nc; /* Number of contours */
    short *sp; /* Start points */
    short *ep; /* End points */
    char *f; /* Internal flags, one byte for every point */
    F26Dot6
    leftSideBearingIn, leftSideBearingOut;
    F26Dot6
    advanceWidthIn, advanceWidthOut;
} fnt_ElementType;

typedef struct {
    long start; /* index to first instruction */
    long end; /* index to one past the last instruction */
} fnt_funcDef;

typedef struct {
```



```

    short error;    /* If not zero, then an error occurred */
} fnt_OutputData;

typedef struct {
    Fract x, y;
    short distance;
} fnt_AngleInfo;

#define MAXANGLES 20

/*
 * This is the global graphics state, which persists between characters.
 * NOTE: A character should not change the global graphics state
 *       in such a way that it might impact another character !!!!!
 */
typedef struct {
    voidFunc *function; /* pointer to instruction definition area */
    F26Dot6 engine[4]; /* Engine Characteristics */
    F26Dot6 *stackBase; /* the stack area */
    short filler;
    short init; /* Lets us know if we are executing the pre program or not */
    short pixelsPerEm; /* number of pixels per em as an integer */
    short pointSize; /* the requested point size as an integer */
    /* Global Graphic state stuff the pre-program may change below */
    F26Dot6 *store; /* the storage area */
    fnt_funcDef *funcDef; /* function Definitions */
    char *prePgm; /* pointer to the preProgram, which may contain f() definitions */
    F26Dot6 *controlValueTable; /* the control value table */
    F26Dot6 wTCl; /* width table cut in */
    F26Dot6 swCl; /* single width cut in */
    short sW; /* single width, expressed in the same units as the character */
    F26Dot6 scaledSW; /* scaled single width */
    short autoFlip; /* The auto flip Boolean */
    short deltaBase;
    short deltaShift;
    short angleWeight;
    fnt_OutputData outPut;

    /* BELOW WE HAVE PRIVATE STUFF */
    longFunc ScaleFunc; /* Call back function to do scaling */
    /* These are parameters used by the call back function */
    Fixed fixedScale; /* fixed scaling factor */
    long nScale; /* numerator required to scale points to the right size */
    long dScale; /* denominator required to scale points to the right size */
    short shift; /* 2log of dScale */
    fnt_AngleInfo *angleInfo;
} fnt_GlobalGraphicStateType;

```

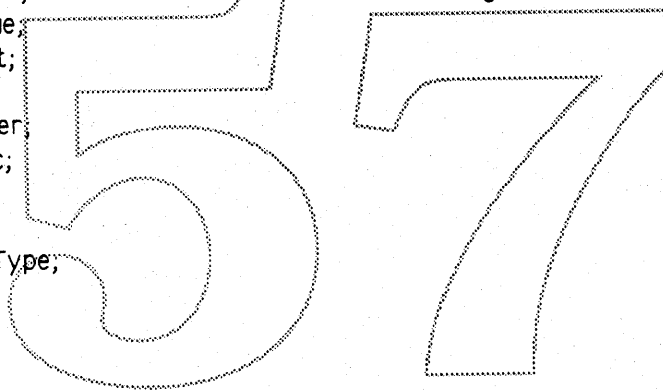


```

/*
 * This is the local graphics state which has no inter character memory
 */
typedef struct {
    short Pt0, Pt1, Pt2; /* The internal reference points */
    fnt_ElementType *CE0, *CE1, *CE2; /* The character element pointers */
    short roundToGrid;
    F26Dot6 minimumDistance;
    Fract pvx, pvy; /* Projection Vector */
    Fract fvx, fvy; /* Freedom Vector */
    F26Dot6 *stackPointer;
    long loop; /* The loop variable */
    unsigned char *insPtr; /* Pointer to the instruction we are about to execute */
    fnt_ElementType *elements;
    fnt_GlobalGraphicStateType *globalGS;

    /* BELOW WE HAVE PRIVATE STUFF */
    Fract pfProj; /* = pvx * fvx + pvy * fvy */
    unsigned char opCode; /* The instruction we are executing */
    longFunc RoundValue;
    voidFunc MovePoint;
    FractFunc Project;
    voidFunc Interpreter;
    voidFunc TraceFunc;
    jmp_buf
    env;
} fnt_LocalGraphicStateType;

```



```

/*
 * Executes the font instructions.
 * This is the external interface to the interpreter.
 *
 * Parameter Description
 *
 * elements points to the character elements. Element 0 is always
 * reserved and not used by the actual character.
 *
 * ptr points at the first instruction.
 * eptr points to right after the last instruction
 *
 * globalGS points at the global graphics state
 *
 * TraceFunc is pointer to a callback function called with a pointer to the
 * local graphics state if TraceFunc is not null. The call is made just before
 * every instruction is executed.
 *
 * Note: The stuff globalGS is pointing at must remain intact
 * between calls to this function.

```

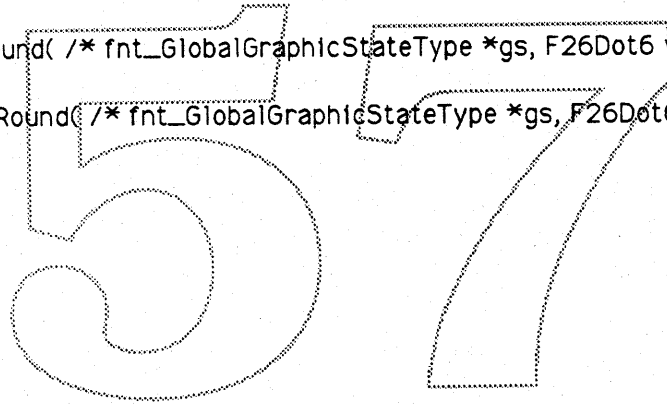
```

*/
extern int fnt_Execute( /* elements, ptr, eptr, globalGS, TraceFunc */ );
/*
  fnt_ElementType *elements;
  char *ptr, *eptr;
  fnt_GlobalGraphicStateType *globalGS;
  voidFunc traceFunc;
*/

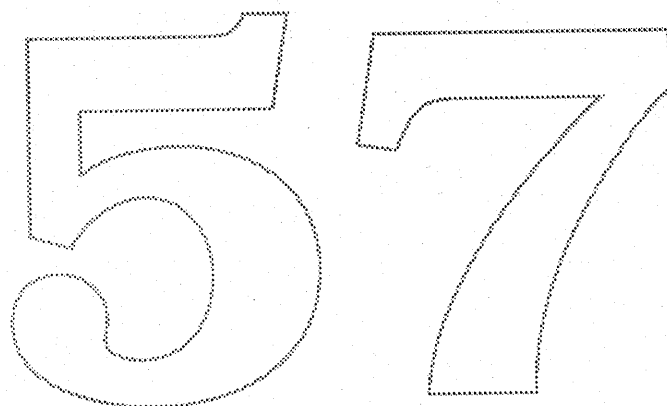
/*
* Init routine, to be called at boot time.
*/
extern void fnt_Init( /* globalGS */ );
/* fnt_GlobalGraphicStateType *globalGS; */

extern F26Dot6 fnt_FRound( /* fnt_GlobalGraphicStateType *gs, F26Dot6 value */ );
extern F26Dot6 fnt_SRound( /* fnt_GlobalGraphicStateType *gs, F26Dot6 value */ );
extern F26Dot6 fnt_FixRound( /* fnt_GlobalGraphicStateType *gs, F26Dot6 value */ );

```

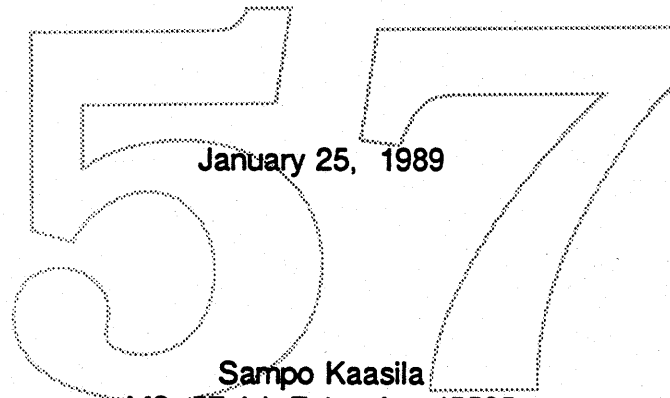


### 17.2.3 Scan Converter SubERS (includes Quadratic B-Spline description)



# The Bass Project

## Scan Converter Sub ERS



January 25, 1989

Sampo Kaasila

MS: 27-AJ; Extension 45505

Product Development

Apple Computer, Inc.

# Contents

1.0 Introduction

2.0 Goals

3.0 Functional Description

4.0 Implementation Details

4.1 The Mathematical Foundation for the Forward Differencing

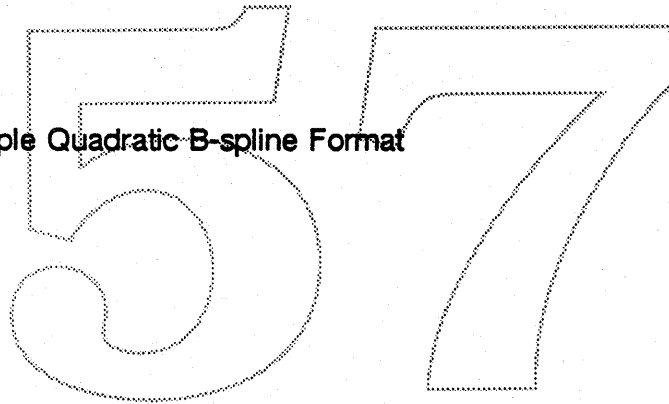
4.2 The Modified Bresenham Algorithm

4.3 The Painter

4.4 Memory Usage

5.0 Interface

Appendix A: The Apple Quadratic B-spline Format



# The Scan Converter

## 1.0 Introduction

The scan converter is an internal module in the outline font package and not directly callable by an application. The outline font package will be an integral part of the Macintosh system software. Its function is to take an outline description of a shape and return a bitmap description. The outline is described by 2nd order B-splines, see appendix A.

## 2.0 Goals

The scan converter is a likely bottle neck in the outline font package. Therefore great attention has been focused on minimizing this overhead. The algorithms are designed to work optimally, in terms of speed, at small point sizes like 12 pt at 300 dpi.

## 3.0 Functional Description

The scan converter works according to a simple model:

- (1) If a pixel center is inside or on the outline then the pixel is always turned on. There is one exception to this rule and this is when the pixel center is exactly on a local Y-minimum on the curve or exactly on a horizontal straight line which is not a Y-maximum. This improves the efficiency of the algorithms, since if the scan converter only includes one end of every spline, cases where a scan line goes through the end point of two splines are easy to deal with. So, the range  $Y_{min} < y \leq Y_{max}$  will be searched for scan line intersections and therefore we have the exception.
- (2) Also, the scan converter does not evaluate the B-spline exactly. Instead it approximates it by straight line vectors. This results in an error. During the development cycle outline fonts has been subject to this error limit, and a sixteenth of a pixel has been found to be a reasonable error.
- (3) To support large glyph generation, the scan converter is also able to do banding of glyph bitmaps at a loss of speed.

## 4.0 Implementation details

The scan conversion process has 3 or 4 steps:

- (1) If 32 bits does not provide enough bits for full precision for the forward differencing coefficients then, since forward differencing is numerically instable, break down the curve using recursive subdivision which is numerically stable until it is within range for the forward differencing.
- (2) Break up the curves into straight line vectors using forward differencing.
- (3) Find scanline intersections using a slightly modified Bresenham algorithm.
- (4) Paint the bitmap using x-oring.

When the bitmap is finished it is be handed to the font manage.

## 4.1 The Mathematical Foundation for the Forward Differencing

First break down the B-spline into 2nd degree Bezier curves. Each Bezier curve is described by three control points ( A, B, C ). Please, see picture in the explanation of the Apple quadratic B-spline spline format in appendix A.

The parametric equation for a 2nd degree Bezier curve is:

$F(t) = (1-t)^2 * A + 2 * t * (1-t) * B + t * t * C$ , and t goes from 0 to 1 =>

$F(t) = t * t * (A - 2B + C) + t * (2B - 2A) + A =>$

Now say that s goes from 0...N, =>  $t = s/N$

set:  $G(s) = N * N * F(s/N)$

$G(s) = s * s * (A - 2B + C) + s * N * 2 * (B - A) + N * N * A$

In the scan converter N will always be made to be a power of two. This means that we can compute F(t) from G(s) simply by using shift instructions.

Let's look at the first values:

=>  $G(0) = N * N * A$

=>  $G(1) = (A - 2B + C) + 2 * N * (B - A) + G(0)$

=>  $G(2) = 4 * (A - 2B + C) + 4 * N * (B - A) + G(0) =$   
 $3 * (A - 2B + C) + 2 * N * (B - A) + G(1)$

From this we can derive the forward differencing coefficients:

$D(G(0)) = G(1) - G(0) = (A - 2B + C) + 2 * N * (B - A)$

$D(G(1)) = G(2) - G(1) = 3 * (A - 2B + C) + 2 * N * (B - A)$

$DD(G(0)) = D(G(1)) - D(G(0)) = 2 * (A - 2B + C)$

Now we need to be able to compute an error term. Would it not be wonderful if we could express the error in terms of the 2nd order difference, since this is what

differentiates this from a straight line and we need to compute the difference anyway to be able to do the forward differencing. This would cut down on the overhead required to compute the error. Well, this is exactly what we are going to do !

Let's define the error as the distance between the mid point between point A and C  $((A+C)/2)$  and the spot on the curve where  $t$  is 0.5  $(A/4 + B/2 + C/4)$ . The spot where  $t$  is 0.5 is exactly the mid point on the curve in the parametric space.

$$\text{error} = (A+C)/2 - (A/4 + B/2 + C/4) = A/4 - B/2 + C/4 = (A - 2B + C) / 4 = DD(G) / 8 !!!$$

It can also be shown that if we double the number of steps then new DD = old DD / 4. Explanation: As, Bs, Cs are one half of the new points generated from a subdivision of A, B, C.

$$As = A$$

$$Bs = (A + B) / 2$$

$$Cs = ((A + B) / 2 + (B + C) / 2) / 2 = A/4 + B/2 + C/4$$

Now the new error is:

$$es = (As - 2Bs + Cs) / 4 = (A - (A + B) + A/4 + B/2 + C/4) / 4 = (A/4 - B/2 + C/4) / 4 = (A - 2B + C) / (4 * 4) = \text{old error} / 4 !!!$$

Since the error term is directly proportional to the 2nd order difference we can safely conclude that the subdivided 2nd order difference is exactly a quarter of the old 2nd order difference.

All this can be used to implement an algorithm that breaks 2nd order Bezier curves into straight line vectors without using any multiply or divide instructions. It simply involves determining how many steps are necessary for the forward differencing given a maximum acceptable error threshold and the 2nd order forward difference. After this it is only a question of looping around and incrementally updating the first order differences (by adding the 2nd order difference) and storing away the coordinates.

## 4.2 The Modified Bresenham Algorithm

This is implemented as a fairly plain vanilla Bresenham line drawing algorithm. The main difference is that we have fractional pixel positions for the straight line vectors. The way this is solved is by using two multiplies to calculate the error term at the first Y-scan line intersection. After this the error terms are changed (shifted left by the number of fractional pixel position bits) and we change the step-size to be a whole pixel. The algorithm also includes pixels that are centered exactly on the intersection of the vector and the scanline. To summarize, this algorithm takes straight line vectors and produces the pixel coordinates for the intersections between the scan lines and the vectors.



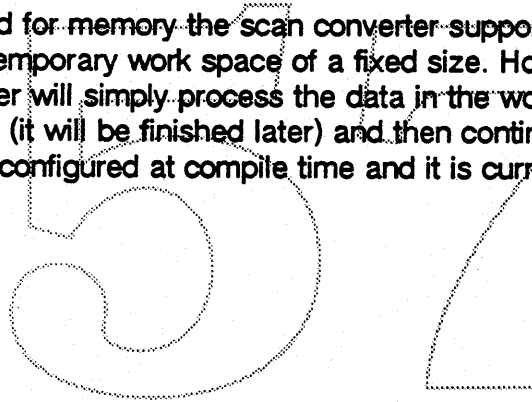
### 4.3 The Painter

This is implemented using an x-oring scheme on 32 bit wide data. It simply takes all the scan line intersections produced by the Bressenham algorithm and inverts the scan line starting from the intersection to the right border of the bitmap. After this has been done to all the scan line intersections the result will be the correct bitmap. A scan line intersection is the intersection between the scan line and the spline describing the character.

### 4.4 Memory Usage

The scan converter relies on the caller to provide all the memory and hence it does not directly deal with allocating and deallocating memory.

To minimize the need for memory the scan converter supports banding. Also there is a need for some temporary work space of a fixed size. However if more is needed then the scan converter will simply process the data in the work space and produce an unfinished bitmap (it will be finished later) and then continue. The size of the work space can be configured at compile time and it is currently set at 3376 bytes.



## 5.0 Interface

```
/*
 * File: sc.h
 *
 * This module scanconverts a shape defined by quadratic bezier splines
 *
 * © Apple Computer Inc. 1987, 1988, 1989.
 *
 * History:
 * Work on this module began in the fall of 1987.
 * Written June 14, 1988 by Sampo Kaasila.
 */

/* DO NOT change these constants without understanding implications:
   overflow, out of range, out of memory, quality considerations, etc... */

#define PIXELSIZE 64 /* number of units per pixel. It has to be a power of two */
#define PIXSHIFT 6 /* should be 2log of PIXELSIZE */
#define ERRDIV 16 /* maximum error is (pixel/ERRDIV) */
#define ERRSHIFT 4 /* = 2log(ERRDIV), define only if ERRDIV is a power of 2 */

/* The maximum number of vectors a spline segment is broken down into
 * is 2 ^ MAXGY
 * MAXGY can at most be:
 * (31 - (input range to sc_DrawParabola 15 + PIXSHIFT = 21)) / 2
 */
#define MAXGY 5
#define MAXMAXGY 8 /* related to MAXVECTORS */

/* RULE OF THUMB: xPoint and yPoints will run out of space when
 * MAXVECTORS = 176 + ppem/4 ( ppem = pixels per EM ) */
#define MAXVECTORS 257 /* must be at least 257 = (2 ^ MAXMAXGY) + 1 */

/* RULE OF THUMB: scanData will run out of space when
 * SIZEOFSCANDATA = 2.2 * ppem (think about an S) */
#define SIZEOFSCANDATA 660 /* must be at least 4 and a multiple of 4 */
/* NOTE: when we run out of space speed is reduced, but we still function */

#define sc_outOfMemory 0x01 /* For the error field */
#define sc_freeBitMap 0x01 /* For the info field */

typedef struct {
    uint32 *bitMap;
    int32 xMin, yMin, xMax, yMax;
} sc_BitMapData;
```

```

typedef struct {
    int16 yxData[ SIZEOFSCANDATA ]; /* y,x y,x,... */
    int32 xPoints[ MAXVECTORS ]; /* vectors */
    int32 yPoints[ MAXVECTORS ];
} sc_GlobalData;

typedef struct {
    int32 *x, *y;
    int16 ctrs;
    int16 *sp, *ep;
    char *onC;
} sc_CharDataType;

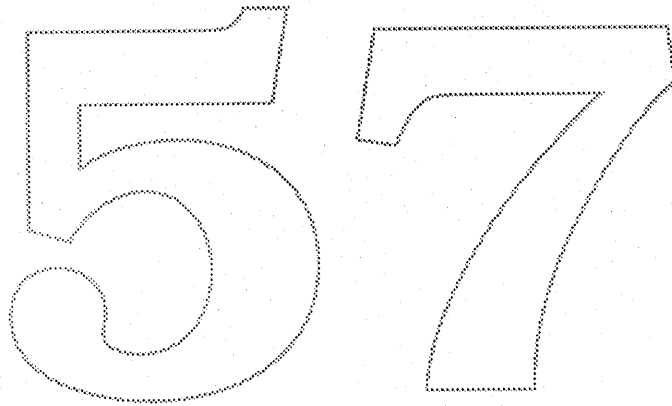
/*
 * Returns the bitmap
 * This is the top level call to the scan converter.
 *
 * Assumes that (*handle)->bbox.xmin,...xmax,...ymin,...ymax
 * are already set by sc_FindExtrema()
 *
 * PARAMETERS:
 *
 * charPtr is a pointer to sc_CharDataType
 * scPtr is a pointer to sc_GlobalData.
 *
 * lowBand is lowest scan line to be included in the band.
 * highBand is the highest scan line to be included in the band.
 * if highBand < lowBand then no banding will be done.
 * Always keep lowBand and highband within range: [ymin, (ymin+1) ..... ymax];
 * scPtr->bitMap always points at the actual memory.
 * the first row of pixels above the baseLine is numbered 0, and the next one up is 1.
 * => the y-axis definition is the normal one with the y-axis pointing straight up.
 */
extern void sc_ScanChar( /* charPtr, scPtr, bbox, lowBand, highBand */);

/*
sc_CharDataType *charPtr;
sc_GlobalData *scPtr;
sc_BitMapData *bbox;
int16 lowBand, highBand;
*/

/*
 * Finds the extrema of a character.
 *
 * PARAMETERS:
 *
 * charPtr is a pointer to sc_CharDataType
 *
 * bbox is the output of this function and it contains the bounding box.
 */

```

```
extern void sc_FindExtrema( /* charPtr, bbox */ );  
/*  
sc_CharDataType *charPtr;  
sc_BitMapData *bbox;  
*/
```



## Appendix A: The Apple Quadratic B-spline Format

The glyphs are described by a 2nd order B-spline format. In this format there are only two kinds of points: points that are on the curve and points that are not on the curve. All combinations of points on and off the curve are legal.

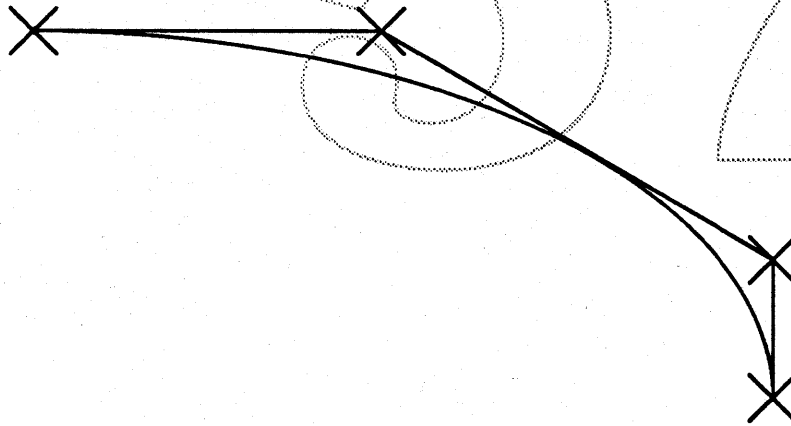
A straight line segment is simply described by two consecutive points on the curve. A non straight B-spline can be broken down to a quadratic Bezier format where a curve segment is described by three points (A, B, C). Point A and C are on the curve and B is a tangent point outside the curve. The tangent direction at point A is the vector AB, and the tangent direction at point C is the vector BC.

The parametric equation for the quadratic Bezier format is:

$F(t) = (1-t)(1-t) * A + 2t(1-t) * B + t*t*C$ , where t is 0 at point A and 1 at point C.

The quadratic B-spline format can be broken down into quadratic Bezier splines by creating new points on the curve exactly in between every two consecutive points laying outside the curve.

For example, in the picture below there are four points. Lets number them 1 to 4 starting from the left. Point 1 and 4 are on the curve while point 2 and 3 are not. This can be broken down to two 2nd order Beziers where the first spline would be described by the following three points (A,B,C): Point 1, Point 2, (Point 2 + Point 3)/2. The second Bezier would be described by: (Point 2 + Point 3)/2, Point 3, Point 4. Note that this B-spline format guarantees first degree continuity.



To describe the shape of a character there is a need for the following information:

1. How many contours ?
3. Last point of every contour. (The last point does not overlap the starting point.)
4. A flag to indicate if a point is on or off the curve.

Also, it is not necessary to have points at curve extrema if the two surrounding points, laying outside the curve, have the same x-value for an x-extrema and the same y-value for a y-extrema. In this case the surrounding points provide enough control to handle the "grid-fitting". The direction of the curves has to be such that, if you follow a curve in the direction of increasing point numbers, black (the filled area) will always be to the right.

## 6.0 Interface

```
/*
 * file: fnt.h
 *
 * © Apple Computer Inc. 1987, 1988, 1989.
 *
 */
#ifndef __SETJMP__
#include <SetJmp.h>
#endif

#define F26Dot6 long

typedef long (*longFunc) ();

typedef Fract (*FractFunc) ();

typedef void (*voidFunc) ();

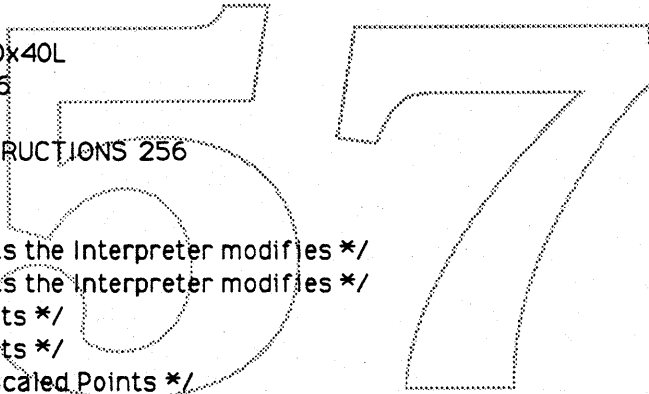
#define fnt_pixelSize 0x40L
#define fnt_pixelShift 6

#define MAXBYTE_INSTRUCTIONS 256

typedef struct {
    long *x; /* The Points the Interpreter modifies */
    long *y; /* The Points the Interpreter modifies */
    long *ox; /* Old Points */
    long *oy; /* Old Points */
    long *oox; /* Old Unscaled Points */
    long *ooy; /* Old Unscaled Points */
    char *onCurve; /* indicates if a point is on or off the curve */
    short nc; /* Number of contours */
    short *sp; /* Start points */
    short *ep; /* End points */
    char *f; /* Internal flags, one byte for every point */
    F26Dot6
    leftSideBearingIn, leftSideBearingOut;
    F26Dot6
    advanceWidthIn, advanceWidthOut;
} fnt_ElementType;

typedef struct {
    long start; /* index to fist instruction */
    long end; /* index to one past the last instruction */
} fnt_funcDef;

typedef struct {
```



```

    short error;    /* If not zero, then an error occurred */
} fnt_OutputData;

typedef struct {
    Fract x, y;
    short distance;
} fnt_AngleInfo;

#define MAXANGLES 20

/*
 * This is the global graphics state, which persists between characters.
 * NOTE: A character should not change the global graphics state
 *       in such a way that it might impact another character !!!!!
 */
typedef struct {
    voidFunc *function; /* pointer to instruction definition area */
    F26Dot6 engine[4]; /* Engine Characteristics */
    F26Dot6 *stackBase; /* the stack area */
    short filler;
    short init; /* Lets us know if we are executing the pre-program or not */
    short pixelsPerEm; /* number of pixels per em as an integer */
    short pointSize; /* the requested point size as an integer */
    /* Global Graphic state stuff the pre-program may change below */
    F26Dot6 *store; /* the storage area */
    fnt_funcDef *funcDef; /* function Definitions */
    char *prePgm; /* pointer to the preProgram, which may contain f() definitions */
    F26Dot6 *controlValueTable; /* the control value table */
    F26Dot6 wTCl; /* width table cut in */
    F26Dot6 sWCl; /* single width cut in */
    short sW; /* single width, expressed in the same units as the character */
    F26Dot6 scaledSW; /* scaled single width */
    short autoFlip; /* The auto flip Boolean */
    short deltaBase;
    short deltaShift;
    short angleWeight;
    fnt_OutputData outPut;

    /* BELOW WE HAVE PRIVATE STUFF */
    longFunc ScaleFunc; /* Call back function to do scaling */
    /* These are parameters used by the call back function */
    Fixed fixedScale; /* fixed scaling factor */
    long nScale; /* numerator required to scale points to the right size */
    long dScale; /* denominator required to scale points to the right size */
    short shift; /* 2log of dScale */
    fnt_AngleInfo *angleInfo;
} fnt_GlobalGraphicStateType;

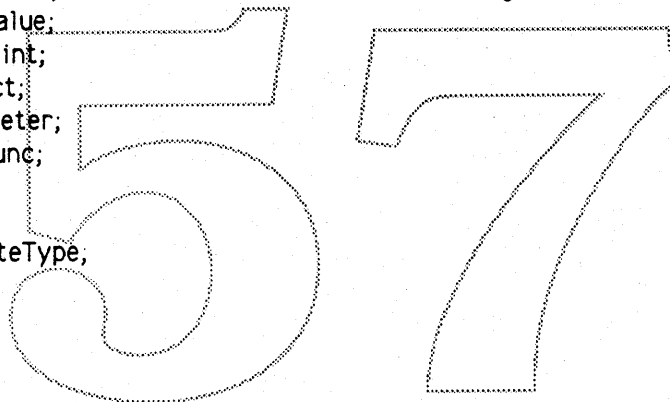
```

```

/*
 * This is the local graphics state which has no inter character memory
 */
typedef struct {
    short Pt0, Pt1, Pt2; /* The internal reference points */
    fnt_ElementType *CE0, *CE1, *CE2; /* The character element pointers */
    short roundToGrid;
    F26Dot6 minimumDistance;
    Fract pvx, pvy; /* Projection Vector */
    Fract fvx, fvy; /* Freedom Vector */
    F26Dot6 *stackPointer;
    long loop; /* The loop variable */
    unsigned char *insPtr; /* Pointer to the instruction we are about to execute */
    fnt_ElementType *elements;
    fnt_GlobalGraphicStateType *globalGS;

    /* BELOW WE HAVE PRIVATE STUFF */
    Fract pfProj; /* = pvx * fvx + pvy * fvy */
    unsigned char opCode; /* The instruction we are executing */
    longFunc RoundValue;
    voidFunc MovePoint;
    FractFunc Project;
    voidFunc Interpreter;
    voidFunc TraceFunc;
    jmp_buf
    env;
} fnt_LocalGraphicStateType;

```



```

/*
 * Executes the font instructions.
 * This is the external interface to the interpreter.
 *
 * Parameter Description
 *
 * elements points to the character elements. Element 0 is always
 * reserved and not used by the actual character.
 *
 * ptr points at the first instruction.
 * eptr points to right after the last instruction
 *
 * globalGS points at the global graphics state
 *
 * TraceFunc is pointer to a callback functioned called with a pointer to the
 * local graphics state if TraceFunc is not null. The call is made just before
 * every instruction is executed.
 *
 * Note: The stuff globalGS is pointing at must remain intact
 * between calls to this function.

```



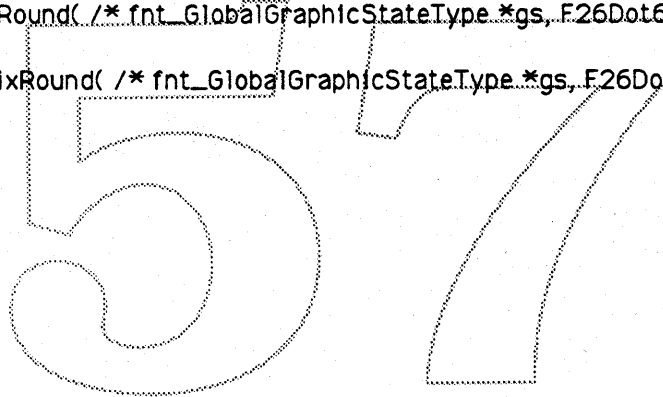
```
*/
extern int fnt_Execute( /* elements, ptr, eptr, globalGS, TraceFunc */ );
/*
  fnt_ElementType *elements;
  char *ptr, *eptr;
  fnt_GlobalGraphicStateType *globalGS;
  voidFunc traceFunc;
*/
```

```
/*
 * Init routine, to be called at boot time.
 */
extern void fnt_Init( /* globalGS */ );
/* fnt_GlobalGraphicStateType *globalGS; */
```

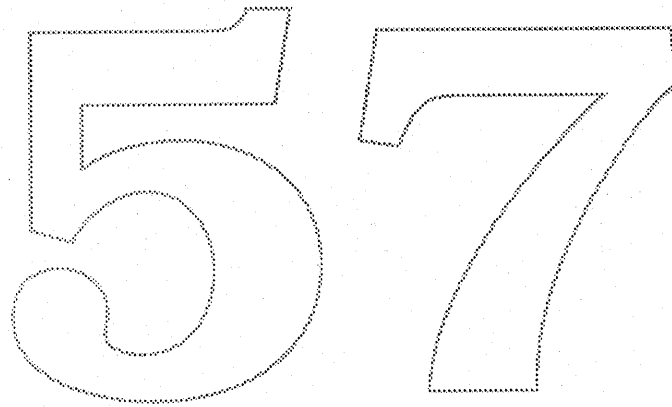
```
extern F26Dot6 fnt_FRound( /* fnt_GlobalGraphicStateType *gs, F26Dot6 value */ );
```

```
extern F26Dot6 fnt_SRound( /* fnt_GlobalGraphicStateType *gs, F26Dot6 value */ );
```

```
extern F26Dot6 fnt_FixRound( /* fnt_GlobalGraphicStateType *gs, F26Dot6 value */ );
```



## 17.2.4 Instruction Editor SubERS



# Bass Edit

## SubERS

January 29, 1989

Bass Edit is a developer's tool for assigning Bass font instructions to 'sfnt' fonts. These 'instructions' are a stack based language that operates on the control points of an outline character. The font designer opens a font, types a character, and can then view its outline, instructions, and how it will look as a bitmap at a given size. The edited font can then be saved (optionally with a new name), turned into a suitcase file for Font/DA Mover, or dumped out as a text file. The editor also has a number of auxiliary views: Control Values, Info (advance width and side bearings), TextEdit, and the PreProgram.

To edit a font, choose *Open* from the **File** menu and select the font. Its Creator should be 'SEdt', and its Type should be 'Sfnt'. The **View** menu will now contain a list of the displays available. Choosing one from the menu brings that display to the front. Clicking in the go-away box of a display hides it. If all the windows are closed, the font file is automatically closed, as if *Close* was chosen from the **File** menu. *Dump* will write out the font into a text file. This is intended only as a means of verifying the data. The *Print* item in the **File** menu applies to the front window only, allowing the user to print only the information wanted. Most dialogs may be dragged like normal windows by clicking on the double-line border. This is useful if the dialog covers some other part of the screen that the user needs to see.

### Alphabet

#### Display

This view shows all 256 characters in a normal quickdraw FONT. The user may double click on any character to select it for editing.

This view also handles

The screenshot shows a window titled "NuHelu41" with a grid of characters. The grid is organized into rows and columns, with some characters appearing to be standard ASCII characters and others being special characters or ligatures. The characters are displayed in a serif font style.

□	□	0	@	P	`	p	À	é	†	—	¿	—	□
□	□	!	1	A	Q	a	q	Ä	ë	°	±	ì	—
□	□	"	2	B	R	b	r	Ç	í	¢	≤	¬	"
□	□	#	3	C	S	c	s	É	ì	£	≥	√	"
□	□	\$	4	D	T	d	t	Ë	î	§	¥	f	'
□	□	%	5	E	U	e	u	Ö	ï	▪	μ	≈	'
□	□	&	6	F	V	f	v	Ü	ñ	¶	ð	Δ	÷
□	□	'	7	G	W	g	w	á	ó	β	Σ	«	♦
□	□	(	8	H	X	h	x	à	ò	⊗	Π	»	ÿ
□	□	)	9	I	Y	i	y	â	ô	⊙	π	...	ÿ

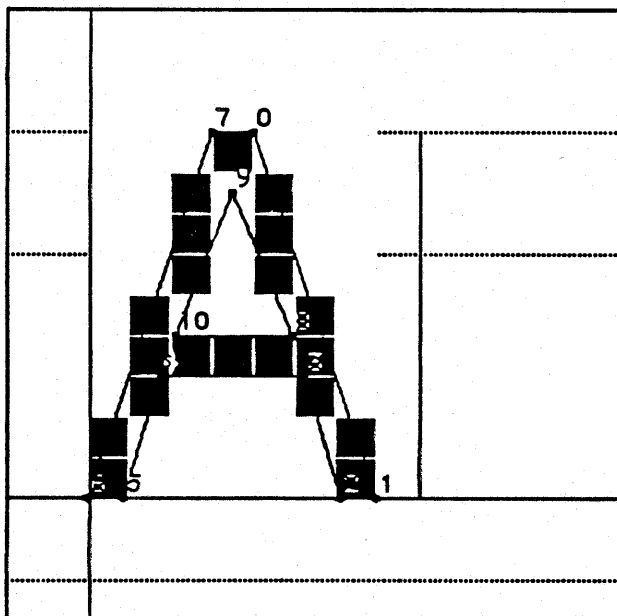
cutting and pasting of entire characters, within and between sfnt files. The font displayed is Helvetica, so that a designer of a Roman font can match the sfnt characters to those defined by

quickdraw.

## Info Display

The Info Display is the central place numerical information about the font and the current character is displayed. The first line and first column of three apply to the whole font, while the rest apply only to the current character and point size. The user can edit the left-side-bearing, advance-width and right-side-bearing. Notice that they are in integer units in UPEM space, just like the control point coordinates. In addition they can enter a character number or point size directly, just by clicking on the value and editing it. The check box for instructions operates just like the *Hints* item in the SFNT menu, toggling the instruction interpreter on and off.

H 1.0a52•Info			
Font	H 1.0a52	UPEM	2048
Style	0	H - height	13
First glyph	32	x - height	10
Last glyph	255	p - height	-4
pxl width 10		pxl height 13	
-----			
1.4531	LSB 166	Char	66
12.0059	AW 1366	Size	18
0.5527	RSB 92	<input type="checkbox"/>	Instructions



## Glyph Display

The glyph display allows the user to view the character being edited. The user may change the point size and the amount of fat-bit zooming to get the character at a size for viewing. There are a number of options available to customize what information is displayed about the character. These options are toggled on and off from a menu. The character in the picture has metrics, gridlines, numbers, instructions, bits and outline ON, and foci and blobs OFF.

- *Metrics*: the lines for character origin,

baseline, advance width cap height, x-height and descender

- *Grid Lines*: the white lines that delineate each pixel when the character is in fat-bit mode
- *Outline*: the scaled outline that was used to create the bitmap
- *Point Numbers*: the internal numbering scheme that the instructions operate on
- *Show Bits*: the bitmap produced by the scaled outline
- *Instructions*: the glyph can be displayed with or without executing the instructions
- *Pixel Foci*: shows pixel centers when the character is zoomed in for viewing
- *Pixel Blobs*: use circular pixels instead of square, approximating dot sizes for different imaging engines, see Engine Display.

The character is placed relative to the character origin line to show its left and right side bearing.

## Instruction Display

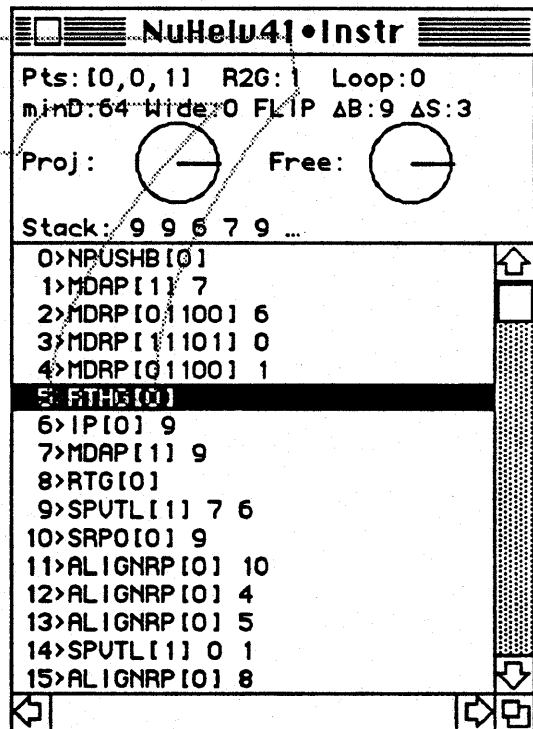
This is where the real work gets done. The display is split into two views. The bottom is the list of instructions that apply to the current character. They are numbered starting with 0 so their context can be identified when they are shown in the glyph display. The Edit menu, in addition to *Cut*, *Copy*, *Paste*, *Clear and Select All*, contains two new items which apply to instruction editing: *Insert*, which creates a new entry just above currently selected instruction, and *Append*, which creates a new entry after the last instruction.

An instruction is selected by clicking on it. A range of instruction may be selected by holding down the SHIFT key, as in editing text. To edit an individual instruction the user has two options.

One is to double click, which invokes a dialog box split into three fields: the instruction name (upper or lower case), its booleans, and its arguments (separated by spaces). The second is to

4>MIRP[10100] 6 35
5>MIRP[01101] 29 37
6>SRPO[0] 22
7>MIRP[11101] 37 37

click on the text after it is selected, as in the finder. The text may then be directly edited, including using the keyboard commands for cut, copy and paste. CMD-PERIOD or CMD-Z undoes any edit-



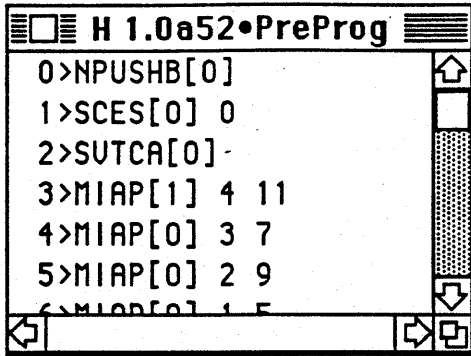
ing, clicking away or hitting RETURN accepts the changes. Changes to the instruction list do not take effect immediately. They must be compiled into binary form. To do this, choose *Recalc* from the **Instruction** menu. All instructions should edit as they are documented in Sampo's ERS, with their arguments listed with them. The major exception are the PUSH instructions. No arguments are given with a push instruction, since they appear with their respective instruction. During recalc, all arguments following a push instruction up to the next push are collected and applied. In addition, the compiler takes care of converting between PUSHB and NPUSHB instructions, with the correct booleans. The same holds true for PUSHW and NPUSHW. At this time the compiler does not check for arguments out of range of a PUSHB or NPUSHB. The minor exception is the DELTA instruction. Because its arguments are packed, a separate dialog box appears for editing. The user is presented with a large, scrolling fields to enter delta triplets: Size, Move, Point. Size is the point size is the relative to the delta-base variable. Move is in integer units to be shifted by the delta-shift variable. Point is just the point number of the control point to be moved. The interpreter expects the delta exceptions to be sorted. This is done for the user. Note also that the user may click on the Recalc button while editing the DELTA, to see the effect immediately. While in the delta editor, the RETURN key moves the cursor down to the next line, but the ENTER key activates the OK button.

The upper view displays the state of the interpreter's internal graphic-state just before executing the current instruction.

- Pts[]: the three internal reference point numbers
- R2G: round-to-grid
- Loop: loop-count
- minD: minimum distance
- wide: single width
- ΔB: delta base
- ΔS: delta shift
- Proj & Free: Projection and Freedom vectors
- Stack: the first five values on the interpreters stack, starting with the top

When instructions have been added or modified, the graphics-state display will not work until the instructions have been *Recalced*.

**Keyboard short-cuts:** ESCAPE deselects the instructions, just like clicking in the graphic-state. The DELETE key is the same as choosing *Clear* from the **Edit** menu. The UP and DOWN arrows move the current instruction selection. RETURN invokes the instruction editor for the current instruction, just like double clicking.



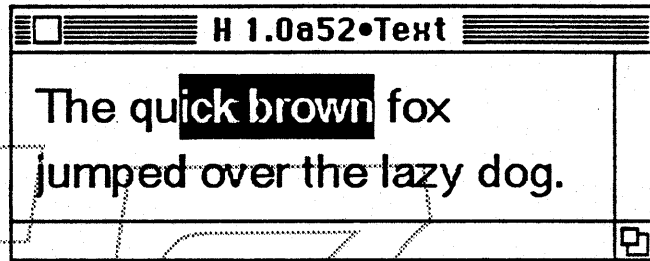
## PreProgram Display

This displays the instructions in the preprogram. It operates just like the character instruction view in that it may be edited, cut and pasted, and must be *recalced* to record changes. One major difference is that there is no associated graphics-state display.

## Text Display

This is a simple text window that the user may type in to see the font as it will appear as an installed resource. Point size is determined by the **Size** menu.

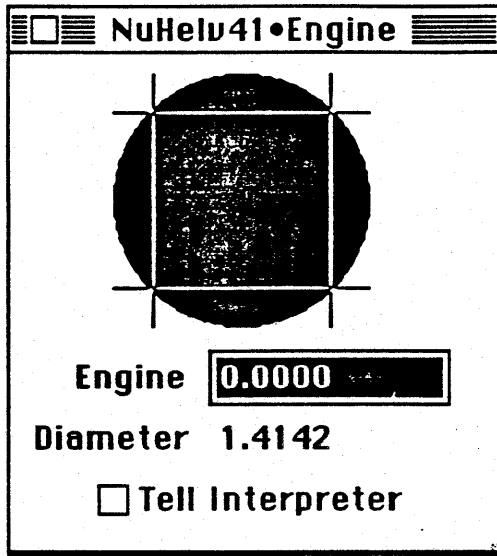
There are currently no scroll bars, so the user may scroll by dragging with the mouse or with the arrow keys. *Cut/Copy/Paste* are supported as well.



	Raw	Scaled
0>	-29871	56328.76
1>	0	56329.01
2>	-27	57344.25
3>	-27	5924.06
4>	-27	456704.23
5>	1081	456704.25
6>	1109	457737.00
7>	1493	457737.01
8>	1519	472072.78
9>	1462	472073.00
10>	1489	473088.23

## Control Values Display

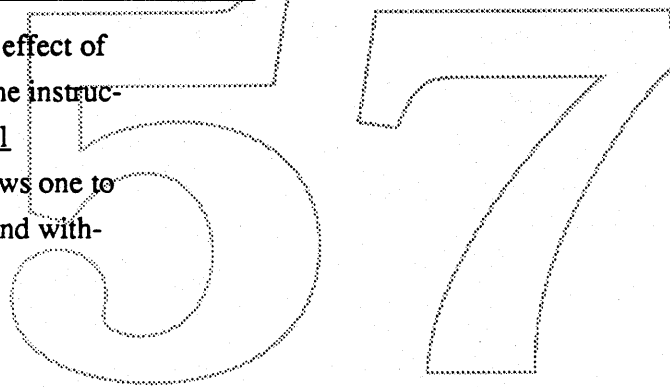
This is just a list of the values stored in the sfnt's control value table. They are numbered starting with 0 as they would be referenced from the instructions. Just like instructions, they may be edited by either double-clicking to invoke a dialog box, or by selecting and then clicking to edit. Also, as with instructions, the table must be 'recalced' to record the changes made to it. The third column is to display the control values scaled to the current point size. This is not yet implemented, so the user should ignore that column for now.



## Engine Characteristics Display

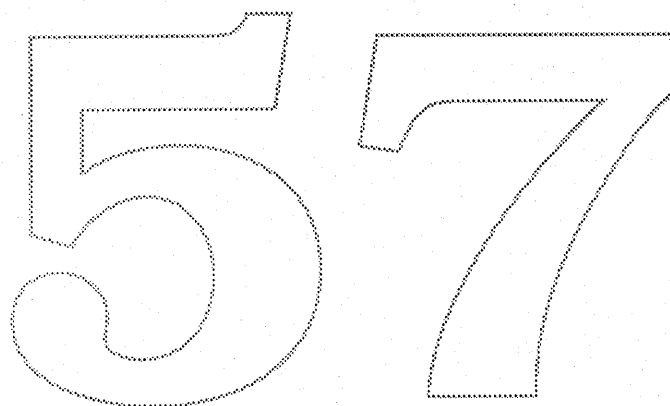
The Engine Display allows the user to test his instructions with different engine characteristics on screen. By choosing *pixel Blobs* from the SFNT menu the glyph display window will draw the character using circular pixels instead of square. The user can now vary the engine diameter and see the resulting pixels that are affected. An engine value of 0 represents an "ideal" print engine. The grid lines show the mathematical pixel boundaries. The user

can choose to see the effect of the engine value on the instructions by selecting Tell Interpreter. This allows one to compare blobs with and without compensation.



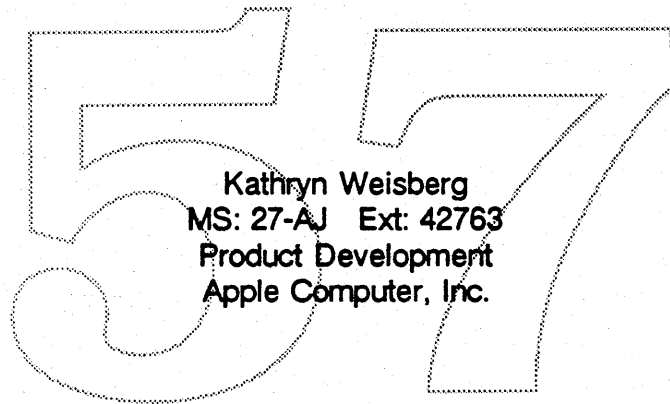


## 17.2.5 Outline Font Data SubERS



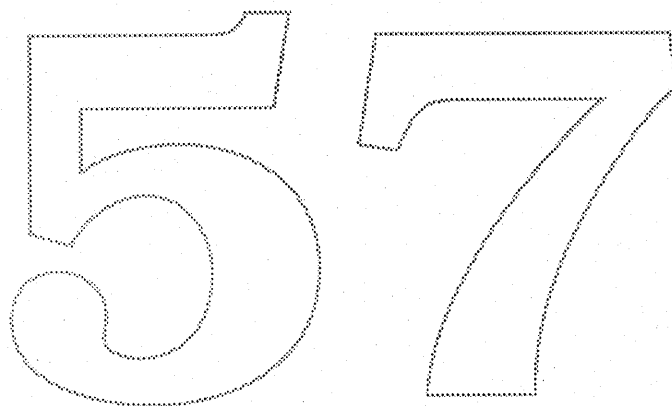
# Outline Font Data SubERS

February 18, 1989



⌘ Confidential Information

1.0	Introduction.....	2
2.0	Description of Task.....	2
3.0	Core Font Requirement.....	2
4.0	Sourcing.....	4
5.0	Digital Masters.....	4
6.0	Spline Format Requirement.....	4
7.0	Instruction Editing.....	4
8.0	Screen Fonts.....	8
9.0	Contingency.....	9



## 1.0 Introduction

In order to best represent the Bass Instruction Set and Intelligent Scaling Algorithms, it is necessary to develop the core set of fonts for the Macintosh using those instructions. The alternative to this approach is to purchase outlines already hinted in another format and convert those fonts to Bass format using a translator. This alternate approach is discussed in Section 9.0 as a contingency. Since the Bass Instruction set is more powerful than any currently known formats, it is in Apple's best interest to develop fonts using that set of primitives. For detailed information on the instruction set and Interpreter, see Appendix 17.2.2 of the Bass ERS.

## 2.0 Description of Task

The task requires that splines be purchased from digital font foundries, convert the splines from their existing format (Bezier, Line & Arc, etc.) to Quadratic B-Splines. The resulting splines must then be converted to the `sfnt` data structure in order to be read by the Instruction Editor. The splines are examined for control point integrity and asymmetries resulting from spline conversions are corrected where possible. The `sfnt` is then subjected to an automated hinting program which analyses the font's structure. Using a clustering technique, like distances are measured and a Control Value Table and Preprogram Table are generated. This level of instruction application provides the core set of distances to control heights, stroke-weight, and relational distances. Further instructions to control diagonals and fine-tune the font are applied manually using the Instruction Editor<sup>1</sup>. Bass is a language and as such requires specific knowledge of its syntax. The instruction programs must be robust if integrity of design is to be maintained through the scaling and scan conversion process and to ensure accurate representation of the character when applying engine characteristics. Since Bass is not Alphabet or Typographic dependent, it will apply equally well to instructing non-roman fonts and perhaps even logotypes and icons.

Once all the instructions are applied the fonts must be tested for errors in Boolean arguments which control engine characteristics. Instructions will be applied on a character by character basis to control scaling through 25• at 72dpi. Sizes from 24• through 9• at 72 dpi will be controlled using delta exceptions. These special instructions will be applied on both a size and character basis. In this way, the outline can be distorted at any given size to turn on/off specific pixels by exception. Only the distance (delta) between the original control point and its new location is stored.

## 3.0 Core Font Requirement

25 fonts will be under development for the Bass Project. Not all of those fonts will be available at introduction and will be phased in as completed. The metrics of these fonts will match the existing LaserWriter ROM metrics for the fonts currently

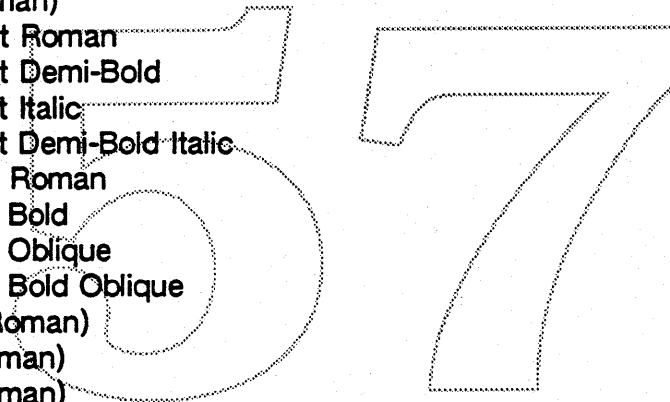
---

1. For a detailed description of the Bass Instruction Editor, See Appendix 17.2.4

available in that environment to avoid document reformatting.

Core Font List

Times New Roman  
Times New Roman Bold  
Times New Roman Italic  
Times New Roman Bold Italic  
Helvetica (Roman)  
Helvetica Bold  
Helvetica Oblique  
Helvetica Bold Oblique  
Courier (Roman)  
Courier Bold  
Courier Italic  
Courier Bold Italic  
Symbol (Roman)  
Lucida Bright Roman  
Lucida Bright Demi-Bold  
Lucida Bright Italic  
Lucida Bright Demi-Bold Italic  
Lucida Sans Roman  
Lucida Sans Bold  
Lucida Sans Oblique  
Lucida Sans Bold Oblique  
New York (Roman)  
Geneva (Roman)  
Monaco (Roman)  
Chicago (Roman)



The Times New Roman font will require font substitution to the LaserWriter and NT/NTX devices. Although the design is similar to Times Roman the name is protected by trademark and will be listed in the Font Menu as Times New Roman. This version of Times is superior to the current Times offered by the LaserWriter in that the stroke weight is finer and should print better at low resolutions and small point sizes as well as appearing more elegant at large sizes and at very high resolutions. The Symbol font was specifically designed to be compatible with the design of the Times New Roman font which will make their characters interchangeable within a document and work well when Symbol characters are interspersed throughout a document. The Lucida font family will be introduced as an alternative serif and sans-serif faces. The Apple fonts of New York, Geneva, Monaco and Chicago will be designed exclusively for Apple and incorporate details specifically tuned to digital scaling technology.

#### 4.0 Sourcing

Digital data (splines) are available from several sources, however, those fonts protected by trademark require that the splines also be obtained from that source. Fortunately, those were the best sources for those splines. Times New Roman was obtained from Monotype Ltd. of England along with the data for Symbol.

Helvetica was obtained from Linotype AG in West Germany. Courier was obtained from URW, also in West Germany. The Lucida fonts will be obtained from Bigelow & Holmes of Menlo Park, California who will also design the Apple Exclusive fonts. Several of the fonts in the core set listed are protected by trademarks and require licensing outside the existing Adobe/Apple agreements. Licensing has been obtained on all but one of the faces (Helvetica) and negotiations are underway for use of this trademark.

#### 5.0 Digital Masters

Digital Masters for the core set will be at an effective resolution of 2048 units to the EM square.

#### 6.0 Spline Format Requirement

Apple's spline format will be Quadratic B-Splines. This requires that all splines obtained for the core set be converted from their source description (Bezier, Conic, Line & Arc, etc.). Conversion of the splines will be done at URW in West Germany and sent to Apple in Macintosh data format (Macintosh Floppy Diskettes) for conversion to sfnt format and instruction application. Tolerance on the conversion to Quadratics will be  $7/1000$  upem based on tests conducted prior to development.

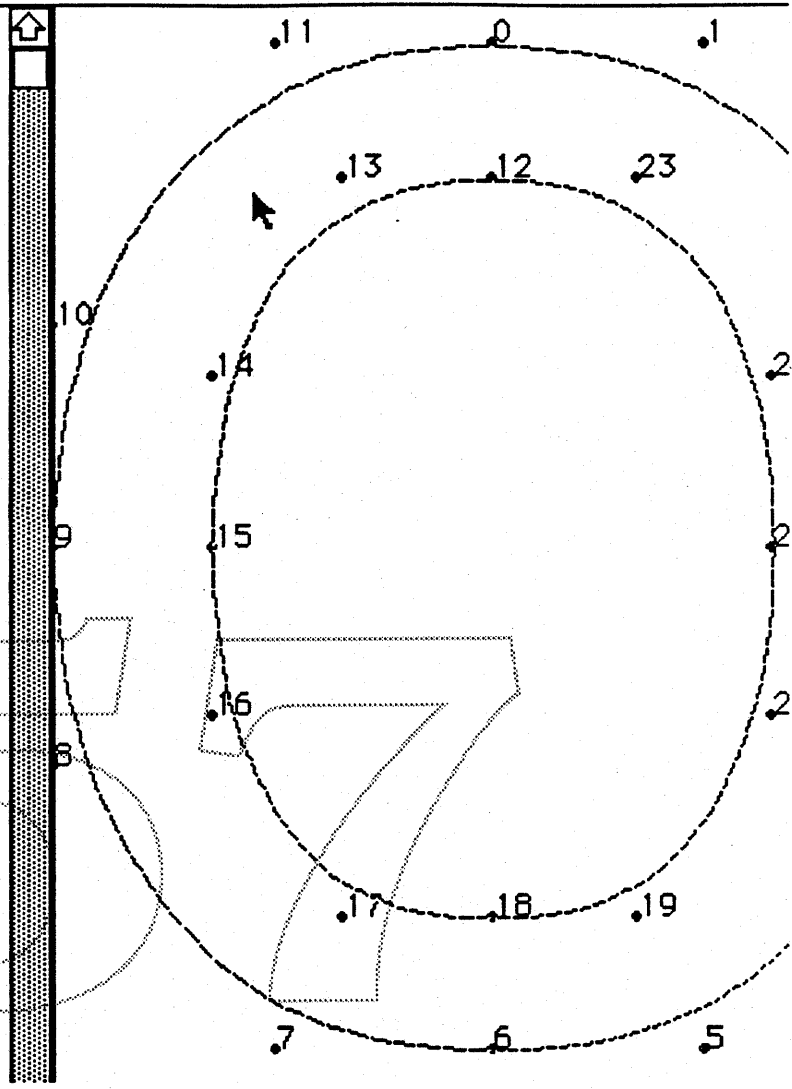
#### 7.0 Instruction Editing

After conversion to Quadratic B-Splines the outline font data will be examined on a character by character basis for anomalies in character shapes and size, starting measurements will be verified against Typeset masters for Times New Roman and Helvetica. Metrics will be checked against Adobe ROM listings and asymmetries will be adjusted where possible. Verification of control point data will include forcing all points to x and y extrema as required. Instruction editing will begin when all digital masters meet the required format. In the following example points 5, 6 and 7 are y extrema points and should be identical. The same is true of points 0, 1 and 2 ...etc.

```

AW = 1139
LSB = 78
0: x = 492, y = 1108
1: x = 730, y = 1108
2: x = 985, y = 790
3: x = 985, y = 540
4: x = 985, y = 292
5: x = 730, y = -27
6: x = 492, y = -27
7: x = 254, y = -27
8: x = 0, y = 292
9: x = 0, y = 540
10: x = 0, y = 790
11: x = 254, y = 1108
12: x = 492, y = 958
13: x = 328, y = 958
14: x = 178, y = 732
15: x = 178, y = 540
16: x = 178, y = 348
17: x = 328, y = 123
18: x = 492, y = 123
19: x = 656, y = 123
20: x = 805, y = 348
21: x = 805, y = 540
22: x = 805, y = 732
23: x = 656, y = 958

```



Each character in a font contains its own set of instructions. Instruction editing can begin with an automated MPW tool which applies the base level instruction set to each character based on an analysis of like character elements, (i.e., horizontal and vertical minimum and maximum stroke-weights, cap, descender, ascender and x-heights as well as overhangs for each height measurement). These consist primarily of "direct" and "indirect" absolute and relative moves (MDAP, MIAP, MDRP and MIRP)<sup>2</sup>. The global values for each of these elements is stored in the Control Value Table in two forms, as a fraction of the master resolution of 2048 upem or in the scaled size displayed in the preview window of the Instruction Editor.

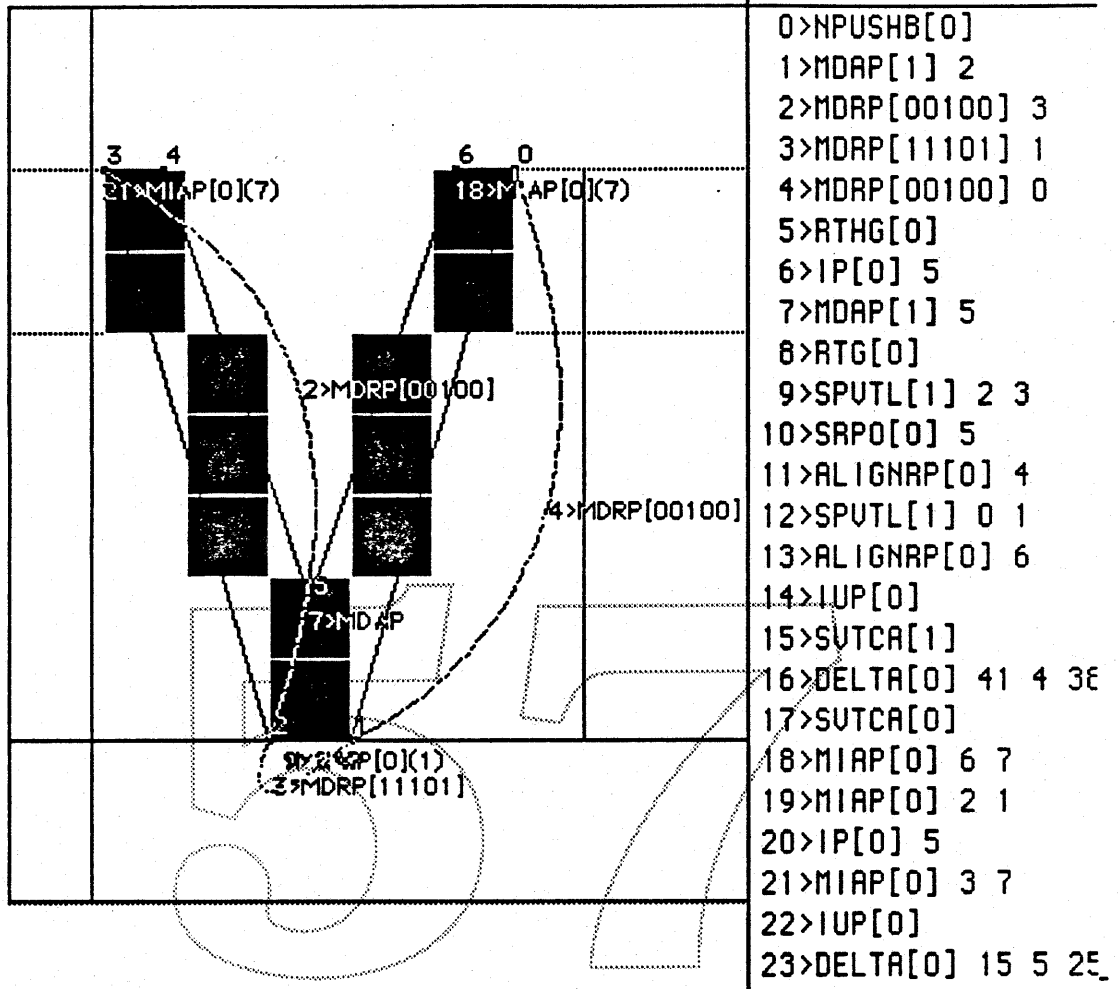
2. See Font Instruction Set and Interpreter SubERS - Appendix 17.2.2

H 1.0A88•CVT		
	Raw	Scaled
0>	-29871	10240.00
1>	0	11.59
2>	-27	757777.21
3>	-27	8.87
4>	-27	424977.21
5>	1081	424977.45
6>	1109	424977.46
7>	1493	426001.20
8>	1519	426001.45
9>	1462	0.00
10>	1489	0.00
11>	1493	0.00
12>	1521	0.00
13>	-420	0.00
14>	-437	0.00
15>	-27	0.00

Note: The scaled values in this example are incorrect and will display correctly once the editor is handed the scaled values by the glue code.

This table can be added to or changed based on fine-tuning of measurements for a given font. There is one Control Value Table for each font. The vectors can be set to either axis and all subsequent instructions will correspond to that axis. Diagonal strokes are controlled via the "freedom" and "projection" vector settings or Align Angle Instruction.





These type of controls are unique in instruction based scaling techniques. Additional global controls such as changing the % of x-height relative to the Cap Height can be obtained using the PreProgram. The following example exhibits the use of instructions in the Control Value Table used to control the % of x-height relative to the Cap height for the Helvetica font.

```

H 1.0A88•PreProg
0>NPUSHB[0]
1>SCES[0] 0
2>SUTCA[0]
3>MIAP[1] 4 11
4>MIAP[0] 3 7
5>MIAP[0] 2 9
6>MIAP[0] 1 5
7>MIAP[1] 0 1
8>SRP1[0] 0
9>SRP2[0] 4
10>IP[0] 1
11>IP[0] 2
12>IP[0] 3
13>MDAP[1] 1
14>MDAP[1] 2
15>MDAP[1] 3
16>RC[0] 0
17>WCUT[0] 1 (*)
18>RC[0] 1

```

Values in this table effect the entire font. Function calls can also be used to compress frequently used instruction sequences into a single instruction using the PreProgram as a type of look-up table.

The Instruction Set is an extremely powerful language and as such requires and intimate knowledge of its syntax and a robust approach to its use. The greater the knowledge of the language the greater and more economical the programming possibilities. All instructions are used by the Interpreter to establish rules for distorting the character during scaling. For an indepth description of each instruction see the Font Instruction Set and Interpreter SubERS attached as Appendix 17.2.2 of the Bass ERS.

## 8.0 Screen Fonts

With enough knowledge of the Instruction Set, instructions could be applied to obtain adequate scaling to any size, at any resolution. An alternate approach was developed to aid in the development of fonts at very low sizes in very low resolutions (i.e., 9• - 24• at 72 dpi.). This instruction is called the Delta Exception instruction and allows bits to be arbitrarily turned on or off by selectively moving a control point on a given character, at a given size. Where the

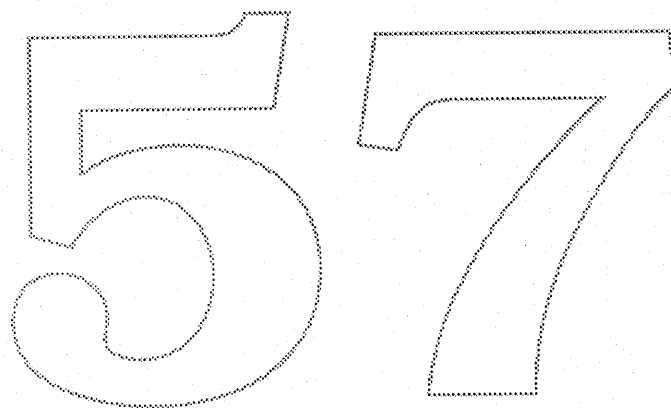
Size	Move	Point	DELTA
5	2	44	<input type="button" value="Recalc"/>  <input type="button" value="OK"/>  <input type="button" value="Cancel"/>
6	2	44	
6	2	32	
7	-4	8	
7	-4	9	
7	-3	13	
7	-2	4	
8	-2	4	
14	-3	39	
15	2	44	

first digit represents the size + 9 (14), the second digit represents the distance required to move the control point to turn a bit on or off as 2/8 of a bit and the third digit represents the point number (44). All sizes can be included in a single delta instruction for a single character in a given axis and are placed in ascending order by the Instruction Editor.

## 9.0 Contingency

As a contingency and for the protection of project goals, a VS Translator has been developed which converts fonts hinted in URW VS format to the Bass format. For a detailed description of this translator, see Appendix 17.2.7.

## 17.2.6 Outline Font Data Protection SubERS



## ERS - OUTLINE FONT DATA PROTECTION

### INTRODUCTION

#### Purpose of This Document

This document describes procedures that Apple could use to protect a typeface manufacturer's digital font data from being copied. The document also describes the costs of implementing these procedures in terms of degradation of functionality and performance of Macintosh applications. Export licensing problems that may result from implementing these procedures are also discussed.

No decision has yet been made to implement any of the procedures described in this document. There are strong engineering reasons to not implement any protection procedures. However, outline fonts have traditionally been kept secret and protected by encryption. It may be necessary to provide some protection measures to persuade typeface manufacturers to make their fonts available for the Macintosh.

#### Product Definition

Two products are described in this document. The first product is an SFNT Protection program to be delivered to typeface manufacturers who want to have font protection. This program will convert a standard version 1.0 SFNT file into a Protected SFNT file in which the outline coordinates and/or grid fitting instructions will be encrypted.

The second product is a set of procedures to be integrated into Macintosh Font Management System software that will decrypt encrypted files and discourage reverse engineering attempts to steal the data from Macintosh memory after the data have been decrypted.

### PROBLEM STATEMENT

#### Basic Need

Typeface designs are valuable intellectual property, but United States copyright law does not provide for their protection. The grid fitting information that is used to improve the rendering of an outline at a particular size may also not be protectable in the legal system. Therefore, it is desirable to provide technical methods of protecting outlines and grid-fitting information so that Royal typefaces can not be easily copied.

In the past, digital typeface data have only been used in closed systems such as typesetting machines and postscript printers. The data have always been encrypted in these environments, making it difficult for anyone to copy the outlines. Because of this tradition, typeface manufacturers also expect their outline font data to be protected when used in Macintosh applications. A typical paragraph proposed by a font manufacturer as part of their contract with Apple is: "Apple shall encrypt XXX Outline Fonts in Bass File Format and in any other Derivative form so as to prevent usage of the XXX Outline Fonts except with Apple system software designed to decrypt the XXX Outline Fonts for purposes of display and printout only. The original digital data of XXX Outline Fonts licensed by XXX to Apple and all Derivative forms of the XXX Outline Fonts shall be Confidential Information which Apple shall not disclose nor allow to be disclosed (e.g by software tools, documentation, or other means) to third parties in clear (unencrypted) form. ... "

The traditional encryption protection of outline fonts is not well suited to the Macintosh for two reasons:

1. Typography should be completely integrated with graphics. Many Macintosh applications need to use text in creating graphics images. For example, a user might want to run text along an arch or some other free flowing curve. To do this properly, the graphics application program must have access to the character outlines so that they can be properly rotated and distorted before being converted to bitmaps. But this character outline information is exactly the data that the manufacturers have traditionally kept secret.

In other words, providing complete functionality to graphics applications requires that the Macintosh system software provide toolbox routines that allow applications to obtain font outlines. Therefore, anyone can write a simple application that copies the outline data for any font.

2. Macintosh software is accessible. There is no provision to keep any part of the Macintosh operating system inaccessible to the owner. While, the source code for system software is not published, powerful debugging tools exist that can be used to inspect all parts of memory and to trace the execution of all parts of the system software including the system ROMS. To copy an encrypted font, a thief need only determine the point at which a character is decrypted and the place in memory where the decrypted information is stored.

Traditional cryptographic means can be used to encrypt outline font files and thus protect the data up to the point it is read into the Macintosh. At that point, the data is vulnerable to attack via examination of the font rendering software which is much easier than cryptanalysis of the encrypted fonts.

### **Educating Typeface Manufacturers**

Attempting to protect outline font data will cripple graphics applications (including Apple's own future graphics software). Protecting the data against a programmer skilled in reverse engineering software is almost impossible without adding hardware to the Macintosh which would require retrofitting all existing Macintoshes before they could use outline fonts. Therefore, these facts are being explained to manufacturers so that they can rationally calculate the tradeoffs involved in publishing unprotected fonts.

Basically, each manufacturer must decide if the large market for unprotected fonts that work in all Macintosh applications is worth the potential loss of proprietary data to unscrupulous competitors. It appears that several major manufacturers may be willing to accept the idea that they should publish their fonts in unprotected form.

If most manufacturers are unwilling to publish completely unprotected outline fonts, some compromise protection plan will need to be implemented. For example, it may be possible to protect the grid-fitting instructions attached to each character, but not the actual outline coordinates. Or a decision may be made to provide graphics applications with bitmaps for rotated and stretched characters but not actual master outlines.

### **Protection Levels**

If it becomes necessary to protect outline fonts from copying, three levels of protection can be visualized. The minimum level of protection that can be provided is encryption of the parts of an SFNT file that contain the outline coordinates or the grid fitting instructions or both. A second level of protection involves making it difficult to use examination of the system software to copy the grid fitting instructions. The third level of protection attempts to protect the actual coordinates of font outlines from being copied.

## File Encryption

Encryption methods that will encrypt any file to any required degree of safety are well understood. In particular, a standard encryption method called DES (data encryption standard) can be used. However, DES and similar methods that are normally used to protect financial transactions, diplomatic messages, etc, are very slow unless special hardware is used.

Techniques such as DES are not required to encrypt SFNT files. Unlike natural language messages, ASCII encoded computer programs, and other highly formatted files, the SFNT glyph data table has very low statistical redundancy. The "alphabet" of this table is large and almost all combinations of "letters" are meaningful. Furthermore, the encryption key can be different for each font.

Therefore, the glyph data can be encrypted using a simple permutation cipher on a table of 256 bytes. This encryption method allows very quick deciphering and will be very resistant to cryptanalytic attack, providing that a different permutation cipher is used for each font.

Using a different cipher for each font requires that the key to the cipher be transmitted from the typeface manufacturer to Apple, and that the key be known to a Macintosh system that is going to use the enciphered font. The traditional method of key management would require each manufacturer to register each font with Apple; Apple to securely transmit a key to the manufacturer to be used for that font; and for the key to be included in the system software of a Macintosh before the font could be used.

A better solution is to use the Public Key Exchange method invented by Whitfield Diffie and Martin Hellman. This method allows the manufacturer to choose a random key for the font and to transmit a one-way function encoding of the key in the SFNT itself. The critical element of this method is that the information about how to encode the keys is completely public, but it does not provide information about how to decode the key; this is known only to Apple, and is incorporated into the Macintosh System software.

The use of the Public Key Exchange method to transmit the key for each font obviates any need for font registration, record keeping, etc. The right to use the method must be licensed from the patent holder, which is Stanford University.

## Barriers to the Copying of Grid Fitting Instructions

Instead of attempting to cryptanalyze a protected font, a potential thief may decide to mount a "reverse engineering" attack on the system software. The purpose of this attack is to either find and understand the code that decrypts the font file, or to find the time and place that the instructions have been decrypted and to copy them at that point.

Because Macintosh system software source code is not published, the thief must examine the binary code stored in memory and disassemble it into meaningful procedures. Many powerful tools are available to assist in this process, including the debugging tools MacsBug and SADE provided by Apple.

A reverse engineering attack can be static or dynamic or both. A static attack is mounted by using a disassembler on the object code to create pseudo source code. The programmer can then "walk through" the source code, attempting to understand what it does, how the data are structured, and so on. A dynamic attack uses a debugger to actually step through the instructions executed in accomplishing a particular task.

Barriers against a static attack include:

1. Spreading the functionality of critical procedures across many program modules. This requires a great deal of jumping back and forth from page to page to follow the flow of task execution.
2. Using intertwined goto branching so that the flow of program control is very difficult to follow.
3. Introducing many false variables that ultimately have no effect. The thief must keep track of these variables and what happens to them until he realizes that they are inconsequential.
4. Creating thousands of possible paths through the code. If many branches are introduced, the thief is forced to examine many paths through the code regardless of whether the paths are ever taken in the actual task execution.
5. Including potential calls to almost all other parts of the system code. By including potential calls to other parts of the system code, the thief is forced to examine all of that code. The thief can not ignore these calls if the code is designed so that a few of them actually do important parts of the decryption task.

Barriers against a dynamic attack include:

1. Using very long nonrepetitive sequences of instructions. Obviously the more instructions that need to be stepped through, the more difficult it is to comprehend how the program works. Instruction paths can be lengthened by introducing procedures and instructions that are irrelevant to the task function, however this obviously decreases the execution speed of the task.
2. Creating equivalent instruction sequences to do the same task. If more than one instruction path can be used to accomplish a task and the choice of path is made dependent upon environmental variables such as the time of day, then the same data will not always be processed by the same instruction path, thus increasing the difficulty of charting the flow of program control.
3. Side effects programming. An example of an ideal side effect is a procedure that stores a value in a register of some peripheral chip but the action is inconsequential to the purpose of the procedure and it is completely undocumented. Knowledge of this information allows the creation of a procedure that uses that value without someone being able to trace what was stored there in the first place.
4. Using anti-debugger execution sequences. The selection of the correct path through an instruction sequence can be made dependent upon instructions being executed very close together in time, so that a debugger using a trace function will slow the procedure down and cause an incorrect sequence of instructions to be executed. Using this barrier can cause problems in an environment where interrupts can have lengthy service routines, so it should probably not be used as part of general system code.
5. Disabling debuggers. Debuggers such as MacsBug and SADE use the TRAP command to set breakpoints. When the 680xx gets to that TRAP instruction an exception is generated and an exception routine starts executing at a special memory address. The address of the debugger is put in that memory address so that it gains control at that point. If the procedure being spied upon changes the memory address so that it no longer points at the debugger, the debugger can not gain control.

The disabling of debuggers during decryption may cause problems to legitimate developers attempting to debug their applications and may also cause problems for the manufacturers of the debuggers that are disabled by the code.



Any of these barriers can ultimately be surmounted by a persistent thief. The degree of protection is probably proportional to the resources put into erecting the barriers. However, there are disadvantages to implementing these barriers and the cost of the disadvantages may also be proportional to the degree of protection that is created.

The main disadvantage of these techniques is that system maintenance is made more difficult. The Macintosh operating system software is already very large and complex. Deliberately introducing convoluted code to make it even more complex may be very unwise. As noted above, some of these techniques may also impede legitimate debugging efforts of legitimate developers. These barriers may also impose a substantial degradation in execution speed. Because the degradation occurs only the first time that a character is rendered at a particular point size, it may be reasonable to pay a significant time penalty for protected fonts in order to obtain added security; however, it is crucial that there be no effect on the execution speed for unprotected fonts.

### **Barriers to the Copying of Outline Coordinates**

The anti-reverse engineering tactics discussed in the previous section can also be used to impede copying of character outlines. An additional level of protection can be created through the use of distorted outlines. To create distorted outlines, the manufacturer first creates a standard outline and grid fitting instruction set for each character in the font. Then for each character, one or more of the outline points is deliberately moved to an incorrect position. To compensate for this distortion, instructions are added to the grid fitting instructions to move the point back to its correct position prior to the point being used for grid fitting.

This distortion must be done very carefully to account for movements of other points that may be affected by the movement of the distorted points. However, the end result is a new distorted outline and instruction set that provides the same grid fitted outlines as did the original outline, but the coordinates of the original proprietary outline are concealed. Even if a thief can obtain the distorted outlines and grid fitting instructions, each character will have to be laboriously analyzed to reconstruct the original master outlines.

The major disadvantage of using distorted outlines is the extra effort that would be required of manufacturers in preparing their fonts, although it is possible that some kind of distortion could be automatically introduced at the time the file is encrypted.

The major disadvantageous consequence of protecting font outline coordinates is that the outlines can not be passed to graphics applications that need to use text as part of the graphics. Thus, any plan to implement protection of font outlines must include a plan for responding to such applications requests and a plan for substituting some other kind of text rendering such as bitmaps or outlines from substitute fonts.

### **National Security and Export License Requirements**

If Apple provides any protection at all for outline fonts, the SFNT files will need to be encrypted, and every Macintosh will need to have decryption software as part of its operating system software. This could result in restrictions on the export of Macintoshes because any product that contains file decryption software must be approved for export by the U.S. government.

It is possible to export software with decryption capability. However, to do the exporting under normal Commerce Department licensing procedures, the software must first be approved as not compromising

National Security interests. While, several U.S. government departments have votes on this decision, the major technical vote is cast by the National Security Agency.

It appears that NSA is most concerned about preventing the shipment of software that provides a general capability to do very good encryption and decryption of files. In particular, software that implements the Data Encryption Standard (DES) is restricted; software that implements the RSA public key encryption technology with large keys (greater than 256 bits) is also restricted. On the other hand, NSA is aware of the problems that their policies may pose for American competitiveness, and it appears that they want to cooperate in finding solutions that meet the needs of American companies.

Apple will need to either submit the decryption algorithms and implementation code to NSA for approval, or apply for a Commerce department ruling that the decryption capability is incidental to the purpose of the software and does not require approval. In either case, the likelihood of approval is very high because of the following factors:

1. The permutation cipher used to encrypt the glyph data table is not related to DES, and can be broken relatively easily by an agency such as NSA.
2. The Public Key Exchange procedure used to transmit the cipher key uses a very small key (about 96 bits).
3. The Macintosh software will contain only decryption software and not encryption software.
4. The encryption procedures are specially for protection of fonts and would have little usefulness for encrypting natural language messages or other kinds of files.
5. The decryption procedures are completely user transparent. The owner of the Macintosh need not even know the decryption capability exists. He will be given no access to the decryption capability, indeed, every effort will be made to prevent him from having such access.

#### Hardware Requirements

In order to receive Commerce department approval of the decryption software, it is necessary that it be for used on computers classified as less than 42 megabits/second ( a commerce department metric). Since the Macintosh II exceeds this classification, it is important that the software also run on other Macintoshes such as the Plus and standard SE.

### IMPLEMENTATION OF FONT PROTECTION

#### Protection Decision

No determination has yet been made whether any font protection will be required in order to persuade a sufficient set of manufacturers to provide their digital fonts for the Macintosh. Nor has the level of protection that would be provided been determined in the event that some protection is required.

Therefore, no attempts have been made to introduce the kinds of anti-reverse engineering modifications of the system code that were described in the first section.

#### File Encryption and Decryption

If any degree of protection is required, it will be necessary to encrypt and decrypt the glyph data table in the SFNT file. The specifications for doing this are as follows:

### Encryption Program

Apple will provide manufacturers a program to convert a standard SFNT file into a Protected SFNT file. The protected SFNT file will have exactly the same format as the standard file, and it will differ only in that the grid fitting instructions and/or the outline coordinates in the Glyph Data Table will be encrypted and the Encryption Table will be filled in with a transformation of the encryption key.

The glyph data will be encrypted using a single permutation cipher. A 96 bit random number R will be generated by timing many asynchronous events such as the intercharacter typing intervals of the user. R will be used to calculate a key K according to the formula:  $K = A^R \text{ modulo } P$ , where A and P are 96 bit numbers contained in the Encryption program and  $\wedge$  is the symbol for exact exponentiation. That is,  $A^R$  stands for A multiplied times itself R times. Calculating this product modulo P means that the very large number  $A^R$  is reduced to a 96 bit number by dividing it by P and using only the remainder.

The key K will in turn be used as a seed to a byte shuffler that will create a shuffled table containing a permutation of the numbers 0...255. This permutation table P maps each of the numbers between 0 and 255 into some other number between 0 and 255. The table can be used to encrypt the glyph data table S into an encrypted table X according to the following rule:

$$\begin{aligned} X(0) &= P(S(0)) \\ X(i) &= P(S(i) \text{ XOR } X(i-1)) \end{aligned}$$

To decrypt the glyph data, the Macintosh System must have access to the key K so that the corresponding Permutation table can be generated and used to reconstruct the original S data. K will be transmitted to the Macintosh by encrypting it using the Diffie-Hellman Public Key Exchange method.

This method relies upon the fact that discrete exponentiation modulo an integer is a one-way function for large numbers. That is, if we calculate the number

$$Y = X^V \text{ modulo } P,$$

then it is very difficult to recover V if you only know X, Y and P.

To encrypt the key K, we calculate the number  $E = B^K \text{ modulo } P$  where B is another number included in the Encryption program. E is put in the Encryption table. The secret to the decryption lies in the fact that the numbers A and B are related by the formula  $A = B^D \text{ modulo } P$  where D is a number carefully hidden in the Macintosh System.

By the power law of exponents, we have:

$$A^R = (B^D)^R = B^{(D \cdot R)} = B^{(R \cdot D)} = (B^R)^D = E^D$$

so to recover the key  $K = A^R \text{ modulo } P$ , we simply calculate  $E^D \text{ modulo } P$ ; but this can only be done by someone who knows D which is known only to the Macintosh System. Although the numbers A, B and P can be discovered by reverse-engineering the Encryption Program, this information does not allow the thief to calculate D (without spending months of computational time) provided that A, B and P are carefully chosen very large numbers.

### SFNT Verification

The Encryption program will also provide a verification mode. In this mode, the manufacturer will specify a point size and the program will create bit maps from both the original unprotected font and the encrypted font for every character. The bit maps will be compared and any discrepancies will be reported. When running in this mode, the program will use the normal Macintosh System software to create the bit maps, so that no extra vulnerability is created for the decryption methodology.

### Public Key Decryption

When an application selects a protected font, the first step will be recovering the encryption key K and using it to create the permutation table needed to unscramble the glyph data. This will require modifying STDTXT at the point that it calls the font manager.

The first step is to get the key K by calculating  $E^D$  modulo P. This is a time consuming calculation and it will probably be desirable to provide some precalculation by having the Encryption Program calculate  $E^2 \text{ mod } P$ ,  $E^4 \text{ mod } P$ , ...  $E^{(2^{40})} \text{ mod } P$  and including these in the Encryption table in addition to the number E. If D is chosen to be equal to the sum of about 20 of these powers of 2, then the calculation of  $E^D$  can be reduced to about 20 multiprecision multiplications. By keeping the choice of which 20 numbers are used secret, exhaustive search of all choices of 20 numbers out of the 40 precalculated numbers is still kept computationally unfeasible.

Once K is calculated, it will be used as a seed in the same byte shuffling routine as was used by the Encryption Program to create the permutation table. The inverse of the permutation cipher will then be calculated: that is G(i) will be calculated so that  $G(P(i)) = i$ . This table can then be used later to recover the original glyph data.

### Character Unpacking

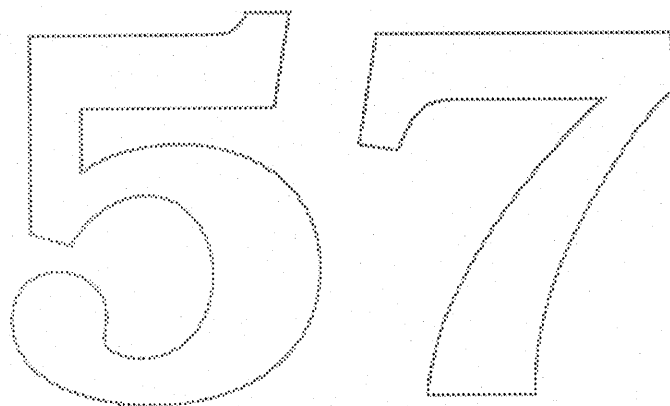
The first time that each character in a font is used by an application, it must be grid fitted and scan converted by the BASS software. The bitmap can then be cached in case the character is used again. When the font is protected, the glyph data for the character must be decrypted. Using the inverse permutation table G, the original glyph data can be recovered from the encrypted glyph data table X by the rule:

$$\begin{aligned} S(0) &= G(X(0)) \\ S(i) &= G(X(i)) \text{ XOR } X(i-1) \end{aligned}$$

### Prototype Code

Procedures have been written in C to do most of the multiprecision arithmetic, byte shuffling, etc required to implement public key encryption and decryption of SFNT files. Timing measurements indicate that decryption would not significantly degrade execution speed on Macintosh II computers.

## 17.2.7 VS Translator SubERS

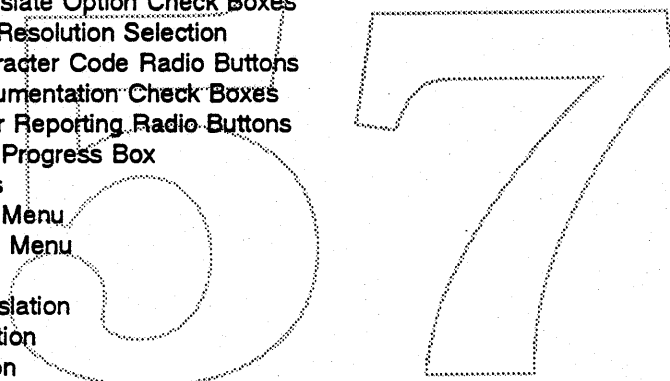


# USER MANUAL FOR TranVS

## A Program to Translate from VS format to ROYAL format

Richard Becker  
January 19, 1989

- I Introduction
- II User Interface
  - File Menu
  - Dialog Box
    - Action Buttons
    - Translate Option Check Boxes
    - Em Resolution Selection
    - Character Code Radio Buttons
    - Documentation Check Boxes
    - Error Reporting Radio Buttons
    - The Progress Box
  - Other Menus
    - Edit Menu
    - Help Menu
- III Algorithms
  - Outline Translation
  - Hint Translation
  - Point Deletion
  - Peak Movement
  - Curve Normalization
  - Em Resolution
- IV System Requirements
- V Appendices
  - A. URW File Format for VS Digital Typeface Description
  - B. Apple File Format for SFNT Digital Typeface Description
  - C. Warning Messages
  - D. Changing File Attributes
  - E. File Format for International Character Code conversion



## ERS - VS TRANSLATOR

### Purpose of this Document

This document explains the need to translate descriptions of typeface characters that are in URW VS format to descriptions that are in APPLE Royal format. A Macintosh application program called TranVS has been written to do this translation. The user manual for TranVS (minus appendices) is attached as the functional specifications for this ERS.

### Product Definition

Royal is Apple's new outline font technology. Every character in a font is described by the coordinates of points that trace an outline of the character and a set of instructions that cause the outline to be correctly converted to a bitmap at any scale factor. Without the "intelligence" supplied by the instructions, scaled outline fonts generally do not provide pleasing bitmaps.

Many typeface manufacturers have digitized outline fonts and provided their own methods of doing intelligent scaling. In particular, URW UNTERNEHMENSBERATUNG of Hamburg, Germany has many typefaces in a format called VS (for vector scaling).

TranVS is a computer program that translates a Macintosh file containing a font in URW VS format to a file containing the font in Apple Royal format. The program translates the outline description from URW's arcs and straights description to the Royal quadratic B-spline format. It also translates the scaling "hints" in the VS format to the grid fitting instructions of the Royal format.

### Hardware and Software Compatibility

The program will run on any Macintosh with at least 1 Megabyte of memory. It runs as an application and has been tested under version 6.0 of the System software.

### Intended Users

An authorized user of a particular URW font can use TranVS to create an equivalent Royal font either for use directly in Macintosh applications or as a font to be further edited by a typeface designer and eventually used in Macintosh applications.

### Performance

A typical 220 character font can be translated in about 1 minute on a Macintosh II computer.

## I. Introduction

TranVS is a Macintosh Application Program that translates descriptions of typefaces in URW VS format into descriptions in Apple Royal SFNT format. TranVS can be used to translate oriental and Arabic alphabets and symbol sets as well as the standard Roman alphabets.

The program operates on a Macintosh file containing a description of a font as created by the URW (URW Unternehmensberatung) type manufacturing company in their VS (vector scaling) format.

Two major transformations are made in this translation. In the VS format, the character outlines are described by a series of straight lines and arcs of circles. The arcs and vectors must be transformed to a description in terms of quadratic splines which are used in the SFNT format.

The second transformation involves supplemental grid-fitting information. URW provides information about each character that can be used as "hints" in deforming the character to fit on a particular pixel grid. This information includes things like the coordinates of lines that ought to be forced to be on a grid boundary, identification of the points that mark the maximum excursion of a curve, points that are peaks, etc. This supplemental information must be transformed to a grid-fitting program using the BASS instructions.

The translation results in the creation of three output files. One of the files contains the the SFNT description of the typeface. This file can be transformed into an SFNT RESOURCE by using the utilities described in Appendix D. The other two output files provide documentation about the typeface and its translation.

## II. User Interface

The user controls the translation of a Font with the File Menu and a large modeless Dialog Box that appears on the display when TranVS is launched and remains on the screen until the user quits the application.

### The FILE menu:

QUIT exits the TranVS program.

CLOSE is not enabled in TranVS. It is provided for the use of Desk Accessories.

OPEN requests the user to select the file to be translated, e.g. Courier.VS This file should contain a VS description of a font as created by URW and described in Appendix A. The Type of the file must be 'DATA' (see Appendix D for a discussion of file types). TranVS attempts to open the specified file. If it is successful, it attempts to create 3 new files with the same name as the opened file, but with extensions .sfnt, .doc, and .err added to the file name. If the original file can not be opened, or the new files can not be created, the user is notified by an ALERT message.

[ NOT YET IMPLEMENTED - If the disk already has files with the same names as those that would be created, the user is asked whether he wishes to overwrite these files. If not, he should cancel the alert, quit the program, and rename the file that he wishes to translate. -]





R

**US Translator**

**Translate Options**

Do Not Delete Points

No Double Grid Rounding

No Curve Normalization

Em Resolution

**Character Codes**

ASCII  URW  International

**Documentation**

Font Summary

Hints

Instructions

Coordinates

**Error Reporting**

Full  Minimum

**Translate**

**Cancel**

**Progress:** File- courier.us -Open for Translation.

## The DIALOG Box:

The dialog box is shown in figure 1. It contains check boxes, radio buttons, and action buttons that control the operation of TranVS. It also contains a Progress box that displays the progress of the translation operation.

### Action Buttons:

After a file has been opened for translation, the **Translate** button starts the translation process. As each character in the font is translated, the name and code of the character is shown in the progress box. If TranVS is being run under MultiFinder, the translation can proceed in background mode, while the user is doing other applications in the foreground.

At any time, the user can stop the translation by clicking on the **Cancel** button. TranVS will stop translating the input file and create an SFNT file and documentation files for the set of characters that were translated prior to the cancellation.

The file containing the SFNT description of the typeface is named the same as the original VS file, but .sfnt is added as an extension. For example, translating the input file Courier.vs would generate the output file Courier.vs.sfnt. This file can be transformed into an SFNT RESOURCE through the use of the utility programs described in Appendix D.

### Translate Option Check Boxes:

Three control options are provided that actually change how the typeface is translated. In the translation of circular arcs to quadratic splines, it is necessary to introduce an additional control point for every arc segment in the original outline. However, under some circumstances the BASS scan converter will generate intermediate points from existing points. This generation can be anticipated and points close to those that would be generated by the scan converter can be removed. This reduction of points in the outline can be prevented by clicking on the **Do Not Delete Points** check box.

The VS format identifies outline points that are peaks, such as the vertical peak in V or the horizontal peak in <. The standard URW grid fitting method is implemented by moving vertical peaks to a point halfway between gridlines when moving in the X direction and to a gridline when moving in the Y direction, and vice versa for horizontal peaks. However, this algorithm can cause major distortions in diagonal stems at small point sizes. A better diagonal can often be created by using the algorithm: move vertical peaks to the nearest half way point or grid point in the X direction and do nothing in the Y direction, and vice versa for horizontal peaks. This is the default algorithm used in the translator. Clicking on the **No Double Grid Rounding** check box disables this improvement and uses the original URW Round-to-Half-Grid algorithm.

The VS format identifies curve beginnings, endings, and extreme values, although beginnings and endings are not always identified in closed curves such as o's. Implementation of the URW grid fitting method requires interpolating curve boundaries to be consistent with movement of straight stem lines, moving curve extrema to be on grid lines, and interpolating other points on the curve to be consistent with the movements of the surrounding curve boundaries and extrema. This procedure can result in deforming a character containing symmetric closed curves such as Courier O into an unsymmetric outline. This degradation can be avoided by requiring that whenever a character contains a curve with two opposite extrema (i.e. left and right or top and bottom) that all intervening points on the curve be moved in such a way that the points retain their original relationship to the two extrema. This improvement is the default option in TranVS, but it can be defeated by clicking on the **No Curve**

**Normalization check box.**

These options must be selected before the Translate button is clicked. During translation these controls are disabled and can not be changed.

**Em Resolution Selection:**

When the user selects a file to be translated with the OPEN command in the File menu, the header of the file is read, and the Em resolution used in the digitization of the typeface is put into the Em Resolution box. The VS file header actually specifies both an X-direction Em resolution and a Y-direction resolution. If these numbers differ, a warning is printed in the .err documentation file, and the X-direction is put in the box. This number is included in the SFNT header table and it is used by the BASS interpreter to calculate the scale factor used to transform the SFNT point coordinates to pixel coordinates at any particular point size.

At any time prior to the finish of the file translation, the user can type into the Em Resolution box a number that is different from the one specified by URW. This can be used to slightly shrink or expand the outlines to compensate for differences in the way that scan converters and grid fitting methods work. For example, one might specify an Em resolution of 1002 instead of 1000.

**Character Code Radio Buttons:**

Each character must be identified by a character code. VS file codes are either 16 bit URW codes or 8 bit Apple Extended ASCII codes. The URW code is partially described in Digital Formats for Typefaces by Peter Karow published by URW Verlag. The Apple Extended ASCII codes are shown in Inside Macintosh and in the documentation for the ROYAL font technology. These character codes must be translated into SFNT character codes which are either the 8 bit Extended ASCII code or the 16 bit Apple International Character code.

If the ASCII button is clicked, the VS file is assumed to contain ASCII codes, and they are used directly as SFNT codes. Values outside the range of 0 to 255 are flagged as errors. If the URW button is clicked, the VS file is assumed to contain URW codes and they are translated to extended ASCII codes according to a built-in table that includes the correct mapping for many Latin characters.

[ Not Yet Implemented - If the International button is clicked, character codes are translated according to an external file provided by the user. The format of this file is described in Appendix E. -]

**Documentation Check Boxes:**

During the translation process, TranVS creates a documentation file. The file has the same name as the file being translated, but an extension of .doc is added, e.g. Courier.vs.doc. The amount of information in this file is controlled by the 5 documentation check boxes. Any or all of these boxes can be checked in any combination. If none of the boxes is checked, no information is provided; however, a zero length file is still created.

If the Font Summary box is checked, information is provided about both the original VS file and the translated SFNT file. The header block of the VS file as described in Appendix A is printed. This includes the name of the Font, date of creation and typographical information like Cap height, x height, and so on. Summary information about the SFNT file includes the size of the file, the lengths and offsets of the SFNT tables, values of the Control Value table entries, and information such as maximum advance width, minimum right sidebearing, and so on.

If the **Hints** box is checked, the VS file stemblock information is printed for each character. This information includes the number of contours in each character, straight stem edges that should be placed on grid boundaries, outline points that begin and end curves, and so on.

If the **Instructions** box is checked, the BASS grid fitting instructions generated by the translator are printed. These are printed as a list of hexadecimal numbers rather than alphabetic mnemonics, so this option is mostly useful for detailed debugging of the translation algorithms.

If the **Coordinates** box is checked, all of the original VS information about each character is printed to the documentation file. This includes set width information, contour details, and actual coordinates of each outline point.

#### Error Reporting Radio Buttons:

TranVs also creates another text file that describes any errors encountered in the translation process. This file has the same name as the file being translated, but the extension .err is added, for example, Courier.vs.err. These errors include both fatal errors that stop the translation process and data integrity errors that occur because of inconsistencies or incompleteness in the original font description that may result in a poor translation but do not stop the translation process.

If the **Minimum** radio button is clicked, only very serious errors are reported such as a VS character has more contours than the maximum of 8 that are allowed in TranVS. If the **Full** radio button is clicked, all warnings are included. For example, if a contour contains a point identified as a serif beginning but there is not a corresponding point identified as a serif ending, the character number and point number will be printed out along with an identifying message. The possible warning messages and their likely causes are described in Appendix C.

#### The Progress Box:

The progress box shows the progress of the translation process. It shows the name of the file that is selected for translation, and then shows each character as it is translated. The information displayed is the position in the VS file, the character itself (providing that the character exists in the default font used by TranVS), and the hexadecimal character code in the SFNT file.

#### Other Menus

The **EDIT** menu is not enabled in TranVS. It is included so that Desk Accessories can use the menu if the application is run under Finder rather than MultiFinder.

The **HELP** menu provides online information about each of the controls, similar to the information in this document.

#### IV. Algorithms

This section describes the algorithms used to do the Font translation. The effects of control options are explained, and alternatives that were considered but not implemented are also discussed.

#### Outline Translation:

In the VS format, the outline of a character is described by a sequence of straight and circular

segments. Each straight segment is specified by the X and Y coordinates of the endpoints of the segment. Each circular segment is specified by the X and Y coordinates of the endpoints and by a chord height which is the distance from the chord connecting the two endpoints to the circle as measured along its perpendicular bisector. This information implicitly provides the radius of the circular segment.

In the SFNT format, an outline is represented as a sequence of straight line and quadratic B-spline segments. Each VS circular segment must be translated to a B-spline. This is done by adding a third outline point between the two endpoints of the circular segment. This point is not on the outline curve, but it controls the shape of the curve according to the parametric vector equation:

$$F(t) = (1-t)^2(1-t)A + t^2(1-t)B + t^2tC$$

where A and C are the circular segment endpoints and B is the inserted off-curve control point. The B-spline can not be exactly the same as the circular arc segment, so a decision must be made as to what feature of the circular segment to match in addition to matching the positions of the endpoints.

The solution used in TranVS is to choose the position of B so that the resulting curve has the same first derivatives at the endpoints as the VS circular arc. This has the desired feature that a long curve made up of many curved segments will have a continuous first derivative, rather than containing sharp corners.

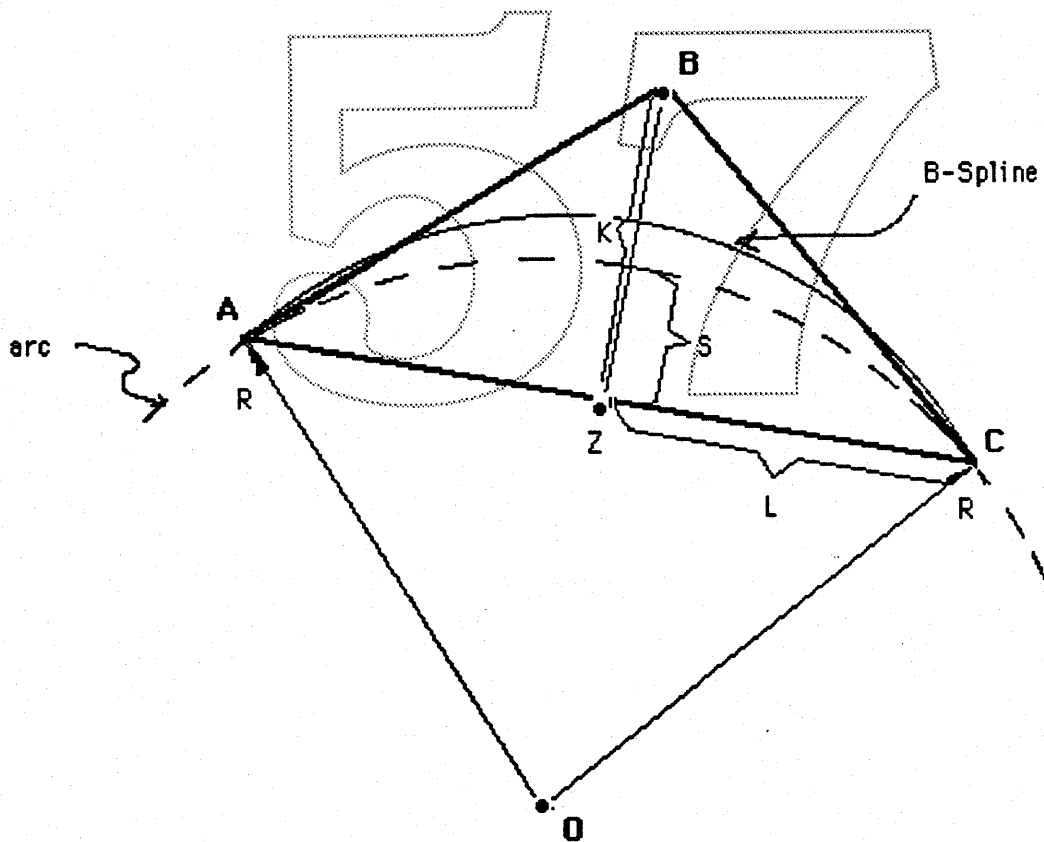


Figure 2 - Inserting Control Point B

As shown in figure 2, this requires that the point B be at the intersection of the two lines that are tangent to the circular arc at points A and C. In other words, B is at the apex of an equilateral triangle whose base is the line AC and whose height is K.

Because BZC and CZO are similar right triangles, K is related to L, R and S by the formula:

$$K/L = L/(R - S)$$

where R is the radius of the arc, and L equals (length of AC)/2. R is related to the chord height S and L by the formula:

$$R^2 = L^2 + (R - S)^2$$

Therefore, the height K can be derived from S and L by the formula:

$$K = (2*S*L) / (L^2 - S^2)$$

Other criteria for choosing the control point B can be imagined. For example, one might ask that the B-spline match the circular arc at its maximum excursion from the chord. This criterion is met by setting the distance K to 2\*S. Procedures for using this criterion are included in the source code and can be selected by changing the function calls delx1 and dely1 to delx2 and dely2 in the file decodeA.c and recompiling.

The final step in the outline translation is to delete excess points. This procedure is done unless the Do Not Delete option is checked. The delete procedure is detailed later in this section.

### Hint Translation

In the VS format, supplemental grid fitting information consists of point labels and stemblock data. Point labels are embedded into the actual image data of each character by reserving values greater than 120 or smaller than -120 as identifiers rather than vector increments. The labels that can be provided are: Start of Curve, End of Curve, Start and End of Curve, Vertical Curve Extrema, Horizontal Curve Extrema, Start of Serif, End of Serif, Vertical Peak, and Horizontal Peak.

Stemblock data consists of the coordinates of lines that ought to be placed on grid lines according to the URW grid fitting philosophy. These lines generally correspond to edges of straight stems in characters, although sometimes regularizing lines such as x-height are also included. The coordinates can be provided as absolute values, or as distances relative to a previous coordinate. In the latter case, the presumption is that the relative distance is the important variable to be grid fit, rather than the absolute value of the second line. The stemblock also contains the coordinates of curve extrema which can be provided either as absolute or relative values.

In some cases there is more than one stemblock, each stemblock providing information for a different set of contours. This generally happens with relatively complicated characters such as the percent sign.

Successful translation of the VS format requires the generation of a sequence of BASS instructions that will implement the intention of the URW grid fitting method for a particular character. This sequence of instructions is generated by going through the following steps for each stemblock; first for movements in the X direction and then for movements in the Y direction:

A. Mark all points that are inside a serif (but not the boundaries) as *untouchable*. Mark all points

that are inside a curve (but not the boundaries and not curve extrema) as *untouchable*. Mark all points that are not *untouchable* as *touchable*.

B. If the stemblock contains lines identified by absolute coordinates, generate instructions to move all *touchable* points on those lines to the nearest grid line. Mark each of points as *moved*.

C. If the stemblock also contains lines identified by relative coordinates, generate instructions to move all *touchable* points on those lines to the nearest integer number of pixels away from the corresponding absolute line. Mark these points as *moved*.

D. For each pair of relative and absolute lines, generate instructions to interpolate the positions of all points between the two lines provided that the points:

1. are *touchable*.
2. have not been *moved*.
3. lie on a contour containing points on the two lines.
4. are not a serif-start if moving in the X direction.
5. are not a serif-end if moving in the Y direction.

E. Find the stemblock line with the smallest absolute coordinate and generate instructions to move all points with coordinates less than the minimum line the same amount that the line was moved, provided that the points:

1. are *touchable*.
2. are not a serif-start if moving in the X direction.
3. are not a serif-end if moving in the Y direction.

F. Find the maximum stemblock line and generate instructions to move all points with coordinates greater than the maximum the same amount that the stem edge was moved, provided that the points:

1. are *touchable*.
2. are not a serif-start if moving in the X direction.
3. are not a serif-end if moving in the Y direction.

G. Sort the remaining stemblock lines and generate instructions to interpolate all remaining points between the two closest stemblock lines provided that the points:

1. are *touchable*.
2. have not been *moved*.
3. are not a serif-start if moving in the X-direction.
4. are not a serif-end if moving in the Y-direction.

H. If moving in the X-direction, generate instructions to move all serif-start points the same amount that their matching serif-end points have been moved. If moving in the Y-direction, generate instructions to move all serif-end points the same amount that their matching serif-start points have been moved.

I. If the stemblock did not identify any straight lines, generate instructions to mark all *touchable* points as *moved*. (but do not move the points).

J. Find all the curve extrema points identified in the stemblock by absolute coordinates and generate instructions to move these points to grid lines.

K. Find all the curve extrema points identified in the stemblock by relative coordinates and generate instructions to move these points to the nearest integer number of pixels from the corresponding absolute curve extrema.

L. If the No Curve Normalization box is not checked, generate instructions to implement curve

normalization. These instructions are explained later in this section.

M. Generate instructions to move peaks to grid lines or half-way between grid lines. The actual instructions depend on whether the **No Double Grid** box is checked. These instructions are described later in this section.

After all stemblocks have been processed, an IUP instruction is generated to interpolate each unmoved point between the the pair of moved contour points that enclose the point.

### Point Deletion

The outline translation procedure causes a new off-curve control point to be created for every circular arc segment in the original outline. As a result, the new outline may have many more points than the original. Some of these points can be deleted.

When the scan converter encounters two successive off-curve points, it generates a new on-curve point that lies exactly half way between the two off-curve points. These points are then used to generate the splines that outline the character.

The point deletion algorithm is applied to all 3 point sequences consisting of off-curve, on-curve, off-curve points. The same mid-line point is calculated from the off-curve points that would be calculated by the scan converter. If this point is very close to the existing on-curve point, the existing point is deleted.

Currently, the definition of very close is that the point is deleted if the distance between the calculated point and the existing point is less than 1/8 of the distance between the two off-curve points or the distance is less than 1/250 of the Em Resolution.

If the **Do Not Delete Points** box is checked, the point deletion algorithm is not applied.

### Peak Movement

Peaks can be made crisp by moving them so that exactly one pixel represents the peak instead of two adjacent pixels being both on or off yielding a flat peak. The URW grid fitting method does this by moving vertical peaks (as in v) so that the outline peak is on a horizontal grid line and exactly half way between two vertical grid lines. Similarly the outline is deformed so that horizontal peaks (as in >) occur on a vertical grid line and halfway between two horizontal grid lines.

While this deformation produces pointed peaks, it can cause major degradation of diagonal stems for coarse rasters. A reasonable compromise between not grid fitting peaks at all and over deforming the character is to generate instructions that move vertical peaks so that the peak is on a vertical grid line or halfway between vertical grid lines whichever is closest. Similarly horizontal peaks are moved so that they are on a horizontal grid line or halfway between two grid lines, whichever is closest. In this compromise algorithm, vertical peaks are not adjusted vertically at all, nor is the x coordinate of the horizontal peaks changed.

The instructions generated by this compromise algorithm seem to yield better characters for most standard fonts for English, than does the original URW algorithm. This algorithm is the default translation used in TranVS, but it can be replaced by the original URW algorithm by checking the **No Double Grid Rounding** check box.

### Curve Normalization



The original URW grid fitting method deforms curves according to the rule: Adjust the start of the curve, the end of the curve and extrema of the curve to be consistent with any straight lines that have been grid fit, then interpolate the other curve points to be consistent with the movement of the curve start, end and extrema.

For coarse rasters, this deformation can drastically degrade the relationship between parts of the curve. For example, a horizontal extrema that had a vertical value exactly half way between two vertical extrema may end up much closer to one or the other of the vertical extrema.

TranVS looks for curves that have at least two extrema of the same kind (vertical or horizontal). Instructions are generated for these curves so that points between the minimum and maximum extrema are moved to retain the same relationship to the two extrema that was present in the original outline.

This extra curve processing can be defeated by checking the **No Curve Normalization** check box.

### Em Resolution

The VS file header specifies an X direction Em Resolution and a Y direction Em Resolution. These numbers represent the degree of resolution used in digitizing the original master characters. For example, a Y direction Em resolution of 1000 means that 1000 different points could be distinguished between the lowest possible line that could be digitized and the highest line that could be digitized.

The BASS interpreter and scan converter uses the Em Resolution to calculate a scale factor to convert outline coordinates to display coordinates. The formula for this scale factor is:

$$\text{factor} = (\text{Display Resolution} / \text{Em Resolution}) * (\text{Point Size} / 72)$$

where Display Resolution is the resolution of the intended display device in dots per inch (e.g. 300 dpi for the Apple Laser Writer) and Point Size is the approximate size of the type face to be displayed (where 1 inch equals 72 points).

Typically the X direction and Y direction resolutions are the same. If they are not, a warning is printed and the X direction resolution is used for the Em Resolution of the SFNT file. This number is printed in the Em Resolution box whenever a new VS file is opened for translation. The user can change the size of the typeface that will be produced at a particular point size by typing a different number in the Em Resolution box.

### IV System Requirements

TranVS can be run on a Macintosh II computer with 5 megabytes of memory under version 6.1a2 of Multifinder or under version 6.1 of the Finder. It will also run [ - Not Yet Tested - ].

**LaserWriter 6.0 ERS  
Version 0.2**

**Jay Patel, x44905, MS 27-AJ  
Monday, January 16, 1989**

**57**

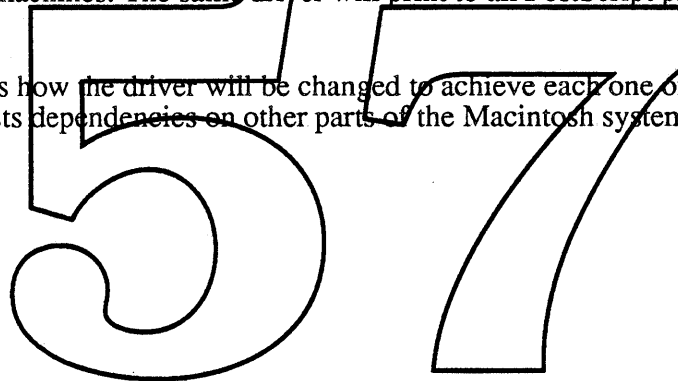
**Introduction**

This document describes changes planned for the LaserWriter driver version 6.0. The goals for this driver are: 1) to support color PostScript™ printers (currently only the QMS ColorScript™ 100), 2) to make sure that the driver is 32-bit clean, 3) to improve the fonts querying mechanism, 4) to support 32-bit color QuickDraw, and 5) to add support for the Kanji PostScript printer. Another goal of this driver is to fix bugs found in the last version (5.2) of the driver.

The User Interface of this driver remains the same as the previous driver except for two changes. The Page Setup dialog gets a pop up menu for additional page sizes (these could be user defined custom page sizes) and the Print dialog gets a set of radio buttons to select Color/Grayscale or black & white printing. Both of these changes are described in detail later.

This driver is expected to work on all Macintosh CPUs except Mac 512K and 512Ke. Although the driver does not rely on the presence of 128K or 256K ROMs, memory requirements might prevent it from running on 64K ROM machines. The same driver will print to all PostScript printers, color, monochrome, and Kanji.

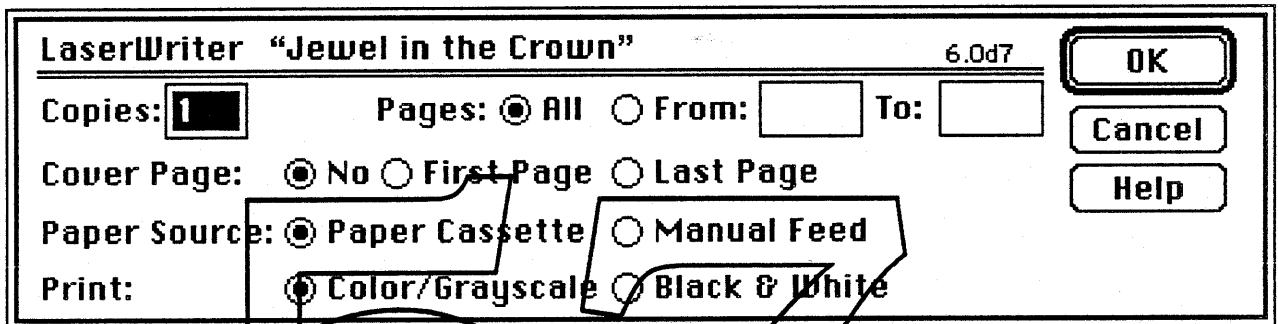
This document describes how the driver will be changed to achieve each one of the goals listed above. Where appropriate, it lists dependencies on other parts of the Macintosh system software.



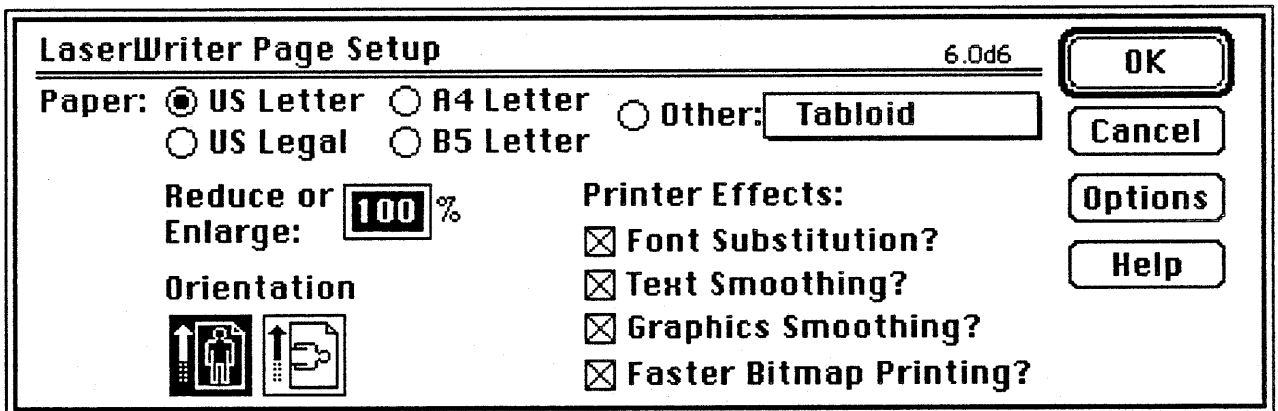
### User Interface Changes

There are two user interface changes planned for the LaserWriter driver 6.0.

In order to support color PostScript, the LaserWriter driver 6.0 will return color GrafPort to all applications when running on a CPU that has color QuickDraw installed. It is known that certain applications write directly into the GrafPort data structure instead of using the procedural interface provided by QuickDraw. It is possible that these applications will crash if given a color GrafPort for printing. For this reason, there will be two radio buttons to select the "Printing mode" in the Print dialog. These radio buttons will be labeled *Color/Grayscale* and *Black & White*. The default setting will be to print using *Color/Grayscale* because we want the third parties to fix their applications if they crash printing into a color GrafPort. The new Print dialog is as follows.



The imageable area of the QMS ColorScript printer for various paper sizes is different than the Apple LaserWriter imageable areas for the same paper sizes. This is expected to be the case for other printers as well. In order to support the QMS and other printers with different imageable areas, the LaserWriter driver version 6.0 allows users to choose from a variety of page sizes, in addition to the standard page sizes supported by the driver. These additional page sizes appear in a popup menu in the Page Setup dialog and can be selected much like the other standard page sizes. This document describes how to add new page sizes to the LaserWriter driver. The new LaserWriter Page Setup dialog looks as follows.



All the additional page sizes are stored in a resource of type "pgsz"( id -8192). The standard page sizes that the driver will ship with are **Tabloid**, **A3 Tabloid** and **No. 10 Envelope**. If the "pgsz" resource is

not present, the driver does not put up the popup menu and the radio control labeled "Other" is made invisible. The page sizes can be added using ResEdit. A ResEdit template for "pgsz" resource will be created and distributed with the driver.

The first word in the "pgsz" resource is the number of page sizes in the resource. This is followed by the page size entries. Each page size entry has following fields:

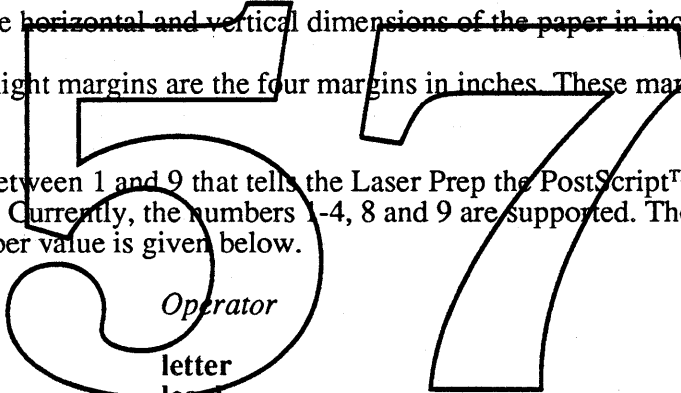
<i>Name</i>	<i>Type</i>
PageH	16.16 Fixed point number
PageV	16.16 Fixed point number
Top Margin	16.16 Fixed point number
Left Margin	16.16 Fixed point number
Bottom Margin	16.16 Fixed point number
Right Margin	16.16 Fixed point number
PageType	Integer
Name	Pascal String (first byte is the length)

PageH and PageV are the horizontal and vertical dimensions of the paper in inches.

Top, Left, Bottom and Right margins are the four margins in inches. These margins describe the imageable area on the paper.

PageType is a number between 1 and 9 that tells the Laser Prep the PostScript™ page type operator to be used for the print job. Currently, the numbers 1-4, 8 and 9 are supported. The page type operator used for each PageType number value is given below.

<i>PageType</i>	<i>Operator</i>
1	letter
2	legal
3	a4
4	b5
8	11x17
9	a3



This means that a page size that requires Laser Prep to use an operator other than those listed above cannot be defined. Note also that for user defined page sizes, the "Larger Printable Area" checkbox in the "Options" dialog of the Page Setup dialog has no effect on the page rectangle returned to the applications to image in.

### Support for Color PostScript

As stated earlier, when the *Color/Grayscale* button is selected in the Print dialog, the LaserWriter driver 6.0 will return color GrafPort to all applications when running on a CPU that has Color QuickDraw installed. Applications will image in this GrafPort instead of the old style GrafPort that they currently image in. At PrOpenDoc time, if the application has passed in a pointer to the storage for the port, the driver will check it's size before allocating a color GrafPort in that storage. If the size of that storage is as big as the structure for the old GrafPort, the driver will ignore this storage pointer and allocate a new

structure big enough to hold the color GrafPort and other fields for the driver's internal use. On CPUs where Color QuickDraw is not installed, the driver will return old style GrafPort to the application and print old QuickDraw colors to a color device. This means that color PICTs created on the MacII when used in a document on a Mac+ or SE will be printed in old QuickDraw colors.

LaserWriter 6.0 will send the same PostScript code to color and monochrome printers. The LaserPrep will have routines that will do the "right thing" on each device. The LaserPrep for this driver will be incompatible with previous versions and as always every machine on the network will have to upgrade to this new version.

Due to incompatibilities between Color QuickDraw and Color PostScript, not all documents printed to the color PostScript printer will be WYSIWYG. Some of the limitations are listed below.

PostScript supports only copy mode. None of the color QuickDraw transfer mode will be supported in this driver. The driver will use the copy mode for all of the Color QuickDraw transfer modes.

Color QuickDraw allows pixel patterns that can be of any color and of any size. This is not possible to do in PostScript. There are three types of Pixel patterns in color QuickDraw. Old-style patterns (patType = 0) are fully supported by the driver. Full-color pixel pattern (patType = 1) are supported in a limited way. The driver will treat a full-color pixel pattern as an 8x8 pattern and use this pattern to define a color screen at the printer. RGB pattern (patType = 2) will be supported by setting the color at the printer to be the requested RGB color in the pattern.

Color QuickDraw allows patterns to be drawn in any RGB color. For example, you can draw the QuickDraw *bricks* pattern or the *fish scales* pattern in any RGB color. This again is not possible to do in PostScript. In these cases the color will be matched to one of the eight colors (three additive, three subtractive, black and white) that PostScript can draw this pattern in. This scheme of matching an RGBColor to one of the eight colors will also be used to implement color QuickDraw pixel patterns described earlier.

There are other problems/limitations of the driver when printing in color to the QMS ColorScript 100 printer.

There is no legal tray for this printer. If you try printing legal documents to this printer, they will be clipped.

The LaserWriter driver *invert* option does not really make sense when printing in color. The driver will print the document on a black background if the *invert* option is selected.

Color pattern filled polygons from MacDrawII will not be filled correctly by the driver. This is because the driver does not have enough pattern information to do things right.

The driver should be thoroughly tested to make sure that it prints correctly on both color and monochrome postscript devices. Color bitmaps (pixMaps) in particular should be tested to make sure that they print in color on the QMS and in gray scale on monochrome printers. This driver must be tested on all versions of PostScript interpreter.

### 32-bit Clean Driver

The LaserWriter driver code modules will be examined to make sure that it is 32-bit clean. This driver is expected to run under 32-bit clean ROMs. The driver will have to be tested thoroughly with different applications running on a machine (a MacII) with 32-bit clean ROMs.

### Improved fonts querying mechanism

The LaserWriter driver will try to improve the time taken to inquire the printer for its fonts. This involves receiving multiple font names per AppleTalk packet and changing the algorithms and data structures used to store and retrieve font names during the processing of the document.

### Support for 32-bit Color QuickDraw

The LaserWriter driver will correctly handle printing of documents in 32-bit CQD mode. This is listed as a separate issue here just to stress the point that the driver should be tested under 32-bit CQD as well. The driver will create its own GDevice for printing. This GDevice will be 32 bit deep if 32-bit QuickDraw is present otherwise it will be 8 bit deep. The driver will create this GDevice at PrOpenDoc time and allocate the printing GrafPort in this GDevice. The current GDevice will be saved and restored by the driver's PrOpenDoc and PrCloseDoc routines respectively.

### Support for the Kanji PostScript Printer

To support printing on the Kanji PostScript printer, the LaserWriter driver and KanjiTalk need modifications. These changes are necessary for the following reasons. Today, KanjiTalk prints Kanji text as bitmaps. This means that it intercepts applications text drawing routines and if the text is being drawn in a Kanji font, it changes the text call into a bitmap call so that the LaserWriter driver sees StdBits instead of StdText and everything works fine. On the Kanji PostScript printer however, KanjiTalk would like to use the real font on the printer to print Kanji characters. Therefore, KanjiTalk needs to know if the Kanji font exists on the printer. Since the Kanji Screen fonts have double byte characters and their widths are not available in the form of standard FOND width tables, the LaserWriter driver driver needs to call KanjiTalk to get these widths.

Listed below is the interface between KanjiTalk and the LaserWriter driver to make printing possible on the Kanji PostScript printer.

LaserWriter driver will provide a PrGeneral opcode to be used by KanjiTalk to determine if a font exists on the printer. This routine can be called anytime after the document is opened, i.e. after the international proc is called by the driver.

```

CONST
    ISPrinterFont = -4;
TYPE
    TPrinterFontData = RECORD
        iOpCode: Integer; /* set this to ISPrinterFont */
        iError: Integer; /* result code from the driver */
        lReserved: Longint;
        hPrint: THPrint; /* pass NIL here */
    
```

```

    iFontid: Integer; /* the Fontid in question */
END;
TPPrinterFontBlk = ^TPPrinterFontData;

```

KanjiTalk will allocate the above structure, initialize it and call PrGeneral.

```

Procedure PrGeneral (pPrintFontBlk:Ptr);

```

KanjiTalk will call PrError after this. If it returns resNotFound, KanjiTalk should call PrSetError (noErr). This is the case where the drive does not support any PrGeneral calls. If PrError returns noErr then check iError in TPrinterFontData record. If it is ftNotFound (=3) then the font was not found on the printer. If it is opNotImpl (=2) then you are talking to a driver that does not support this PrGeneral opcode. In either failure case, send bitmaps to the driver. iError will be noErr if the call is successful.

KanjiTalk will provide a routine ISSpecialFont (Script Manager selector = 42) that will take font id as the parameter and return true if the font is a double byte font.

```

Function ISSpecialFont (id:Integer):Boolean;

```

If ISSpecialFont returns true then the driver will call RawPrinterValues (Script Manager selector = 44) to get printer widths for the text to be printed using this font.

```

Function RawPrinterValues (textPtr:Ptr;textLen:Integer;
    var printerWidth:Fixed;
    var numSpaces:Integer;
    var totalChars:Integer;
    var spaceChar:Integer;):Boolean;

```

textPtr and textLen are the parameters passed to StdText. printerWidth is the width of text at the given point size (fontSize in the grafPort). numSpaces is the total number of spaces in this string. totalChars is the total number of characters (not bytes) including the spaces in this text string. spaceChar is the character code for the space character in this text string. This function will return FALSE if for some reason KanjiTalk could not get the widths.

The above interface is exercised only when running under KanjiTalk. These changes affect the line layout code of the LaserWriter driver and should be tested thoroughly. I don't expect this code to affect the line layout code when running on a system without KanjiTalk.

**List of bugs to be fixed**

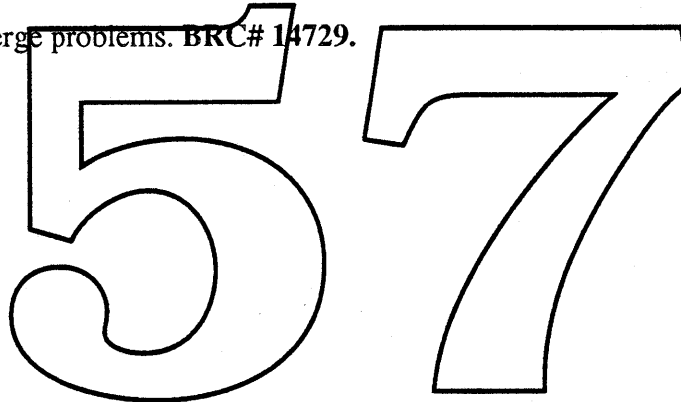
Following is the list of bugs to be fixed in the LaserWriter 6.0

1. Restore correct spot function after printing a QuickDraw pattern. **BRC# 32812.**
2. When you cancel a print job that is using the low-level print manager interface, the driver will crash. The cancel has to happen while the driver is opening the connection with the printer. **BRC# 33440.**
3. Unable to print to a LaserWriter selected number of pages from multipage Guide document.



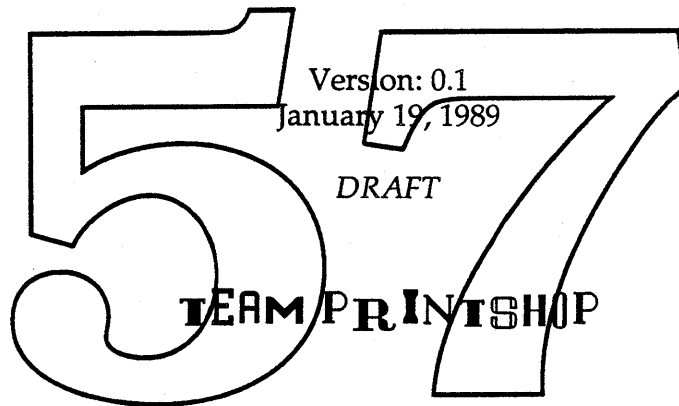
**BRC# 29185.**

4. Excel 1.06 does not correctly print a range of pages from Finder to a LaserWriter. **BRC# 29620.**
5. When trying to initialize a printer from an application residing on a volume other than the startup volume, user is presented with an incorrect dialog. **BRC# 30367.**
6. The Print Directory command from the Finder to a LaserWriter prints the "Locked" icon in random sizes and locations on the printouts. **BRC# 31508.**
7. When printing large number of polygons using FramePoly, one of the polygons is printed incorrectly. **BRC# 31831.**
8. Printing docs in background, changing printer in the foreground, will change the cover page of docs in background to have the name of the printer in the foreground. **BRC# 32653.**
9. Mac Plus and SE crash when printing to a LaserWriter from RedRyder. **BRC# 33208.**
10. Setting the copies field in the LaserWriter print dialog to 0.1 prints 141 copies. **BRC# 31338.**
11. MacPaint 1.5 documents when printed from Finder (with BGP) are not smoothed. **BRC# 32904.**
12. The Print dialog box is too small to display a message that is over two lines long. **BRC# 19178.**
13. Fix PrJobMerge problems. **BRC# 14729.**



# GINSU

## PROJECT ERS



# GINSU TEAM

## Project Responsibilities

*(Where to go with questions)*

### FULL-TIME PARTICIPANTS

- **William Stein**  
Project Leader.
- **Amy Rosenstock**  
Dialog Handler, Memory Handler, Debug Handler, Error Handler, Miscellaneous Handler, Application Programming Interface
- **Nik Bhatt**  
Package Architecture, Spooling Handler, Despooling Handler, File Handler, Generic Drivers, PrintStation
- **Sean Parent**  
Imaging Engine Architecture, QuickDraw to Raster Imaging Engine
- **Naresh Gupta**  
QuickDraw to PostScript Imaging Engine, PostScript Model
- **Daniel Lipton**  
Food Processor (Image Processing)
- **Tom Dowdy**  
Background Handler, PrintMonitor, PrintStation
- **Scott Jenson**  
User Interface

### PART-TIME PARTICIPANTS

- **Bayles Holt**  
QuickDraw to PostScript Imaging Engine, PostScript Model
- **Jay Patel**  
QuickDraw to PostScript Imaging Engine, PostScript Model
- **Andy Axelrod**  
ASYN I/O Handler (All Communications Packages)
- **Sam Weiss**  
SmartDriver
- **Hugo Ayala**  
QuickDraw to Vector Imaging Engine Investigation
- **Michael Hopwood**  
Product Marketing

# GINSU TEAM

## Documentation Status

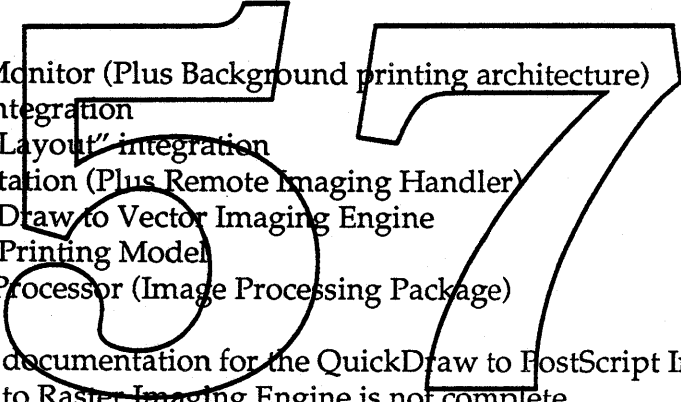
### The good news.

Complete low level ERS documents exist for a number of the Ginsu handlers. These documents are not included in this package, but are available on request for review.

### The bad news.

Not documented, but not forgotten...

Several major functional and architectural areas are not documented in this packet. Specifically:

- 
- 1) PrintMonitor (Plus Background printing architecture)
  - 2) Bass integration
  - 3) "Line Layout" integration
  - 4) PrintStation (Plus Remote Imaging Handler)
  - 5) QuickDraw to Vector Imaging Engine
  - 6) Color Printing Model
  - 7) Food Processor (Image Processing Package)

In addition, the documentation for the QuickDraw to PostScript Imaging Engine and the QuickDraw to Raster Imaging Engine is not complete.

### The ugly news.

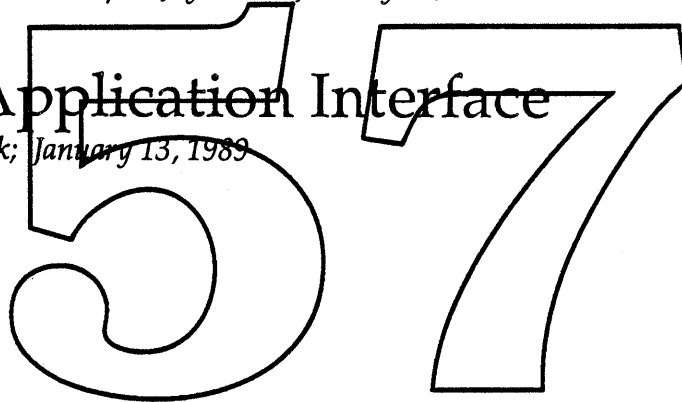
The new and improved schedules for all this stuff are still in process, and are not included here.

## Please Note

This package contains DRAFT documentation. Read at your own risk.

# PACKAGE CONTENTS

- **Ginsu User Interface ERS**  
*Scott Jenson and Andy Axelrod; Version 0.3, January 18, 1989*
- **The Ginsu Architecture**  
*Nik Bhatt; January 18, 1989*
- **Imaging Engines**  
*Sean Parent, Naresh Gupta, Jay Patel; January 19, 1989*
- **Ginsu Application Interface**  
*Amy Rosenstock; January 13, 1989*



*Captain Ginsu Says: "Read this, and join the PrintShop..."*

# *Ginsu User Interface* *ERS*

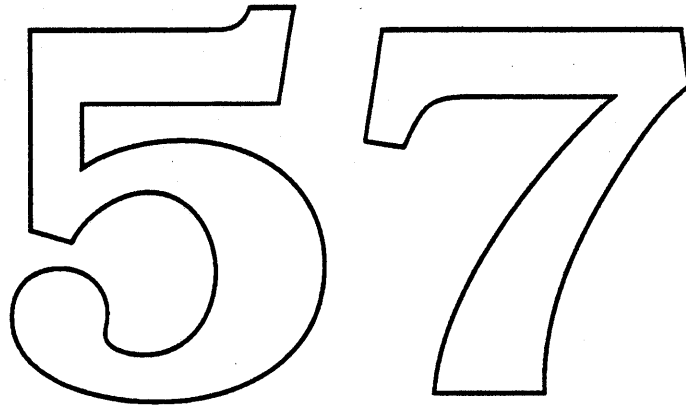
Version 0.3, Wednesday, January 18, 1989

Scott Jenson and Andy Axelrod

57

# Table of Contents

1.0	About This Document.....	1
2.0	Project Goals.....	2
3.0	User Scenarios.....	3
4.0	The New Dialogs.....	6
5.0	Document Formatting Model.....	10
6.0	Printing of Document Pages.....	20
7.0	View from the Finder.....	28
8.0	Page Setup Dialog.....	34
9.0	Print Job Dialog.....	36

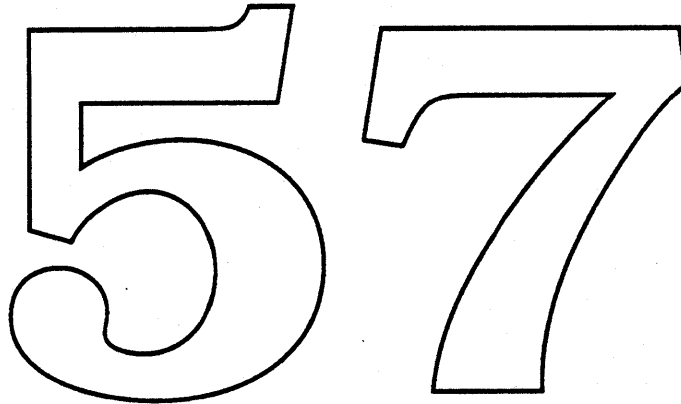


## 1.0 About This Document

This document describes the User Interface for Ginsu, the new version of the system printing architecture. Send any comments, questions, or general diatribes to either:

Scott Jenson	MS 27-AO
AppleLink:	jenson
VAX mail	jenson
Telephone	4-1576

Andy Axelrod	MS 27-AJ
AppleLink	Axelrod2
Telephone	4-5712

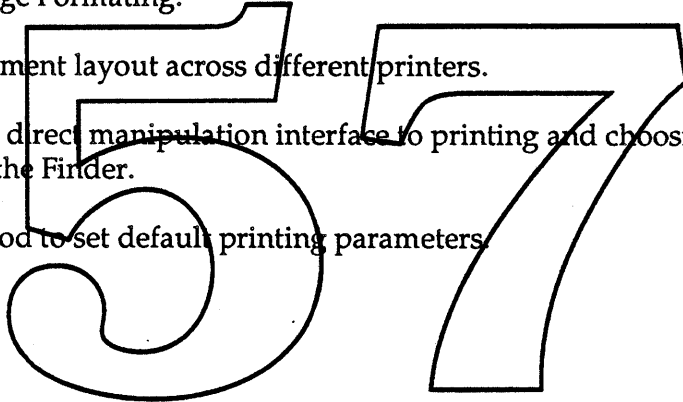




## 2.0 Project Goals

The goals for the User Interface are to:

- eliminate the need for dialog boxes for basic user printing.
- design a "scalable" interface to the PageSetup and Print Dialogs that is simple and intuitive for novice users yet gives access to more advanced features as a user grows to need more functionality.
- have a consistent set of printing dialogs across all applications and printers.
- have a universal method to access the extra printing functionality added by applications or printers.
- support By Page Formating.
- preserve document layout across different printers.
- design a more direct manipulation interface to printing and choosing printers from the Finder.
- a simple method to set default printing parameters.



## 3.0 User Scenarios

The new printing architecture is not a simple "one-shot" improvement to printing. There are several competing goals, representing very diverse levels of users. The challenge always comes down to making the difficult tasks "doable" without sacrificing the simplicity necessary for the normal user. This section provides a rough sketch of the various types of people who we feel are going to use the new printshop software. We've created six groups or classes of users, progressing from the most basic single user up to a very advanced user/office environment. Each class builds upon the previous one adding new features and constraints related to printing.

We've broken up our user base in this way to better understand how to design and evaluate the interface. In adding a new feature, we try to use this organization to determine which group will use it, how often they'll use it, and also how accessible it needs to be. It also provides a rough guide for determining the number of users affected.

While we could quibble forever on the "proper" breakdown of our user base, or even if an increasing progression of sophistication is a good metric for comparison, even an imperfect model is very effective in designing and evaluating different user interface designs.

### User Class 1 - Home/Education

single user  
single paper size(US Letter)  
single printer (serial)

- They don't make use of Page Setup.
- They don't make use of PrintJob.
- They usually use "Print One" style of printing.
- They need some way, automatic or explicit, of choosing a printer once.

### User Class 2 - Small Business/Student

add to the previous class:  
ImageWriter and/or LaserWriter (appleTalk or SCSI)

- First use of Page Setup: Page Size(A4 for Europe, Legal for some US firms).
- First use PrintJob: Copies, Quality, Manual Feed.
- Implicit change to another printer (by moving to another machine)
- Changing simple Core Defaults (page size, print quality)

### User Class 3 - Small Business II

add to the previous class:

Multiple Page Sizes: US Letter/LetterHead/Envelope  
Multiple Printers(appleTalk)

- Still only using Novice Dialogs.
- Explicit change to another printer
- By Page Formatting using scaling or orientation (no PaperTypes)
- Document wide use of application/printer specific PaperTypes.

### User Class 4 - Sophisticated Small Business

add to the previous class:

multiple page sizes (All of the above/Forms/Slides)  
Non-Basic Printers: High End LaserWriter/ SlideMaker  
Simple Color Printing  
a Print Server

- First use of extended dialogs
- User defined Paper types --letterhead, labels, special forms
- Printer Specific Special Effects
- Complex Manual Feed  
"Application Driven" -- only some PaperTypes are manually fed.
- Redirection of manual feed directives.
- Interface to print server.

### User Class 5 - Graphics Designer and Engineer/CAD user

add:

Multiple Paper Types (All of the above, larger graphic paper, E size paper)  
Complex Color (24 bit)  
Very High Resolution  
Multiple Printers (Plotters, LaserWriters, Linotronic, QMS, slide makers)  
Applications (Spreadsheet, Word Processing, Desktop Publishing, and CAD)

- "Proofing" or fitting of one page size to another
- Color mapping between different output devices given a single document
- Scaling required by QuickDraw coordinate space limitations  
Not up to us, this is an application guideline.

### User Class 6 - Large Business Environment:

add:

Multi-Bin input/output printer  
Collation/Sorting  
Mailboxes  
Large volume production of documents

Multiple Printers (LaserWriters, Sequoias, slide makers, QMS)

- Mapping document pages to printer input trays
- mapping output to mailbox
- flexible output bin selection (ranges of mailboxes)
- Sharing paper types among different machines
- print job time switching of paper type
- General output bin selection
- overflow bins.

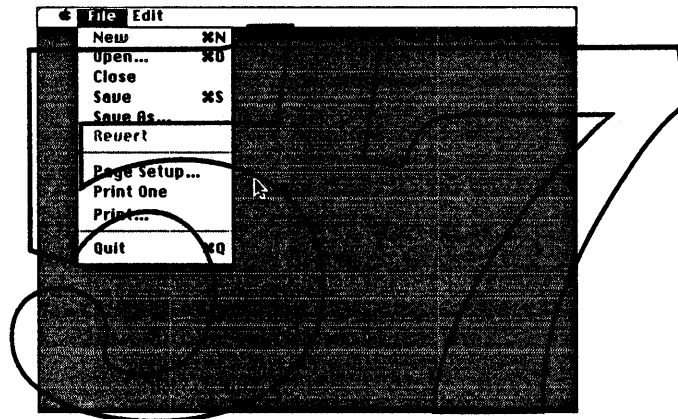
57

## 4.0 The New Dialogs

Both the PageSetup and PrintJob dialogs are going to have a tiered interface. This chapter describes this layering as it applies to both dialogs. For specific details of each, go to the later chapters devoted exclusively to their design.

### 4.1 The World of No Dialogs

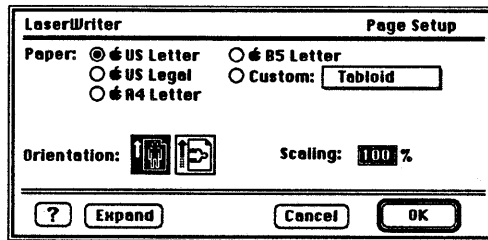
For most casual users, they won't need to use the dialogs at all. Through an improvement in the overall printing model, there is no longer any need to choose PageSetup each time you change printers (see the chapter "Document Formatting Model" for more information.) This eliminates most of the need to ever bring up PageSetup. To reduce the need for the PrintJob dialog, we would like to encourage applications to provide in addition to "Print..." a "Print One" menu item similar to MacDraw.



There will also be a simple method to set system printing defaults (see the chapter "View From the Finder") so European users who tend to prefer A4 paper will also be able to avoid mandatory use of the print dialogs as well. This isn't just a convenience for our European users, the defaulting mechanism will also help large installations "make all of the hard choices" so the users of that particular installation can continue to avoid the dialogs.

### 4.2 The Novice World

The next level up is to bring up a very simple dialog for both PageSetup and PrintJob. These dialogs ask the user to choose from no more than 3 or 4 options and then hit "OK" to continue. These novice dialogs have 2 very important constraints: They look identical across both applications and printers. Applications will no longer be able to add various buttons and other assorted slime to the bottom the the dialogs. Printer drivers as well, will now all bring up the same basic dialog.



The Novice Dialog

Like the “No Dialogs” case above, the novice world should be very adequate for a large group. The choices in the dialogs will be limited but a large group of users only need for example, to choose legal size paper, or print 4 copies of a document.

In addition, the PrintRecord normally stored with a document will now retain almost all user print choices, including which printer to use. This will encourage a “one-shot” use of the novice dialogs and further printing can fall back to the no dialog model.

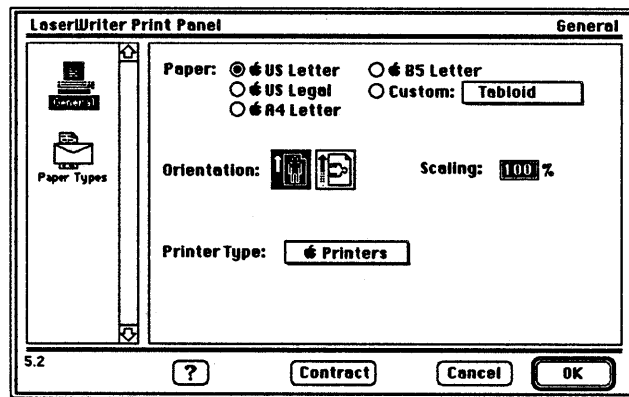
Also at this level, the user will have access to the new Help system in Big Bang. Bubble help will always be available through the <Special-Key> click metaphor while “How Do I \_\_\_?” help will be available from a button in the dialog. As the Help system model includes more than just these two types of help, it was suggested that on clicking the help button (currently the “?” character) instead of going directly to “How Do I \_\_\_?”, the user gets a popup menu identical in form to the one found in the Finder. While this is certainly “featurefull”, the concern is that forcing a novice user to navigate a popup from a button in a dialog may be counter productive. If this help is not available for any reason, we’ll fall back on our own version of the “How Do I \_\_\_?” help browser. We sincerely want to avoid this.

#### 4.3 The Extended World

The next level up is for the user to click on the “Expand” button at the bottom of the novice dialog. The simple dialog now expands into a control panel interface with the first selected panel showing the same choices of the novice dialog. In expanding to the control panel, it is very important to keep the same novice choices visible. This reinforces the feeling that the user is still looking at the same choice set with the only difference being that a browsing interface to more settings has become available.

Just like the control panel, clicking on one of the icons in the scrolling list on the left brings up a different panel of choices. The number of system panels initially available will be kept small (just 2 or 3). Applications as well as printers can then use this mechanism to add their own extensions to the printing interface such as

margins, special effects, etc.



The Extended Dialog

Help, just like in the Novice World above will also be available.

#### 4.4 Why Choose the Control Panel?

Most high-end applications add extra choices to our dialogs. Many users don't need these extra features but are forced to wade through this full screen dialog as applications have no choice but to add them "up front", directly under the standard print dialog. This is very confusing to users as some applications change the dialogs so much there is little consistency going from one application to another. Also, given the limited real estate of the Mac screen, many options are crammed too closely together further confusing their use.

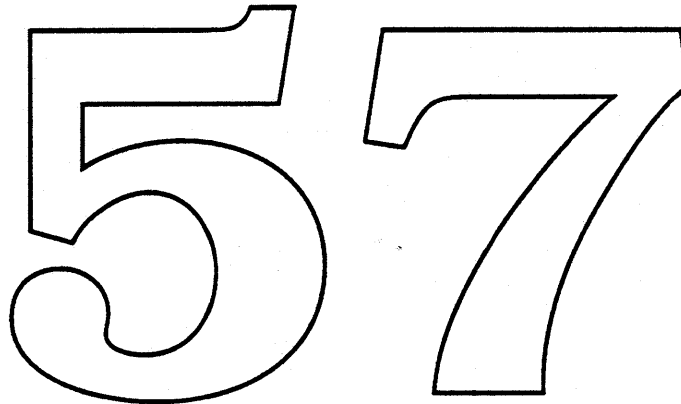
Our goal was to reduce this clutter and provide a consistent set of dialogs across all applications. Users then learn how to print once and leverage that skill across each new application. Applications though, still need some way to add their own extensions. The control panel interface is a known browsing metaphor which is easily extendible and has the benefit of making the most use of the smaller Mac screen.

It does have the disadvantage of making users "make an extra click" to get to the application specific features. This is a feature not a bug. We can't stress enough how important printing must fall in line with the rest of the Mac Interface; namely, once I learn how to print the first time, I never need to learn it again. Full screen print dialogs crammed with a varying list of printing choices, DON'T let me do this.

All well and good, but why the control panel? We looked at several techniques of "cascading" the information in a dialog box but none were as "clean" and modular as the control panel. We're able to avoid a major extension to dialog box

interfaces as the a control panel interface already exists and users can leverage off of the experience they've gained there. Also, we've tried to keep this "doorway to advanced features" an area of extra functionality. As most users will never need to use these features we can hopefully avoid any confusion which novice users may have. Of course, we want to start some user testing quickly to determine how easily users acclimate to it.

On a more graphical note, there is some concern that the expanding from novice to expanded dialogs will be confusing. The current model is to have the novice dialogs be reasonably small and centered on the screen and the expansion extends in both horizontal and vertical directions. The visual "anchor" is that the choices in the novice are still laid out identically in the extended dialog. Needless to say we've got a few other visual effects and we'd like to do user testing on all to see which one will keep the users focus best.





## 5.0 Document Formatting Model

This chapter describes the changes to document formatting that Ginsu will introduce. In general, Ginsu, in coordination with third party applications, will provide document formatting capabilities, which are much more flexible than those supported by the current printing model.

### 5.1 Support for Multi-Format Documents

In the current printing model, a document has only one format as far as the printing software is concerned. This format information is specified via the Page Setup dialog and includes the paper size (e.g. US Letter and US Legal), orientation (Landscape or Portrait), scaling factor, and other more device specific formatting characteristics. In the current model, a document is formatted for a specific type of device (e.g. LaserWriter, ImageWriter, etc.), and the format settings apply to all pages of the document.

#### 5.1.1 Problems with the Single Format Model

The current printing model's adherence to a single document format creates many difficulties for our users. The primary problem is the inherent lack of flexibility. Consider the following situations: 1) a user wants to have two different page sizes within a document, such as envelope and US letter; 2) a user has merged spreadsheet data into a word processing document and he wants the document text to be in portrait orientation and the spreadsheet data in landscape orientation (because it won't fit otherwise); 3) a user wants a particular graphic image within a document scaled so it will fit entirely within a document page, but doesn't want all pages affected by the scaling factor. These are but a few examples of this inflexibility. In each of these cases, users will typically address the problem by creating separate documents to circumvent the single format restriction.

#### 5.1.2 Moving to a Multi-Format Model

In Ginsu, we're extending the document formatting model to allow separate formatting characteristics for different portions of a document. In this multi-format model, assuming the proper support is provided by the application, a user can format any page(s) of a document in anyway he/she chooses. For example, a user might want some pages of a document to be formatted for portrait orientation and some for landscape orientation. Or, he might want to create a single document which contains different size pages, such as one with envelope and standard US letter sizes. In order to specify per page formatting in this model, a user might first select the pages of the document which are to be format a specific way, and then invoke the Page Setup dialog to specify the pages' formatting characteristics. In the extreme case, a user could choose to format every page of a document differently. In general, the formatting characteristics that can be specified on a per document basis in the current formatting model (e.g. paper size, orientation, scaling, etc.) can also be specified on a per page basis in the multi-format model. One exception to this rule is that a document can only be formatted for

one printer; users cannot format different pages of a document for different printers. As with the existing model, the multi-format model requires that a document be formatted for a specific device (e.g. LaserWriter, ImageWriter, or "Apple Printers" (see below)).

## 5.2 Impact of Multi-Format Documents on Application User Interfaces

A major issue raised by the introduction of multi-format documents is the impact they will have on existing application interfaces.

Applications will be affected, because their interfaces will likely need to distinguish between formatting on a per page basis and formatting on a per document basis. For example, in an application which supports multi-format documents, what would be the interpretation of the "Page Setup..." file menu item? Does it perform per page formatting, per document formatting (as in the past), or both? One possibility is to extend the existing definition of "Page Setup..." to include per page and per document formatting. Another possibility is to introduce a new menu item, "Document Setup..." which allows the user to specify per document formatting, and change the definition of the "Page Setup..." item to encompass only per page formatting. This problem is still an open issue and we are continuing to evaluate these and other alternatives.

Another problem posed by the introduction of multi-format documents is the need to recommend methods for how users should select a range of pages to format. Since the page selection method will vary for different classes of applications (e.g. consider the differences between a spreadsheet document (which has no page metaphor) and a word processor document), we plan to recommend appropriate guidelines for each class. The list of classes currently includes word processors, spreadsheets, graphics programs, and database/mail merge applications. We're evaluating a number of alternatives and plan to develop a final list of guidelines shortly. It's important to note that applications are not required to support multi-format documents; applications can continue to support only single format documents if they choose. Obviously, the degenerate case of a multi-format document is today's single format document.

## 5.3 Paper Types and their Relationship to Multi-Format Documents

In the current printing model, the size of a document's pages is defined by the paper size that is selected from the Page Setup dialog. Current drivers provide a fixed set of paper sizes, and an application may, in some cases (e.g. ImageWriter), augment that set with additional sizes specific to the function of the application (e.g. paper sizes which correspond to preprinted forms). There is no provision, however, to allow users to add additional custom paper sizes to augment those already defined by the printer driver.

### 5.3.1 Why Do We Need Custom Paper Types?

A significant problem with the current printing model is the inability for users to

define their own custom paper types. This results in a number of problems.

First, printing on special types of paper (e.g. mailing labels, checks, preprinted forms, etc.) is either difficult or impossible for our users. Given that the printing software provides only a standard set of paper sizes, users must rely on applications to provide the custom paper sizes they may require. If no application can be found which supports the required custom paper size(s), our users are out of luck. Custom paper sizes need to be defined and maintained on a system wide level so that all applications can utilize them.

Second, users need to be able to format a document for one printer and print the document on another printer without causing the document to reformat. In order to prevent the reformatting of documents, two factors need to be considered, including the size of a document page's printable area and the imaging resolution.

Third, over the next few years we will experience the introduction of many new printers that will have multiple input trays and output bins. Obviously users will want to exercise these printer facilities to their fullest, and will want to maximize the relationship between a document's format and the printer's configuration. For example, they will want to have, say, envelopes, letterhead, and blank sheet stationery in a printer's input bins, and will want to define envelope, letterhead, and blank sheet formatted pages within their document. When such a document is printed, they will expect that the appropriate stationery will automatically be drawn from the correct input tray. Without the ability to define custom paper types (e.g. types for envelope and letterhead stationery), users would not be able to maintain within their documents a natural correspondence between document pages and printer paper. And the capability to automatically map document pages to printer paper would be lost.

### 5.3.1 Definition of Paper Types

During the course of the Ginsu design effort we evaluated a number of possible definitions for paper types. The one we finally settled on is described here. The next section provides brief a historical perspective on how we arrived at the current paper type definition.

A paper type is intended to represent only the formatting characteristics of a sheet of paper. Its definition includes three components: a name, the physical dimensions of the paper it represents, and the borders within the paper that delineate the imageable area. The name is a simple text string which uniquely identifies the paper type definition within a single machine. The physical dimensions of the paper specifies the width and height of the paper in inches, centimeters, picas, or points. (In the remaining sections of this document, reference to a paper type's size is used as a shorthand way to refer to a paper type's physical page dimensions). The imageable area component defines the actual printable area of the paper. This area is defined by specifying the borders of the imageable area rectan-

gle within the paper's boundaries. A border is simply an offset from one side of the paper, so to specify an imageable area, we need to specify the top, left, bottom, and right offsets from their respective edges of the paper. The imageable area dimensions are also specified in inches, centimeters, picas, or points.

The imageable area component of the paper type definition is always expressed in terms of an imaging resolution of 72 dpi. A resolution of 72 dpi was chosen since it's the imaging resolution supported by QuickDraw and all of our devices. Defining imageable area in this way ensures that paper types are not defined for a particular class of device. When a document page is actually printed, the resolution at which it will be imaged is determined by the target application and device. For applications which image pages at 72 dpi (i.e. screen resolution), those 72 dpi imaged pages will be scaled by the print driver to the target device's resolution (e.g. 144 dpi for the ImageWriter and 300 dpi for the LaserWriter). For applications which image at device resolution directly, (e.g. PageMaker), those imaged pages will be printed without being scaled by the print driver.

A paper type's size can be defined to be arbitrarily large, so long as it doesn't exceed the limitations imposed by QuickDraw coordinate space. A following section describes in detail how we handle printing arbitrarily large pages.

### 5.3.2 Historical Evolution of Paper Types

In order to understand why we defined paper types in this manner, it's useful to look at the historical evolution of paper types. This section provides a brief description of each paper type definition we considered and the main problems associated with each.

#### Paper Type Definition #1:

##### Components of Definition:

- Paper type name
- Physical paper dimensions
- Imageable area dimensions
- Imaging resolution, determined by the device for which the paper type was defined
- Each paper type created for a specific device

##### Problems:

Paper types defined at one resolution, say 72 dpi, which were to be used on a device (or application) that imaged at a higher resolution, say 1200 dpi, might not be able to print at the higher resolution. This was because the paper type's size might exceed QuickDraw coordinate space. From this definition we realized the need to somehow vary a paper type's resolution in order to allow the same paper type to print on a variety of devices.

**Paper Type Definition #2:**

**Components of Definition:**

Same as #1, but decided to include physical paper attributes such as color, medium (e.g. paper, transparency, and film), and punch (e.g. 3 hole, 2 hole, and spiral).

**Problems:**

We added these attributes to more accurately model users real world concept of paper. The problem with these attributes was that they had no affect on the formatting of document pages. Up to this point, every component of a paper type definition influenced document formatting.

**Paper Type Definition #3:**

**Components of Definition:**

Same as #2, but decided to include device specific attributes such as ASA rating for film paper types.

**Problems:**

Though this definition more accurately reflected the attributes of the physical medium, it suffered from the same problems as definition #2; it added attributes that did not affect document formatting.

**Paper Type Definition #4:**

**Components of Definition:**

- Paper type name
- Physical paper dimensions
- Imageable area dimensions
- Common set of paper type attributes (e.g. medium - attributes common to all devices)
- Device specific paper type attributes (e.g. ASA rating, gamma correction tables, etc.)
- Each paper type created for a specific device
- Imaging resolution considered a device specific attribute so user could change it (resolution defaulted to that of the device for which the paper type is defined)

**Problems:**

This definition was bad because it continued to promote the inclusion of non-formatting related information in the definition. It also results in the creation of large numbers of paper types (different ones for different attribute settings), and might cause users difficulty printing document pages, which have one set of device specific attribute settings, to paper in a printer which has different

settings.

### **Paper Type Definition #5:**

#### **Components of Definition:**

Paper type name

Physical paper dimensions

Imageable area dimensions

Paper types no longer defined for a specific device; could be used on any device.

Paper type defined at resolution of 72 dpi (so paper type could be used on all devices)

#### **Problems:**

The primary deficiency of this definition was that for some devices, users might need to specify a paper type's device specific settings each time a document is printed. By preserving print job settings (see a following section), we felt this problem could be minimized.

After much discussion, we finally agreed to definition #5. By allowing a paper type to be used on all devices, the resolution at which pages are imaged can float from device to device.

### **5.3.2 Classes of Paper Types**

Ginsu will define four classes of paper types. The four classes are the Apple standard types, device specific types, user defined types, and application specific types. Please note that this class designation is simply used to distinguish between categories of paper types; every paper type is still defined in the same way (i.e. name, paper size, imageable area, etc.).

#### **Apple Standard Paper Types**

The Apple standard paper types are a set of paper types composed of the standard paper sizes (at least US Letter, US Legal, A4 Letter, and B5 Letter), which have imageable area definitions that are common across all Apple printers. Maintaining common imageable areas means that, for example, the dimensions of the US Letter type's imageable area is exactly the same for all Apple printers. Since the maximum imageable area for a specific paper type will vary across the range of Apple printers (due to varying hardware characteristics), the common imageable area for the paper type is defined to be the least common denominator imageable area defined across those printers. In other words, it's defined to be the intersection of all of the imageable areas. The key reason for providing paper types with common imageable areas is to address the reformatting problem. Recall that a document may reformat when either the imageable area or imaging resolution changes. By providing paper types whose imageable area definitions don't change across all Apple printers, documents formatted with any of these

paper types will never reformat due to variations in imageable area.

In order to designate these standard paper types, the Ginsu Page Setup dialog will precede the names of these paper types with an '⌘'. For example, the standard paper types would be listed in the dialog as "⌘ US Letter", "⌘ US Legal", "⌘ A4 Letter", and "⌘ B5 Letter". The standard paper types will always be made available for users to choose from, regardless of the device for which a document is being format. There will be one set of Apple standard paper types defined for all Apple printers.

### Device Specific Paper Types

The device specific paper types are a set of paper types, which are composed of the standard paper sizes (e.g. US Letter, US Legal, A4 Letter, etc.) and any additional device specific paper sizes (e.g. Tabloid for the Linotronic). A set of device specific paper types will be defined for every Apple printer we market. These paper types differ from the Apple standard paper types in that each type's imageable area is defined to be the maximum that can be defined for the given device and paper size. The device specific paper types define a class of "best fit" paper types for each Apple printer. As an example, the definition of the existing LaserWriter US Letter paper type specifies a physical paper size of 8.5 in. by 11 in, and an imageable area whose left and right margins are .25 in., and whose top and bottom margins are .11 in. Note that the "⌘ US Letter's" imageable area would be smaller.

These paper types are required, because all existing documents are formatted for these "best fit" sizes. If we didn't provide these types, all existing documents would reformat when opened using Ginsu drivers.

In order to designate these device specific paper types, the Ginsu Page Setup dialog will precede the names of these paper types with the name of the device to which the paper types applies. For example, the device specific paper types for the LaserWriter would include "LaserWriter US Letter", "LaserWriter US Legal", "LaserWriter A4 Letter", "LaserWriter B5 Letter", and "LaserWriter Tabloid". As the name implies, the device specific paper types are only made available for users to choose from when a document is formatted for the corresponding device.

### User Defined Paper Types

The user defined paper types refer to the set of all paper types defined by the user. Paper types defined by users are shared by all devices, and the only restriction that applies when defining a new paper type is that the name of the type must be unique across all paper types on the user's machine.

### Application Defined Paper Types

The application defined paper types refer to the set of all paper types defined by

applications. Unlike the previous three classes of paper types, these paper types are only defined in the context of an application; that is, paper types defined by an application are only accessible when the application is executed. The printer driver will resolve problems which arise when an application paper type has the same name as another paper type.

### 5.3.3 Printing Pages that Won't Reformat Across Devices

Recall from an earlier section that in order to prevent documents from reformatting across printers, we must ensure that a document page's imageable area doesn't change, and that the resolution at which it's imaged also doesn't change. By formatting documents using the Apple standard paper types (i.e. "Apple US Letter", "Apple US Legal", etc.), we can be sure a document page's imageable area will not change. In order to prevent reformatting due to changes in resolution, we must be able to image a document on all Apple printers, at the same resolution. The only resolution common to all printers (i.e. the lowest common denominator) is 72 dpi; therefore, using the Apple standard paper types in conjunction with a resolution of 72 dpi will prevent documents from reformatting. The only remaining problem is to determine how a user should indicate that a resolution of 72 dpi should be used when formatting a document. Since imaging resolution is determined by the device for which a document is format, and a user must select a device in order to format a document (types of devices such as ImageWriter and LaserWriter) we've extended the list of available printing devices to include "Apple Printers". "Apple Printers" is used to signify all Apple printers. Thus, choosing "Apple Printers" from the device list in the Page Setup dialog, indicates that the target document is being format to print on all Apple printers (with an implied resolution of 72 dpi).

### 5.3.4 Printing Oversize Pages

Given that document pages can be defined to be arbitrarily large, it's possible for QuickDraw coordinate space to be exceeded for large pages imaged at high resolution. An example of such a situation would be to attempt to print an E-size document page (size of large plotter paper) on a 1200 dpi Linotronic printer. At such a high resolution, this large a page cannot be represented in QuickDraw coordinate space. To resolve this problem, we allow the resolution of the device to vary. In our example, the printer driver would return a resolution low enough to represent the document page in QuickDraw coordinate space, possibly 600 dpi. The general strategy is to allow document pages to be imaged at the highest possible resolution of the target device, while still being able to represent the entire page in QuickDraw coordinate space.

### 5.3.5 Mapping Old Paper Sizes to New Paper Types

Many existing documents are formatted with paper sizes which will directly map to new paper types defined in Ginsu. For these documents, the old paper sizes will map directly to Ginsu device specific paper types; for example, existing documents formatted for the LaserWriter using the US Letter paper size will map di-



rectly to the new "LaserWriter US Letter" paper type. Once a document has been converted, the user will manipulate paper types, rather than paper sizes. No references to the old paper sizes will be preserved.

There are a number of documents, however, which will be formatted for paper sizes that do not directly map to new Ginsu paper types. An example of such documents are those formatted for the ImageWriter using the US Letter paper size. (In Ginsu, the definition of the "ImageWriter US Letter" paper type will not correspond to the existing paper size definition, because of errors in the existing definition's imageable area). In order to accommodate these incompatible paper sizes, Ginsu will preserve the old sizes by creating new (though temporary) paper types from the old paper size definitions. These temporary paper types will not be stored with the other system wide paper types, but rather, with the document from which they were derived. These temporary paper types will be given names such as "Old ImageWriter US Letter" and "Old ImageWriter US Legal", in order to distinguish them from the new Ginsu paper type definitions. Users will be allowed to format new pages of an old document using one of these temporary paper type definitions. When a user reformats a document such that no pages of the document are formatted for a temporary paper type, then the temporary paper type definition will be removed. Thus, we will allow users to continue to use their old paper type definitions, but once they reformat, they will implicitly move to the new Ginsu paper type definitions and lose the old ones.

### 5.3.6 Creating and Invoking Paper Type Definitions

Users can create new paper types from within the Page Setup dialog. A following section of this document provides screen shots of the new Ginsu Page Setup dialog, which include a screen for creating new paper types. All paper types created by users are defined on a per machine basis, and so are accessible to all applications on the target machine. Paper type names must be unique within a machine. As is illustrated in a following section, the Apple standard paper types are selected from a range of radio buttons in the Page Setup dialog, and all other paper types are accessible from a pop-up menu.

### 5.3.7 Attaching Device Specific Attributes to Paper Types

Recall from our earlier discussion that device specific attributes of paper types will be specified at print job time. At print job time, the user has finally selected the target device on which to print. Since the target device is known, we know what device specific attributes need to be specified in order to print. The printer driver will augment the standard Ginsu Print Job dialog with additional items, which specify the device specific attributes. These settings will be preserved on a per document basis, so all device specific attribute settings will be retained.

## 5.4 Binding Printers to Documents

In the current printing model, a document is always formatted for and printed to, the currently selected system printer. The primary deficiency in this model is that

changes to the system printer affect all documents the user may wish to view and possibly print. If the specification of the system printer changes, documents typically need to reformat before they can be viewed or printed. Such behavior causes unnecessary problems for our users. In Ginsu, we've removed the concept of a system printer and have bound the printer specification directly to a document. Now, the specification of the device for which a document is format, and the device to which a document will print, is stored with each document.

In the Ginsu Page Setup dialog, a user must select the type of device for which his document should be format. The user will be allowed to select from such device types as "Apple Printers", "ImageWriter II", "LaserWriter II SC", and "LaserWriter II NT/NTX". The device type specification applies to an entire document and will be retained with the document until subsequently changed by the user.

In the Ginsu Print Job dialog, users will select the device to which they wish to print. A pop-up menu item listing the user's selected set of printers will be available from the print dialog. When a user selects a printer from this menu, the specification, as well as all other print job information, will be retained by the document being printed (see note below). Each subsequent time the document is printed, the most recent print job settings will be restored. Thus, once a user selects a desired printer, the document will always print to the same printer (assuming it's available). We're still debating exactly how much of the print job information should be retained, but it's likely we will retain everything except the "copies" setting. The "copies" setting would always be reset to one everytime a document is printed.

Problems can arise when a document is moved to a machine, which does not contain the print driver of the device for which the document is formatted. The formatting information included in the document will be sufficient enough to allow a user to continue editing his document, just as if the proper print driver were available. If the user attempts to print the document, and the previously selected printer is not available from the new machine, the print driver will inform the user of this fact (via an alert) and instruct him to select a new target printer (note: the user could select a printer whose type is different than that for which the document is format). If the previously selected printer is available, the user could print to it. If the user invokes the Page Setup dialog in order to reformat a portion of his document, he would be allowed to specify generic format settings (e.g. paper type, orientation, scaling, etc.), but would not be allowed to specify device specific format settings, if any. The generic format settings would be settings which are common across all devices.

## 6.0 Printing Document Pages

This chapter describes the model for mapping pages within a document, to paper within a printer. It describes how document pages will be mapped to single input and multi-input tray printers, how printed pages will be routed to single and multi-output bin printers, and discusses how mismatches between document page size and printer paper size will be handled. All of the settings described below will be available in the Ginsu Print Job dialog.

The intent of the model is to provide intelligent defaults, so in the event a user hasn't explicitly specified a mapping, or he wants the print software to "do its best", the software will perform a reasonable mapping. The model supports intelligent (i.e. can determine paper size) and unintelligent (i.e. can't determine paper size) printers, and printers with single and multiple input and output sources.

In general, the print software determines how to map document pages to printer pages by taking into account the default printer paper type(s), the actual types of paper in the printer (if it can be determined), and the user specified page mapping preferences (described below). The following sections describe how each of these specifications factor into the page mapping equation.

### 6.1 Default Printer Paper Types

In order to map document pages to printer pages, the print software needs to know the size of the paper currently loaded into the printer. It would be useful to know other characteristics of the printer paper (e.g. its medium), but at a minimum we need to know the size of the printer paper. In Ginsu, we will define a default paper type for every printer we support. For most printers, the default will probably be the "US Letter" standard paper type. As a result of specifying a default paper type for each device, we're implicitly defining a default paper size. Users will be allowed to change a printer's default paper type. For devices which have multiple input trays, a default paper type can be specified independently for each tray.

For a printer which cannot determine the size of the paper it contains (i.e. dumb printers like the ImageWriters), the driver will assume the default paper type reflects the actual paper in the printer. Without maintaining the concept of a default paper size, the printer driver would not know to what size paper the document pages should be mapped. For a printer which can determine the size of the paper it contains (e.g. the LaserWriter II NT/NTX), the default paper size will be overridden by the actual size of the paper in the printer.

Using a default paper size or actual paper size in order to determine a printer's paper size is adequate for paper which is automatically fed into the printer. However, when a user chooses to manually feed paper, the printer driver cannot know what type of paper the user will actually place in the device, and so cannot know to what printer paper size a document's pages should be mapped. For

manually fed input sources, we have to assume users will place the correct type of paper into the printer when prompted. We define the "correct" paper to be paper which has the same size as the document page currently being imaged.

## 6.2 User Specified Page Mapping Preferences

In addition to default printer paper types, a user is allowed to configure preference settings which specify to what extent he wants to be involved in resolving a paper type mismatch, and how a paper type mismatch should be resolved. A mismatch occurs anytime a document page's physical paper size (and possibly imageable area if it can be determined) does not match the size of the paper in the printer. As an example, consider a document which is formatted for E-size paper (i.e. plotter paper), and is to be printed on a printer that contains US Letter size paper. In this situation, the print driver needs to know what method to employ in order to map this large page to such a small piece of paper (e.g. we could tile the big page across many US Letter size pages, clip to the small paper, or scale to the small paper). The driver also needs to know whether the user wishes to temporarily stop printing and be informed of a mismatch condition.

The following three sections describe the preferences we'll provide in order to help users address the paper mismatch problem.

### 6.2.1 Resolving a Paper Mismatch

The Ginsu Print Job dialog will provide a preference setting which allow users to specify how a paper type mismatch should be resolved. A mismatch can only be determined at the time the printer driver establishes a connection to the device. If a mismatch is encountered by the driver, it will use this preference setting to determine how to resolve the mismatch. As a result, users will never be prompted at print time to specify how to resolve a mismatch.

This preference setting allows a user to choose one of four mismatch resolution methods. These include allowing the user to tile a document page across multiple printed pages, scale a document page so it will fit within a single printed page, clip a document page so it will fit within a single printed page, or not print the document. If a user selects the "don't print" option, none of the remaining pages of the document will be printed. The default setting for this preference will likely be to scale the image.

In the event a user (or the system default) specifies that tiling or scaling is to occur when a mismatch is detected, then if the document page size is larger than the printer paper size, the document page will be appropriately tiled or scaled, and will also be centered within the paper. If clipping were chosen in this situation, no centering would occur; the top, left corner of the document page would coincide with the printer paper's top, left corner. If the document page is smaller than the printer paper, the document page will not be altered, but simply centered on the printer paper.

### 6.2.2 Informing the User of a Paper Mismatch

Another preference setting allows a user to specify to what extent he wants to be involved in resolving a paper type mismatch. This preference setting allows a user to specify whether he wants to be alerted in the event a mismatch is detected.

The default setting for whether to inform a user of a paper mismatch is determined on a device by device basis. For example, with an ImageWriter LQ, the default would be to inform the user (because of possible printer damage that could result if we don't inform him), whereas for a LaserWriter, the default would be to not inform the user.

If a user specifies that he does not want to be informed of a mismatch, the driver will do its best to resolve any mismatch that may occur, and will not bother the user. In this situation, the printer driver will resolve the mismatch in the manner specified by the mismatch resolution preference setting; that is, the driver will tile, scale, clip, or not print the document pages.

In the event a user specified that he does not want to be alerted when a mismatch occurs, then if a mismatch is detected, an alert is issued which describes the mismatch condition and allows the user to "continue", "retry", or "cancel". Selecting "cancel" terminates the print job and none of the remaining document pages are printed. Selecting "continue" instructs the printer driver to continue printing and use the mismatch resolution preference setting to determine how to resolve the mismatch(es) it encountered. Selecting "retry" instructs the driver to reexamine the state of the printer to determine if the mismatch condition still exists. The "retry" option is useful in situations where after being alerted to a mismatch problem, the user realizes that the wrong paper is in the printer. Normally the user will want to put the correct paper in the printer and have the driver try again.

### 6.2.3 Verifying the State of the Printer Before Printing

In addition to the aforementioned preference settings, one other setting is provided which allows a user to specify that a print job should block immediately before printing. The purpose of this setting is to provide a user the opportunity to verify that the printer is in the proper state to print the job (e.g. ensure it has the appropriate paper installed). If this preference is selected, then immediately before beginning to print the document (the connection to the device will have been established at this point) the print driver will alert the user to the fact that the job is waiting to print. The alert will list the paper type requirements of the print job, so that the user can determine the required configuration of the printer. When the alert is issued, the user can choose to continue the job or terminate it. Selecting "continue" initiates printing. Selecting "cancel" terminates the print job.

### 6.3 Matching Document Pages to Printer Input Tray(s)

The previous sections described how document pages would be mapped to printer pages once the printer driver had selected the appropriate printer input tray. Deciding which input tray to use for single input tray printers is obvious; the printer driver will always select paper from the single input source. For printers which have multiple input trays (e.g. ImageWriter LQ or Sequoia), selection of the appropriate input tray(s) will depend upon how the user specified that a document's pages should be matched to the printer input trays. Users will be provided two methods for matching document pages to input trays: match by explicit page number (e.g. print page 1 from tray 3), and match by paper type (e.g. document page formatted for "🍏 US Letter" is printed from tray containing this paper type).

No matter which paper matching model is chosen by the user, mismatches between document pages and printer paper is handled as previously described.

#### 6.3.1 Matching by Paper Type

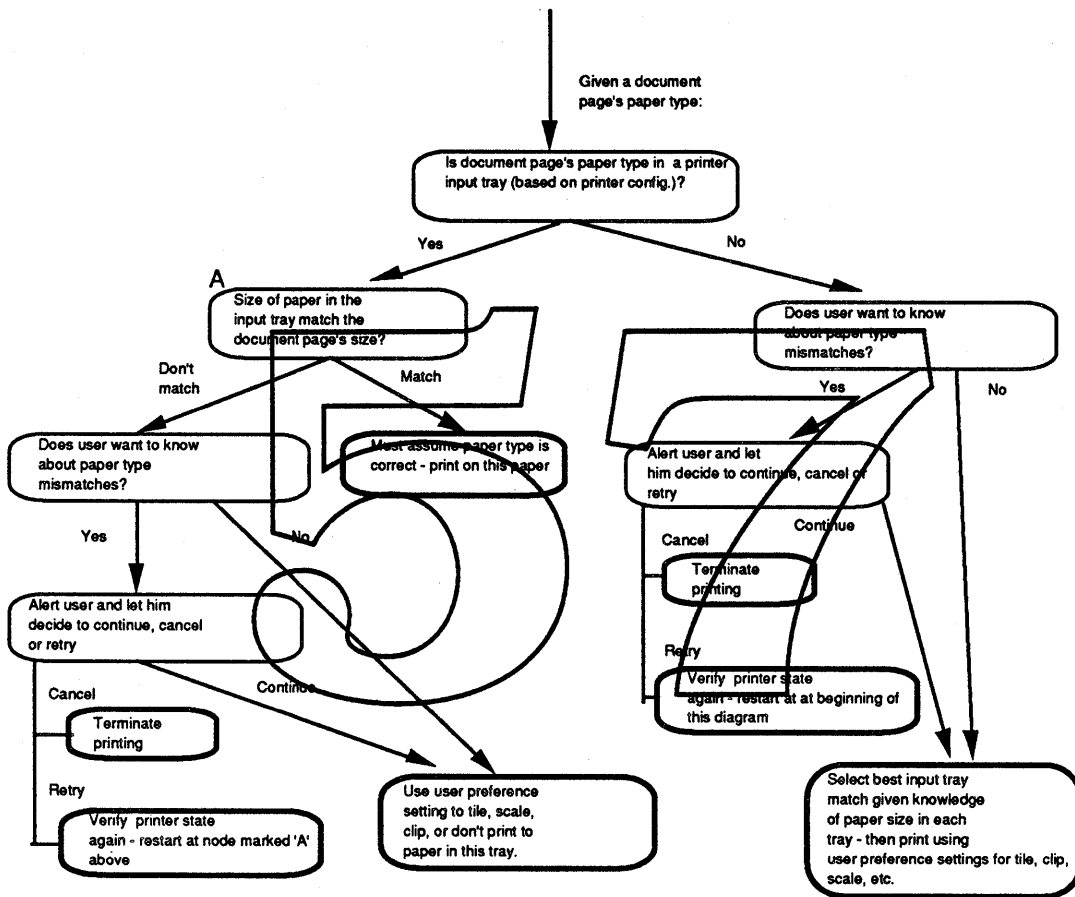
The default, and most natural method of matching document pages to input trays, is to match by paper type. This method matches document pages to input trays by simply matching paper types contained within the document, to paper types contained within the printer input trays. For example, a document contains pages formatted for envelopes and US letter paper types, and the printer contains envelopes in tray 1 and US Letter blank sheets in tray 3. When the document is printed, the appropriate paper will automatically be drawn from the correct input tray. In order for this mapping to occur, users must specify which paper types are contained in the printer's input trays. (If no paper types have been specified, the device's default paper type would be assumed, such as "🍏 US Letter"). Specification of the printer paper types is required, because most devices cannot determine the type of paper each input tray contains (no Apple printers can, and many cannot even determine the size). In order to find matches, the printer driver will always compare respective paper type names and physical paper sizes.

In the event a document contains a paper type which is not in any of the printer's input trays, the user can specify that the paper type be mapped to another paper type that is contained within a tray. An example of such a situation is a user who has a document formatted for "🍏 US Letter", with a printer which contains transparencies, and he wants to print the document on the transparencies. In this situation, the user would specify that the "🍏 US Letter" format pages be mapped to the printer's transparencies.

In order to provide users greater control over the selection of input trays, users will be allowed to specify whether a job can automatically select a new input tray, if the one it was previously using becomes empty. In situations where some number of trays contain the exact same paper types, users may want a job to switch trays automatically. However, there are situations where this is undesirable; for

example, when two trays indicate they contain the same paper type, but one actually contains transparencies. Choosing the wrong tray in this situation could be quite costly.

The following diagram summarizes the algorithm for mapping document pages to printer paper using the paper type matching method.



### 6.3.2 Matching by Document Page Number

The other method provided for matching document pages to printer input trays is to match by explicit page number. This method requires users to explicitly define which document pages should be printed from which printer input trays. It's most useful for applications which do not support multi-format documents, because it allows the user to treat the document as if it contained multiple formats as far as printing is concerned. The method for specifying the mapping is similar to that currently supported by the ImageWriter LQ driver. Users will be allowed to specify any sequence of commands which are of the form: "Pages i thru j from

tray <tray #>". The "i" and "j" designators represent page numbers, and the "j" designator can additionally be the string "end", which signifies the end of the document or sub-document (as in a mail merge application). An example sequence might be:

Pages 1 thru 1 from tray 3  
 Pages 2 thru 3 from tray 2  
 Pages 4 thru end from tray 1

Within the actual Print Job dialog, a graphical method will be provided with which users can generate these sequences.

In order to provide users greater flexibility, they can choose to repeat a sequence of page matching instructions. Users can specify that a sequence of instructions be repeated for all pages of a document. For example, the following sequence would print all odd document pages on paper from tray one, and all even pages from tray two (note the implied resetting of page numbers as the sequence repeats - page 3 is treated as a new page 1):

Pages 1 thru 1 from tray 1  
 Pages 2 thru 2 from tray 2  
 Repeat

The actual mechanism for specifying a repeated sequence is still being evaluated. A possible alternative method for expressing the same matching sequence as above might be:

One page from tray 1  
 One page from tray 2  
 Repeat

The page matching sequence is be applied to each copy of the document which is printed.

#### 6.4 Routing Printed Pages to Printer Output Bins

This section contains a description of how printed pages might be routed to printer output bins. This description is very preliminary and has yet to be reviewed in detail by the PrintShop. It's only listed here for completeness.

Once document pages are printed, they must be properly routed to printer output bins. As with the input trays, there are two cases to consider: printers with only one output bin and printers with many output bins. The single output bin case is trivial; all printed pages are routed to the single output bin. In the multiple output bin case, users can route individual copies of a document to output bins. (There are two additional routing alternatives being considered: route by explicit



page number (e.g. print page 1 from tray 3 and place in bin 6), and route by printer paper type (e.g. pages printed on "⌘ US Letter" will be routed to a bin designated to contain "⌘ US Letter" type paper).

In order to provide greater flexibility in referring to output bins, the Ginsu Print Job dialog will provide a facility for users to name the output bins of a printer. Thus, users will be allowed to refer to output bins either by name or explicit bin number within the Print Job dialog. This facility is useful when users want to treat a printer's output bins as mailboxes for naming specific users or groups, for example.

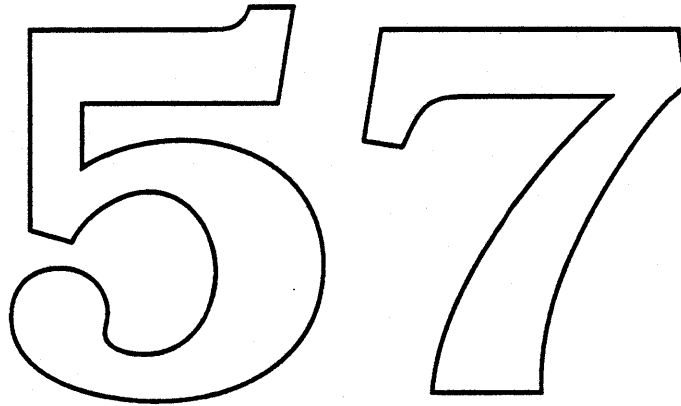
In order to route individual document copies, a user first selects a range of output bins in which to place the copies. Users can specify this range to include all bins, or only a subset. (We may also allow users to specify "any" bins, and allow the driver to choose the bins).

Given some range of output bins, users can then specify whether or not the document should be collated. If collation is chosen, then the print driver would cycle through the range of output bins in a round robin fashion, placing completed copies in each bin. The default setting for all devices would be to collate. If collation is not chosen, all copies of each document page will be placed into a single output bin in the selected range of bins. For example, given a range of three bins (1 - 3), the print driver would place all copies of page one into bin 1, all page two's into bin 2, all page three's into bin 3, etc. If the number of document pages exceeded the number of selected output bins, the print driver would cycle through the bins in a round robin fashion (so all page four's would go into bin 1 in our example).

In the event any of the bins become full, we could simply block the job and inform the user of the printer's condition. Another option would be to allow the user to designate overflow bins which could be used to hold the additional pages, or possibly let the driver select an appropriate bin(s) in which to place the additional pages. These and other possible alternatives to this problem are still being evaluated.

In addition to collation, Ginsu will also allow users to specify the stacking order of their document pages. The two options they can choose from are "front to back", and "back to front". When an entire document can be spooled before printing, the Ginsu print drivers will be able to adhere to the user's stacking order specification. In the event an entire document cannot be spooled (e.g. a no disk space condition), the stacking order specification can only be maintained with the cooperation of the application. If the application knows the user's stacking order specification, and can determine the paper stacking mechanism of the printer, then it can send pages to the driver in the order needed to maintain the desired stacking specification. The Ginsu Application Interface contains a description of

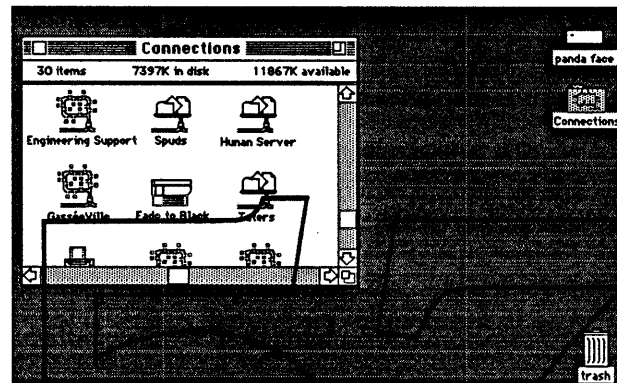
the routines which will enable applications to do this. In the absense of application support under these conditions, the stacking order cannot be maintained.



## 7.0 View from the Finder

### 7.1 The Basic Chooser Interface.

Recent designs of the chooser have stressed browsing the network through a Finder level window<sup>1</sup>. This "Connections" tool would be the user's gateway into the network. In addition to a more Mac-like direct manipulation of network devices, users could also copy favorite devices to the desktop or folders, drag documents directly onto printers or file servers, apply NuFinder Views to the window displays, and use "Get Info..." on all network objects.



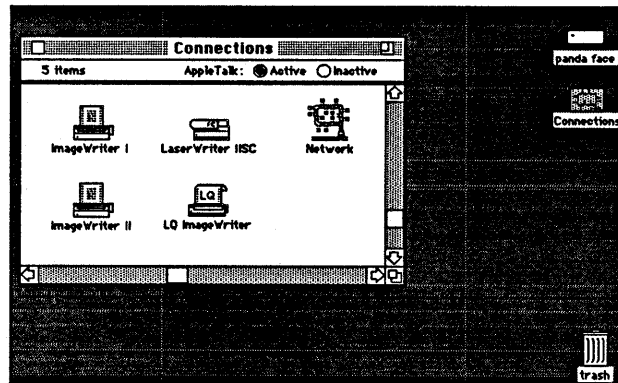
Network only Chooser (Annette Wagner)

Unfortunately, its very "Finder window" look could cause some confusion as many Finder actions like dragging to the trash or duplicating icons would be either impossible or significantly changed. Also, if this approach wants to capture the full functionality of the original Chooser, it needs to be extended to include non-AppleTalk devices.

### 7.2 The Solution to all the World's Ills.

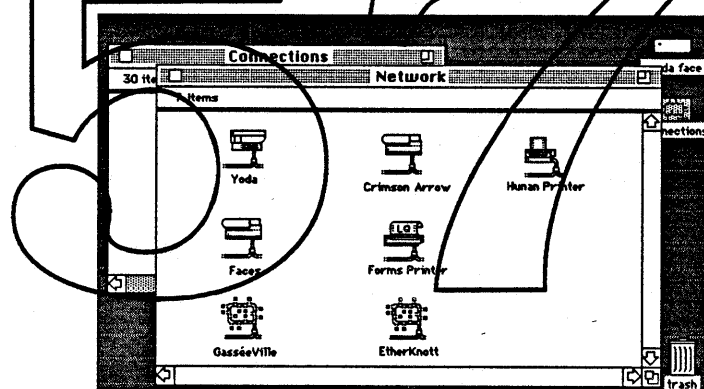
The current proposal is to put the network into a Network icon within the Connections tool and all non-network devices are now put at the top level. All devices in this window appear if their accompanying drivers are in the system folder. To keep users from mistakenly using the Connections Tool exactly like a Finder Folder, an attempt has been made to visually distinguish the window. This is only an attempt, there is no doubt we can do more visually.

1. "Report on Interface Designs for Choosign Network Devices" by Laurie Vertelney and Jack Palevich and Annette Wagner's "NuFinder Protos" Stack.



Current Proposal

Opening the Network icon displays all devices in the current zone with additional icons for other zones on the network. The Network icon and the zones are explicitly not made to look like folders to avoid their implied behavior. Folders can be made, renamed, and shuffled; none of which can be done to AppleTalk zones (yet...)



This keeps the same advantages of the original design but also has the disadvantage of making users dig through one more level to get to the network. While this is slightly annoying from a speed of access point of view, the extra hierarchy also organizes the devices a bit and for the moment, is not considered a critical problem.

Another disadvantage is the "appearance" of the direct connect devices at the top level. As many of these devices are quite dumb, it's technically difficult to have them appear (and disappear) as cleanly as AppleTalk devices. More importantly, these devices will have a different behavior than AppleTalk devices in that their appearance doesn't guarantee existence. If the user drags a document to the ImageWriter icon it's possible to get a "Printer not there. Check connection" type of alert. There is concern that these dual behaviors between

network and direct connect devices might be confusing.

An open question is adding and removing the Connections Tool from the desktop. It is not a CDEV or a DA. It needs to be an integral part of the Finder environment to encourage users to drag their favorite devices to the desktop or folders. This shouldn't be a difficult problem, it's just unresolved.

The other open issue is turning AppleTalk on and off. The current design is to put it only at the top of the Connections Tool window. This gives good graphical feedback in that by turning it off, the Network icon will immediately gray.

A final concern is that this model is stretching the Finder's definition and use of a window. We want to make the Connections Tool an integral part of the environment so users intuit the benefits of copying the devices to the desktop. Yet, as the physical model of the net prevents some Finder actions from applying to the window we don't want it to imply more than it is by making it look too similar. While the current design seems to fit within these two paradoxical constraints, we need to treat this as a major extension of the Mac interface and test it accordingly.

### 7.3 Another Solution to all the World's Ills.

A simpler approach could be to totally wimp out and keep the existing chooser interface but link it with SOFAs (no pun intended). The SOFAs would give a large chunk of the functionality described about in that users could still "Open" and "Get Info..." on the SOFAs. It would also maintain compatibility with bizarre third party RDEVs that would need to be revved to work within the new model. While practical, this solution just isn't that much fun.

### 7.4 Printing icons in the Desktop.

Whatever approach we take, once we have printer icons on the desktop, "Get Info..." will give configuration information: attached hard disks, input bin contents, plug in cards, etc. could all be shown here. Opening or double clicking on a printer would show all pending jobs. Printing servers, by having jobs spooled directly to their local hard disk, would be able to show all network jobs as opposed to a classic LaserWriter just showing the user's queued jobs.

One of the biggest advantages of desktop printing icons is that in larger installations, an administrator can "pre-choose" a group of printers. A new user can just walk up to a system and start printing and switching between printers without ever needing to use the Connections Tool or even understanding the concept of a network.

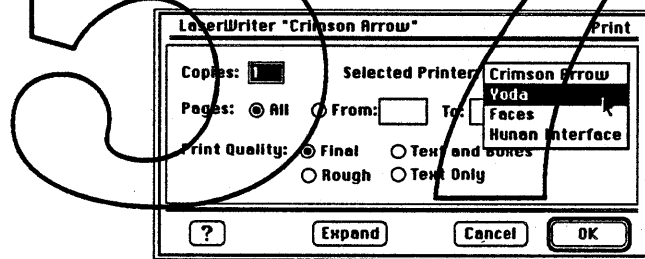
The biggest drawback to this model is that by making the choosing task move from the original "One at a time" model to the group of printers, the novice user

must now copy at least one printer to the desktop in order to print. Even though we can argue the Connections model is intuitive, flexible, and all sorts of other nice things, it is **still** new and must somehow be learned the first time. The act of copying a printer to the desktop doesn't obviously mean that we will now be using it.

In our favor, a newly installed system has no known printer. At print time we can now determine this and give appropriate feedback. The most popular idea is at print time just tell the user to go to the Connections Tool and copy a printer to the Desktop. A minimalist approach to be sure but this is significantly better than today, where we just throw up an alert that basically says the ImageWriter is unplugged.

### 7.5 Dialog Printing with the new Chooser.

Printing poses a particular constraint in that it is usually done from within a modal dialog; there is no direct manipulation environment to select a printer at print time. There are currently two proposed methods for selection. The first is to let the user choose between a group of "favorite" printers. These would be printers dragged out of the Connections Tool onto the desktop or into a folder. The Finder could keep track of all printers wherever they are on the system disk and then at print time, let the user select a printer from a popup menu:



Selecting from Desktop Icons

Another approach is to access the Connections Tool through a StdFile interface. As the Connections Tool zone hierarchy now mimics a nested set of folders, it would be very easy to map into StdFile.

These two methods are not exclusive. The popup could certainly have "Choose Printer..." as the last item in the menu to mix the two approaches. The concern here is the existence of two parallel interfaces to the Connections Tool along with the problem of designing around "power networking" concerns. For printing, browsing the Connections Tool for available printers and then copying it to the desktop would be a distinct task from selecting one of these printers for output. Most users use very few printers, even in a complex network environment. They would browse the network a few times to find their printers and then never use it again. **It isn't critical to integrate browsing with selecting for the majority of**

our users. The critical judgment is whether is it worth the potential user confusion to give power network browsers this valuable feature. It's not an obvious call. For this reason, we are going to start soon some user tests on a mix of the discussed models to get some feedback.

### 7.6 Defaults and Advanced Options.

One important task that came out of our user classes was that some users need to change the system defaults for some basic choices. For example, most European users want PageSetup to default to A4 paper and many law and accounting firms want to use US Legal paper all the time.

At the other end of the spectrum, there are high end users who will need to configure their large Mac shops for specific print tasks. For example, diskless Mac shops need to stop spooled files from going to the boot disk (which may be too busy or too small) to another AppleShare volume or even a dedicated printing server.

From the Finder, opening or double clicking on the Printing icon, which is now distinct from the individual driver icons, will let the user set system defaults. There are currently two approaches to what exactly will happen when the printing icon opens: 1) a small CDEV window will open or 2) a small application will launch. The CDEV has a more limited interface but also has smaller memory requirements, keeps the user centered in the Finder environment, and is also potentially available from a DA. The application has much more flexibility in design and growth potential in future versions. The choice is currently unresolved.

In either case, the user will be able to bring up the PageSetup and PrintJob dialogs, having full use of all panels and their choices, and then save their settings. There will also be the choice to return to "Standard Defaults." In addition, they will be able to permanently turn on the extended dialogs. This is for those users who find they are constantly using either our extended panels or an applications and don't want to always go through the novice dialog.

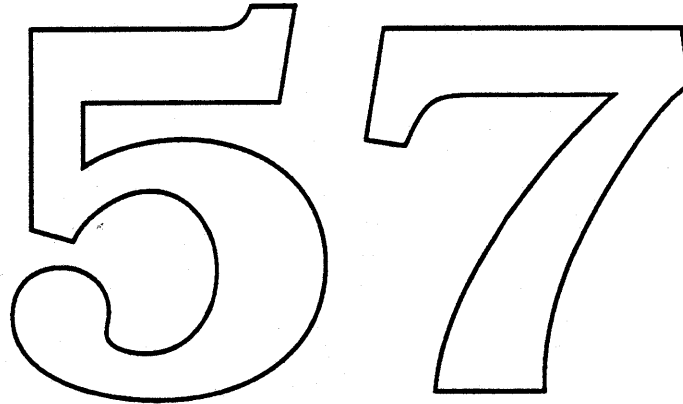
This defaulting mechanism is not meant to be exclusively for power users; it's important for any user to easily change the default PaperType to A4. There is some concern though, that users won't know to open or double click on the printing icon. As NuFinder will make opening Finder icons a more rewarding task with DAs, CDEVs and some AppleTalk devices all "openable", this might not be as obscure as we think.

As an extension for administrators who have special configuration needs for printing, they will be able to choose to see in addition to the extended panels the advanced panels. These advanced panels control many of the system level behaviors such as spooling, input bin mapping, and PaperType mismatching.

Administrators can then see and change these panels and save them as new system defaults. This is the last tier in our scalable interface. We feel that the most common use of these advanced panels will be by system administrators that need to tune their specific site by setting the parameters for all users. They will only turn them on in the defaulting application, set their defaults, and then turn them off from view again.

Note that as these panels show up in the extended dialogs. Even though they are turned on in the defaulting mechanism, they will be available each time the extended panels are used at print time. While it is not normally to be used on a job by job basis, high end users have that choice.

Another approach was to put a "Defaults" panel in the extended control panel interface for both PageSetup and PrintJob dialogs. Clicking on this panel would then let the user then move between all panels and change their default settings. This was rejected as it seemed to complicate the user model of the dialogs.





## 8.0 Page Setup Dialog

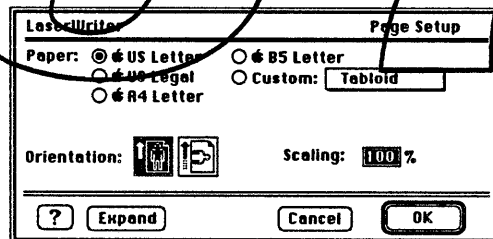
### 8.1 The World of No Dialogs

It is very important to remember that many users will never have to use the PageSetup dialogs. This is especially true as changing between printers no longer requires PageSetup. Also, larger installations can now have an administrator use the defaulting mechanism to tune the system to their specific environment.

### 8.2 The Novice World

The current dialog design is very similar to the old dialog with the exception of the "Custom: Tabloid" control. With variable number of PaperTypes, we needed to extend the standard radio button grouping of fixed page sizes. As most users will only use the standard sizes, we didn't want to have the new choice set confuse the entire selection process. We opted for the standard radio button set with the addition of a pop-up radio button choice for the other types. This was preferred over a scrolling list as it was less intrusive on the users first view of the dialog.

Originally, only the paper choices and orientation were to be in this dialog. Scaling was also put in because many users in groups 2 and 3 may need only this feature and we didn't want to force them to go into the extended dialogs. We've gone back and forth on the placement of all 3 of these sets and feel we may want to explore further simplification by moving some to the extended dialogs.

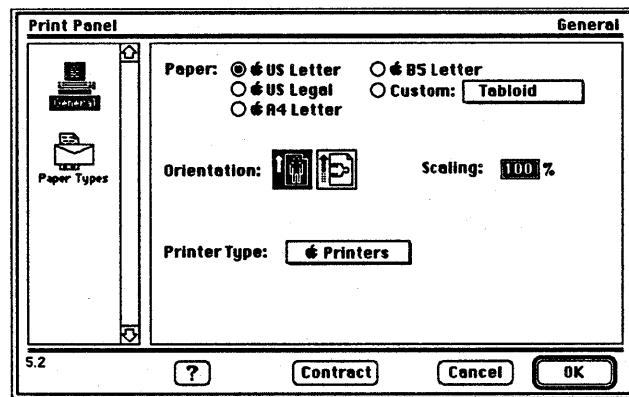


The Novice Dialog

### 8.3 The Extended World

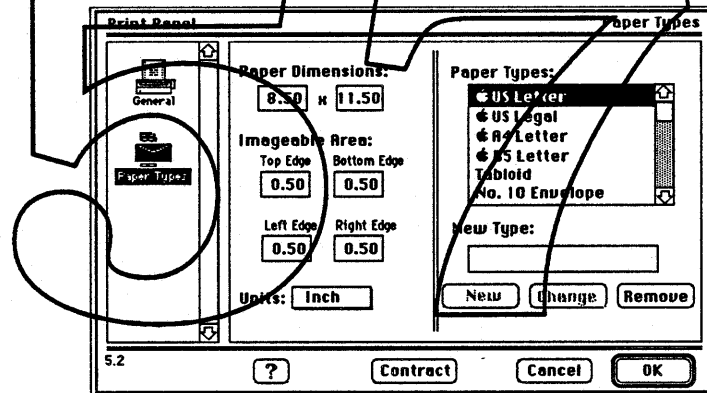
On clicking in the "Expand" button, the Novice dialog expands into the control panel interface. The same choices in the previous dialog are there, with the addition of the "Printer Type:" pop-up menu. This menu allows more advanced users to select a specific printer for output. The default settings is Printers. This name is intentionally similar to the PaperTypes to imply a standard "no-fuss" choice. With the Printers chosen, the application is told the device resolution is 72 dpi, something which allows output to be easily printed on most printers. By choosing another printer, advanced painting and drawing applications can image at a specific device resolution. Also, printer specific

Papertypes would be available from the "Custom:" popup menu.



The Extended Dialog

By clicking on the PaperTypes icon, the following panel appears:



Custom PaperTypes

The panel lets a user define a new PaperType. The two common uses for the panel would be to 1) just give a new name to an Apple type or 2) define a very specific size/imageable area combination. The first will be very common when we have printers that support multiple input bins. Users can get all of the benefits of the Apple paper types but also get automatic bin mapping. The obvious case here would be defining a "3-Hole Punch" PaperType.

The second is a more advanced use of the panel. It would be useful in creating a custom size for a company letterhead or an odd sized form. Note that applications specializing in custom papersizes like check writing or label making programs, can add their own types choices available in the novice dialog box. Most users will be able to use custom PaperType and never see this box.

## 9.0 Print Job Dialog

This chapter provides a brief introduction to the new Ginsu Print Job dialog. It provides descriptions for each of the perceived users of the dialog: the "Print One" user, the novice user, and the advanced user.

### 9.1 The World of No Dialogs

The world of no dialogs corresponds to the "Print One" user of the Print Job dialog. For this user, the Print Job dialogs are never displayed. Printing of the target document is initiated using the most recent Print Job settings saved with the document, and with the "copies" attribute set to one. These settings may be the default dialog settings. This option is most useful for the user that has already configured the Print Job settings for his document, and simply wants it to print without being bothered by the dialog.

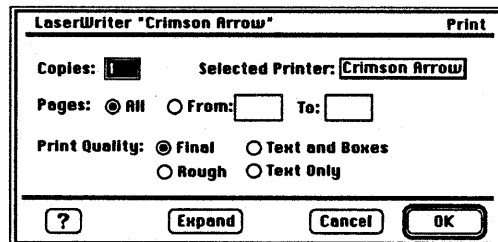
### 9.2 The Novice World

The novice world corresponds to the novice user's use of the Print Job dialog. The novice Print Job dialog includes only the most simple print job settings, and has been designed for use by the most novice Macintosh users. By specifying good default settings for options which do not appear in the novice dialog, we're able to provide users with a much simplified dialog. The novice Print Job dialog includes the same settings and screen layout for every device we support. Thus, for the novice user, printing to different devices does not require the user to adjust to new Print Job interfaces.

One open issue we're currently debating is whether to include additional information in this dialog, which describes some of the "extended" dialog settings that may impact novice users. For example, on a film recorder it's potentially important for even the novice user to know what type of film is loaded into the device; and for an ImageWriter LQ user, it's important for him to know what size paper we think is in the printer, so that his printer isn't inadvertently damaged. In these situations as well as others, it may be important for novice users to know some of the other print job settings. The problem is to decide what information is important, and how to present it in a palatable fashion to the novice user. We are still evaluating possible alternatives.

An example screen shot of the novice dialog for the LaserWriter follows. This dialog contains a specification for the number of copies to print, the page range to print, and the quality at which to print the document. In addition, the dialog includes the specification of the printer on which to print the document. The pop-up menu includes a list of the user's preferred set of printers. This facility allows the user to select the printer he/she wishes to print to, regardless of the type of printer. The dialog includes the standard "OK" and "Cancel" buttons, as well as "Help..." and dialog expansion buttons. Though the help facility functions are still being debated, at a minimum, clicking "Help..." will produce a scrollable screen of context sensitive help related information. The dialog expansion button

extends the capabilities of the dialog to provide many more options. When the dialog is expanded, the window is enlarged and new items will appear, but the existing items will not change location. The expanded dialog is primarily for use by more advanced users and is discussed in the following section.



The Novice Print Job Dialog

### 9.3 The Extended World

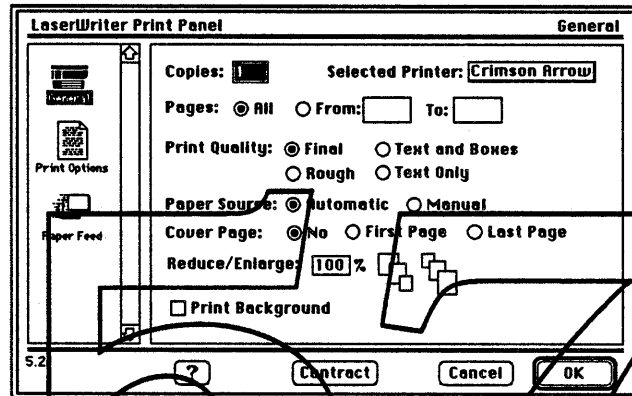
The extended world corresponds to the intermediate to advanced user's use of the Print Job dialog. The extended dialog contains numerous other application and device specific options that the user can choose from. Example screen shots for the extended dialog of the LaserWriter follow. These screen shots list a possible minimal set of dialog panels. There are a number of other panels that might appear (much of it depends upon the outcome of how defaults are eventually handled), and we're still evaluating the appropriate mix.

The most notable difference between the novice and extended versions of the dialog is the addition of the scrollable area on the left side of the screen. Each of the icons listed there provides additional print job options. In order to view an icon's options, the user would select the icon, and the panel to the right of the scrollable area would change to reflect the new options. This dialog works in a manner analogous to the current Control Panel. Depending on the device, the number of available icons (and thus print job options) will vary.

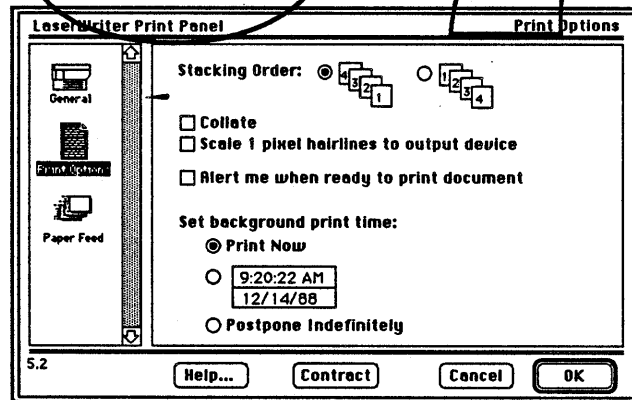
The Print Options panel includes common optional settings for printing to a LaserWriter. At varying times, these settings have been spread across a number of different panels. Currently, they're combined into one panel, but this may change. The General panel includes the novice dialog settings, as well as some other general settings such as the "Paper Source" specification. The "Paper Source" setting indicates whether the printer is to be manually or automatically fed paper. If manual is chosen, and the document to be printed contains multiple paper types, the user can optionally invoke the Paper Feed panel, in order to specify that only some of the document's paper types are to be manually fed into the printer.

As was mentioned in an earlier section of this document, device specific attributes of paper types would be specified at print job time (e.g. medium). The attributes

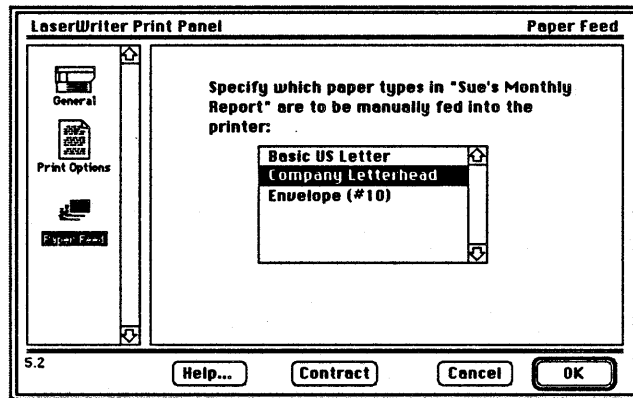
that could actually be set would vary from device to device, and many devices would not have any device specific settings. We're still debating the exact mechanism for specifying these attributes. One possibility is to allow users to create a "media list", which would simply be a list of device specific settings, with each list entry designated by a unique name. To attach device specific settings to a paper type, users would be allowed to bind an entry in the "media list" to a paper type used in the document being printed. Another alternative is to simply allow users to select a paper type (from a list of types used in the document), and then set the device specific settings which apply to it. We're continuing to evaluate these and other alternatives.



LaserWriter General Dialog Panel



LaserWriter Print Options Dialog Panel



LaserWriter Paper Feed Panel

57

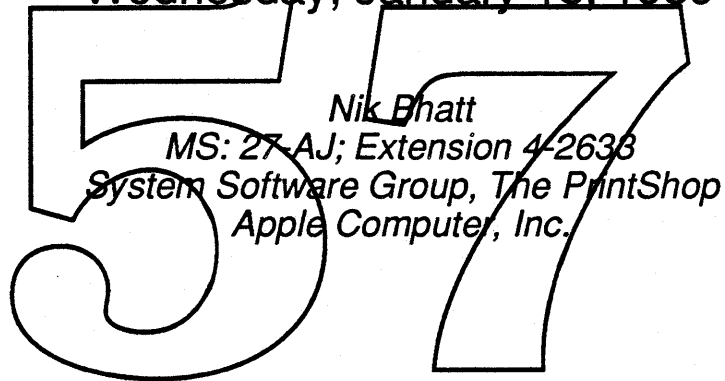
57

# The Ginsu Project

The Ginsu Architecture

Just-Poured-Concrete version

Wednesday, January 18, 1989



*Nik Bhatt*

*MS: 27-AJ; Extension 4-2638*

*System Software Group, The PrintShop  
Apple Computer, Inc.*

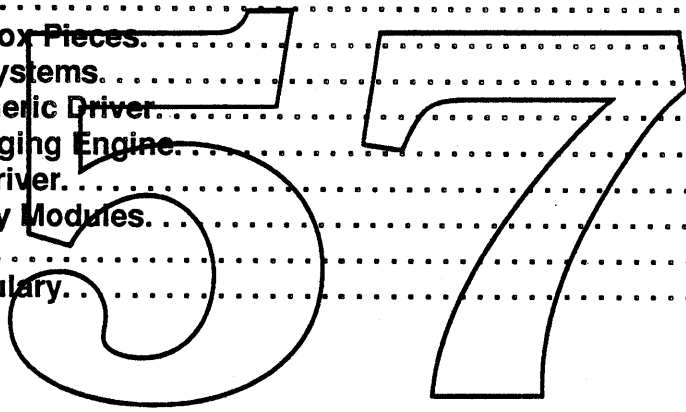


**Captain Ginsu Says: Eat This Architecture!**



# Table of Contents

Architectural Philosophy.....	1
Top 10 PrintShop Slogans.....	4
Feature Set .....	5
Quality Modes.....	7
The Glorious Architecture.....	9
Functional Description (MacDraw II).....	9
I. Overview.....	9
II. The Old World.....	9
III. The New World.....	9
IV. Definitions.....	10
V. Into the Breach.....	15
Architectural Pieces (MacDraw II).....	15
Job Flow (MacDraw II).....	15
1. PrGlue.....	15
2. Glue Interface.....	16
3. TackleBox.....	17
a. TackleBox Pieces.....	18
4. Imaging Systems.....	20
a. The Generic Driver.....	21
b. The Imaging Engine.....	21
5. Specific Driver.....	23
6. Personality Modules.....	25
Open Issues.....	27
Architecture Vocabulary.....	28



## Architectural Philosophy

**[0] Life is meaningless.**

**[1] People are inherently evil.**

**[2] I print, therefore I am.**

**[3] Segmentation:**

[a] We will adopt the system's segmentation model, if one, or

[b] A "son of PrGlue" dispatcher which will load in code resources on demand, but leave the entire execution chain locked.

The segmenting model for the system is the preferred choice, but must be designed with Printing in mind. Our code will be heavily segmented and we will make calls across files (for example, the Specific Driver will call the TackleBox, which is in a different file). This implies the development of a clean jump table definition that persists easily across files.

**[4] Low-level driver:**

The low-level driver will still be supported for old applications. Its use is strongly discouraged, and may be dropped at any time. Print streaming and other uses of the low-level driver are not "Macintosh printing" — after all, you can't stream to a LaserWriter. Further, print streaming is a device-dependent method of printing. For old applications, we will render low-level calls through the draft interface of the driver. Via draft, we will offer most of the old functionality people wanted.

**[5] PrGlue: Old glues will be supported for old apps. New apps must use the new glue.**

The trap will be patched to support the TackleBox. PrClose, for example, will call PrCleanup in the TackleBox to deallocate storage and close files. The current PDEF structure will be maintained, since it's a reasonable method of division and lets us support old glue and new glue the same way.

PDEFs (code resources) are defined in the old drivers as follows:

PDEF 0: Draft — the mode the LaserWriter is always in, and the "draft" quality in ImageWriters. See also Selective Service, Wind, and Beer.

PDEF 1: Spooling — exists in all devices. Support backgrounding too

PDEF 4: Dialogs — self-contained unit to display dialogs and validate print records

PDEF 5: Imaging — images a spool file

PDEF 7: PrGeneral — considered obsolete in Ginsu. Supported for old apps

PDEF 10: PAP — AppleTalk protocol code called internally. Obsolete in Ginsu.

Technically, there are two spooling PDEFs in the LaserWriter because it operates differently in the foreground from the background. Ginsu will support all these PDEFs for old apps, and PDEFs 1, 4, and 5 for new apps.

In Ginsu, all devices will have four quality modes (see below), so the "Draft" mechanism will be rolled into the Imaging Systems. PrGeneral is being replaced by a new call set which is a superset of the old call set. Protocol support is rolled into the Async I/O Handler in the TackleBox.

**[6] Mixing of imaging models.** PostScript and QuickDraw can still be mixed each has abilities the other does not have. We *strongly* discourage model mixing, because it prevents us from providing all of the functions of QuickDraw (e.g., transfer modes).

*Important: We will not allow mixing of Skia and anything else.*

**[7] Additions to QuickDraw.** The PrintShop does not wish to extend QuickDraw wholly inside the Printing Manager. Any extensions to Printing QuickDraw should be available for the screen. Though we are removing state comments from the model, we must also unify imaging in general. As Whorf on the new Start Trek would say, "We'll do whatever it takes to clean up our model."

**[8] WYSIWYG or Death!**

**[9] There is no [9].**

**[10] Data structure access:** Applications will have *no* direct access to internal printing data structures. We will offer a procedural access to relevant fields.

**[11] Avoidance of incest:** Ginsu drivers and Handlers will have narrow interfaces among them, low data flow, and procedural access to the internal data structures of other units. This allows us to make substantive changes with confidence.

**[12] Device-independent Spool Files:** We will make spool files as device-independent as possible. Users will be able to re-route spool files to a different type of device from the one the job was spooled to.

**[13] Joining System Software:**

Versions of printing will be tied to versions of System Software. It will no longer be possible for a user to mix and match Systems and LaserWriters. This permits us to get feature enhancements from the rest of the group, without building them into the drivers like we do now — and you all want to help us out, don't you...

**[14] Device-is-master model:**

We actually have some of the functions in place to move to a "device is master" model from the current "screen is master" model. Pink uses the model of "*space* is master", but they have a "device-independent" imaging model. For the "device is master" model, the application should use the metrics of the device class (say, the LaserWriter at

300 dpi) and render the screen based on those metrics; currently, applications render with screen metrics and we have to scale to printer metrics. This scaling creates artifacts and decreases the quality of the final image. If the application renders at device resolution, it is already using the metrics of the printer. However, it should format the screen in a way that both follows the printer metrics, and is easy to read. Some ideas:

[1] *Use the Layout Manager to display text.* The Layout Manager should provide "good-looking text" at any resolution.

[2] *Applications should provide a "zoom" function to draw fine details.* Zoom is extremely powerful because it can give the user pixel-by-pixel control over the final image. An application should offer a wide range of zoom factors (not 25, 50, and 100%), as well as "smart zoom." Smart zoom guides the user toward zoom factors which look best on the target device (300% and 150% on the LQ). Thus an application would offer the entire integer range from, say, 25% to 400%, but flag 150% and 300% as "recommended" zooms.

[4] *Bitmaps can be rendered via Zoom.* They would be displayed on the screen reduced by CopyBits. However, one could zoom in to view the complete map (à la Draw II).

[5] *There's always the screen.* Applications can always format to the screen. This is not a break in the model — the screen is a device too.

[6] *Formatting should be done to the non-volatile features of the device class.* We do not want applications formatting to the fonts on a specific printer, nor to the type of color ribbon installed (e.g., three-color or four-color). One formats to the device *class*, not to a particular device. This is inherently more stable, and lets the user print the document on a variety of physical devices without trouble. An application can query for a table of resolutions paired with color space, and then select a pair. In general, with higher resolution, fewer colors are available.

**From the home office in Cupertino, CA:  
Top 10 PrintShop Slogans**

[1] Save, then print

[2] Save, print, crash, reboot, print...

[3] Whine, then print

[4] Print, then whine.

[5] WYPIWYG: What you print is what you get.

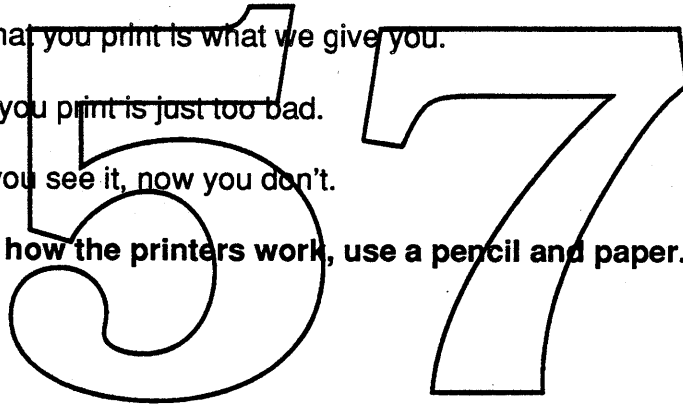
[6] WYGIWYG: What you get is what you get.

[7] WYPIWWGY: What you print is what we give you.

[8] WYPIJTB: What you print is just too bad.

[9] NYSINYD: Now you see it, now you don't.

[10] If you don't like how the printers work, use a pencil and paper.



---

## An incomplete feature set of the architecture

---

[1] **All devices** will be able to print in the background or foreground.

[2] **We will** implement the same feature set on *all* devices (e.g, PicComments).

[3] **We will** support four quality modes on all devices (Final, Rough, Text and Boxes, and Text Only — see below).

[4] **All devices** will be able to print with little or no disk space. This is called “Dire Straits Printing,” and requires some extremely minor application assistance. Dire Straits actually comes in two flavors — first, low disk space, in which we could not spool a single page to disk, and second, no disk space, in which all disks are locked to us. This will significantly help us handle complex documents on a wide range of systems.

[5] **We will** be able to support transfer modes in the LaserWriter if all of the data sent is QuickDraw. This is an option the application can enable (because it may be slow). We can also support regions in the LaserWriter, to a limited degree.

[6] **We will** be able to support Skia when it is available. We will **not** support mixing QuickDraw and Skia on a page; the application will have to emulate QuickDraw with Skia if it insists pixel-for-pixel compatibility with QuickDraw.

[7] **We will** support full color with Color QuickDraw. We will half-tone and dither images on all devices to increase the color and grey-scale capabilities of the device.

[8] **We will** support the Bass group by providing them with whatever rendering information and support they need. We will do the same for Boffin (Layout Manager).

[9] **The architecture** will be *fully* re-entrant. We will be able to support multiple print jobs at once within an application; each of these jobs can be destined for a different device. This will require that services we use be re-entrant as well (*notably QuickDraw*).

[10] **PrintMonitor** will be able to run more than one print job on a machine at a time.

[11] **We will** attempt to perform some analysis of print jobs and printer use; for example, if a Print Station is connected to three identical NTXs, it will re-route output to the least busy NTX. Re-routing jobs over between geographically distant printers will not be done automatically — we will provide load information on devices for users, but they will have to re-route the jobs themselves. This avoids unhappy surprises, like, “Whoops, the driver sent a copy of my Career Limiting Memo to Gifford’s printer. Time to work on the ol’ resumé.”

[12] **We will** permit users to take a spool file destined for a device and drag it to a different device. The user can also drag spool files across logical device classes (e.g., drag an ImageWriter spool file to a LaserWriter), but will be alerted that the output may not be perfect.

[13] **We will** create a remote transmission protocol which will enable us to image print jobs remotely, and to send imaged jobs to another Macintosh for printing. The first capability is new — we will send the spool file over the network and let another Macintosh (an Imaging Station) render the data. This will speed up background printing, reduce network usage, and reduce the chance of nuclear war. The second capability is similar to a upgraded LaserShare and will fight crime in the inner cities.

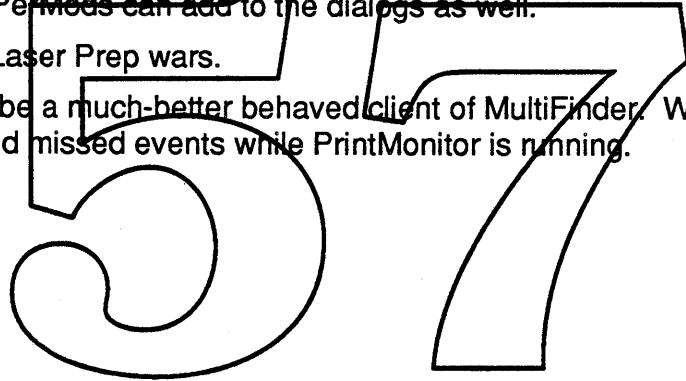
[14] **We will** provide a substantial application interface to printing data structures so that applications get be more intelligent about how they print. No direct access is permitted.

[15] **We will** provide an extensible dialog mechanism which simplifies printing for novice users, and while providing applications with a mechanism to support power users.

[16] **Personality Modules** provide a mechanism by which third-party hardware companies can add hardware extensions to our devices and control those extensions through our drivers. ~~PerMeds can add to the dialogs as well.~~

[17] **We will** end Laser Prep wars.

[18] **Printing will** be a much-better behaved client of MultiFinder. We will strive to eliminate jerkiness and missed events while PrintMonitor is running.



---

## Quality Modes available on all devices

---

**Final:**

The best we can do. Highest resolution on the device, with full color (including dithering and halftoning), all objects rendered, Line Layout via the Layout Manager. All fonts available.

**Rough:**

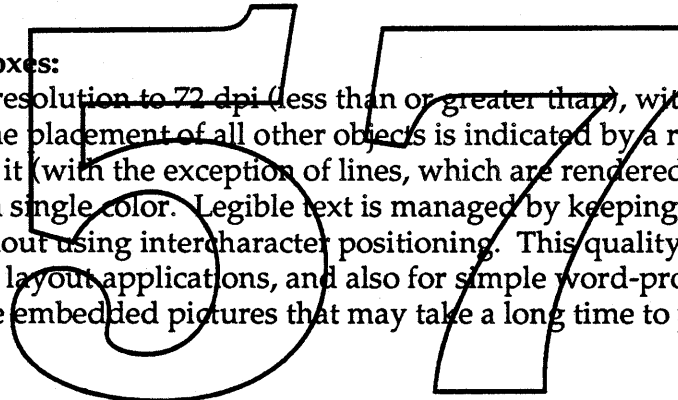
Closest resolution greater than or equal to 72 dpi, with all objects rendered, with partial color (the printer's native set). Full Line Layout performed if necessary (i.e., if the resolution is not 72 dpi). All fonts available. This mode is especially helpful for graphics applications and for tasks in which the non-text, possibly colored, objects have special significance. This is the "big picture" or proofing quality.

**Text and Boxes:**

Closest resolution to 72 dpi (less than or greater than), with text rendered "legibly". The placement of all other objects is indicated by a rectangle with an X through it (with the exception of lines, which are rendered). All fonts available; we use a single color. Legible text is managed by keeping logical lines together, but without using intercharacter positioning. This quality is especially helpful for page layout applications, and also for simple word-processing tasks where there are embedded pictures that may take a long time to print.

**Text Only:**

Resolution is defined to be that which lets us render text most quickly. Single color, no line layout (but logical lines are maintained), Font Substitution with printer's native set (e.g., ImageWriter's) or with cached fonts (e.g., LaserWriter). No other objects are rendered. This quality is especially helpful for printing long text documents (including listings) at very high speed.





*Examples of Different Qualities:***[1] ImageWriter I:**

Final: 144 dpi, monochrome, half-toning available, Boffin (Layout Manager) used, all objects drawn. All fonts.

Rough: 72 dpi, monochrome, no half-toning, Boffin used (though no extra layout is necessary), all objects rendered. All fonts.

Text & Boxes: 72 dpi, monochrome, no half-toning, no extra layout done, all objects but text and lines are rendered as boxes. All fonts.

Text Only: Native character set, quality set on front panel, no extra layout done, no graphics.

**[2] ImageWriter II:**

Final: 144 dpi, eight colors, half-toning and dithering available, Boffin used, all objects drawn. All fonts.

Rough: 72 dpi, ~~eight colors, no half-toning or dithering~~, Boffin used (though no extra layout is necessary), all objects rendered. All fonts.

Text & Boxes: 72 dpi, ~~monochrome, no half-toning~~, no extra layout done, all objects other than text and lines are rendered as boxes. All fonts.

Text Only: Native character set, quality set on front panel, no extra layout done, no graphics.

**[3] LaserWriter NT/NTX:**

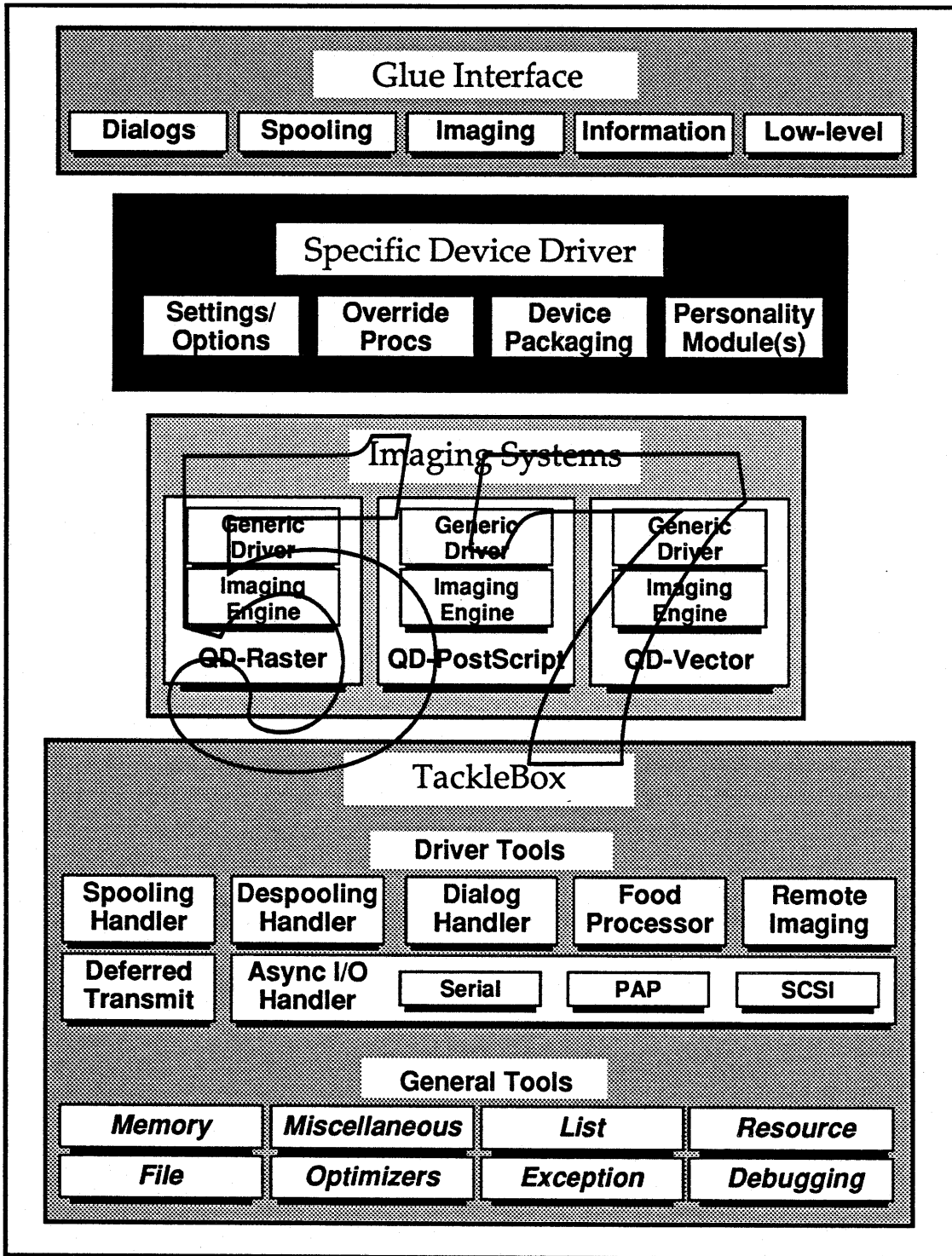
Final: 300 dpi, monochrome, half-toning available, Boffin used, all objects drawn. All fonts.

Rough: 75 dpi, monochrome, no half-toning, Boffin used, all objects rendered. All fonts.

Text & Boxes: 75 dpi, monochrome, no half-toning, logical lines maintained, all objects other than text and lines are drawn as boxes. All fonts.

Text Only: 75 dpi, Courier 12pt., logical lines maintained, no graphics.

# Ginsu Architecture: Functional Components



## The Glorious Architecture

### I. Overview:

This document describes the Ginsu Architecture by defining its various pieces. These pieces interact to execute a print command, and there is a section describing each major chunk.

---

### II. The Old World:

First, a brief description of how printing struggles in the current system. There are "drivers", and there is the "glue." Drivers are self-contained entities which handle all parts of the job; they have their own dialogs, their own spooling systems, and their own imaging and transmission code — no code is shared among drivers. The glue takes Printing Manager calls and routes them to a driver; the driver opened is the one for the "system printer." The system printer is selected from Chooser and affects all open documents. Users install a new driver by dragging it into their system folder.

---

### III. The New World:

Printing under the Ginsu model is somewhat different. There are still drivers and the glue, but there is also the TackleBox and Imaging Systems. We added the latter two in order to share code better.

The TackleBox provides most of the general purpose utilities and high-level tools needed by all drivers. For example, to manage memory better, it is allocated through calls in the TackleBox. To avoid the duplication of spooling code, there is a Spooling Handler in the TackleBox which all drivers call to spool.

Imaging Systems contain all of the code necessary to image QuickDraw. Rather than have  $n$  copies of the code necessary to render QuickDraw into rasters for ImageWriters, there is only one. Drivers do relatively little — for the most part, they simply call the TackleBox or Imaging System to perform the desired task; however, they also have the ability to override any functionality we provide. The glue does roughly the same stuff it does now, except that the concept of "system printer" goes away, in favor of the "binding documents to printers" model (see Vocabulary).

## IV. Definitions:

**Interface Levels:** There are four sections to the new interface. They are described below:

- Dialog Calls* : All dialog and print record access calls (e.g., PrValidate)
- Spooling Calls* : PrOpenDoc, PrOpenPage, PrClosePage, PrCloseDoc  
: PrStartDoc, PrStartPage, PrEndPage, PrEndDoc.
- Imaging Calls* : PrNewPicFile
- Information Calls* : Any calls which return information about the state of printing.
- [ *Low-level Interface* : Supported for Ginsu-unaware applications only.]

**PrGlue:** The dispatcher between the application and the current driver. PrGlue is always linked to the application, and there are a number of different versions of the glue. The earlier glues were self-contained libraries; however, the most recent glue calls the PrGlue trap to do the bulk of the work. Ginsu-aware applications must link to a new version of the Glue which will call the Glue Interface.

**Glue Interface:** The front-end to PrGlue. It loads in a Specific Driver for use and routes calls from the glue to that driver. The Glue Interface lives with the TackleBox in an icon called "Printing." This icon is not a file, but a folder masquerading as an icon (this is possible in NuFinder). *Where stuff goes and how it is installed is a big open issue.*

**TackleBox:** The shared code which spools, despoils, and handles memory, files, and dialogs. It also contains the Async I/O stuff for transmission, the Food Processor for bitmaps, and remote imaging support code.

**Imaging System:** One of QuickDraw-to-Raster, QuickDraw-to-PostScript, or QuickDraw-to-Vector. It does most of the work of imaging — it contains both a Generic Driver, and an Imaging Engine. A single Imaging System serves more than one Specific Driver; therefore, it serves more than one type of device (i.e., device class). For example, the QuickDraw-to-Raster Imaging System serves all raster printers. Thus, LQs, ImageWriter IIs, PaintJets, and LaserWriter IISCs all use the same Imaging System. They use, however, different Specific Drivers (see below).

**Generic Driver:** A driver which calls the TackleBox to spool a job, despool it, and image. It controls that job, via the TackleBox, and is the foundation on which all specific

drivers are built. It is not capable of packaging data for a device or putting up custom dialog panels — it does, however, put up a standard set of panels for the target imaging model. There is exactly one Generic Driver per imaging system.

*Put another way...* The generic driver is the base class definition of the “driver” object. It includes code, calls, and tabular information about all device-independent aspects of a job. Some of the code or tabular information is stubbed out, if it is specific to a device class (e.g., device packaging).

---

**Imaging Engine:** A section of an Imaging System which takes a page of QuickDraw and renders it into another format (such as into PostScript).

---

**Specific Driver:** A combination of tables, override procedures, and code to drive a particular Device Class (defined below). It loads in the correct Imaging System and uses the Generic Driver to do most of the work of controlling the job. Calls are routed by the Glue Interface to the specific driver; that driver has the choice of executing the call, or routing it to the Generic Driver. For the most part, the call will be passed on; in special instances, however, the call will be handled by the specific driver (for example, if the specific driver wants to handle imaging differently for some reason).

*Put a different way...* A Specific Driver is also an instantiation of the “driver” object. It can override calls or provide any information contained in a Generic Driver. It can also call the “inherited” proc of the Generic Driver.

---

**Device Class:** A set of devices which share formatting attributes — resolution, paper/page sizes, color space, imaging system, *and user interface* definitely must be common. A user, *using the driver’s paper types*, must be able to print without clipping or scaling.

For example, “LaserWriter II US Letter” is a guaranteed “good size” on all LaserWriter II NTs and NTXs.

*Issue: Should a device class require a base definition of firmware instructions? That is, can two devices which share formatting attributes, but have different firmware instruction sets, be part of the same Device Class? We may need to find this out experimentally but I am against it.*

Justification for definition: We do not want unnecessary bulk in the system. Driving diverse devices (ah, alliteration) with the same driver automatically increases the disk requirements for users who have at most one or two devices. We are sharing code by putting the device-independent stuff in the Generic Driver, but the specific stuff should

be logically separate. Further, we shouldn't saddle the ImageWriter II owner with the firmware definitions of the ImageWriter I and of the Seikosha clone.

*Examples of Device Classes:*

[1] ImageWriter I, which is an incredibly stupid, 144 dpi, monochrome device. It uses the QD-Raster Imaging System.

[2] ImageWriter I Wide Carriage, because it can print paper sizes which the vanilla ImageWriter I cannot.

[3] ImageWriter II, because it supports color. AppleTalk and Serial versions are represented by the same driver.

[4] ImageWriter LQ, because it is 216 dpi. It uses the QD-Raster Imaging System as well.

[5] LaserWriter II NT/NTX, because they are both 300 dpi and support identical paper capabilities. They are both monochrome. They use the QD-PostScript Imaging System.

[6] LaserWriter Classic/Plus: They handle envelopes differently, and have a smaller imageable area in their default memory configuration.

[7] LaserWriter IISC, because it uses a different Imaging System from the other LaserWriter IIs. It also cannot image the same sized legal page as the others (due to memory limitations). It uses QD-Raster imaging.

[8] QMS ColorScript, because it has a smaller page size and a different color system than LaserWriters. It uses the QD-PostScript Imaging System.

**Personality Module:** A plug-in module for a Specific Driver which contains code to drive a particular extension to the device class. It supports anything which differs from the base definition of the device class. The following restrictions apply:

*The extension cannot affect imaging.*

*The extension can operate correctly, when given control at a small set of points during a job.*

*The extension must drive a piece of hardware, not a piece of software.*

**Examples:**

[1] Apple hardware extensions; therefore, the ImageWriter II sheet feeder is a PerMod. However, the color ribbon is not, since color is an imaging concern. Further, the ImageWriter II by definition is a color device (which also supports monochrome).

[2] LaserWriter Sheet Feeders (e.g., from BDT)

The set of PerMods need not be overlapping; if there is a LaserWriter IINT specific extension, then it is not used when printing to LaserWriter IINTXs, and vice-versa.

**If an extension affects formatting, a new Specific Driver must be rolled because that change violates the definition of the Device Class.**

Thus, if a sheet feeder requires so much memory that the NTX's ability to image a legal page is affected, a new driver must be created because the user can no longer print a simple legal page on all NTXs without worry that scaling or clipping may occur. Paper path is a different problem. We can alert the user if envelopes cannot be fed into the particular device because the sheet feeder blocks the envelope tray. We can offer different options at that point, including removing the sheet feeder, or perhaps scaling the page on that can be fed.

Personality Modules are *not* an instantiation of the driver object. They are clients of the Specific Driver, and are called by it at well-defined moments during a job. They do not override driver procedures. They are called by the specific driver at these times:

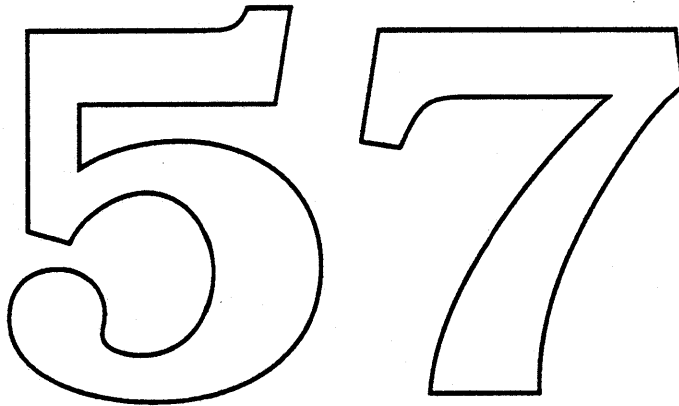
*OpenConnection, CloseConnection;*  
*StartPage, EndPage;*  
*StlDialog, JobDialog, PrintDefault, PrValidate;*  
*AbortPage, AbortJob, ResetDevice.*

Personality Modules have priorities; if it is necessary for one PerMod to be called before another, the first would be assigned a higher priority.

*Issue: who sets the priority: the user, the vendor, or Apple?*

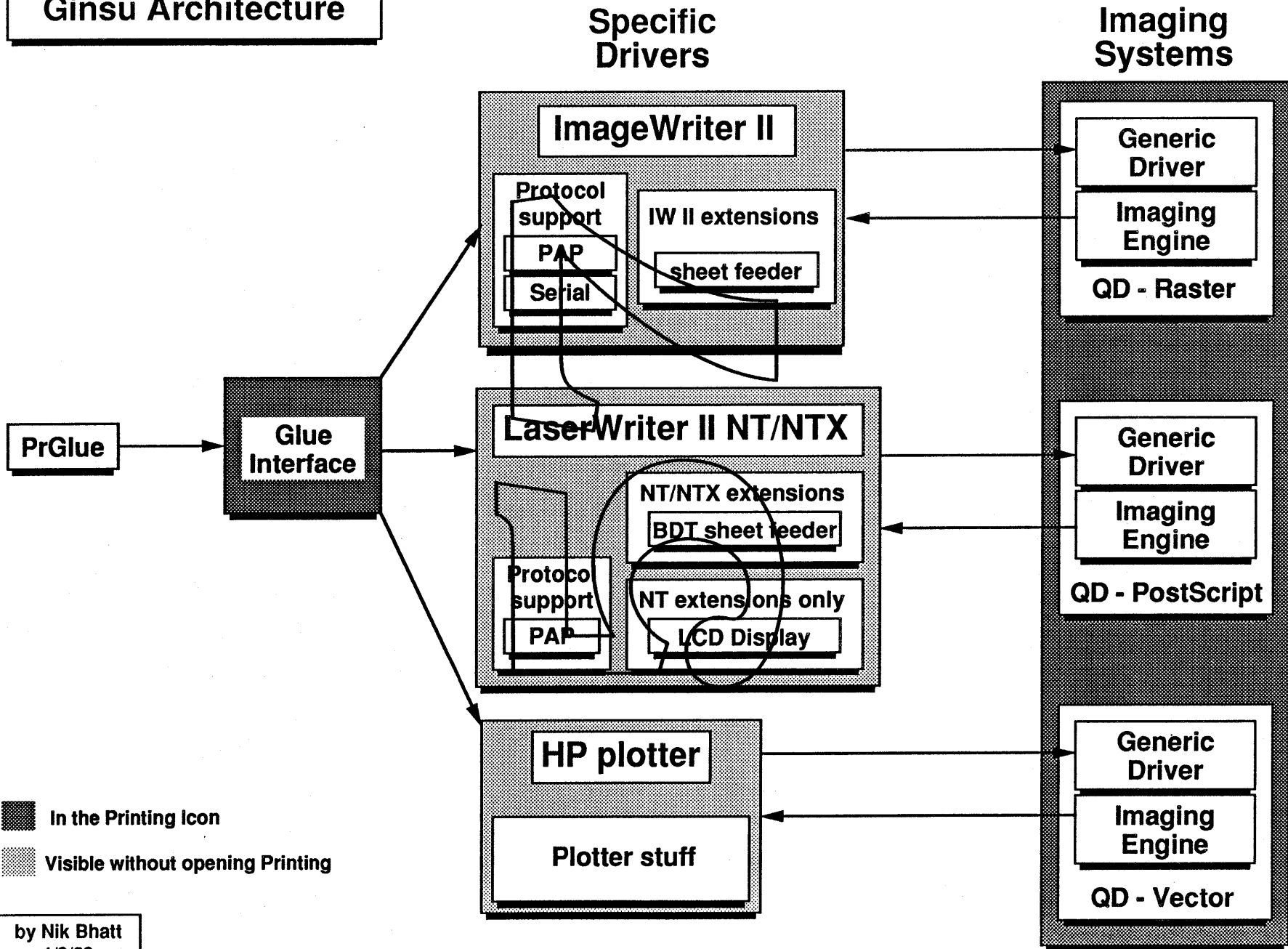
**Example:**

Given two PerMods — an LCD display to which we can send text for the user to read at the printer, and a BDT sheet feeder. It is of utmost importance that the display receive the StartPage call before the sheet feeder, so that the user can know which page to feed.

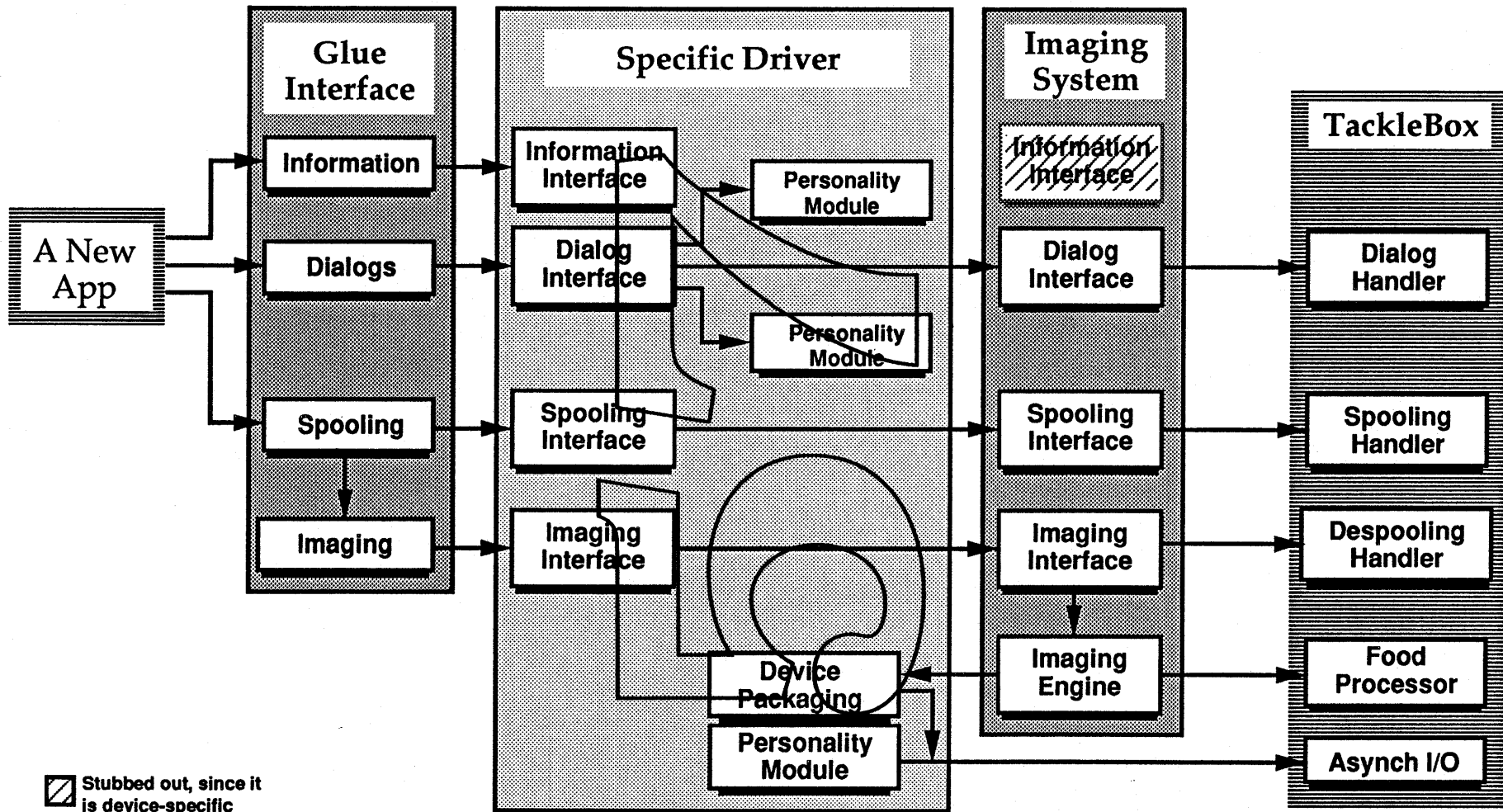
The image shows two large, hollow outline numbers, '5' and '7', positioned side-by-side. The '5' is on the left and the '7' is on the right. Both numbers are rendered in a simple, bold, sans-serif style with a consistent stroke width.



# Ginsu Architecture



# Ginsu Architecture: Job Flow



▨ Stubbed out, since it is device-specific

by Nik Bhatt  
1/17/89

An application makes a call to one of the three interfaces. The Glue Interface routes the call to the appropriate interface in the Specific Driver, loading that driver if necessary. The Specific Driver will usually route the call to the Imaging System. The Imaging System calls the TackleBox to do most of the work. The Imaging Interface is never explicitly called by an application (except PrintMonitor); it is called by PrCloseDoc.

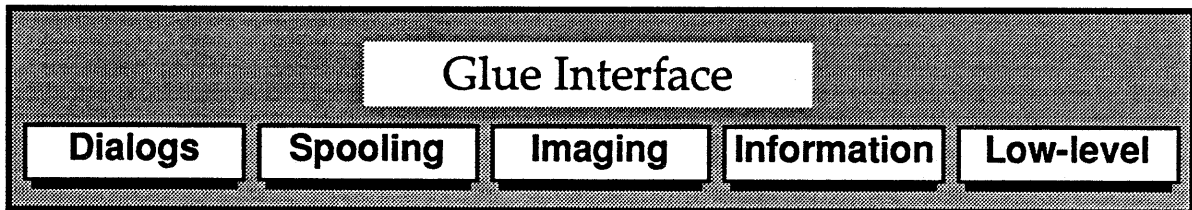
## PrGlue

**Definition:** PrGlue is the entry point for all Printing Manager calls. If an application makes a printing call like PrOpenDoc, the linker replaces the PrOpenDoc() call with a jump to a linked module called PrintCalls plus a selector for the call made. PrintCalls dispatches the call either within itself or to the \_PrGlue trap. Earlier versions of PrintCalls were self-contained; the most recent version jumps to the \_PrGlue trap to implement most of the calls.

**Currently:** The glue looks in the System file for a string resource which contains the name of the current system printer (e.g., "LaserWriter"). The glue opens that driver, opens the resource fork, and loads in the correct code resource for the call made. Then the glue locks down the resource and JSRs to the appropriate function. On return, the glue may unlock the resource, depending on a bit in the selector long word.

**In Ginsu:** The glue will always open the "Printing" resource file, and call the Glue Interface. The Glue Interface will decide which driver to open and what to call. This way, no matter what changes we make to the internal structure, the glue can remain unchanged. **Note:** The Glue Interface will never open an old driver. Old drivers from all vendors are considered obsolete and unavailable under Ginsu.

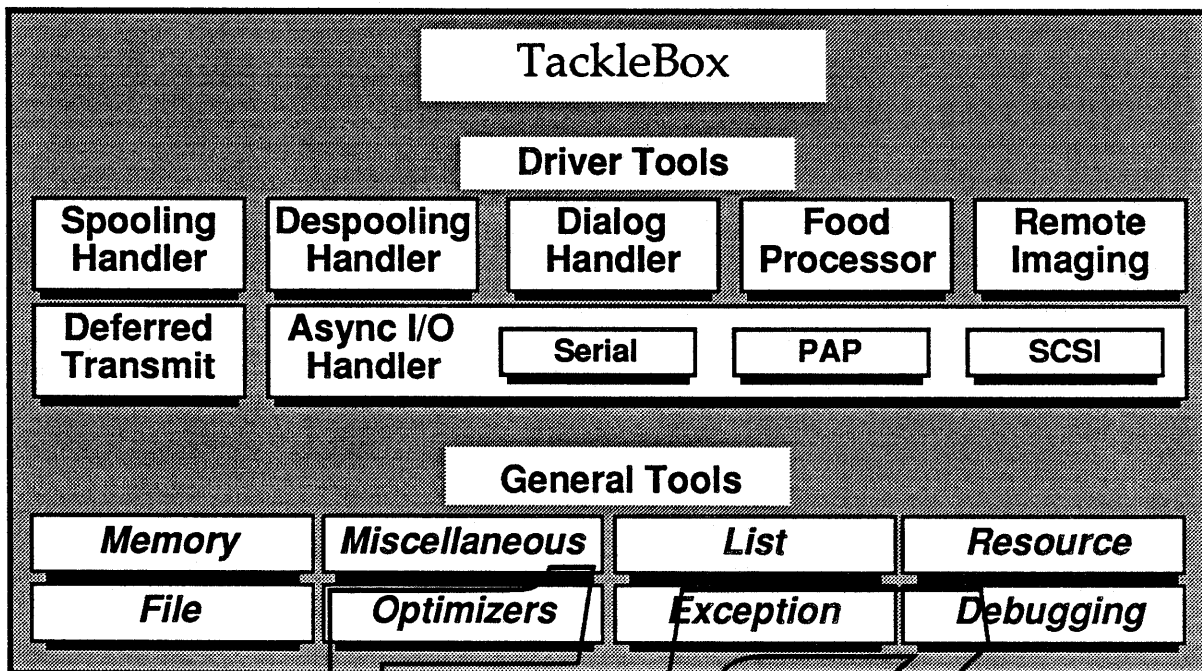
**Where:** The glue will continue to be a linked library which lives a double life. It will be able to support both the old calls and new calls; however, it will dispatch all calls to the PrGlue trap for processing. There is one sticky wicket, though: it is a fact that some developers (notably Microsoft) want to continue to support the 512KE. It would be unfair to force them to use a Ginsu glue which does not run on the 512KE, since it makes it impossible for them to print on one of these machines. I propose the following: users of the 512KE do *not* get access to Ginsu functionality; they will see old dialogs and get old functionality *with old drivers*. Applications link with a single, unified glue. Note that an application which wants to support owners of the 512KE, but get Ginsu functionality, will have to make old calls on 512KEs and new calls on the Plus and up. Since applications do not have to make Ginsu calls, they can operate on all machines identically if they wish. Realize that documents may print differently on 512KEs from other Macs, since they are being printed on radically different drivers. This is not our problem; we shouldn't make it impossible for developers to support older machines.



**Definition:** The Glue Interface, in conjunction with PrGlue, serves as the interface between the application and Printing. The Glue Interface provides a consistent interface to PrGlue, regardless of the Specific Driver used. PrGlue calls the Glue Interface in response to both old and new Printing Manager calls made by the application. The Glue Interface will refuse an application's attempt to call old Printing Manager calls once PrStart (the first new call) has been called; similarly, it will refuse use of a new call once PrOpen has been called. The Glue Interface supports all existing versions of PrGlue; to simplify the task of supporting old PrGlues, the Glue Interface uses the current PDEF (code resource) structure.

**Where:** The Glue Interface lives with the TackleBox in the "Printing" icon.

**What happens:** PrGlue calls the Glue Interface in response to a Printing Manager call made by the application (e.g., PrOpenDoc). The Glue Interface calls either the TackleBox or the Specific Driver; the TackleBox handles setup and cleanup calls. The Specific Driver handles the Spooling, Imaging, Dialog, Information, and Low-Level Interfaces. Note that for those calls in which the Glue Interface calls the TackleBox, the TackleBox will subsequently call the Specific Driver. For example, when PrStart is called, the Glue Interface will call the TackleBox to allocate storage for a job, and then call the Specific Driver to initialize for a job.



**Definition:** The TackleBox is a loosely allied group of "handlers" which provide most of the common functionality of the old drivers. For example, all of the old drivers spool; rather than have  $n$  copies of the spooling code, there is one *Spooling Handler* which performs that task for the new drivers. The TackleBox can be broken up into two pieces: general tools and driver tools.

**General Tools:** These are the handlers which could be called by any entity in Printing. Two tools are the Memory Handler and List Handler which provide services to everyone in printing. The File Handler is general only in that it can be called by anyone; however, calls from any handlers other than the Spooling, Despooling, and Remote Handlers are discouraged.

**Driver Tools:** These are the handlers which provide the driver with specific, high-level calls to perform a complex task. For example, driver tools include the Spooling and Despooling pair, as well as the Food Processor. Async I/O is also a driver tool, and provides the driver with the ability to send to and receive data from the device in any one of a number of protocols (e.g., SCSI, AppleTalk).

**Where:** The TackleBox lives in the "Printing" icon.

## TackleBox Pieces: A cast of tens.

### Driver Tools

#### Dialog Interface

##### Dialog Handler

It manages the interactions among the user, the application, and printing. Puts up the Page Setup and Print dialogs. It permits additions to the dialogs by the application, the driver, and Personality Modules. It is the keeper of the Print Record and accepts calls to access and change fields in the Print Record.

#### Spooling Interface

##### Spooling Handler

It writes and maintains a spool file. This file can be despoiled in the foreground or background. Spooling Handler also manages disk errors (e.g., disk full) and is Dire Straits-aware. It calls the Optimizers (see below) to keep a font database and to perform line layout and bitmap splitting.

#### Imaging Interface

##### Despooling Handler

It reads a spool file, either in the background or foreground. It is Dire Straits-aware. It calls the Optimizers to decode the font database and reconstruct bitmaps.

##### Asynch I/O Handler

It manages the interactions between the Macintosh and the device. Checks status, queues packets, and supports multiple protocols.

##### Imaging Engines

They take QuickDraw data from bottlenecks and render them into another format. Formats include raster, PostScript, and vector. They support Color QuickDraw, Classic QuickDraw, and Full Color QuickDraw.

##### Food Processor

It takes bitmaps and pixmaps and munges them. It can halftone, dither, and scale. It supports Color QuickDraw data structures. It also performs color separations. Note: This unit is mostly undefined, but it'll be cool. Trust me.

##### Remote Imaging

It manages the interaction between the user's Macintosh and a remote imaging station. This station is also a Mac. It communicates via the to-be-established remote protocol and transmits files across the network. It also checks status and

reports it.

### **Deferred Transmission**

Handles the decoding and transmission of an image file. This is a file which contains already imaged and packaged data. This permits imaging of a complex document without worry that the device will timeout. It also supports the Remote Spooler's ability to print to several devices simultaneously.

## **General Tools**

### **Memory Handler**

It allocates and keeps track of all memory used during a print job. It permits up-front allocation and reservation of memory.

### **List Handler**

It provides calls to create, maintain, access, and destroy linked lists.

### **Debugging Handler**

It provides calls to dump information to TestShell, to a file, or to MacsBug.

### **Exception Handler**

It manages error conditions.

### **Miscellaneous Handler**

It provides calls for things that don't fit anywhere else.

### **Resource Handler**

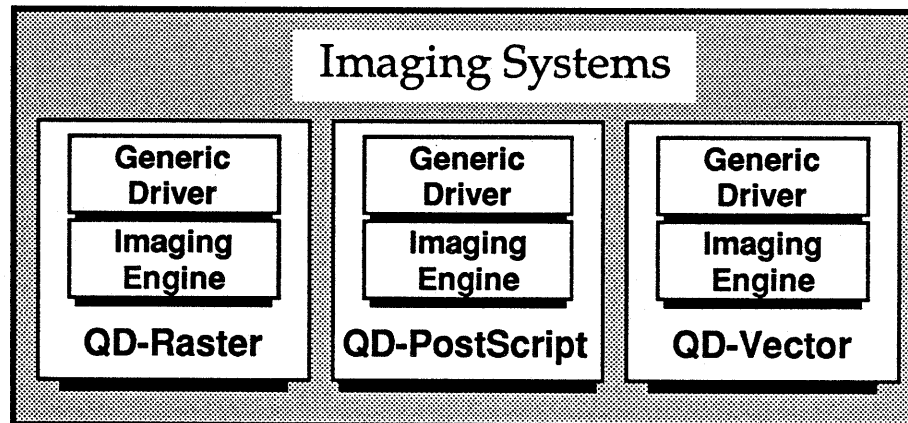
It provides calls to get data from resource files. It preserves the resource chain and ensures that printing resource files are not lost from the resource chain. It supports forced-loading of resources from disk.

### **File Handler**

It manages the disk. The File Handler supports all of the data written by the Spooling Handler and Optimizers with direct calls. It supports three file formats: spool file, print file, and image file. The first two contain information used to image a job. The third consists of already imaged and packaged information. Such a file is used to defer printing of a job, without deferring the imaging of the document.

### **Optimizers**

They determine logical lines to give to the Layout Manager and split bitmaps. They also maintain a font database for use by the driver. They are called by the Spooling and Despooling Handlers and by the driver.



**Definition:** An Imaging System controls spooling and despooling, and also translates between QuickDraw and another format (e.g., raster). It is completely device-independent — any device-specific information required is passed into the Imaging System in tables or procedures. Consequently, the Imaging System does not package data for a device nor does it put up custom dialog panels or items. There are three Imaging Systems currently defined: QuickDraw-to-Raster, QuickDraw-to-PostScript, and QuickDraw-to-Vector. There are two pieces to an Imaging System: a Generic Driver and an Imaging Engine. The *Generic Driver* controls spooling and despooling of an entire job; the *Imaging Engine* renders a single page of QuickDraw.

**Where:** The Imaging System lives with the TackleBox and Glue Interface in the "Printing" icon. Users will install new Imaging Systems via NuFinder. My belief is that Imaging Systems are files added to the Printing icon via the Installer.

**What happens:** The Glue Interface jumps to a procedure pointer in the Specific Driver, which either points to code in the Specific Driver or to code in the Imaging System's Generic Driver. The Generic Driver publicizes its entry points into its jump table so that the Specific Driver can call it; the Specific Driver also publicizes its entry points, so that the Imaging Engine can call the Specific Driver when data has been rendered.

**Note:** QuickDraw bottlenecks are not publicized in this manner. There will be proc pointers in the grafPort for the overridden routines, and the pointers will reference those jump table entries. If the Specific Driver wants to override the Imaging System's bottlenecks (at spooling, despooling or imaging), it does so by grabbing the grafProcs record from the grafPort.



## I. The Generic Driver

The Generic Driver for a given Imaging System is an object of sorts; this object has private data and methods for imaging a job. It can spool, despool, put up dialogs, and image. It provides interface calls for every call the Glue Interface or TackleBox can make. It also provides packaging and Food Processor procedures; however, for the most part, those latter services are stubbed out. It contains no interface to the Async I/O Handler; all transmission must be performed by the Specific Driver.

*Issue: Are remote imaging and deferred transmission things to put into the Generic Driver, with override procs for the specific driver?*

### Capabilities and Limitations:

The Generic Driver does useful work for Spooling, Imaging, Dialogs, and Low-level Interface calls; it will do a simple RTS from Information calls (e.g., a GetResolution call). The Generic Driver displays a set of general purpose panels and options by calling the Dialog Handler. It can also validate and default a print record. However, if the Specific Driver has information it needs to append to the print record, the Specific Driver will have to override the Generic Driver. The Specific Driver will also have to override if it wants to add any panels or options to the dialogs.

The Generic Driver supports all quality modes (Final, Rough, Text and Boxes, and Text Only) via the Imaging Engine.

## II. The Imaging Engine

The Imaging Engine for a given Imaging System is capable of rendering a page of QuickDraw into another format (e.g., into PostScript). It has a small set of entry points to perform the following tasks:

*Install bottlenecks.*  
*Remove bottlenecks.*  
*Abort page.*

These entry points are public — a Specific Driver can also access them to override the Generic Driver's imaging controller. It might do this if it wants to deny the imaging controller access to part of the page. The Imaging Engine, when it has data rendered (either a page or a band) will call the packaging code of the Specific Driver (DrDataIn). The Specific Driver's DrDataIn should package the data and transmit it to the device.

The Imaging Engine will call the Food Processor, and has full access to the TackleBox.

**Qualities:**

**As defined earlier in the document. What follows are notes on the two non-WYSIWYG modes, Text Only and Text and Boxes (from Sean). In general, the Imaging Systems support all four quality modes — help from the Generic Driver or Specific Driver is unnecessary.**

*International issue: Text and Boxes and Text Only are not appropriate for most non-roman languages because the Layout Manager is not used. Where they cannot be supported, the options will be removed.*

**Text Only:**

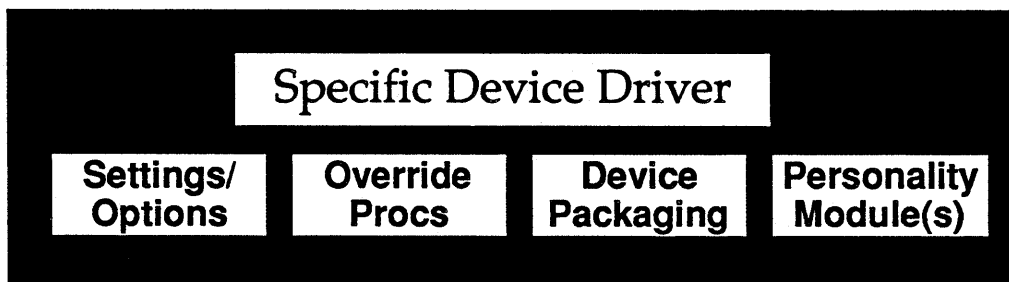
It will be built as a *fast* drafting tool and not a tool to make best use of the printer's native characteristics. Style changes are dropped. Text that is smaller than the printer's native font may be dropped. Text that is too tightly spaced or overlaps other text may be dropped. Dot matrix printers will Y sort the text for speed (it will not be required that the application do this). Support for logging (for terminal applications) is not provided. Character set mapping will be provided for available characters though some characters may not be available and will not appear.

To provide applications (e.g. database apps) such as those that print on forms a path through Text Only with guaranteed results, the following "yellow brick road" will be support on all printer where it is reasonable to do so.

Monaco 12 Point will map to a 10 Pitch (fixed) space font. In PostScript, this will map to Courier. Style changes should not be used. A text strike should always start at the left edge of imageable area with its vertical position some multiple of 12 Points (6 lines per inch) from the top edge of imageable area. Spacing for columns should be accomplished via a MoveTo, in which the distance is a multiple of 10 characters-per-inch (7.2 pixels). Fractional results will be truncated. The use of MoveTo rather than the use of spaces will permit good results with the Layout Manager for higher quality settings. We selected Monaco 12 Point because it is very close in size to a 10 Pitch font (10.29 Pitch). If an application wishes to improve the mapping for the other modes it may do so via space extra (space extra is ignored in Text Only mode).

The "yellow brick road" is not guaranteed to always be present and may not be available on future systems.

Support for Text Only will be rolled into each Specific Driver since its implementation is very driver-dependent. The imaging may or may not be performed by the Imaging Engine, depending on the device. For example, the ImageWriter II driver will do its own Text Only implementation, but the LaserWriter will simply use the PostScript Imaging System.



**Definition:** The Specific Driver is related to the concept of the current ImageWriter driver. It contains *all* of the device-specific information and code necessary to drive a particular device class. Specific Drivers in Ginsu drive a single device class, as defined earlier, and are meant to be quite small. Most of the hard work is done either in the TackleBox or the Imaging System. The expectation is that drivers for Apple printers are trivial to write and debug (given a bug-free TackleBox!).

**Where:** The Specific Driver is a separate icon from the rest of Printing. It should be very easy for the user to determine which printers he or she can print to by looking at the set of installed drivers.

**What happens:** Any call which the Specific Driver does not want to handle entirely on its own will require an Imaging System. Only the Specific Driver knows which Imaging System it needs, and consequently, the Specific Driver must load in the correct Imaging System.

**Errors:** If the Imaging System is not present, then the Specific Driver will fail gracefully with an alert to the user. Further, if the Imaging System is an earlier version than the Specific Driver, the Specific Driver will also alert the user.

**Dialogs:** Any Specific Driver may add panels or items to the Page Setup and Print dialogs. Examples of panels are ones to handle bins; examples of additional items are extra paper sizes (e.g., LaserWriter II US Letter). Items are added to the main printing panels — the Specific Driver can only add items to existing categories (like paper types); the driver cannot add new categories to these panels. If it has new categories (e.g., bins), the driver should add panels. There is a strict procedural interface to accomplish these tasks. *Issue: How do old print records fit into the rest of the structure? Who creates and validates them? This is an issue for the Amy and us.*

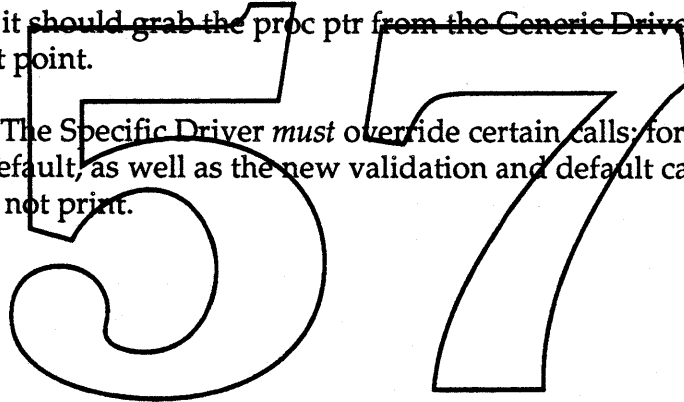
**Structure:** The Specific Driver for a given device class contains four sections:

- [1] Tables : resolution, and other settings
- [2] Override Procs : for all Dialog, Spooling, and Low-Level Interface calls
- [3] Device Packaging : for transmission. Should support the necessary protocols
- [4] PerMod support : Code to call the set of installed Personality Modules.

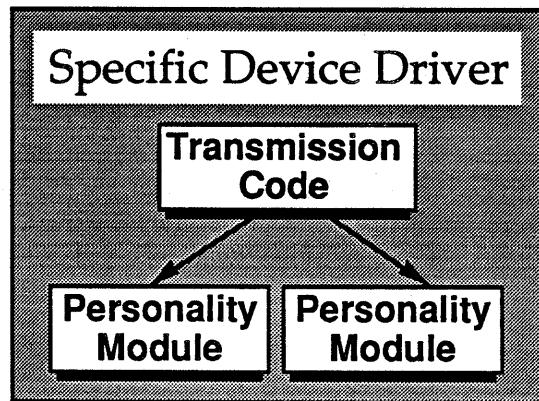
The fields in the tables are readable by the Glue Interface, by the Imaging System, and by the TackleBox; procedure pointers will be resolved at run-time and executed. These procedure pointers should be pointers to a jump table which can load in the necessary code.

Like an object-oriented system, the higher-level construct (the Specific Driver) has the option to override the call. If it doesn't, the next level construct (the Generic Driver) receives the dispatch. Thus, if the Specific Driver wants to handle a particular call, it should install a reference to an internal routine in one of its tables. Otherwise, it should grab the proc ptr from the Generic Driver and stick it into its table at that point.

**Special Note:** The Specific Driver *must* override certain calls; for example, PrValidate and PrintDefault, as well as the new validation and default calls. If it doesn't do so, the job will not print.



## Personality Modules



**Definition:** Personality Modules are plug-in code blobs which drive a hardware "extension" to a device. An extension is anything which affects the base definition of the device, and does not interfere with imaging or packaging of data.

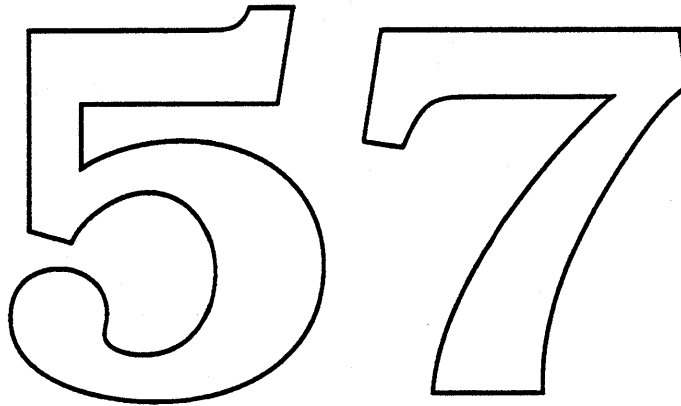
**Where:** It resides in the Specific Driver for the device class it supports, and is installed by the user by dragging. *The installation interface is undefined.*

**What happens:** The Specific Driver calls the Personality Modules during dialogs and during transmission. No part of printing, other than the Specific Driver knows of the existence of PerMods. The PerMod is allowed access to the TackleBox and communicates via Async I/O. Note that the PerMod ~~does not~~ have direct access to the Async I/O — it communicates to it via the Specific Driver, using an undefined call structure.

**Dialogs:** A Personality Module can add panels and items to the Print Dialog, just like a Specific Driver. It can add them to both the main panels and to the Specific Driver's panels if it likes. Since a PerMod can't affect formatting, it can't add to Page Setup.

**Priority:** A PerMod has a priority, which the Specific Driver uses to determine the order in which it calls the set of PerMods under its wing. Priorities will be assigned with large gaps in between: for example, the LCD display might receive a priority of 16000 and the BDT sheet feeder might receive one of 24000, permitting a wide range of devices to be supported later. Priorities are important because of the conceptual inter-relationships between the extensions. For example, pretend an LCD display is used to prompt a user to insert a paper type for manual feed, and there is a sheet feeder installed as well. Then, for manual feed to work, the LCD must receive control before a sheet feeder does, since sheet feeder will time-out on a manual feed and

pull a sheet from a default bin. Since the sheet feeder will not return control until it has successfully received a sheet, the LCD display would be rendered useless.



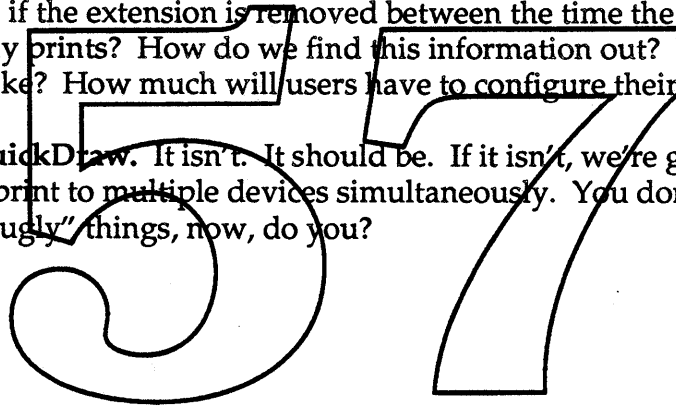
---

## Open Issues

---

- [1] **Installation of Imaging Systems.** How does this work in the context of NuFinder?
- [2] **Installation of Personality Modules:** Can we really just drag 'em in?
- [3] **Priorities of Personality Modules:** Who assigns them?
- [4] **Determining which Personality Modules to use:** This is not mentioned in the document, but it is a problem. Personality Modules can add panels to the Print Dialog, but that implies that we know which set of PerMods are currently connected to the device to know which PerMods should be called at PrJobDialog time. For example, I might have installed an LCD display PerMod and a BDT sheet feeder PerMod to my NTX driver. The device I'm printing to, however, may have any combination of those PerMods (none, LCD, BDT, or both). Only the PerMods connected to the actual device should be given control — not only at dialog time, but upon connection.

What happens if the extension is removed between the time the user hits "OK" and the job actually prints? How do we find this information out? What would the perfect world be like? How much will users have to configure their devices? I'll solve it later.
- [5] **Re-entrancy of QuickDraw.** It isn't. It should be. If it isn't, we're going to have to do ugly things to print to multiple devices simultaneously. You don't want the PrintShop doing "ugly" things, now, do you?



---

## Architecture Vocab: The terms you need to be in the know

---

### General Vocab:

#### Raster Printer:

A printer which takes bitmaps (or some form of them) as its primary input. This is compared to printers which take other input, like PostScript.

#### Binding Documents to Printers:

A document once printed to a device will always print to that device, unless the user explicitly changes the assignment. The concept of "system printer" is abandoned, and the new concept of "default printer" is wholeheartedly embraced.

#### Logical Lines:

Grouping together all of the strings of text which fall on a "line of text." For example, a line of text with bold Times and plain Palatino appears as two distinct strings in QuickDraw, but they would be grouped together during imaging. This results in *substantially* improved output, because layout error can be distributed across more characters, and because we can ensure that style runs do not collide.

### Spooling Vocab:

#### Spool File:

A temporary file which holds everything the application wishes to print, including QuickDraw data, page format, optimizers we use during imaging, and state information needed to recreate the world in which the application wrote the data.

#### Image File:

A temporary file which holds everything the driver would send to a device. Normally, the driver packages imaged data as it arrives from the Imaging Engine and ships it off to the device. However, it has the choice of deferring transmission of the data, and writing an image file instead. This image file is page-based. Since two-way communication must occur during printing (we don't just fire data out the port), the error recovery and status is not encoded in the image file. Thus, when the driver actually transmits data, it should do its usual status and error recovery actions at the start, and at the end, of each image file page. **Note:** Personality Module interactions are not stored in the image file, since they occur at page boundaries as well.



**Link File:**

A very small file which contains information needed to find the spool file and information PrintMonitor needs to initiate imaging. For example, the link file contains the driver name and the icon to use in the dialog. The link file is guaranteed to be in the system folder of the startup disk; the spool file may be written to any mounted volume. We can use the link file to find a spool file across restarts and across application spaces.

**Spooling for the background:**

Writing a spool file which will be imaged in the background by PrintMonitor

**Spooling for the foreground:**

Writing a spool file which will be imaged in the foreground.

**Partial Despooling:**

When the Spooling Handler runs out of disk space and tries to free up space by despooling the completed pages. It does this by calling PrGPicFile. This is a purely foreground operation.

**Dire Straits:**

When there is not enough room to spool a given page. Only entered when all other attempts to free up space for this single page have failed (e.g., despooling earlier pages, waiting for PrintMonitor to finish printing a job).

**Retransmission:**

Requesting the application send the page data again. Used in Dire Straits.

**Dumb spooler:**

LaserShare. A spooler which pretends to be a printer.

**Smart spooler:**

A spooler which retains its identity as a Macintosh

# The Ginsu Project

## Imaging Engine

Ginsu Project Proposal, Version 1.0d8  
Thursday, January 19, 1989

Sean Parent with Nik Bhatt, Jay Patel, & Naresh Gupta  
MS: 27-AJ; Extension 4-2270  
System Software Group, The PrintShop  
Apple Computer, Inc.

*Summary:* Imaging Engines take a stream of QuickDraw and transform it into another standard format. In Ginsu there will be at least three imaging engines; QuickDraw to Raster, PostScript, and Vector. The Imaging Engines are a peer of the Generic Driver and part of the Imaging System.



**Captain Ginsu Says: Eat This Page!**

**This Page left 98% blank**

57

# Table of Contents

<b>1. Introduction:</b> .....	<b>1</b>
<b>1.1. Terminology:</b> .....	<b>2</b>
<b>1.2. Overview:</b> .....	<b>3</b>
<b>2. Goals &amp; Features:</b> .....	<b>5</b>
<b>2.1. General:</b> .....	<b>5</b>
<b>2.2. Standard Objects (The Bottleneck Crew):</b> .....	<b>5</b>
2.2.1. Text:.....	5
2.2.2. Graphic Primitives (Lines, Rectangles, Rounded Rectangles, Ovals, Arcs, and Polygons):.....	5
2.2.3. Regions:.....	6
2.2.4. Maps:.....	6
2.2.5. Comments:.....	6
<b>2.3. Patterns:</b> .....	<b>6</b>
<b>2.4. Comments (A Big Scary Section):</b> .....	<b>7</b>
2.4.1. Problem.....	7
2.4.2. Solution:.....	8
<b>2.5. Maintain Picture Resolution (or “Sean’s Favorite Problem”):</b> .....	<b>11</b>
<b>3. High-Level Interface:</b> .....	<b>13</b>
3.1. Driver to Imaging Engine:.....	13
3.2. Imaging Engine to Food Processor:.....	14
3.3. Imaging Engine to Packager:.....	14
<b>4. QuickDraw to Raster Imaging Engine (QDRIE):</b> .....	<b>15</b>
4.1. Introduction:.....	15
4.2. Additional Features, Goals, and Limitations:.....	15
4.3. Internal Operations:.....	15
<b>5. QuickDraw to PostScript Imaging Engine (QD2PS):</b> .....	<b>18</b>
5.1. Introduction:.....	18
5.2. Additional Features, Goals, and Limitations:.....	18
5.3. Internal Operations:.....	20
<b>6. QuickDraw to Vector Imaging Engine:</b> .....	<b>21</b>

## 1. Introduction:

---

... The Ginsu Saga Continues...

When we last saw the PrintShop that great propeller head samurai, Captain Ginsu, had just sliced the drivers of today into goey chunks and left Dr. Will Stein and his hoard of marauding PrintShoppers to rebuild them...

The new drivers were almost complete except for all this stuff having to do with paper, like imagable area and size. In a crazed and hurried fit, Dr. Stein gathered the remaining mess, rolled it into a ball, labeled it "Paper Types" and stuck it onto the beast. "Until someone has a better place for it," he cackled, "it remains." But even as he spoke the monster changed. The appendage labeled paper types shot spiked tentacles into the beast that tore through the mutated being and transformed it into a nearly unrecognizable mess.

Once again the cry went out...

HELP, CAPTAIN GINSU!!!!

But our hero is being held captive by a band of wandering pink men chanting "seeplusus seeplusus seeplusus."

"Don't worry," yelled the young Nik Sky Writer, "I can handle this," as he whipped out his Junior-Captain-Ginsu-Fan-Club-Official-Swiss-Army-Ginsu-Blade (JCCFCOSAGB for short). "This is going to be COOL!" he cried.

But even as the "L" was rolling off his tongue a voice was heard above his, "Did somebody say 'COOL'?" It was the voice of Dr. Scott Jenson, cosmetic surgeon for the star engineers. "I thought I heard somebody say 'cool'," he stammered. "On my god, It's uglier then I suspected. I must operate at once. Hand me that blade young man!"

And so it began, the reformation of the printing drivers. As Dr. Jenson worked the rest of the PrintShoppers joined in. Together they struggled to build this dream. Dr. Stein was heard calling out orders, "Nik, you build the skeleton. This beast must stand tall. Sean, Jay, and Naresh, you must provide the organs for this beast to eat QuickDraw and spew PostScript and Bits. Amy, the beast must speak with other beasts. Give it a ears and a voice. Andy, it must also be cosmetically appealing to men. Aid Dr. Jenson in this task (the Dr. needs help with his anatomy). And Tom, the beast must be fed at long distances (we wouldn't want to get too close to the thing). Find a way to do this and make sure that you account for multiple feeders at the same time. I hate food on the floor."

And when the beast was complete they took it to Smurf Village (a tranquil place where blue men and women are often seen dancing about). What happens there is another story...

This is the story of that organ known as the Imaging Engine.

The Imaging Engine resides with the Generic Driver as the core of the Imaging System. The function of an Imaging Engine is to convert QuickDraw into one standard format such as Raster or PostScript. To accomplish this the Imaging Engine installs a set of bottlenecks into the printing grafPort that intercepts the QuickDraw information and translates it into the format of that engine's data stream. An Imaging Engine is not device specific but is specific to a set of device classes. (i.e. The Quick Draw to Raster Imaging Engine will be used for a set of printers including the ImageWriters and the LaserWriter SC). This document specifies the framework that each Imaging Engines will fit into but does not go into specifics for the individual Imaging Engines.

### 1.1. Terminology:

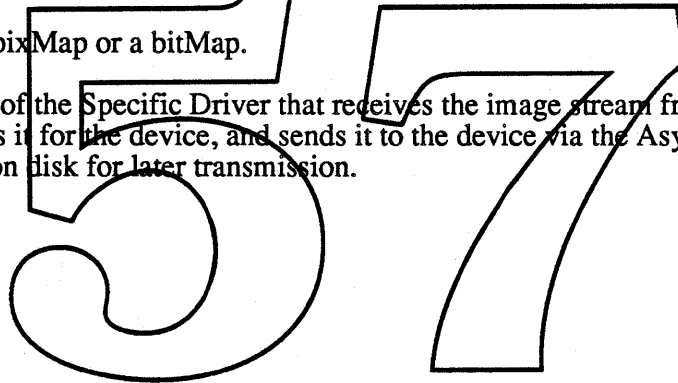
**Food Processor** – Section of code that breaks raster images into layers of separate colors and handles such items as dithering, smoothing and halftoning.

**Raster Image** – A collection of bits in *the* standard raster format of a pixMap.

**QuickDraw** – The set of graphic models including QuickDraw, Color QuickDraw, and, presumably, Full Color QuickDraw (32-bit QuickDraw). Not Skia and Albert.

**Map** – Either a pixMap or a bitMap.

**Packager** – Part of the Specific Driver that receives the image stream from the Imaging Engine, prepares it for the device, and sends it to the device via the Asynchronous IO package or stores it on disk for later transmission.



1.2. Overview:

*Note:* In the following section Driver refers to either the Generic Driver or the Specific Driver if it over-rides the default Generic Driver routines.

Figure 1.2 shows the flow through the QuickDraw bottlenecks from the application to the image stream which is destined for the Packager.

The flow starts with the application drawing and optionally intercepting its own calls in the bottlenecks. There are no real compelling reasons why the application needs to be able to intercept the bottlenecks but the application design may be simplified if it has the opportunity to do so.

The Driver then has the opportunity to filter the information on its way to the Spooling Handler. This could be done to filter out PostScript for non-PostScript devices or filter out QuickDraw for PostScript devices or any other filtering the driver wishes to perform. This would be done purely for performance and would make the print file very much less device independent and is discouraged.

The Spooling Handler then takes the information and pipes it to the print file where the Despooling Handler can pick it up. No one has the opportunity to stand between the Spooling Handler and the Despooling Handler. These two units remain fairly transparent, except for optimization, so there should not be any reason to get in between them. However, since they both reside in the Tacklebox any section can call them for the statistical information that they can gather and optimizations that they perform such as splitting maps. The Imaging Engines will make use of these facilities.

Next, the Driver has an opportunity to step in to handle stuff like substituting patterns and intercepting particular objects that the printer supports directly such as lines or rectangles.

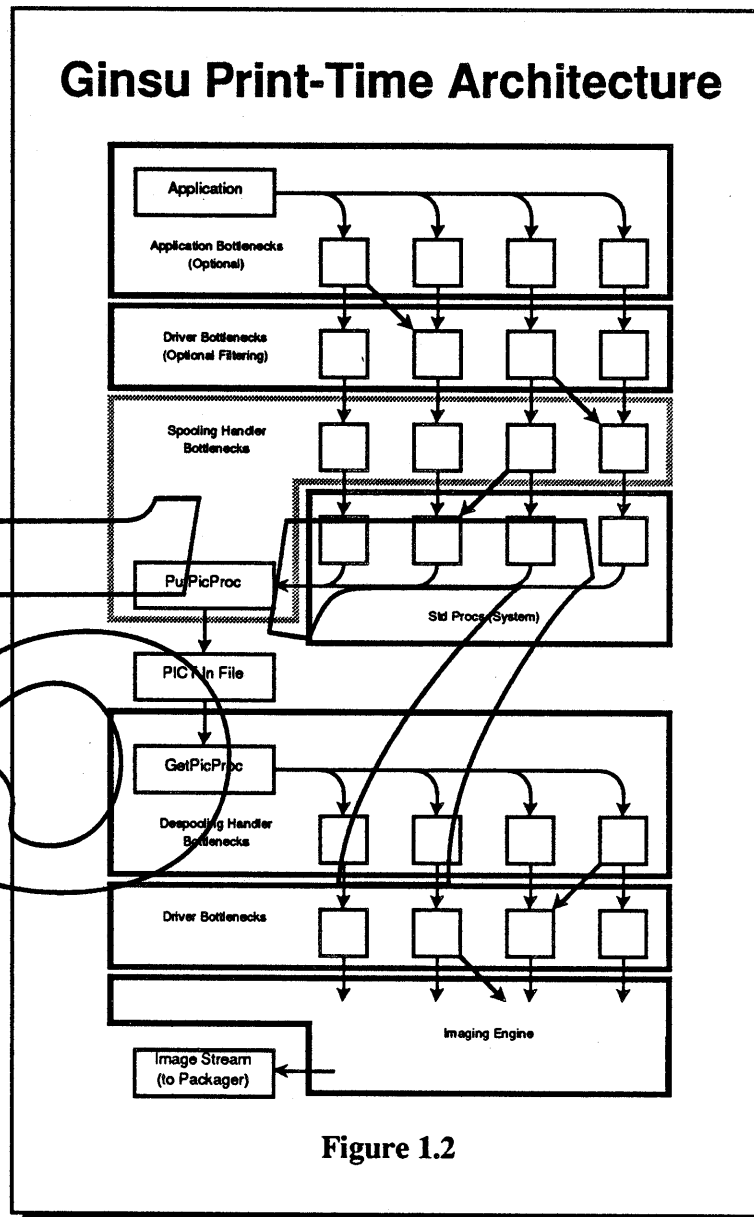
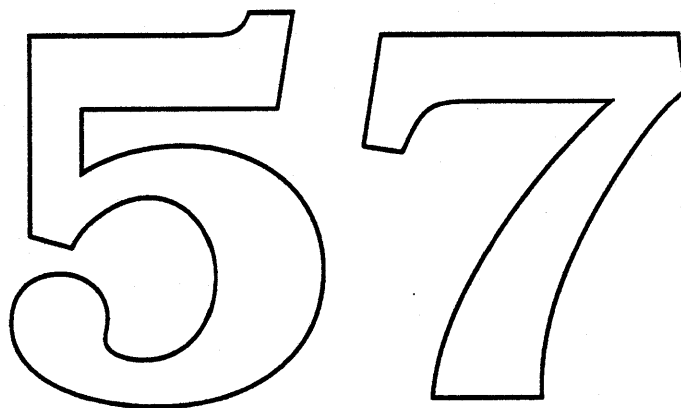


Figure 1.2

Now the QuickDraw information is piped off to the Imaging Engine. The Imaging Engine will take the information and generate a stream of data in one of the generic formats. The stream could be PostScript or a Raster image.

From here the Driver's Packager is called to prepare the information for the device. The Specific Driver has plenty of places to intercept the data and do some optimization.





## 2. Goals & Features:

---

### 2.1. General:

The first goal of the Imaging Engine is to provide the same functionality as the existing drivers. Beyond that, we hope to unify and extend the capabilities of the imaging system as described below.

Traditionally the imaging code has been the bottleneck which keeps printing slow. The new imaging engines will trade off memory and speed dynamically to achieve much faster printing and still be functional in a limited memory environment. (See the individual Imaging Engine ERS's for details.)

Each engine will be able to scale the data on a page from the source resolution (varies by application and document) to the target resolutions of that engine's supported data stream. The engines will support the use of the Layout Manager to scale text.

The engines will support a forms page that behaves in a similar fashion to the FormPrinting PICT comments of today. A form page will stay in effect until either a new form page is provided (could be an empty / null page) or the end of the document is reached. Although a forms page does not provide a great speed advantage on most Raster devices, the use of it will substantially cut down on spool file size.

Special attention will be given to providing the maximum reasonable support for QuickDraw and Color QuickDraw. (See the individual Imaging Engine ERS's for details.)

Support will be provided for each of the four quality modes described in the Architecture document.

Each engine will have a clean / modular design to aid in maintenance and transporting them to other QuickDraw supported devices (i.e. the Apple IIGS). The engines will and must be fully driven via the QuickDraw Bottlenecks.

### 2.2. Standard Objects (The Bottleneck Crew):

#### 2.2.1. Text:

Text will be scaled in resolution and the Layout Manager will be used to lay out the text in those printing qualities that support it (see the discussion of modes in the Architecture section). Underlines will be handled in the by propagating the underscore character. The underline information in Bass fonts will also be used when available.

#### 2.2.2. Graphic Primitives (Lines, Rectangles, Rounded Rectangles, Ovals, Arcs, and Polygons):

These items will be scaled in resolution (pen size is scaled up as well as control points). Operations involving patterns will be handled as described below in the pattern section.

### 2.2.3. Regions:

These are very hard to support because they are tied to resolution and they are very difficult to implement in PostScript. Full support will be provided for applications drawing at device resolution to a QuickDraw device and more limited support will be provided under other circumstances. See the sections on the individual Imaging Engines for more detail.

### 2.2.4. Maps:

Operations on maps will be the function of the Food Processor. Support will include dithering and perhaps smoothing. There may be comments added to invoke these as well. GO DANIEL!!!

### 2.2.5. Comments:

See section below.

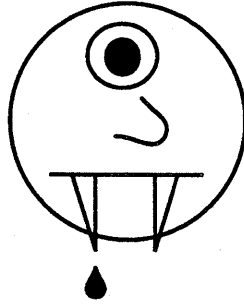
## 2.3. Patterns:

Patterns will be stretched (scaled in resolution) according to the following rules.

- Patterns are stretched by the greatest integer multiple that is less than or equal to the scale factor for the device image. (i.e. A device image with a 2.5x scale factor will have patterns stretched by 2x.) Fractional scale factors will not be supported to avoid scaling artifacts (a very undesirable attribute for patterns).
- Old patterns are assumed to be 72 dpi and are stretched accordingly.
- For old application that do not set resolution, the resolution of Color QuickDraw patterns are assumed to be 72 dpi.
- For Ginsu applications and old application that do set resolution, the resolution of Color QuickDraw patterns is taken from the resolution of the patterns pixmap.

These rules will allow an application to include patterns at various resolutions even if they are drawing at 72 dpi by doing a set resolution to 72 dpi. High resolution patterns are only available from Color QuickDraw. The limitations of Color QuickDraw to handle only patterns of 72 dpi, and a bounding rectangles with dimensions that are a power of two will not apply. This should not break any existing applications since QuickDraw does not support pattern resolutions other than 72 dpi today.

Pattern substitution will be done by default on the standard system patterns by the Specific Driver. The application will not be able to directly supply a high resolution pattern to substitute except through the standard Color QuickDraw pattern mechanism as described above. However, support will be provided for grays via the comments.



## 2.4. Comments (A Big Scary Section):

### 2.4.1. Problem

One of the goals of Ginsu is to unify the print drivers without limiting functionality so applications will not have to write device specific / driver specific code.

An area where the drivers diverge today is in their support for PICT comments. Currently, the PostScript drivers support a variety of comments while other drivers only support a subset of these. The functionality provided in the PostScript drivers includes:

- Rotation of Text and Objects
- Splines (Smoothed Polygons)
- Gray Levels
- Dashing
- Fractional Line Widths

These items, with cubic splines, provide the bulk of the features that most of today's high end applications are providing. The problems with the way these items are supported today include:

- Not Atomic – The PICT comments convey state information and they are not tied to the objects upon which they are meant to operate. For example, `SetLineWidth` works by instructing the reader of the picture to set the line width to the fractional value  $x$  for all subsequent calls. The `Rotate` comments instruct the reader of the picture to rotate all QuickDraw objects by an angle  $y$  until the `RotateEnd` comment appears. Because of this, comments tend to get separated from the object or disposed of all together by graphic editing applications that don't understand them. The result of this is a picture that may bear little resemblance to what was intended, or one which is not syntactically correct (e.g., two `PolyBegins`, with no `PolyEnds`).
- Not Renderable – QuickDraw currently ignores comments for the purpose of rendering. As a result, it does not save changes to the GrafPort when a `PicComment` appears. Since the GrafPort is in an undefined state, the comment cannot be rendered with any certainty. Despite this, since some comments include rendering information the application is forced to draw off-page objects to validate the GrafPort before sending the comment. These off-page objects are an ugly hack. Objects are not drawn on the screen the way they are printed resulting in the 11th PrintShop slogan:

**NYDSINYD: Now you don't see it, now you do.**

- Conditional – Most of the comments today are not interpreted by all drivers. Because of this you get different output on various devices. i.e. if you use rotated text comments then you will get rotated text on a LaserWriter NT but on a LaserWriter SC you will still get the text but it won't be rotated. There is no supplied code to render the comment data to the screen, so applications are discouraged from decoding the information and rendering it. This is not surprising, especially since the comments go outside the bounds of QuickDraw. As a result, what you see on the screen is not what you get. This has forced developers to write code which sends the comment only to PostScript devices, and sends a bit map, or some approximation of the object, to QuickDraw devices.

- Bulky – Sending maps to QuickDraw devices that do not support these comments creates atrocious overhead for in disk space (spool file) and in memory space. Well-behaved applications which send both QuickDraw and PostScript for the same object incur an enormous performance hit because they have to render the object twice. The old model actually encouraged device-specific printing.

- *Cut and Paste is dead. Well, at least dying. Application interchange of data has been significantly compromised by state-variable PicComments. If we could make the comments atomic, all applications could import and export more complex pictures. Cut and Paste would be saved. This is known as CPR: Cut and Paste Resuscitation.*

#### 2.4.2. Solution:

Axiom 1 – Since the data is to be rendered, we have to use a bottleneck procedure.

Axiom 2 – The solution must work for all machines, running old QuickDraw or Color QuickDraw.

Theorem – We cannot use the extended CGrafPort structure or the extra bottleneck procedures in Color QuickDraw. We need to use an existing bottleneck in the grafPort.

Axiom 3 – All applications should be able to display the data, whether they know about the comments or not.

Theorem – This functionality should be built into QuickDraw at some level.

Axiom 4 – State information is replaced by atomic comments.

Theorem – StdComment will have to render data encapsulated in the comments.

Axiom 5 – QuickDraw should flush changes to the grafPort on a PicComment.

Justification – This is important for consistency of imaging, and removes the dummy-object hack described above. Since the next QuickDraw objects will probably not need any additional grafPort changes, there is no data overhead associated.

So, we flush changes to the grafPort on a PicComment and treat selected comments as full class QuickDraw citizens. i.e. Selected PicComments are rendered when StdComment is called, just as any other QuickDraw object would be.

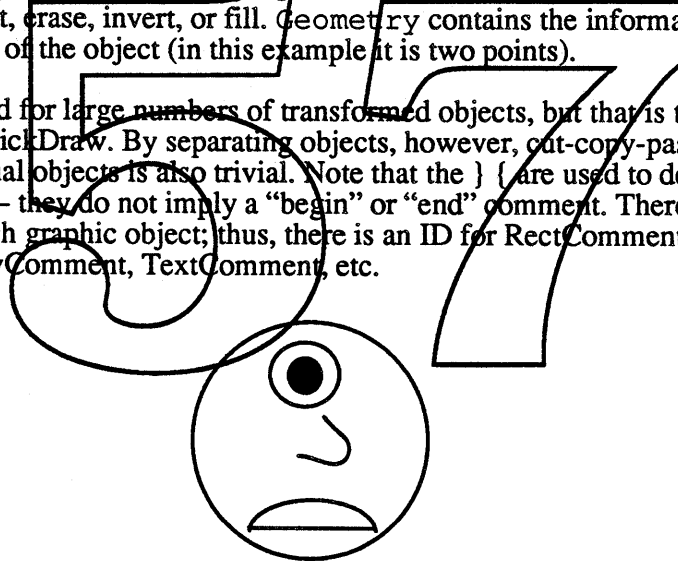
Each comment will contain all state information for rendering a single object.

Example:

```
RectComment
{
  Bounding Rect
  Style Bits
  Style Data
  Operation
  Geometry
}
```

Bounding Rect is the bounding rectangle of the transformed object that can be used for hit testing. Style Bits is used to reduce data overhead by denoting which fields are present in Style Data. Thus, if an application wants to use hairlines but not rotate data, it doesn't have to incur the storage penalty for the rotation information in each object. Style Data will contain attributes of the object such as rotation, center of rotation, line width, gray level, and dashing. Operation is a QuickDraw verb to be performed: frame, paint, erase, invert, or fill. Geometry contains the information to describe the geometry of the object (in this example it is two points).

There is an overhead for large numbers of transformed objects, but that is the penalty for going outside of QuickDraw. By separating objects, however, cut-copy-paste is trivial, and editing individual objects is also trivial. Note that the } { are used to denote scope of a single comment — they do not imply a “begin” or “end” comment. There would be a comment ID for each graphic object; thus, there is an ID for RectComment, LineComment, PolyComment, TextComment, etc.



**Embedded PostScript:** There is also the problem of the comment pair PostScriptBegin / PostScriptEnd. Applications send a mix of PostScript and QuickDraw which is interpreted differently by different drivers: PostScript drivers ignore the QuickDraw, and the QuickDraw drivers skip the PostScript. Since comments are atomic in the new model, we adopt the following new comment:

```
ModelMix
{
  Bounding Rect
  OffsetToPS
  QD Picture
  PostScript data
}
```

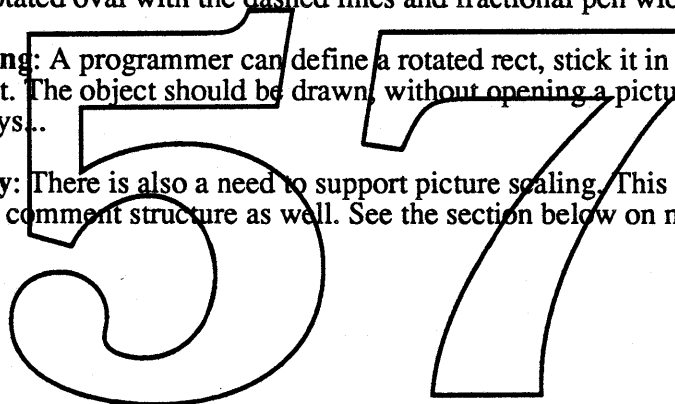
We encapsulate the two models in a single comment. (We may call this comment *IAmAnImbecileComment*). The application provides a QuickDraw picture *inside* the comment, along with the PostScript needed to generate the same image on PostScript devices. The other comments can get away with describing the geometry of a single object; however, since this comment encapsulates a series of objects, simple geometry is not possible.

**Changes to the System:** StdComment will have to be changed to recognize certain comment id's. If these id's are encountered, StdComment should render the encapsulated data with the correct transformations. Pictures should be rendered at 72 dpi. StdComment must not use the bottlenecks again to render. A RectComment should be rendered by a direct call to \_StdRect, not through the bottlenecks, since a recording application like MacDraw will duplicate the data.

**For Printing:** You knew it couldn't be that easy. We image by bottlenecking all the objects, scaling them up, scaling up the pens, patterns, etc., and then calling QuickDraw to render. In this case, we would like to be able to use QuickDraw's rendering code for these comments once we have scaled them up. For example, we take a rotated oval from a OvalComment, scale up the coordinates and pen location, and then call StdComment to render the rotated oval with the dashed lines and fractional pen width.

**One last thing:** A programmer can define a rotated rect, stick it in a comment, and call PicComment. The object should be drawn without opening a picture. That doesn't happen nowadays..

**By The Way:** There is also a need to support picture scaling. This will most likely be done via the comment structure as well. See the section below on maintaining picture resolution.



### 2.5. Maintain Picture Resolution (or "Sean's Favorite Problem"):

When an application draws a picture reduced (using DrawPicture where dstRect < original rectangle) into the print grafPort the items in that picture are scaled and resolution is lost (see figure 2.5).

Example of why this happens:

A picture is created with a two point line. The application scales it down to 25% and the line becomes a one point line (integer values). The print driver scales the page up to 400% to achieve the resolution of the printer and the line becomes a four point line when it should be a two point line again.

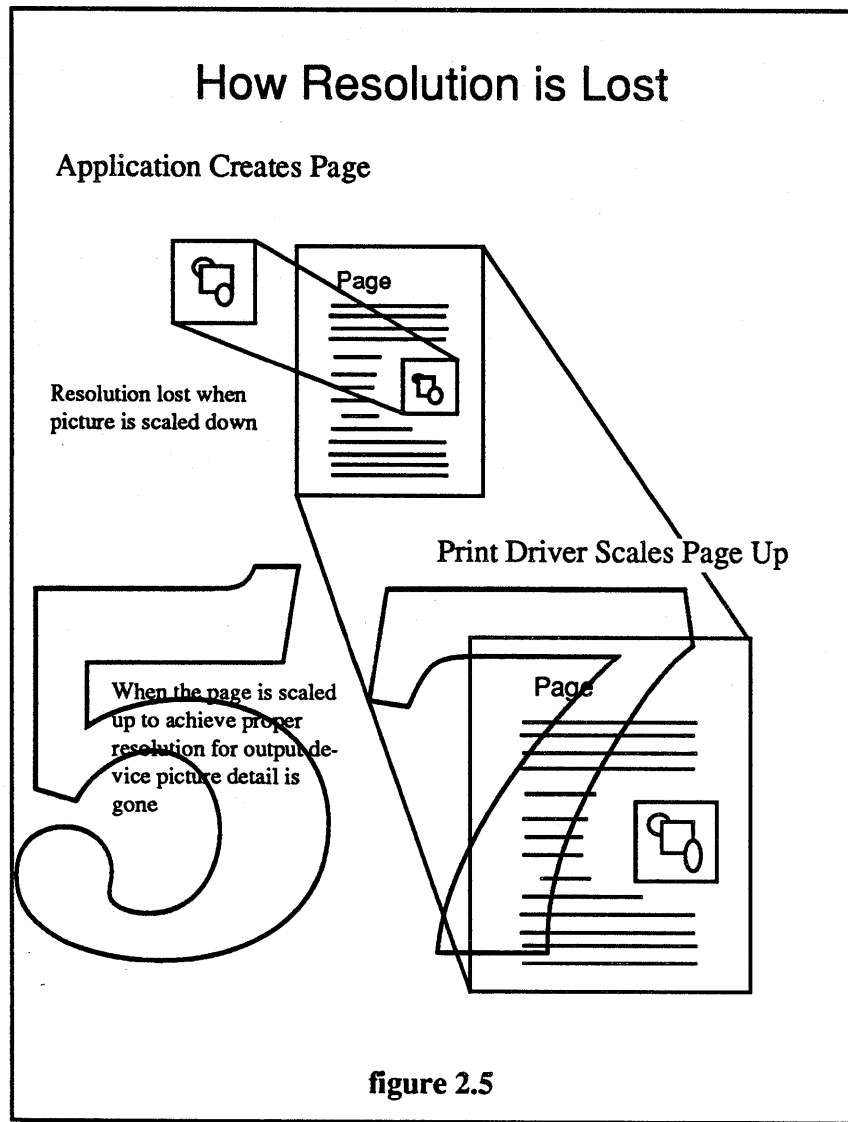


figure 2.5

Text and maps are not affected though their placement is.

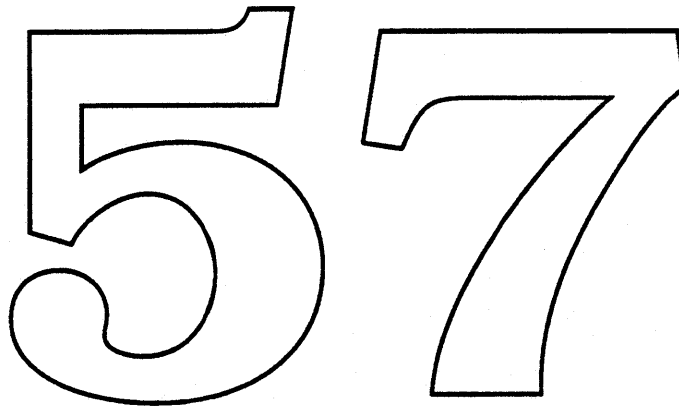
Note that this is a problem in those applications that do not set the resolution to match that of the printer (i.e MacWrite, FullWrite, Word, RagTime, and most other applications). This is not a problem in those applications that set the resolution to match that of the printer (i.e. MacDraw II & PageMaker).

The scaling of the image is being done by DrawPicture which is being called by the application to draw into the print grafPort. The spooling handler will patch DrawPicture to do the following:

- Mark start of picture in the PICT.
- Set dstRect to be equal to the picture's picFrame.
- Draw the picture. (using the normal DrawPicture routine).
- Mark the end of the picture in the PICT.

The Imaging Engine will intercept these marks and adjust the resolution scaling factor for the items in between. The Imaging Engine will be able to handle these marks even if they are nested so that they can also be used by applications.

It is still being debated as to whether the marking mechanism should be made public or not. It will most likely be implemented via some form of PICT comment. If the mechanism was made public then applications could transport a PICT with a resolution attached or transport and generate a PICT inside of a PICT.





### 3. High-Level Interface:

---

#### 3.1. Driver to Imaging Engine:

`IEInit` – Allocate non dynamic data structures and set state to start a new page.

`IEOpenPage` – Set up for one pass of the page. `IEOpenPage` will accept a handle to a `gDevice` record. The `gDevice` record will contain the pertinent information needed to create the target image. This includes the following:

- `gdType` – Device type of CLUT, fixed, or direct.
- `gdResPref` – Preferred resolution for inverse table. (\* I'm still not sure if this is needed \*)
- `gdCompProc` & `gdSearchProc` – Pointers to color search and compliment procedures. Nil for default.
- `gdRect` – Bounding rectangle for the page.
- `gdMap^^.bounds` – (Raster Only) Preferred banding size. Band size will be a multiple of this rectangle.
- `gdMap^^.hRes`, `gdMap^^.vRes` – Horizontal and vertical resolution of the device.
- `gdMap^^.pixelType` – (Raster Only) Storage format for pixel image; either chunky or planar.
- `gdMap^^.pixelSize` – (Raster Only) Preferred bits per pixel (maximum).

`IEOpenPage` will also require a structure that includes some additional information about the page. This information includes the source rectangle for the page so that the scale factor can be calculated and a version number so the record can be extended for a future system.

The `GDevice` record is used as a convenient medium to describe the attributes of the output device. This same convention will be used for both `QuickDraw` and `Color QuickDraw`. The driver will have to allocate the structures with `NewHandle` instead of `NewPixMap`, `NewGDevice`, etc. for consistency. This is really not a problem since most of the initialization done by the `Color QuickDraw` routines is based on the current device and will be meaningless.

`IEAbortPage` – Abort from the processing of this page. `IEAbortPage` should be called in place of `IEClosePage` if a fatal error occurs during the picture playing sequence. `IEAbortPage` will accept a parameter that instructs it to either abort the pass or abort the page all together (a soft abort and a hard abort).

`IEClosePage` – Prepare for either another pass of the same page or, at the end of a page, prepare for a new page. `IEClosePage` will instruct the caller to either retransmit the page or continue (with the next page or end of document).

`IEShutDown` – Dispose of non dynamic data structures and go away.

### 3.2. Imaging Engine to Food Processor:

NOTE – The Food Processor need not only be called by the Imaging Engine. It could also be called by the Packager of a Raster Driver to process the page. If the Food Processor were made public it could be called by an Application but this is probably not desirable since it would be better to have the application hint a map so that the proper algorithm can be selected for the printing device. The Food Processor needs to be able to handle multiple maps since it could be operating on the entire page, called by the Packager, and a pixMap, called by the Imaging Engine, at the same time. (This means that the Food Processor can not keep global state information. All of its state information must be passed around in its map structure.)

FPNewMap – FPNewMap returns an initialized data structure for a new map according to some set of parameters. The map will be created for a specific purpose such as smoothing / scaling or dithering. The Food Processor map structure will get passed among Food Processor routines.

FPDataIn – FPDataIn is called to add data to a specific map. Data is passed in bands where a band could encompass the entire image (one band). FPDataIn returns the area of the map that is complete and ready for use. The completed section of the map may be placed in a picture.

FPDataOut – FPDataOut is called to notify the Food Processor that some portion of the completed section of the map has been used and may be disposed of if the Food Processor no longer needs it.

FPEndMap – FPEndMap is used to tell the Food Processor that the map is complete. It will dispose of the state information and return the remainder of the pixMap that can be copyBits-ed or passed where needed.

### 3.3. Imaging Engine to Packager:

DrDataIn – DrDataIn is called to send the image stream to the packager.

NOTE – The Packager also communicates with the Specific Driver that it is a part of for things like start page / document, and end page / document.

#### Imaging Engine to Despooling Handler:

NOTE – These calls are defined in the documentation for the Despooling Handler. They include calls to get selected maps and to gather statistics. All Imaging Engines will be written so that they are not dependent upon this information for printing but only for speed / memory performance optimization.

## 4. QuickDraw to Raster Imaging Engine (QDRIE):

### 4.1. Introduction:

The QuickDraw to Raster Imaging Engine is the Imaging Engine that will be used to support devices including the LaserWriter SC and the ImageWriters. There are also tentative plans to build support for the QDRIE into the other LaserWriter drivers as an alternate engine to use when complex regions or transfer modes are used.

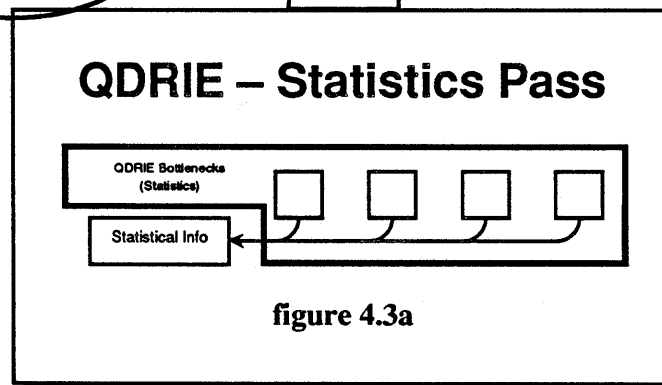
The QDRIE is basically a collection of routines that sit in the QuickDraw bottlenecks. These routines handle splitting a picture into zones and bands and scaling the resolution of the picture (pattern stretching and invocation of the Layout Manager) then calling QuickDraw to handle the rendering into a map.

### 4.2. Additional Features, Goals, and Limitations:

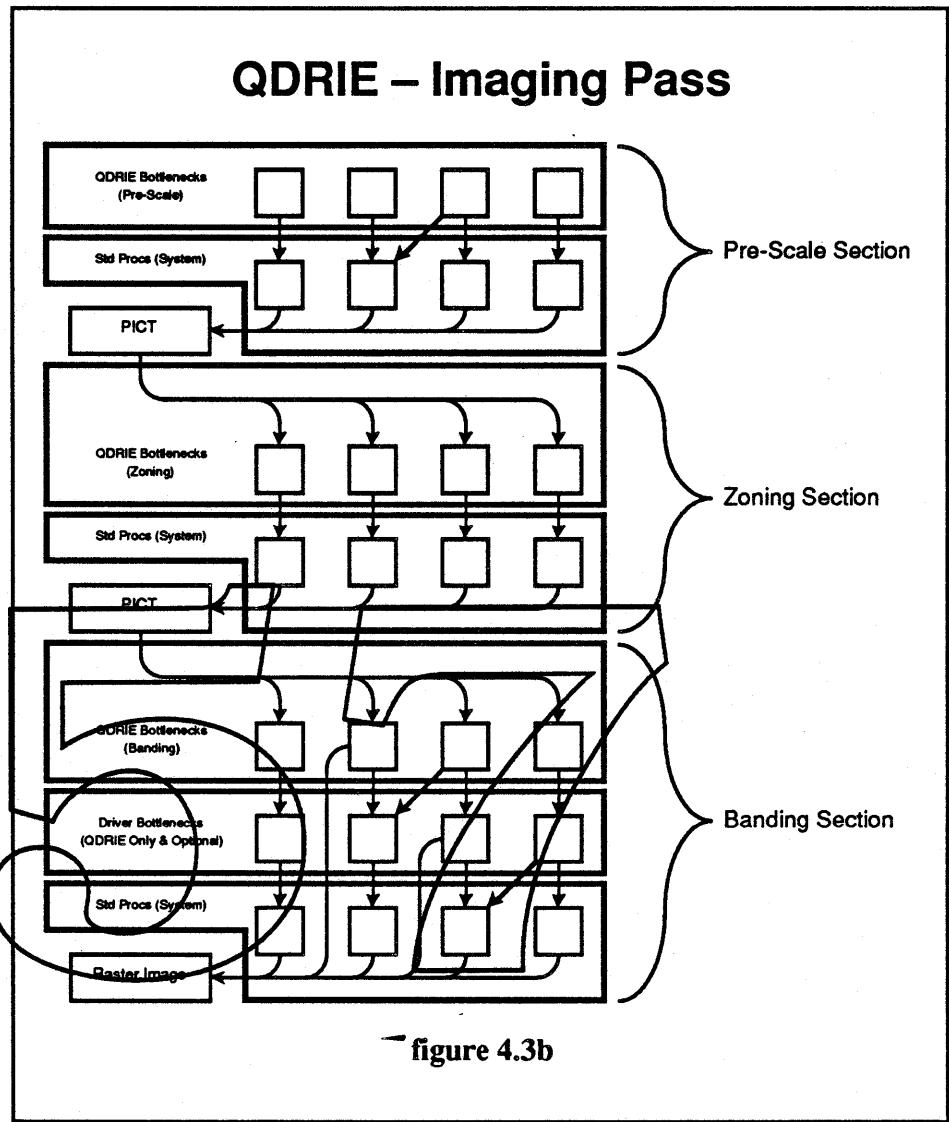
- **Region Support** – Regions will be fully supported when the application is drawing at device resolution. If the application is not drawing at device resolution then regions will be scaled causing scaling artifacts. There are currently no plans to implement a resolution independent region mechanism (but I'm thinking more about it).
- **Speed** – A major goal of the QDRIE is speed. Current raster drivers tend to underdrive the device partially because of their methods for zoning and banding and partially because of their implementation of sync-async io ;-). The QDRIE will do its part to relieve this problem while staying dynamic and capable of running in low memory environments.

### 4.3. Internal Operations:

In order to be as complete as possible in the description of the QDRIE it is important that the functions of the QDRIE bottlenecks are understood. The bottlenecks come in four flavors. There are the statistical bottlenecks that gather information about the QuickDraw objects on the page, the pre-scale bottlenecks that do numerical scaling on the objects and in a low memory situation they may generate zones, the zoning bottlenecks that split the image into zones to speed imaging, and the banding bottlenecks that do the actual drawing of the QuickDraw objects into the raster image.



The statistical bottlenecks are installed as the Imaging Engine for one pass through the page (see figure 4.3a). The statistical information that is gathered is not passed back to the driver as part of the data stream but is held internally for use by the QDRIE. The information that is gathered includes text bounding rectangles, bitMap locations and sizes and any other information that is deemed useful. The statistics pass may generate a list of holes on the page that could be used to adjust zones and notify the Driver of blank areas. This will be experimented with when a prototype is available.



After the statistics gathering pass is completed the rest of the bottlenecks are installed to handle the imaging (see figure 4.3b). The pre-scale section will handle the simple numeric scaling of the page and create a scaled PICT in memory. If enough memory is available then the entire page will be handled in one pass creating one large zone. This will help to eliminate the number of hits to disk to read the PICT. If there is not enough memory to accomplish this, then zones will be generated as large as possible (adjusted for maps if the Despooling Handler is not supplying them on demand).

The zone created in the pre-scaling pass may be split into sub-zones in the zoning section. This is done to cut down on the number of passes through all of the QuickDraw objects. A zone will be split into three sub-zones if each of those sub-zones can contain at least three bands. (\* The magic number three seems to be the optimal. With two you would break even (you may gain or lose a small amount of time) and with four you gain such an insignificant amount that the overhead of creating the additional sub-zones would not be worth it. I will experiment with sub-zones once a prototype is built. \*) Each of the sub-zones may be split into three sub-sub-zones if each of those sub-sub-zones would refer to at least three bands. Because banding sizes are going to be as large as possible sub-sub-zones will be vary rare and even sub-zones often times will not be needed. The depth of the sub-zones will be limited if landscape mode is in effect since there is a greater likelihood that graphics will span zones.

The QDRIE will generate bands from the zones. Once the QDRIE starts banding the actual raster image is created so the QDRIE could make direct hits on the raster image and avoid QuickDraw calls. This may be done for optimization of odd things (like support for b-curves) but would be disabled if the raster image is not directly accessible (it could exist out on a nubus board for a specialized printer). During the banding process, final scaling is done that affect actual imaging. This includes invocation of the Line Layout Manager and pattern stretching. Bands are created of approximately the same size so that there are as few as memory will allow on a page. This is different from being as large as possible where you could make each band a pixel larger but that would not reduce the number of bands on the page.

Before the final image is created, the Driver gets one last shot at the bottlenecks. Note that the location of this hit is specific to the QDRIE and will not exist in the QuickDraw to PostScript engines. The hit will not even be used by the QDRIE when it is gathering statistics. When the Driver does get called it will have a chance to directly manipulate the raster image. The Driver may also redirect the QuickDraw instructions out to a particular device or an accelerator card or some such thing. (\* Are we planning for the future here or what? \*)

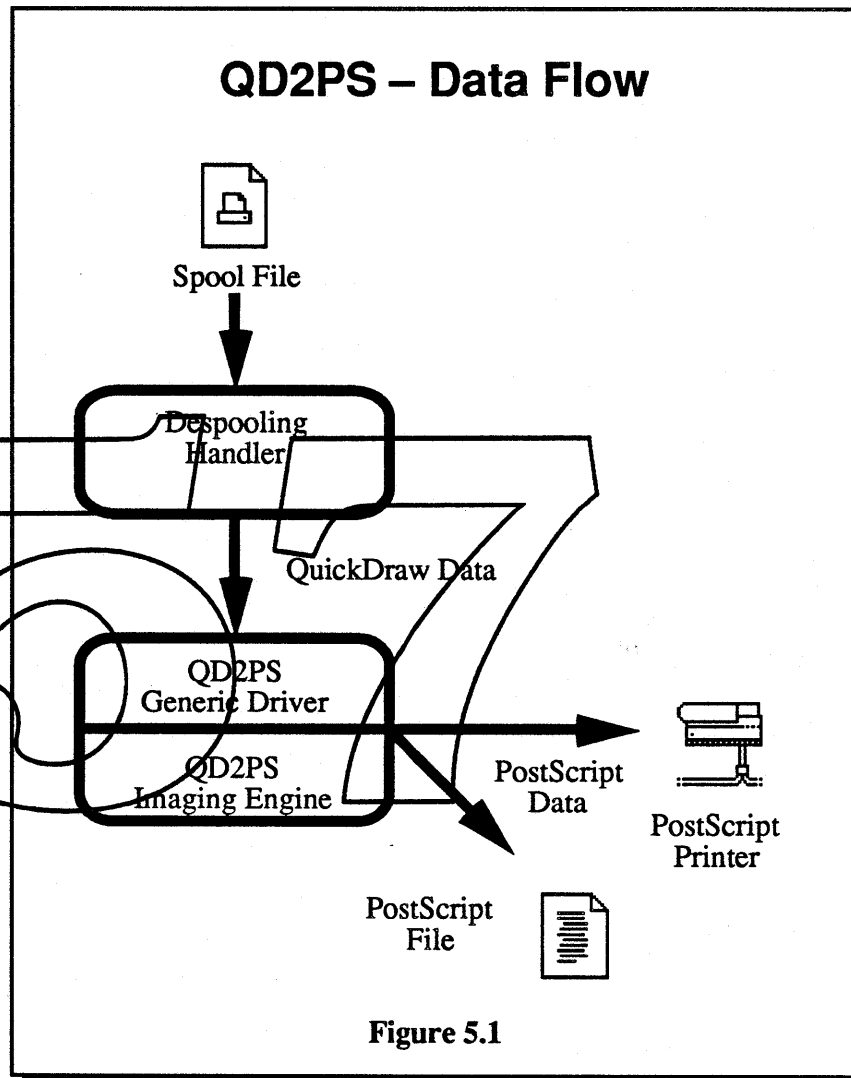
What we are left with is a raster image that is returned to the driver. From here the driver would normally run the image through the Food Processor, package it, and send it to the device.

## 5. QuickDraw to PostScript Imaging Engine (QD2PS):

### 5.1. Introduction:

The QD2PS Imaging Engine translates QuickDraw data into PostScript data. Our aim is to provide an efficient and well-structured postscript generator for QuickDraw. Most of the time-consuming work will be done on the Macintosh rather than on Postscript printers. We will try to make this engine portable enough so that it could be used with Pink (with help of Bayles Holt).

The QD2PS Imaging Engine takes QuickDraw data from via the bottlenecks and converts it to PostScript data to be sent to the Specific Driver's Packager where it will be routed to a PostScript printer or to a disk file to be printed later to any PostScript printer.



### 5.2. Additional Features, Goals, and Limitations:

Following is a rough list of features planned for the QD2PS imaging engine. This engine will provide almost all the functionality of the current LaserWriter driver. Features of the current LaserWriter driver that will or will not be supported in the Ginsu PostScript Driver are listed below:

• **Postscript Dictionary** – The QD2PS imaging engine PostScript dictionary will be sent in the header of each job. There will be an option for installing the dictionary permanently on the printer (until it is turned off). There will be a small prep file that will have to be downloaded at the printer at initialization time. This will include things like the LaserWriter II NT toner light patch, the smoothing (\* Isn't this being handled by the Food Processor? s.p. \*) and stretching code etc. This prep file is expected to be very small and is not likely to change as much as today's prep file. The PostScript dictionary will be segmented so that only the required part of the dictionary will be downloaded with the job header. For example: for text-processing applications, only the text PostScript procedures will be downloaded in the dictionary (unused graphics procedures will not be downloaded).

#### Advantages:

- a) Initialization of the printer will not be required when using different versions of Ginsu drivers. This will avoid "LaserWars".
- b) More virtual memory space will be available on the printer.
- c) It will be easier to take the PostScript file for the job and print it to any PostScript printer.

#### Disadvantages:

- a) The job size would be larger.
- **Fonts** – Each page will be treated as a separate job. This will help PostScript spoolers to manage jobs effectively. This will have some performance penalty in redefining fonts on every page but its benefits outweigh this performance penalty. Since we will know all the fonts required for a given page (refer to the Spooling Handler ERS for fonts use cache discussion), we will be able to download/define all the fonts at the beginning of the page. This will help us manage the printer virtual memory and will eliminate the need for the Unlimited Downloadable Fonts option in the Page Setup dialog.
  - **Font Substitution** – Font Substitution as an option will go away. Applications will be recommended to use LaserWriter resident fonts (for example, Helvetica) as their default font.
  - **Support for Regions** – The QD2PS Imaging Engine will support printing of regions in a limited way. There are two possible ways of supporting regions on a PostScript printer. The first method is to convert the region into a bitmap and print it using the `imagemask` operator just like we print maps today. This has obvious disadvantage since the QuickDraw verbs `fill`, `frame` etc. cannot be applied to a bitmap because a bitmap is not a PostScript path. The second method is to generate a PostScript path from the region data. This path can then be filled, framed, painted, or erased. (Note that the QuickDraw verb `invert` cannot be supported in PostScript because it requires inverting bits in the frame buffer.)

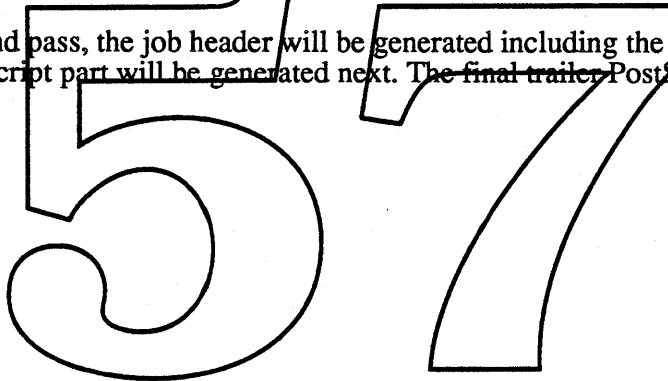
Along the same lines, clipping to an arbitrary region can also be supported in the PostScript driver.

- **Postscript File Structure** – The PostScript files created by the QD2PS Imaging Engine will follow PostScript document structuring conventions as much as possible, but they will NOT be EPS files. However, it will be possible to write a utility to convert these PostScript files into standard EPS files.
- **Bitmap Smoothing** – Bitmap and text smoothing option's fate is unknown. It is likely to remain as an option in the Ginsu PostScript driver. It may either be moved out as a function of the Food Processor or it may be kept as it is today; assembly language code that is downloaded in the prep file.
- **Line Layout** – The Layout Manager will be used to perform line layout of text strings. Applications will be recommended to use the Layout Manager so that the same calls are used when drawing to the screen as when drawing to the printer.

### 5.3. Internal Operations:

There may be a need for a statistic-pass over the QuickDraw data. This pass will help us compact the PostScript dictionary that is sent in the job header. During this pass, information about the type of QuickDraw data in the job will be collected.

During the second pass, the job header will be generated including the PostScript dictionary. The PostScript script part will be generated next. The final trailer PostScript will be generated last.

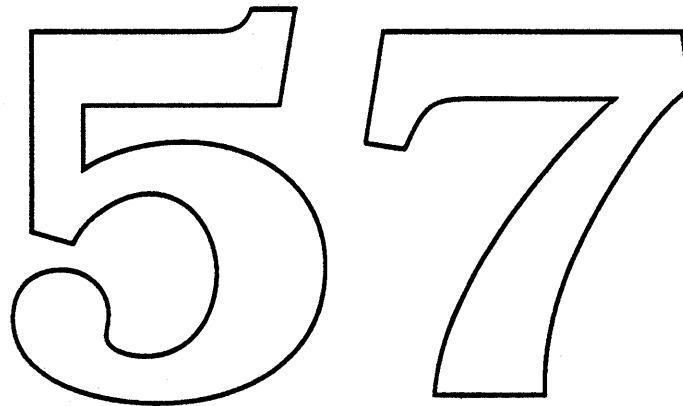




**6. QuickDraw to Vector Imaging Engine:**

---

To Be Supplied. (When will someone start working on this?)



---

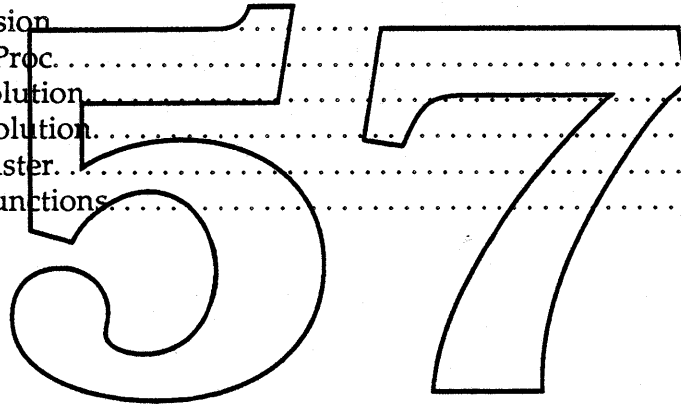
Ginsu Application Interface  
January 13, 1989  
Amy Rosenstock x46799  
The Blue PrintShop  
version 4

57

The Old vs. The New.....	1
The Ginsu Interface.....	4
The New Dialogs And Choosing the Printer.....	4
By Page Formatting And the New Print Records.....	5
Custom Paper Types.....	8
Reformatting Issues.....	9
Binding Documents to Printers.....	9
Fonts.....	9
Resolution.....	10
QuickDraw Space.....	10
Color.....	10
Zoom.....	10
Proposed New Print Manager Interface.....	11
Initiation and Termination of Printing.....	11
PrStart.....	11
PrEnd.....	13
The New Print Records.....	14
PrSizeOfJobRecord.....	14
PrSizeOfFormatRecord.....	15
PrUpdatePrintRecord.....	16
PrValidateJob.....	18
PrValidateFormat.....	19
PrJobDefault.....	20
PrFormatDefault.....	21
The Dialog Calls.....	22
PrDoStyleDialog.....	23
PrDoJobDialog.....	26
PrAddPanel.....	28
Customizing the Style and ByPageFormat Dialogs.....	31
PrAddOrientation.....	31
PrSetOrientation.....	33
PrGetOrientation.....	35
PrAddPaperType.....	37
PrSetPaperType.....	39
PrGetPaperType.....	41
PrSetScaling.....	43
PrDisableScaling.....	45
PrGetScaling.....	46
Customizing the Job Dialogs.....	47
PrSetCustomPageRange.....	47
PrGetCustomPageRange.....	49
PrDisablePageRange.....	50
PrGetPageRange.....	51
PrSetReduction.....	52
PrGetReduction.....	54
PrSetStackingOrder.....	55



PrGetStackingOrder.....	57
PrSendingLastFirst.....	58
PrSetCollation.....	59
PrGetCollation.....	60
The Print Loop.....	61
PrStartDoc.....	61
PrEndDoc.....	62
PrStartPage.....	63
PrEndPage.....	65
Error Handling.....	66
PrErrorString.....	66
PrSetErr.....	67
PrGetErr.....	68
Memory Considerations.....	69
PrMemReserv.....	69
PrGrowFriendly.....	70
Imaging, etc.....	71
PrGetVersion.....	71
PrSetIdleProc.....	72
PrSetResolution.....	73
PrGetResolution.....	74
PrPSToRaster.....	76
Summary of Functions.....	77



## The Old vs. The New

Ginsu introduces a new printing architecture and a new set of device drivers. This revised Print Manager will provide enhanced performance and greater functionality. Current applications which have complied with printing guidelines specified in Inside Mac and the tech notes should run under Ginsu just as they did with the old print drivers. However, to take advantage of new printing features, an application must rev to the new Print Manager.

Applications should find substantial gains in revving to Ginsu. Ginsu supports background printing for all devices, better memory management and error handling, procedural interface to relevant printing information (no more need to delve into the printing structures for clues), better application/printing dialog integration, support for 'by Page' formatting, custom page sizes and more rational page range selections.

With Ginsu, the PrintShop begins a movement to turn more control back over to the application. In the future, all printing effects should be decided on between the application and the user. If the application wants to provide a facility to invert images or do horizontal and vertical flips, etc., the interface should initially be between the application and the user. The application will then pass on its requests to the print software.

Along with Ginsu, new system tools will be available. The Layout Manager will allow the application to perform device independent character positioning. This should replace the need for FractEnable and allow text to look great on the screen and on the printer. We encourage developers to use the new Layout Manager when working with text. It is important to note here that since all printing is done from a spooled file, playing with fields in the font width tables will have no effect on the printed document. The Layout Manager should take care of all character positioning needs.

Ginsu requires basic changes in the application interface. The current Print Manager interface relies on a print record of type TPrint. This data structure will not be supported in future versions of the Print Manager. It will be supported initially to keep current applications running. The print record must change to support new Ginsu functionality. This implies that the current application calling interface must change. The current application calling interface will be supported initially to keep current applications running, but this will go away in time. Applications which wish to run on future systems must rev. This has further implications. All new drivers produced by Apple will be Ginsu drivers. In order for an application to take advantage of new Apple device drivers, the application must rev to use Ginsu.

The PrintShop takes this opportunity to strengthen the interface. All new printing calls will return status. This status should be checked by the application and acted on accordingly. A parallel set of routines will be supplied for those existing in the current

interface with the exception of PrPicFile and the Low Level Driver. Under Ginsu, all print jobs will be spooled unless spooling fails due to low disk space. In this case, a non-spooled printing method will be supplied. This method will not involve a PrPicFile type interface so this interface goes away. The Low Level Driver goes away, too. There will no longer be a data path for sending control commands to the printer.

PrGeneral is another casualty of the new Printing Architecture. This will not be a loss, as the new architecture will supply procedural interfaces for any information that an application may want to access. There will be new routines to get and set resolution and rotation plus numerous other routines to get and set other formatting features. There will be new support for draft printing, including different levels of output quality.

Ginsu provides better support for device independent printing. All supplied PICT comments will be device independent with the exception of a comment which allows the application to send native PostScript directly to a PostScript device. If an application chooses to send native PostScript through the PICT comments, equivalent QuickDraw instructions must be supplied to render the same results for a QuickDraw printer. This will allow application documents to print to any device. QuickDraw devices will emulate those PostScript features described through the current PICT comments. Also, all PostScript drivers will provide some level of support for regions, transfer modes and other non-PostScript features through the power of QuickDraw.

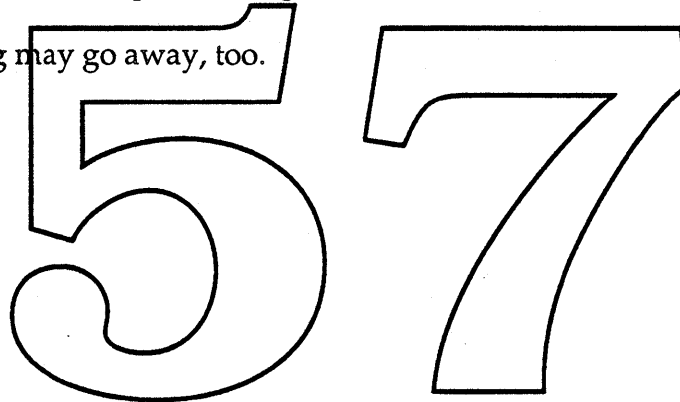
A major change in the Printing Architecture takes place with the Page Setup and Print dialogs. New Ginsu literate applications will have the opportunity to add as much information as they like to these dialogs. They may also set and query fields in the dialog sections controlled solely by the printing software. There is a new mechanism for applications to add to the dialogs. The old method of using the bottom half of the screen will not be supported in Ginsu aware applications. In future versions of Ginsu, this mechanism will go away all together.

In the past, it has been recommended that all applications link with the print Glue in the PrintCalls module so that they may run on 512K Macs. Ginsu does not support 512K Macs. The problem here being that there is too little memory. However, it is possible to add memory to the 512s. There will be no testing done on these machines. A new version of PrintCalls will be provided with Ginsu. Linking with this module should allow printing on all Macs, 512K and above, memory permitting. PrintCalls will invoke the PrGlue trap if it exists.

Old applications running under Ginsu should see no change in printing behavior. These applications will receive old print records and old style dialogs. The PrGeneral opCode to control draft bits will no longer be supported, but if an application checks for the 'opNotImpl' status returned from PrGeneral and acts accordingly, this should not pose a problem. The user, however, will see some change. Ginsu will not support for any application, old or new, certain printer effects. These include:

- font substitution. We encourage applications to use a LaserWriter resident font as the default document font.
- graphics smoothing
- text smoothing
- unlimited downloadable fonts - in Ginsu this will always be true
- larger printable area - the user will determine the printable area solely on the chosen page size.
- precision bitmap alignment - this will be handled through the reduction and enlargement factors in the print dialog.

Faster bitmap printing may go away, too.



## The Ginsu Interface

### The New Dialogs And Choosing the Printer

Print dialogs are undergoing a change. There will still be a basic style and print dialog as before. We attempt to make these dialogs look similar across devices. Two major things come into play with Ginsu, 1) the system printer has gone away and 2) choosing the printer no longer involves the Chooser.

Users will now choose a printer type in the style dialog (ImageWriter, LaserWriter, etc.). This will provide us enough information to do basic formatting. The destination printer will be chosen through the Job Dialog. Since printer types may now change within an application session through the Style Dialogs, applications interested in device resolution must check for a change in resolution after the Style Dialog terminates.

In Ginsu, we add new functionality and more options for the user and the application. Real Estate in the old dialogs had become too scarce a commodity. To remedy this situation, the Print Manager has adopted a new dialog interface. The new interface will follow the Control Panel model, with some adjustments. On selecting Page Setup, Printing or initiating By Page Formatting, the user will encounter a basic dialog. Selecting OK at this point in the dialogs should produce reasonable results. If the user wishes to do more adventurous things with his print job, he may select an Options button which will bring him into the expanded Control Panel type interface. The first panel in this part of the dialog will look very similar to the basic dialog, however, icons will appear on the left side of the panel which will allow the user to move to other panels. An interface is supplied to allow the application to add values to the Printing Basic Dialogs. There are also provisions for the application to add entire panels to the extended dialogs.



## ByPage Formatting and the New Print Records

In Ginsu, we provide support for 'by Page' formatting. This allows the user to have multiple page formats inside a single document. This mechanism also indicates where printer pages should be fed from, providing an automatic mapping between document pages and printer paper flow. The byPage format feature would be most useful in a mail merge type of application, but should prove useful in numerous other instances. In order to support this feature, we introduce two things, a new print record structure and an extended use of the Page Setup dialog.

In the old printing architecture which supported only a single document format, it was sufficient to maintain a single corresponding print record. This print record was stored with the document and used to pass information back and forth between the application and the Print Manager and between the user and the Print Manager. The print record contained formatting information such as page dimensions, job information such as number of copies and various other miscellaneous information. It was used differently by different devices. It contained numerous fields which were only relevant during the life of the print job and had no meaning to either the user or the application.

In order to support 'by page' formatting and reduce the amount of information which needs to be stored with a document, Ginsu splits the print record into three distinct pieces, the Format Record, the Job Record and the Printing GrafPort Structure.

1) *The Printing GrafPort Structure* - This record contains all the information the Print Manager needs to keep around concerning the state of the current print job. This information only has relevance to the Print Manager. We attach all the necessary state information to the bottom of the Printing GrafPort for easy access by all printing routines. The printing grafport structure is passed back and forth between the application and the Print Manager. It will not be stored with the document. Encapsulating all state information in the printing grafport structure permits the printing code to be reentrant, allowing an application to initiate multiple print jobs. This will be most convenient for Print Monitor. Each job will require a unique printing grafport structure.

2) *The Job Record* - This record contains information relating to the current print job which must be stored with the document.

3) *The Format Record* - This record contains information relating to the document format and must be stored with the document.

Both the Job Record and the Format Record contain information relevant to the user and the application. A single document will have one printing grafport structure, one Job Record and one or more Format Records.

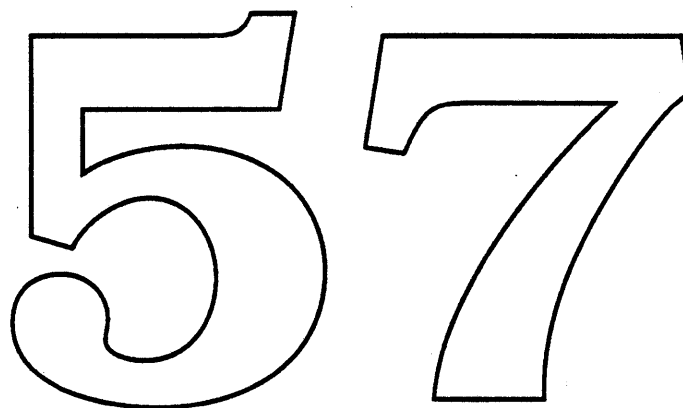
The Print Manager will allocate the printing grafport structure when the Print Manager is initialized ( see PrStart for more information ). The Job and Print Records will be allocated by the application. These records are device specific and may be expected to be of varying lengths depending on the device they are associated with. In order to provide for growth in the printing architecture, a specific device Job or Format Record size may also change in the future. The application should not assume anything about these record sizes. This applies to assumptions about the size of these records during the course of a single application session as well. It is possible for the user to switch printers in the middle of an application. If the device type changes, this may require a different Job or Format record and the sizes of these records will not be guaranteed to be the same size as the previous ones. With this in mind, we provide calls for the application to determine the current size of these records. See the functional interface for the Print Records for more information.

The Page Setup dialog will be available for 'by Page' formatting support. In Ginsu, PrStartPage, which replaces PrOpenPage in the old architecture, takes a format record as a parameter. This tells the printing software how the page is to be formatted. A document may use many formats for a single document or only one. PrDoStylDialog will have a flag, which when set, will indicate that the call is being made specifically for a page format as opposed to a document format. Calling PrDoStylDialog with the byPage flag set will present the style dialog without the option to change printer type, because a single document needs to be tied to a single printer type. The Format Records will be derived from these dialogs. The byPage flag should never be set when PrDoStylDialog is called as a result of the user selecting Page Setup from the File menu.

It is up to the application to provide an interface to the user to initiate creation of the by page formats. Multiple formats may not be appropriate for all applications and need not be supported. Applications supporting by page formatting will find various ways to introduce this concept into the application. There must be a way for a user to select document pages and inform the application that he wishes to create a special format for the selection. At this point the application calls the Print Manager to determine the size of the Format Record to create and then allocates the new Format Record. The application then calls the Print Manager with the new Format Record to display the by Page formatting dialog . Through the user selections, the Format Record is filled in. At the end of the dialog call, the Format Record is returned to the application.

The Print Manager will work with only one Format Record at a time. It is the application's responsibility to keep track of these records. This responsibility includes allocation fo new records, deletion of unused records and sending the appropriate Format Record for the appropriate page at PrStartPage. (see the function description for PrStartPage for more information). An application may create its own custom formats through procedural interfaces without displaying the supplied byPage format dialogs. This would allow an application to provide its own interface. This is not, however, the recommended practice.

Note: The Format and Job Records will be passed by handle. This allows us to grow or shrink these structures between releases as necessary (or between device changes within a single print job) without breaking apps. An app must never assume anything about the size of these structures. The printing software will provide procedures for the application to determine sizes as needed.



## Custom Paper Types

Ginsu supplies a facility for specifying Custom Paper Types. An appropriate set of paper types will ship with every driver. Paper type includes paper size and imageable area. More attributes may be added to these in future releases. A single page format will contain a single paper type.

Paper types shipped with a given printer will include the familiar set of US Letter, Legal, etc. with dimensions custom fit to the particular device. Also included will be a basic set which serves as the generic counterpart to the familiar set. This set will print on any current Apple printer with no need to reformat or map in order to go to a different printer type. This implies that the imageable area and resolution of this second set of paper types fall within the minimum bounds of imageable area and resolution encountered in today's Apple printers.

A major reason for the existence of paper types is to allow an automatic mapping of document pages to printer input trays and output bins. Assigning attributes to document pages and printer trays and bins will dictate the paper flow through the printer. Document pages will map to printer trays with corresponding attributes which will in turn map to output bins with matching attributes.

Custom paper types may be used to limit or expand the imageable area of a document. This feature should lend support to mailmerge type applications. It could also be used to allow users to create posters. An oversized imageable area would result in a user choice to scale, tile, clip or abort at print time.

## Reformatting Issues

In the current printing and system architectures, users are plagued with the threat of having their documents reformat simply by opening them. Ginsu will alleviate some of this reformatting by doing away with the concept of a system printer. Today, an existing application opens a document and calls PrValidate to validate the print record. If the system printer has changed, the print software updates the print record and tells the application that the change has been made. The application checks the page dimensions in the print record, sees that they differ from the current document dimensions and reformats the document. In Ginsu, documents will be bound to the printer type that the user requests when creating the document. A document should not reformat unless the user specifically requests that it reformat for output on a different printer type. The first Format Record sent to validate for a print session will decide the initial printer type. If this is an existing document, the printer type will be set to correspond to the printer type associated with the job the last time it was set up for printing. If this is a new document, a default printer type will be associated with the document at this time. Ginsu provides three methods for dealing with printing documents to devices with non-matching page dimensions or devices containing media with dimensions which do not match those specified for a document page.

- 1) Basic paper types are provided with the drivers which have page dimensions and resolution defined that will fit into the imageable area and resolution of all Apple printers. If a user wishes to print to several printers without reformatting, he may choose one of these basic types.
- 2) A user may choose at print time to print to a device containing media whose page size or imageable area does not match those defined for his document. The user may choose to map what he has without reformatting by choosing to scale, tile or clip the document to the dimensions of the media in the device.
- 3) The user may choose to have his document reformat before outputting to a device containing media with non-matching dimensions as his document.

Reformatting may also occur when a user reopens a document which was created on a system with different fonts than are available at the time of reopening. There is nothing the print software can do to alleviate this problem. An application, however, should notify the user of this situation before reformatting the document.

## Resolution

Device resolution affects the format and content of a document. In the old architecture an application could get and set device resolution. In Ginsu this will still be supported. However, the new support will differ from the old support. Because we allow custom page sizes for any device of any resolution there may arise a resolution/page size combination whose representation would exceed QuickDraw space. Applications should be able to handle negative coordinates to take full advantage of QuickDraw space (-32767 to 32767). This still may not always be adequate to handle all custom page size/ device resolution cases. An example would be a legal size document on a 4000 dpi film recorder. In these cases, when queried for resolution, the Print Manager will consider the device and the page size and return the greatest resolution possible for imaging a page of the given size without exceeding QuickDraw space.

Applications imaging at device resolution should note that the number of colors an application may take advantage of will be limited, as there are no extra pixels left to take advantage of dithering for new colors.

In the current printing architecture, printing supports a feature to allow the user to reduce or enlarge the document imaging area. What actually occurs is that choosing a value less than 100% causes the page size to increase, thus allowing the user to enter more information on the page. Choosing a value greater than 100% causes the page size to decrease, thus allowing less information on the page. At print time, the print code scales the pages accordingly for the device. In Ginsu, we add a second type of enlargement and reduction. This second form is presented in the Job Dialog and follows the copier model. This percentage is applied as the page is imaged to scale up or down.

Following is the proposed new interface for the Printing Manager. The first set of calls is used to initiate and terminate the printing session for a document.

```
PrintError PrStart( TPrGrafPort pPrGrafPort );
```

*what happens:*

With this call, the application has entered the Ginsu world. There is no turning back. The application may use all Ginsu interfaces and features, but none of the old Print Manager calls.

When this call is received, the printing software sets a flag to signal that the application has revved. From this point on, if any old style print call is made, the printing software will return an error. [ There are exceptions here. PrClose, PrError and PrSetError are handled entirely within the Glue. We plan to patch CloseResFile from PrStart so that we can catch the PrClose call. This allows us to cleanup after the TackleBox. We can also take this opportunity to see if the application is mixing calls. There is no comparable opportunity for PrError and PrSetError].

This call replaces PrOpen in the old Printing Manager. PrStart prepares the Printing Manager for use. The printing grafPort is allocated, the appropriate resources are loaded and the Print Manager structures are allocated and initialized. Every call to PrStart must be balanced with a call to PrEnd.

*when to call:*

This is the first call the application must make to begin a job with the Print Manager. Note that an application may have several documents open simultaneously. However, QuickDraw depends on A5 globals, so attempting to image more than one job at a time without saving and restoring these globals will not work. The printing reentrancy feature is most useful to PrintMonitor, which may simultaneously image one job while sending multiple other imaged jobs to print.

*the parameters:*

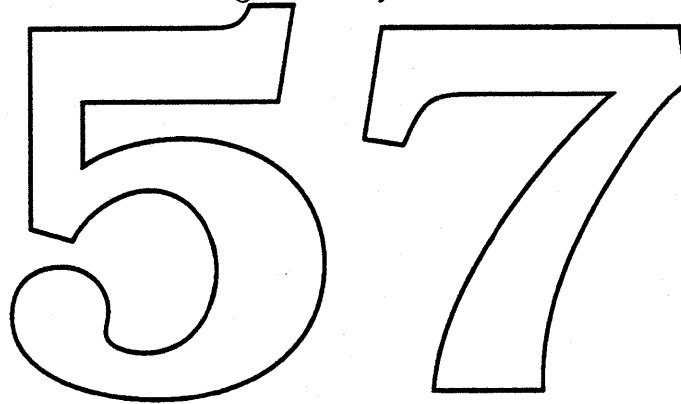
**pPrGrafPort** This contains all the information the printing software needs to maintain the state of the current job. It is allocated through PrStart. This structure replaces the dependency on low memory globals.

*the function result:*

**noErr** Initialization of the Print Manager was successful. pPrGrafPort is a pointer to the Print Manager grafPort and state information.

**resNotFound** a Print Manager resource is missing. (As Ginsu matures this result code may become more specific.)

**memFullErr** there is not enough memory to allocate the structures needed for printing.





**PrintError PrEnd( TPrGrafPort pPrGrafPort );**

*what happens:*

This call replaces PrClose in the old Printing Manager. PrEnd ends a print job. It releases the memory used by the Print Manager. It closes the necessary resource files if no other jobs are open.

*when to call:*

This is the last call the application must make to end a job with the Print Manager.

*the parameters:*

pPrGrafPort This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart. It is deallocated in PrEnd.

*the function result:*

noErr Deallocation of the print job storage was successful. pPrGrafPort is nil.  
badPort pPrGrafPort is corrupt.

The next set of calls will be used to work with the new print record structures.

```
PrintError PrSizeOfJobRecord( Int16          *iJobRecSize,
                              TPrGrafPort  pPrGrafPort );
```

*what happens:*

This call will supply the application with the size of the current Job Record.

*when to call:*

This call should be made after the call to PrStart for a new document to determine the size of the Job Record which the application will allocate. When transforming an old style print record to a new Job and Format record set, this call will determine the size of the Job Record after the conversion. See the PrUpdatePrintRecord function for more information.

Note: The application must store the size of the structure with the document so that it knows how much to read back when reopening the document.

*the parameters:*

iJobRecSize This parameter returns the size of the Job Record in bytes.

pPrGrafPort This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

noErr All is well.  
badPort pPrGrafPort is corrupt.

```
PrintError PrSizeOfFormatRecord( Int16          *iFormatRecSize,
                                TPPrGrafPort pPrGrafPort );
```

*what happens:*

This call will supply the application with the size of the current Format Record.

*when to call:*

This call should be made after the call to PrStart for a new document to determine the size of the Format Record which the application will allocate. When transforming an old style print record to a new Job and Format record set, this call will determine the size of the Format Record after the conversion. See the PrUpdatePrintRecord function for more information.

Note: The application must store the size of the structure with the document so that it knows how much to read back when reopening the document.

*the parameters:*

iFormatRecSize This parameter returns the size of the Format Record in bytes.

pPrGrafPort This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

noErr All is well.  
badPort pPrGrafPort is corrupt.

```
PrintError PrUpdatePrintRecord( THPrint hOldPrintRecord,
                                Handle hJobRec,
                                Handle hFormatRec,
                                TPPrGrafPort pPrGrafPort );
```

*what happens:*

PrUpdatePrintRecord will allow the application to trade in an old style print record for a new Job and Format record set. The rPage and rPaper fields will be mapped from the old print record to the new print records.

Some applications took advantage of PREC 4 which allowed additional paper and imageable area sizes to be added to a print job. For cases where an unknown rPaper or rPage size is encountered, the printing software will map to the Apple Generic page dimensions. Note that an application may change these to whatever they wish through calls to the functions which affect paper types. A user may modify these fields through the dialogs.

*when to call:*

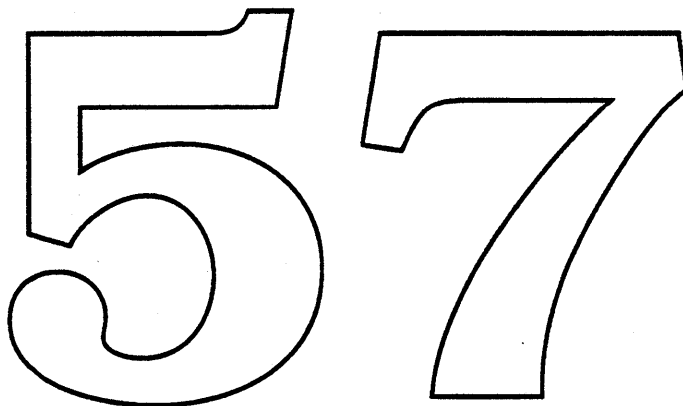
This call should be made when an application needs to transform an old style print record into a new Job and Format Record set. The call should be made after the Print Manager has been initialized with PrStart and the application has allocated the appropriate Job and Format Records.

*the parameters:*

- |                 |   |
|-----------------|---|
| hOldPrintRecord | This is a handle to the old style print record which the Print Manager will reference to initialize the corresponding Job and Format Records. hOldPrintRecord will return unmodified.   |
| hJobRec         | This is a handle to the Job Record allocated by the application. It will return initialized for the current printer type.   |
| hFormatRec      | This is a handle to the Format Record allocated by the application. It will return with initial page dimensions set from the old style print record. If those sizes are unfamiliar to the Print Manager, the paper dimensions will be set to the current printer type defaults. |
| pPrGrafPort     | This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.   |

*the function result:*

noErr	Successful mapping of the old print style record to the new Job and Format records.
badPort	pPrGrafPort is corrupt.
badJobRec	hJobRec is nil or of the wrong size.
badFormatRec	hFormatRec is nil or of the wrong size.



**PrintError PrValidateJob(Handle hJobRec, TPrGrafPort pPrGrafPort);**

*what happens:*

This call replaces half the PrValidate call. See PrValidateFormat for the second half of the replacement. PrValidateJob checks the passed Job Record for compatibility with the current version of the Printing Manager and the currently selected printer. If the Job Record is valid, the function returns noErr (no change). If the Job Record is invalid, the record is adjusted to the default values for the selected printer and the function returns with the result equal to 'updated'.

*when to call:*

This call should be made whenever a Job Record needs to be validated. It should be called after initializing the Print Manager through PrStart when reopening a document.

*the parameters:*

hJobRec This is a handle to the Job Record created by the application.

pPrGrafPort This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

noErr The Job Record did not need updating.

updated The Job Record contained an old version or did not reflect the currently selected printer. It has been updated. This may result in the size of the structure changing.

updateFails The Job Record contained an old version or did not reflect the currently selected printer, resulting in an attempted update which failed. This may be due to a low memory condition.

badPort pPrGrafPort is corrupt.

badJobRec hJobRec is nil or of the wrong size.

```
PrintError PrValidateFormat( Handle      hFormatRec,
                             TPrGrafPort pPrGrafPort);
```

*what happens:*

This call replaces half the PrValidate call. See PrValidateJob for the second half of the replacement. PrValidateFormat checks the passed Format Record for compatibility with the current version of the Printing Manager. If the Format Record is valid, the function returns noErr (no change). If the Format Record is invalid, the record is adjusted to the default values for the selected printer type and the function returns the value 'updated'.

In Ginsu, documents are bound to the printer type specified when they were created. This information is stored in the Format Record. The first Format Record sent to validate for a print job will decide the initial printer type. The current printer type will be stored in the printing grafPort record. This model alleviates reformatting problems related to the old system printer model.

*when to call:*

This call should be made whenever a Format Record needs to be validated. It should be called after initializing the Print Manager through PrStart when reopening a document.

*the parameters:*

**hFormatRec** This is a handle to the Format Record created by the application.

**pPrGrafPort** This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

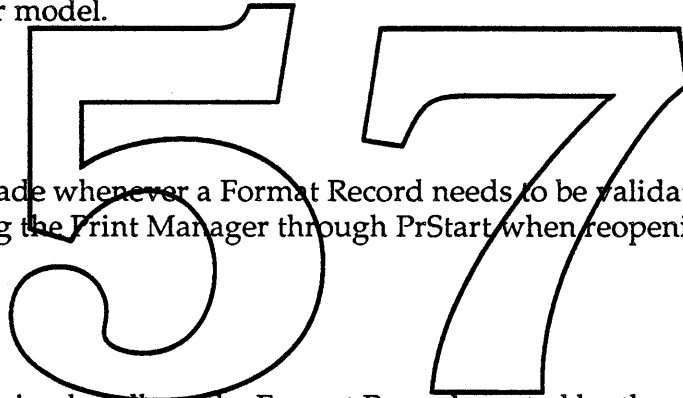
**noErr** The Format Record did not need updating.

**updated** The Format Record contained an old version. It has been updated. This may result in the size of the structure changing.

**updateFails** The Format Record contained an old version, resulting in an attempted update which failed. This may be due to a low memory condition.

**badPort** pPrGrafPort is corrupt.

**badFormatRec** hFormatRec is nil or of the wrong size.



**PrintError PrJobDefault( Handle hJobRec, TPrGrafPort pPrGrafPort);**

*what happens:*

This call replaces half of the PrintDefault call in the old Printing Manager. See PrFormatDefault for the other half of the replacement. This call will fill the passed Job Record with the appropriate defaults for the current driver type. The passed Job Record may be newly created or existing from a document. In both cases, the fields will be updated.

*when to call:*

This call should be made whenever a Job Record needs to be initialized. For new documents this should be called after the Print Manager is initialized through PrStart and the application has allocated the appropriate Job Record.

*the parameters:*

**hJobRec** This is the handle to the Job Record which will be filled with the default values for the current printer type.

**pPrGrafPort** This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

**noErr** Successful initialization of the Job Record.

**badPort** pPrGrafPort is corrupt.

**badJobRec** hJobRec is nil or of the wrong size.



```
PrintError PrFormatDefault( Handle      hFormatRec,
                             TPPrGrafPort pPrGrafPort );
```

*what happens:*

This call replaces half of the PrintDefault call in the old Printing Manager. See PrJobDefault for the other half of the replacement. This call will fill the passed Format Record with the appropriate defaults for the current driver type. The passed Format Record may be newly created or existing from a document. In both cases, the fields will be updated.

*when to call:*

This call should be made whenever a Format Record needs to be initialize. For new documents this should be called after the Print Manager is initialized through PrStart and the application has allocated the appropriate Format Record.

*the parameters:*

hFormatRec This is the handle to the Format Record which will be filled with the default values for the current printer type.

pPrGrafPort This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

noErr Successful initialization of the Format Record.

badPort pPrGrafPort is corrupt.

badFormatRec hFormatRec is nil or of the wrong size.

The next set of calls will be used to work with Dialogs.

In the old Print Manager, dialogs were simple. They were conducted between the user and the printing software. The application made one call to initiate a dialog and had another facility for adding onto the bottom of our screens. In Ginsu we attempt to provide more functionality for all. This involves a lot more interface. Applications will no longer add onto the bottom of our dialogs. However, applications will be able to add entire panels to our dialogs. They will also be able to get, set and lock (though locking is highly discouraged) a subset of the fields in our dialogs.

Certain rules are applied to setting values in the Print Manager Basic Dialogs. The application may add a value, such as a new paper type or orientation. The application may also set the default setting for a value in the Basic Dialogs. The application may lock a setting in the Basic Dialog so that the user may make no other choice and in some instances the application may disable fields in the Basic Dialog. There is one rule to adhere to: The user has the last word. For a new document, the Print Manager will set the values in the Dialogs to a standard set of defaults. The application may then override these settings. As soon as the dialog is displayed to the user, these default settings are frozen. The application may not override a user choice. For existing documents, the settings in the Basic Panel will be taken from the corresponding Format and Print Records. The application may query for most settings in these records. An application is free to do whatever it wants at anytime with the values in the panels it supplies.

Calls are provided to allow the application to add panels to the dialogs. This information will be stored with the Print Manager until the appropriate dialog is initiated and completed. Upon completion of the dialog, i.e. the user hits OK or Cancel to dismiss the dialog, this information will be disposed of, freeing up any memory that it occupied.

At the Print Job Dialog time, the Print Manager needs a list of all formats used in the document to allow the user to identify any document pages which are to be manually fed to the printer. A call is provided for the application to supply these to the Print Manager. This call must be supplied in order for the user to initiate a job which requires some pages manually fed and some pages fed automatically. This is only required for documents containing multiple formats.

**PrintError PrDoStyleDialog(**

```

    Handle      hFormatRec,
    TPrGrafPort pPrGrafPort,
    Boolean     fByPage,
    Int16       PrinterChange(),
    Boolean     Formats(Handle *hFormatRec, Integer iWhichFormat)
);

```

*what happens:*

This replaces the PrStlDialog call in the old Print Manager. This call conducts a style dialog with the user to determine formatting for the document. This call is also provided to support by page formatting. This call conducts a style dialog with the user. The Format Record will be updated to reflect the outcome of the dialog.

*when to call:*

This function should be called in response to the user selecting Page Setup in the File menu or in response to a user requesting to format a page or series of pages in an application that supports by page formatting. The Print Manager must be initialized through PrStart and the application must have a valid Format Record. If the application wishes to customize the Style Dialog through a series of PrAdd and PrSet calls, these calls should be made before the call to PrDoStyleDialog. Customization of the dialogs is optional.

Note: Users now specify a printer type in Page Setup. The printer type must be consistent across all formats in a multi-format document. Applications that support multiple formats within a document must update format records in the case that the user changes printers. If the user changes printer types, the Print Manager will call the Formats routine to update all other Formats to the current printer type.

*the parameters:*

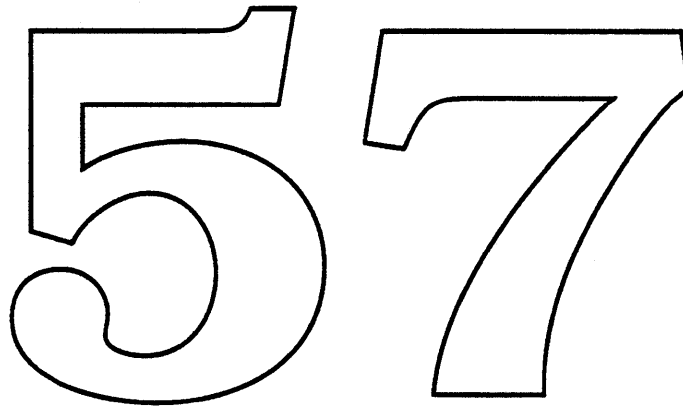
**hFormatRec** This is a handle to the Format Record for the current document.

**pPrGrafPort** This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

- fByPage** A boolean indicating whether the dialog is being supplied to support by page formatting or to support document setup. If the call has been made in response to the user choosing Page Setup from the File menu, fByPage should be false. Otherwise, fByPage should be set to true. If fByPage is true, the user will not be allowed to switch printer types in the style dialog.
- PrinterChange** An application supplied procedure which will be called if the user switches printer type in the style dialog. This will give the application the opportunity to change information it supplies in its panels which would be affected by the device change. The application will not be supplied with the new device type but may query for change in resolution and possible other attributes.
- Formats** An application supplied procedure which will return document formats so that the Print Manager may update them to reflect the current printer type in the case that the user changes printer in the Style Dialog. This is only applicable to applications which support by Page formatting. The call interface for this routine should look like the following:
- ```
Boolean Formats( Handle *hFormatRec, Integer iWhichFormat );
```
- The Print Manager will call this routine repeatedly incrementing the iWhichFormat value until the function returns false indicating that all formats have been delivered. The application must be able to access all document Format Records sequentially. The Print Manager will call the Formats function the first time with iWhichFormat equal to 1. The application should set hFormatRec equal to the first Format Record in the document. Subsequent calls will request subsequent Format Records. The application should continue returning a function value of true and a handle to the next Format Record until iWhichFormat exceeds the number of unique formats in the document. If the application does not support by Page formatting the Formats parameter should be nil.

*the function result:*

- noErr           The user exits the dialog with OK. The printer type does not change.
- prChange        The user exits the dialog with OK. The printer type has changed. Applications that rely on device resolution, should do a PrGetResolution call. Applications that support 'by Page' formatting will have all Format Records updated through calls to Formats(). Note that the Format Record size may have changed.
- cancel           The user exits the dialog with Cancel.
- badPort         pPrGrafPort is corrupt.
- badFormatRec    hFormatRec is nil or of the wrong size or type.



**PrintError PrDoJobDialog(**

```
    Handle      hJobRec,  
    TPrGrafPort pPrGrafPort,  
    Int16       PrinterChange(),  
    Boolean     Formats(Handle *hFormatRec, Integer iWhichFormat)  
  
);
```

*what happens:*

This replaces the PrJobDialog call in the old Print Manager. This call conducts a job dialog with the user to determine how the job will be printed. The user may choose a printer here that does not match the printer type selected in Page Setup. This implies that the Job Record may change size. The Print Manager will map pages accordingly. If the user wants the document to reformat, he should return to Page Setup and change the printer type.

In Ginsu, many new features are added. The Job Dialog may contain numerous panels for a given device. In this light, it does not seem reasonable to support a PrJobMerge call in the new architecture. Job records do not readily map across devices.

For applications supporting 'by Page' formatting, a call must be supplied for passing all Format Records to the Print Manager before the Job Dialog is displayed. The Print Manager will extract the paper type names in these Format Records and display them to the user to allow the user to designate which paper types are to be hand fed to the printer.

*when to call:*

This function should be called in response to the user selecting Print in the File menu. The Print Manager must have been initialized through PrStart and the application must have a valid Job Record. If the application wishes to customize the Job Dialog through a series of PrAdd and PrSet calls, these calls should be made before the call to PrDoJobDialog. Customization of the dialogs is optional.

*the parameters:*

- hJobRec** This is the handle to the Job Record for the current document.
- pPrGrafPort** This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.
- PrinterChange** An application supplied procedure which will be called if the user switches printer type in the style dialog. This will give the application the opportunity to change information it supplies in its panels which would be affected by the device change. The application will not be supplied with the new device type but may query for change in resolution and possible other attributes.

**Formats** A procedure which will return document formats so that the Print Manager may display the various paper types to the user for hand feed selections. The call interface for this routine should look like the following:

```
Boolean Formats( Handle *hFormatRec, Integer iWhichFormat );
```

The Print Manager will call this routine repeatedly incrementing the iWhichFormat value until the function returns false indicating that all formats have been delivered. The application must be able to access all document Format Records sequentially. The Print Manager will call the Formats function the first time with iWhichFormat equal to 1. The application should set hFormatRec equal to the first Format Record in the document. Subsequent calls will request subsequent Format Records. The application should continue returning a function value of true and a handle to the next Format Record until all records iWhichFormat exceeds the number of unique formats in the document. If the application does not support by Page formatting the pFormats parameter should be nil. This call will only be made if the user indicates that he wishes to feed some document pages manually and chooses the dialog panel which allows him to specify the formats to be fed manually.

*the function result:*

- noErr** The user exits the dialog with OK. The printer does not change.
- cancel** The user exits the dialog with Cancel.
- badPort** pPrGrafPort is corrupt.
- badJobRec** hJobRec is nil or of the wrong size.

```

struct {
    char          *panelName;
    EDialogType  eWhichDialog;
    Boolean       fColorIcon;
    Handle        hUserStore;
    Handle        hPanelIcon;
    Int16         iIconID;
    Handle        hPanelDitl;
    Int16         iDitlID;
    Handle        hPanelCode;
    Int16         iCodeID;
    Handle        hFilterProc;
    Int16         iFilterID;
    Handle        hPrintStore;
    some reference for help information;
} TPanelInfo, *TPanelInfo, **THPanelInfo;

```

```

PrintError PrAddPanel(THPanelInfo hPanelInfo,
                    TPrGrafPort pPrGrafPort);

```

*what happens:*

This call signals the Print Manager that the application wants to add a panel to a Print Dialog. The hPanelInfo supplies all the necessary information to make this possible. The application has the choice of 1) sending handles to resources or 2) sending resource ids for certain parameters. If the resource handles are nil, the Print Manager will get the resources when the dialog is to be displayed and release them after the dialog is dismissed. If the handles are not nil, it is the applications responsibility to release them. The passed information will be stored by the Print Manager until the corresponding dialog is executed. After dialog dismissal, the Print Manager will no longer keep this information. This will free the memory taken up by this information.

*when to call:*

This call should be made each time the application wants to add a panel to a dialog. Multiple panels may be added to a single dialog. This function should be called after the Print Manager is initialized through PrStart and before calling the associated dialog routine. This is an optional call.



*the parameters:*

hPanelInfo =

|              |                                                                                                                                                                                                       |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| panelName    | the name of the panel to be displayed with the icon                                                                                                                                                   |
| eWhichDialog | an enumerated type designating which dialog the panel is to be added to. Values include eJob, eDialog, eByPage.                                                                                       |
| fColorIcon   | a boolean value indicating whether the passed icon is color or not. If true, the icon is assumed to be in color.                                                                                      |
| hUserStore   | a handle to storage for the application to be used in any capacity the application finds convenient.                                                                                                  |
| hPanelIcon   | a handle to an icon for the panel, may be nil.                                                                                                                                                        |
| iIconID      | an id into a resource file for the panel icon to be used if hPanelIcon is nil.                                                                                                                        |
| hPanelDitl   | a handle to an item list for the panel, may be nil.                                                                                                                                                   |
| iDitlID      | an id into a resource file for the panel DITL to be used if hPanelDitl is nil. Note that the (0,0) location is situated at the top left corner of the dialog panel (as opposed to the dialog window). |
| hPanelCode   | a handle to a code resource to manage the panel, may be nil. The code resource should have the following calling sequence:                                                                            |

```
Integer DoPrintDialog( Integer iMessage,
                      Integer iItemHit,
                      EventRecord *theEvent,
                      Handle hStore,
                      DialogPtr pDialog );
```

where:

iMessage = a message number referring to the event that just happened. The application would be responsible for responding to all events occurring while in their custom panel with the exception of app and activate events.

The following values will be defined:

|             |      |                                                                     |
|-------------|------|---------------------------------------------------------------------|
| initDialog  | = 1; | this is to allow for setting up of user items                       |
| openDialog  | = 2; | draw yourself - the user has selected the application icon          |
| eventDialog | = 3; | an event occurs while in the application panel                      |
| closeDialog | = 4; | user clicks another icon                                            |
| endDialog   | = 5; | dialog dismissed - the user hits OK or Cancel to dismiss the dialog |

`iItemHit` = in the case of an event involving an enabled dialog item, this parameter contains the item number hit.

`theEvent` = a pointer to the corresponding event record. This should only be of interest for an `eventDialog` message.

`hStore` = private storage for the application to use as it needs.

`pDialog` = a pointer to the dialog.

`CodeID` an id into a resource file for the code resource if `hPanelCode` is nil.

`hFilterProc` a handle to a code resource to filter events for the panel code, may be nil. The code resource should have the following calling sequence:

```
Boolean Filter( Integer itemHit,
                EventRecord *theEvent,
                Handle hStore,
                DialogPtr pDialog );
```

The function parameters share the same definition as those for the other code resource parameter. This function will operate as the filter function described under Modal Dialog in the Dialog Manager. A return value of TRUE indicates that the application has handled the event. A return value of FALSE indicates that the event should be handled by the Print Manager. The Print Manager will either be calling Modal Dialog or emulating its functionality.

`iFilterID` an id into a resource file for the filter resource if `hFilterProc` is nil.

`hPrintStore` This handle contains all the information the printing software needs to maintain the state of the current job. This is the same handle that was allocated at `PrStart`.

*the function result:*

|                            |                                                           |
|----------------------------|-----------------------------------------------------------|
| <code>noErr</code>         | all is well - user exits with OK.                         |
| <code>cancel</code>        | all is well - user exits with Cancel.                     |
| <code>memFullErr</code>    | there is not enough memory to save the panel information. |
| <code>badPort</code>       | <code>pPrGrafPort</code> is corrupt.                      |
| <code>badName</code>       | panel name is reserved.                                   |
| <code>badDialogType</code> | unknown dialog type specified.                            |
| <code>badIcon</code>       | icon name is reserved                                     |

## Customizing the Style Dialog.

```
PrintError PrAddOrientation( Handle      hOrientIcon,
                             Integer     iOrientIconID,
                             char        *orientName,
                             TPPrGrafPort pPrGrafPort );
```

*what happens:*

This call will allow the application to add a custom orientation to the Basic panel of the Style Dialog. The new orientation will be specified with an icon and an orientation name.

*when to call:*

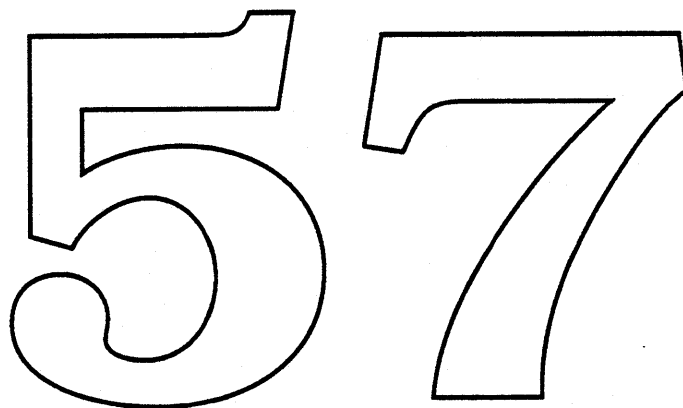
This call should be made each time the application wants to add a custom orientation to a Style dialog. This function should be called after the Print Manager is initialized through PrStart. This custom orientation will be displayed in the next Style. The information concerning the custom orientation will be stored by the Print Manager until the completion of that Style call. If the user selects the custom orientation, the information will be stored with the Format Record. If this custom orientation is selected by the user it will be up to the application to send the document information in such a way as to support this orientation at print time. This is an optional call.

*the parameters:*

- hOrientIcon** a handle to an icon to display with the orientation. This parameter may be nil.
- iOrientIconID** if hOrientIcon is nil, the resource id of the icon for the custom orientation to be displayed. The Print Manager will get the resource before the dialog is displayed and release it after the dialog ends.
- orientName** name of the custom orientation. May not be nil.
- pPrGrafPort** This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

|            |                                                              |
|------------|--------------------------------------------------------------|
| noErr      | all is well.                                                 |
| memFullErr | there is not enough memory to add the custom orientation.    |
| noRoom     | there is no more real estate left for displaying orientation |
| badName    | orientName is nil.                                           |
| badPort    | pPrGrafPort is corrupt.                                      |



```
PrintError PrSetOrientation( EOrientType  eOrientation,
                             char          *orientName,
                             Boolean       kLockIt,
                             Handle        hFormatRec,
                             TPPrGrafPort  pPrGrafPort );
```

*what happens:*

The Basic panel of the Style dialog will display a default orientation. The PrSetOrientation call allows the application to specify which orientation will be defaulted to. This default orientation is specified through an enumerated type (landScape, portrait, custom). If the custom value is sent, the orientation is specified through a name.

If the custom orientation name does not exist when a custom orientation is specified, an error is returned and no updating will be done to the Format record. If an orientation type is specified that is not within the known set, an error is returned and no updating will be done to the Format Record. If the user has previously selected an orientation for this format, the user's choice will override the passed orientation setting.

*when to call:*

This call should be made each time the application wants to set and/or lock a default orientation in a Style dialog. This function should be called after the Print Manager is initialized through PrStart and the application has created a valid Format Record and before the corresponding Style dialog is displayed. This is an optional call.

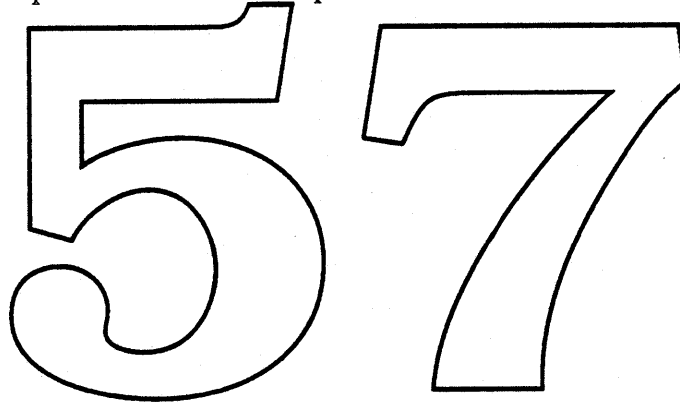
*the parameters:*

- eOrientation** an enumerated type designating the default setting. Values for this field include landScape, portrait and custom. If custom is passed, orientName must not be an empty string.
- orientName** The name corresponding to the custom orientation. May be empty if no custom type is specified.
- kLockIt** A boolean which should be set to true if the application does not want the user to be able to select another orientation. Locking this field is highly discouraged.

- hFormatRec** A handle to a valid Format Record into which the passed orientation will be set.
- pPrGrafPort** This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

- noErr** all is well.
- lockedOut** the user selection cannot be overridden.
- badOrientation** custom orientation selected and orientName is nil or unknown to Print Manager or unknown orientation type.
- badFormat** Format Record is nil.
- badPort** pPrGrafPort is corrupt.



```
PrintError PrGetOrientation( EOrientType *eOrientation,
                             char         *orientName,
                             Boolean      *kLocked,
                             Handle       hFormatRec,
                             TPrGrafPort pPrGrafPort );
```

*what happens:*

The Basic panel of the Style dialog will display icons in order for the user to select an orientation for the document or document section. The PrGetOrientation call allows the application to query for the currently selected orientation for the passed format. This orientation is specified through an enumerated type (landScape, portrait, custom). If the current orientation is a custom value, the orientation name is returned.

*when to call:*

This call should be made each time the application wants to query for a format orientation. This function should be called after the Print Manager is initialized through PrStart and the application has created a valid Format Record. This is an optional call.

*the parameters:*

- eOrientation** an enumerated type designating the default setting. Values for this field include landScape, portrait and custom. If custom is set, orientName will not be an empty string.
- orientName** The name corresponding to the custom orientation. Will be empty if no custom type is specified.
- kLocked** A boolean which returns true if the application has locked the orientation setting. Locking this field is highly discouraged.
- hFormatRec** A handle to a valid Format Record from which the orientation will be returned.
- pPrGrafPort** This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

|           |                         |
|-----------|-------------------------|
| noErr     | all is well.            |
| badFormat | Format Record is nil.   |
| badPort   | pPrGrafPort is corrupt. |

57



```

struct {
    float    fPaperWidth;
    float    fPaperHeight;
    float    fTopBound;
    float    fLeftBound;
    float    fBottomBound;
    float    fRightBound;
    EUnits   eUnits;
    char     *paperTypeName;

    } TPaperType, *TPPaperType, **THPaperType;

```

```

PrintError PrAddPaperType( THPaperType hPaperType,
                          TPPaperType pPrGrafPort );

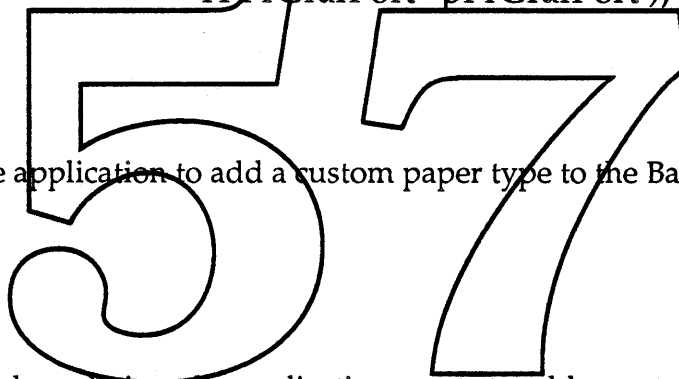
```

*what happens:*

This call will allow the application to add a custom paper type to the Basic panel of the Style Dialog.

*when to call:*

This call should be made each time the application wants to add a custom paper type to a Style dialog. This function should be called after the Print Manager is initialized through PrStart. This custom paper type will be displayed in the next Style dialog. The information concerning the custom paper type will be stored by the Print Manager until the completion of that Style call. If the user selects the custom paper type, the information will be stored with the corresponding Format Record. This is an optional call.



*the parameters:*

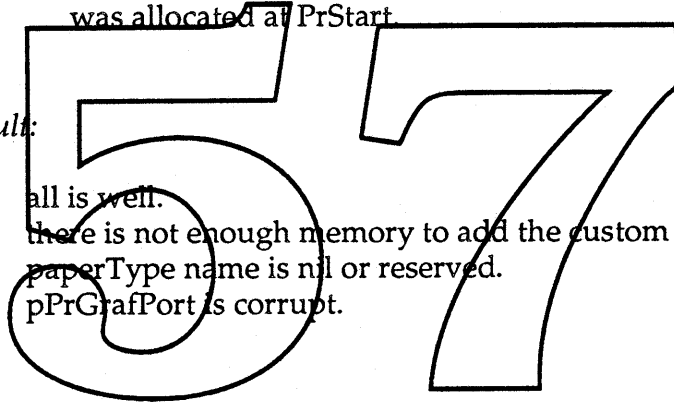
hPaperType =

|               |                                                                                                                                    |
|---------------|------------------------------------------------------------------------------------------------------------------------------------|
| fPaperWidth   | the width of the paper                                                                                                             |
| fPaperHeight  | the height of the paper                                                                                                            |
| fTopBound     | the depth of the top bound of the imaging area                                                                                     |
| fLeftBound    | the width of the left bound of the imaging area                                                                                    |
| fBottomBound  | the depth of the bottom bound of the imaging area                                                                                  |
| fRightBound   | the width of the right bound of the imaging area                                                                                   |
| eUnits        | an enumerated type representing the units the above measurements are made with. Values include inched, centimeters, points, picas. |
| paperTypeName | the name of the custom paper type                                                                                                  |

pPrGrafPort This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart

*the function result:*

|            |                                                          |
|------------|----------------------------------------------------------|
| noErr      | all is well.                                             |
| memFullErr | there is not enough memory to add the custom paper type. |
| badName    | paperType name is nil or reserved.                       |
| badPort    | pPrGrafPort is corrupt.                                  |



```
PrintError PrSetPaperType( char      *paperTypeName,
                          Boolean    kLockit,
                          Handle     hFormatRec,
                          TPrGrafPort pPrGrafPort );
```

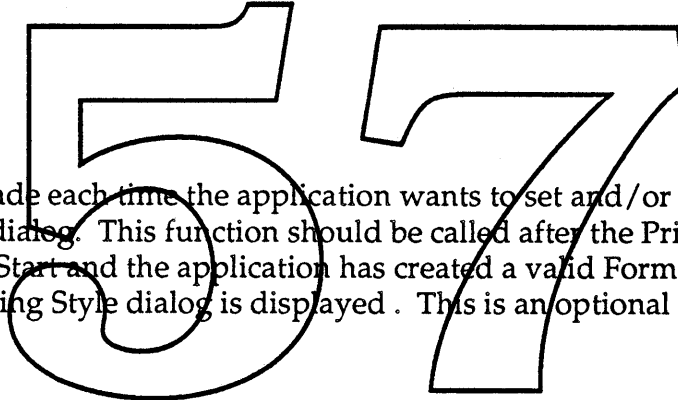
*what happens:*

The Basic panel of the Style dialog will display a default paper type if the user has not selected one. The PrSetPaperType call allows the application to specify which paper type will be defaulted to. This default paper type is specified through a papertype name.

If the paper type name does not exist, i.e., the Print Manager doesn't know about it, an error is returned and no updating will be done to the Format record. If the user has previously selected a paper type for this format, the user's choice will override the passed paper type setting.

*when to call:*

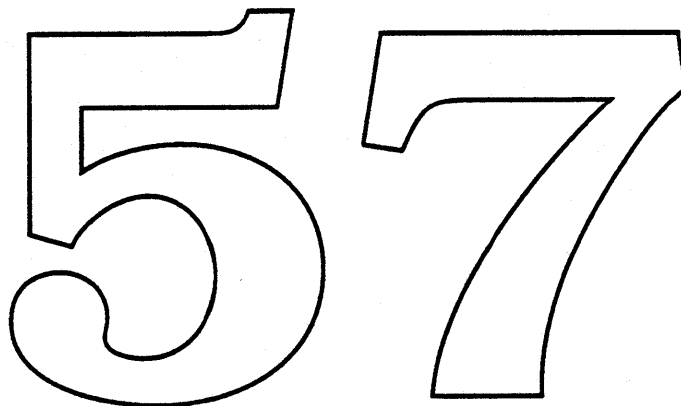
This call should be made each time the application wants to set and/or lock a default paper type in a Style dialog. This function should be called after the Print Manager is initialized through PrStart and the application has created a valid Format Record and before the corresponding Style dialog is displayed. This is an optional call.

*the parameters:*

|               |                                                                                                                                                                    |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| paperTypeName | The name corresponding to the paper type.                                                                                                                          |
| kLockIt       | A boolean which should be set to true if the application does not want the user to be able to select another paper type. Locking this field is highly discouraged. |
| hFormatRec    | A handle to a valid Format Record into which the default paper type will be set.                                                                                   |
| pPrGrafPort   | This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.  |

*the function result:*

|           |                                                     |
|-----------|-----------------------------------------------------|
| noErr     | all is well.                                        |
| badName   | paper type name is nil or unknown to Print Manager. |
| badFormat | Format Record is nil.                               |
| badPort   | pPrGrafPort is corrupt.                             |
| lockedOut | the user selection cannot be overridden.            |



```
PrintError PrGetPaperType( THPaperType *hPaperType
                          Boolean      *kLocked,
                          Handle      hFormatRec,
                          TPrGrafPort pPrGrafPort );
```

*what happens:*

The Basic panel of the Style dialog will display a list of paper types for the user to choose among for the document or document section. The PrGetPaperType call allows the application to query for which paper type is currently selected for the passed format. This paper type is specified through name, paper dimensions and margins.

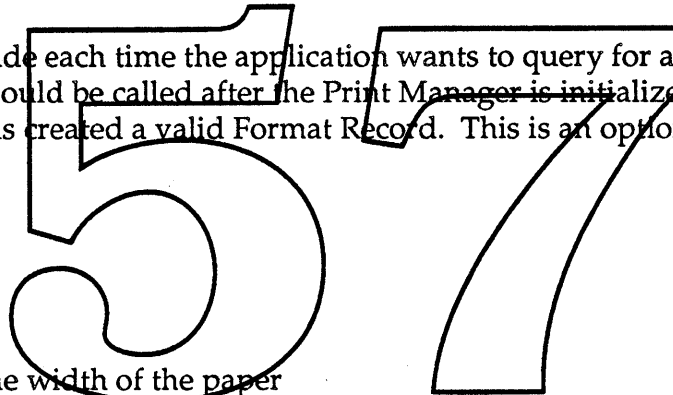
*when to call:*

This call should be made each time the application wants to query for a format paper type. This function should be called after the Print Manager is initialized through PrStart and the application has created a valid Format Record. This is an optional call.

*the parameters:*

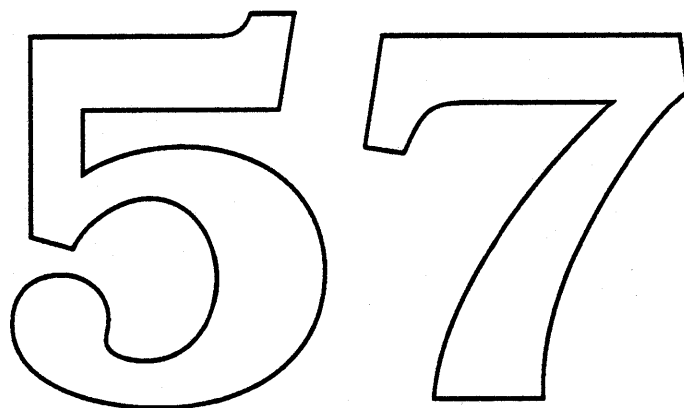
hPaperType =

|               |                                                                                                                                                                   |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| fPaperWidth   | the width of the paper                                                                                                                                            |
| fPaperHeight  | the height of the paper                                                                                                                                           |
| fTopBound     | the depth of the top bound of the imaging area                                                                                                                    |
| fLeftBound    | the width of the left bound of the imaging area                                                                                                                   |
| fBottomBound  | the depth of the bottom bound of the imaging area                                                                                                                 |
| fRightBound   | the width of the right bound of the imaging area                                                                                                                  |
| eUnits        | an enumerated type representing the units the above measurements are made with. Values include inched, centimeters, points, picas.                                |
| paperTypeName | the name of the custom paper type                                                                                                                                 |
| kLocked       | A boolean which returns true if the application has locked the paper type setting. Locking this field is highly discouraged.                                      |
| hFormatRec    | A handle to a valid Format Record into which the default paper type will be set.                                                                                  |
| pPrGrafPort   | This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart. |



*the function result:*

|           |                         |
|-----------|-------------------------|
| noErr     | all is well.            |
| badFormat | Format Record is nil.   |
| badPort   | pPrGrafPort is corrupt. |



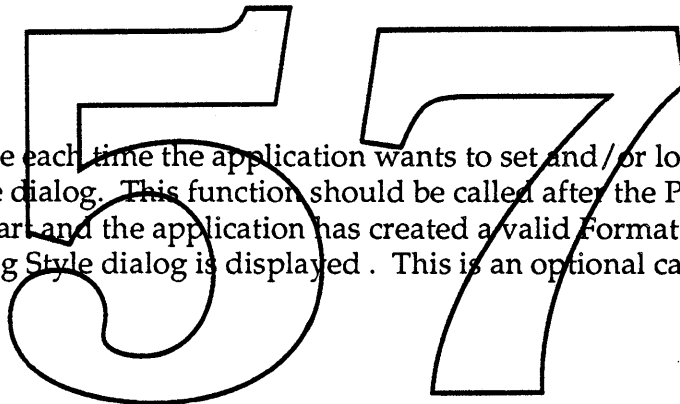
```
PrintError PrSetScaling( Int16      iScale,
                        Boolean     kLockIt,
                        Handle      hFormatRec,
                        TPrGrafPort pPrGrafPort );
```

*what happens:*

The Basic panel of the Style dialog will display a default scaling setting if the user has not selected a setting. The PrSetScaling call allows the application to specify which scaling setting will be defaulted to. This default scaling setting is specified through an integer value. The scaling works as follows: a value less than 100% will produce a larger page size. This will allow the user to fit more information on a page. A value greater than 100% will produce a smaller page size. This will reduce the amount of information the user may fit on a page. Resolution is not effected.

*when to call:*

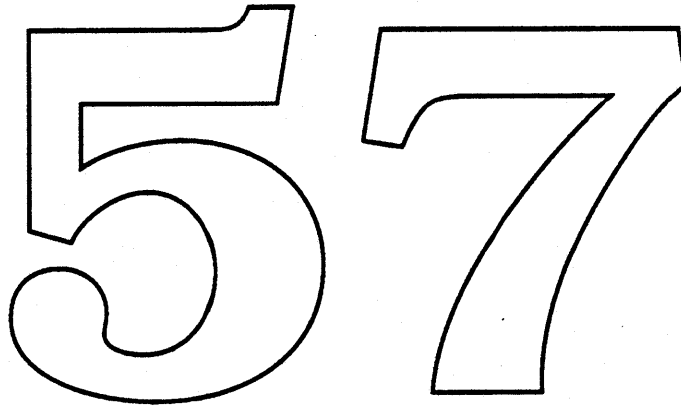
This call should be made each time the application wants to set and/or lock a default scaling setting in a Style dialog. This function should be called after the Print Manager is initialized through PrStart and the application has created a valid Format Record and before the corresponding Style dialog is displayed. This is an optional call.

*the parameters:*

- iScale**            The amount to scale the page by. Must be positive.
- kLockIt**            A boolean which should be set to true if the application does not want the user to be able to select a different scaling setting. Locking this field is highly discouraged.
- hFormatRec**        A handle to a valid Format Record into which the default scale setting will be set.
- pPrGrafPort**        This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

|           |                                          |
|-----------|------------------------------------------|
| noErr     | all is well.                             |
| badNumber | iScale is a negative number.             |
| badFormat | Format Record is nil.                    |
| badPort   | pPrGrafPort is corrupt.                  |
| lockedOut | the user selection cannot be overridden. |





```
PrintError PrDisableScaling( Handle      hFormatRec,
                              TPPrGrafPort pPrGrafPort );
```

*what happens:*

This call allows the application to disable the scaling feature provided in the document and Style dialogs. See PrSetScaling and PrGetScaling for more information on the scaling feature.

*when to call:*

This call should be made if an application wishes to replace the print zoom feature with one of its own. This function should be called after the Print Manager is initialized through PrStart and the application has created a valid Format Record and before the corresponding Style dialog is displayed. This is an optional call.

*the parameters:*

**hFormatRec** A handle to a valid Format Record into which the zoom disable information will be stored.

**pPrGrafPort** This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

**noErr** all is well.

**badFormat** Format Record is nil.

**badPort** pPrGrafPort is corrupt.

**lockedOut** the user selection cannot be overridden.

```
PrintError PrGetScaling( Int16      *iScale,
                        Boolean     *kLocked,
                        Handle      hFormatRec,
                        Handle      hPrintStore );
```

*what happens:*

The Basic panel of the Style dialog will display a field to allow the user to select a scaling factor for the document or document selection. The PrGetScaling call allows the application to query for the current scaling setting for the passed format. This setting is specified through an integer.

*when to call:*

This call should be made each time the application wants to query for a format scaling setting. This function should be called after the Print Manager is initialized through PrStart and the application has created a valid Format Record. This is an optional call.

*the parameters:*

**iScale**      The amount to reduce or enlarge the page by.

**kLocked**     A boolean which returns true if the application has locked the scaling setting. Locking this field is highly discouraged.

**hFormatRec**   A handle to a valid Format Record from which the scaling setting will be taken.

**pPrGrafPort**   This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

**noErr**        all is well.

**badFormat**    Format Record is nil.

**badPort**      pPrGrafPort is corrupt.

## Customizing the Job Dialogs.

**PrintError PrSetCustomPageRange(**

```

char          *startPage,
char          *endPage,
Boolean      parsePageRange(char *first, char *last),
Handle       hJobRec,
TPPrGrafPort pPrGrafPort

);

```

*what happens:*

This call allows the application to customize the Page Range fields in the Basic Print Dialog. The Print Manager stores the passed information until the next invocation of the Print Dialog. The application supplies a start string representing the first page "number" of the document and an ending string representing the last page "number" of the document and a procedure to parse and restrict the user input for the from and to fields.

*when to call:*

This call should be made each time the application wants to customize the Page Range information for the Print Dialog. This function should be called after the Print Manager is initialized through PrStart and the application has created a valid Job Record and before the corresponding Job dialog is displayed. This is an optional call.

*the parameters:*

|                |                                                                                                                                                                   |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| startPage      | string representing starting page number of document                                                                                                              |
| endPage        | string representing ending page number of document                                                                                                                |
| parsePageRange | procedure to parse user entries in From and To boxes. Should return true if values specify a valid range, false otherwise.                                        |
| hJobRec        | handle to valid Job record for the document                                                                                                                       |
| pPrGrafPort    | This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart. |

*the function result:*

|           |                                          |
|-----------|------------------------------------------|
| noErr     | All is well.                             |
| badStore  | hPrintStore is corrupted.                |
| badPort   | pPrGrafPort is corrupt.                  |
| lockedOut | the user selection cannot be overridden. |

57

**PrintError PrGetCustomPageRange(**

```

char      *startPage,
char      *endPage,
Handle    hJobRec,
TPPrGrafPort pPrGrafPort );

```

*what happens:*

This call allows the application to query for the custom page range the user has selected. The string values reflect the user's choice. This information should inform the application as to which pages it must send to the Print Manager. Pages that fall outside the user specified range should not be sent to the Print Manager.

*when to call:*

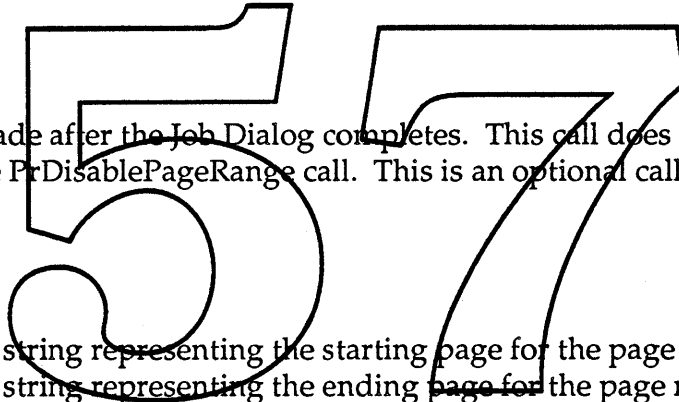
This call should be made after the Job Dialog completes. This call does not apply if the application makes the PrDisablePageRange call. This is an optional call.

*the parameters:*

|             |                                                                                                                                                                   |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| startPage   | a string representing the starting page for the page range                                                                                                        |
| endPage     | a string representing the ending page for the page range                                                                                                          |
| hJobRec     | handle to valid Job record for the document                                                                                                                       |
| pPrGrafPort | This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart. |

*the function result:*

|          |                           |
|----------|---------------------------|
| noErr    | All is well.              |
| badStore | hPrintStore is corrupted. |
| badPort  | pPrGrafPort is corrupt.   |



```
PrintError PrDisablePageRange( char      *replacementString,
                                Handle     hJobRec,
                                TPrGrafPort pPrGrafPort );
```

*what happens:*

This call signals the Print Manager that the application has dealt with page range selection inside the application. The Page Range fields will be disabled in the Print Dialog. A string supplied by the application will be displayed instead. This string should give an indication of the pages to be printed.

*when to call:*

This call should be made each time the application wants to disable the Page Range information for the Print Dialog. This function should be called after the Print Manager is initialized through PrStart and the application has created a valid Job Record and before the corresponding Job dialog is displayed. This is an optional call.

*the parameters:*

replacementString string representing pages user has selected to print  
hJobRec handle to valid Job record for the document  
pPrGrafPort This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

|           |                                          |
|-----------|------------------------------------------|
| noErr     | All is well.                             |
| badPort   | pPrGrafPort is corrupt.                  |
| lockedOut | the user selection cannot be overridden. |
| badJobRec | hJobRec is nil or of the wrong size.     |

```
PrintError PrGetPageRange( Int16      *iStartPage,
                          Int16      *iEndPage,
                          Handle      hJobRec,
                          TPrGrafPort pPrGrafPort );
```

*what happens:*

This call allows the application to query for the page range the user has selected through the default print dialog page range mechanism. This information should inform the application as to which pages it must send to the Print Manager. Pages that fall outside the user specified range should not be sent to the Print Manager. This call and the PrDisablePageRange call are mutually exclusive.

*when to call:*

This call should be made after the Job Dialog completes. This call does not apply if the application makes the PrDisablePageRange call. This is an optional call.

*the parameters:*

|             |                                                                                                                                                                   |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| iStartPage  | the first page the user has selected to print                                                                                                                     |
| iEndPage    | the last page the user has selected to print                                                                                                                      |
| hJobRec     | handle to valid Job record for the document                                                                                                                       |
| pPrGrafPort | This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart. |

*the function result:*

|           |                                      |
|-----------|--------------------------------------|
| noErr     | All is well.                         |
| badPort   | pPrGrafPort is corrupt.              |
| badJobRec | hJobRec is nil or of the wrong size. |

```
PrintError PrSetReduction( Int16      iReduce,
                          Boolean     kLockIt,
                          Handle      hJobRec,
                          TPrGrafPort pPrGrafPort );
```

*what happens:*

The Basic panel of the Job dialog will display a default reduction/enlargement setting if the user has not picked a setting. The PrSetReduction call allows the application to specify which reduction/enlargement setting will be defaulted to. This default setting is specified through an integer value. The image on the document pages is scaled up or down according to the reduction setting. The page size remains the same. Resolution is not effected.

*when to call:*

This call should be made each time the application wants to set and/or lock a default reduction/enlargement setting in a Job dialog. This function should be called after the Print Manager is initialized through PrStart and the application has created a valid Job Record and before the corresponding Job dialog is displayed. This is an optional call.

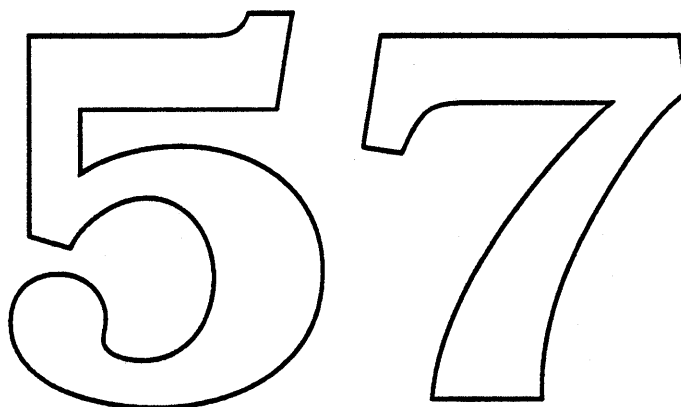
*the parameters:*

- iReduce*      The amount to scale the image by. Must be positive.
- kLockIt*      A boolean which should be set to true if the application does not want the user to be able to select a different reduction/enlargement setting. Locking this field is highly discouraged.
- hJobRec*      A handle to a valid Job Record into which the passed reduction value will be set.
- pPrGrafPort*   This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.



*the function result:*

|           |                                          |
|-----------|------------------------------------------|
| noErr     | all is well.                             |
| badNumber | iReduce is a negative number.            |
| badJobRec | Job Record is nil.                       |
| badPort   | pPrGrafPort is corrupt.                  |
| lockedOut | the user selection cannot be overridden. |



```
PrintError PrGetReduction( Int16      *iReduce,
                          Boolean     *kLocked,
                          Handle      hJobRec,
                          TPPrGrafPort pPrGrafPort );
```

*what happens:*

The user may select a reduction/enlargement setting for the document in the Basic panel of the Job dialog. The PrGetReduction call allows the application to query for the current value of this setting. This setting is specified through an integer.

*when to call:*

This call should be made each time the application wants to query for a document reduction/enlargement setting. This function should be called after the Print Manager is initialized through PrStart and the application has created a valid Job Record. This is an optional call.

*the parameters:*

**iReduce**      The amount to reduce or enlarge the image by.

**kLocked**      A boolean which returns true if the application has locked the reduction/enlargement setting. Locking this field is highly discouraged.

**hJobRec**      A handle to a valid Job Record from which the current setting will be taken.

**pPrGrafPort**   This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

**noErr**          all is well.

**badJobRec**      Job Record is nil.

**badPort**        pPrGrafPort is corrupt.

```
PrintError PrSetStackingOrder( Boolean      kFirstPageOnTop,
                               Boolean      kLockIt,
                               Handle       hJobRec,
                               TPPrGrafPort pPrGrafPort );
```

*what happens:*

The Job Dialog contains a panel where the user may specify stacking order. The user may specify that the first page of the document should end up on the top of the document (generally printing back to front order) OR that the last page end up on the top of the document (generally printing front to back order). This call allows the application to set and potentially lock the default setting.

Because all jobs are spooled in Ginsu, the Print Manager can print pages in any order. However, if there is not enough disk space to spool the entire job, an ordering requiring that the last page be printed first cannot be accommodated without support from the application. If the user specifies a stacking order requiring a document to be printed last page first, the application may support this feature by transmitting the pages in that order. See the PrSendingLastFirst call for more information.

*when to call:*

This call should be made each time the application wants to set and/or lock the default stacking order setting in a Job dialog. This function should be called after the Print Manager is initialized through PrStart and the application has created a valid Job Record and before the corresponding Job dialog is displayed. This is an optional call.

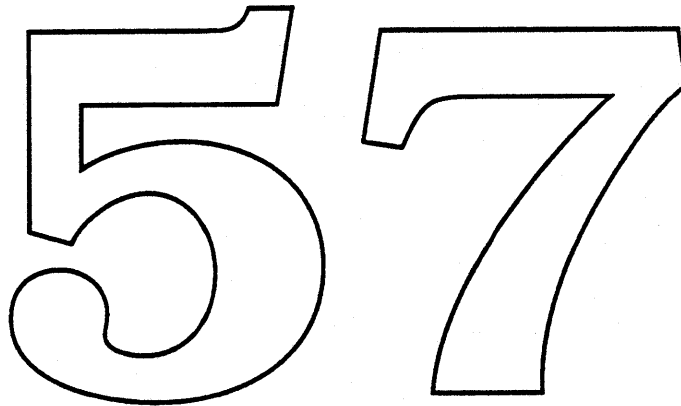
*the parameters:*

- kFirstPageOnTop** A boolean indicating the stacking order for the job. If true, pages will be stacked with the first page ending up on top. A value of false will provide a document with pages in reverse order.
- kLocked** A boolean which, if true, locks the passed stacking order so that the user may not choose a different stacking order. Locking this field is highly discouraged.
- hJobRec** A handle to a valid Job Record into which the passed value will be set.

pPrGrafPort      This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

|           |                                          |
|-----------|------------------------------------------|
| badStore  | all is well.                             |
| badJobRec | hJobRec is nil or of the wrong size.     |
| badPort   | pPrGrafPort is corrupt.                  |
| lockedOut | the user selection cannot be overridden. |

The image shows two large, hollow outline numbers, '5' and '7', positioned side-by-side. The '5' is on the left and the '7' is on the right. Both numbers are rendered in a simple, bold, sans-serif style with a consistent line thickness.

```
PrintError PrGetStackingOrder( Boolean *kFirstPageOnTop,
                               Boolean *kLocked,
                               Handle hJobRec,
                               TPrGrafPort pPrGrafPort );
```

*what happens:*

The Job Dialog contains a panel where the user may specify stacking order. The user may specify that the first page of the document should end up on the top of the document (generally printing back to front order) OR that the last page end up on the top of the document (generally printing front to back order). This call allows the application to query for the stacking order setting the user has selected.

*when to call:*

This call should be made each time the application wants to query for a document stacking order setting. This function should be called after the Print Manager is initialized through PrStart and the application has created a valid Job Record. This is an optional call.

*the parameters:*

- kFirstPageOnTop     A boolean indicating the stacking order for the job. If true, pages will be stacked with the first page ending up on top. A value of false will provide a document with pages in reverse order.
- kLocked             A boolean which returns true if the application has locked the stacking order setting. Locking this field is highly discouraged.
- hJobRec             A handle to a valid Job Record from which the current setting will be taken.
- pPrGrafPort         This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

- noErr                all is well.
- badJobRec           Job Record is nil.
- badPort             pPrGrafPort is corrupt.

```
PrintError PrSendingLastFirst( Handle      hJobRec,
                                TPPrGrafPort pPrGrafPort );
```

*what happens:*

The Job Dialog contains a panel where the user may specify stacking order. The user may specify that the first page of the document should end up on the top of the document (generally printing back to front order) OR that the last page end up on the top of the document (generally printing front to back order).

Because all jobs are spooled in Ginsu, the Print Manager can print pages in any order. However, if there is not enough disk space to spool the entire job, an ordering requiring that the last page be printed first cannot be done. If the user specifies a stacking order requiring a document to be printed last page first, the application can guarantee this feature by transmitting the pages in that order. In order to support the last page first ordering, an application must use this call to inform the Print Manager that the application intends to send the pages in reverse order. If this call is not made, the order assumed is sequential from the start page the user has specified in Page Range.

*when to call:*

This call should be made after the Job Dialog has been executed and before the PrStartDoc call if the user has specified a last page on top ordering and the application wants to support this by transmitting the selected pages in reverse order. This is an optional call.

*the parameters:*

|             |                                                                                                                                                                   |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| hJobRec     | A handle to a valid Job Record which will store this information.                                                                                                 |
| pPrGrafPort | This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart. |

*the function result:*

|           |                                      |
|-----------|--------------------------------------|
| noErr     | all is well.                         |
| badJobRec | hJobRec is nil or of the wrong size. |
| badPort   | pPrGrafPort is corrupt.              |

```
PrintError PrSetCollation( Boolean      kCollationOn,
                          Boolean      kLockIt,
                          Handle      hJobRec,
                          TPrGrafPort pPrGrafPort );
```

*what happens:*

The Job Dialog contains a panel where the user may specify that document copies be collated. This call allows the application to set and potentially lock the default setting.

*when to call:*

This call should be made each time the application wants to set and/or lock the default collation state in a Job dialog. This function should be called after the Print Manager is initialized through PrStart and the application has created a valid Job Record and before the corresponding Job dialog is displayed. This is an optional call.

*the parameters:*

**kCollationOn** A boolean indicating whether collation is to be turned on. If kCollation is true, collation will be turned on. Otherwise, it will be turned off.

**kLocked** A boolean which, if true, locks the passed collation state so that the user may not choose a different stacking order. Locking this field is highly discouraged.

**hJobRec** A handle to a valid Job Record into which the passed value will be set.

**pPrGrafPort** This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

**badStore** all is well.

**badJobRec** hJobRec is nil or of the wrong size.

**badPort** pPrGrafPort is corrupt.

**lockedOut** the user selection cannot be overridden.

```
PrintError PrGetCollation( Boolean      *kCollationOn,
                          Boolean      *kLocked,
                          TPrGrafPort  pPrGrafPort );
```

*what happens:*

The Job Dialog contains a panel where the user may specify that document copies be collated. This call allows the application to query for the state of the collation setting.

*when to call:*

This call should be made each time the application wants to query for collation state. This function should be called after the Print Manager is initialized through PrStart and the application has created a valid Job Record. This is an optional call.

*the parameters:*

**57**

**kCollationOn** A boolean which returns true if the user has selected collation, false otherwise.

**kLocked** A boolean which returns true if the application has locked the stacking order setting. Locking this field is highly discouraged.

**hJobRec** A handle to a valid Job Record from which the current setting will be taken.

**pPrGrafPort** This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

**noErr** all is well.

**badJobRec** Job Record is nil.

**badPort** pPrGrafPort is corrupt.



The following calls pertain to the Printing Loop.

```
PrintError PrStartDoc( TPrGrafPort  pPrGrafPort,
                      char           *documentName,
                      Handle         hJobRec );
```

*what happens:*

This call replaces the PrOpenDoc call in the old Printing Manager. This call initializes a printing grafPort for use in printing a document, makes it the current port, and returns a pointer to it. Every call to PrStartDoc must be balanced with a call to PrEndDoc. Note that there is no provision in this new call for the application to provide its own IO buffer. Buffering is treated dynamically in the new print architecture so it no longer makes sense for the application to provide a static buffer.

*when to call:*

This function should be called after the Print Manager is initialized through PrStart and the application has a valid Job Record. This call initiates print spooling.

*the parameters:*

pPrGrafPort      This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

documentName    a string to identify the document. This name will be used to help the user track the status of the job.

hJobRec          a handle to a Job record which should have already been validated.

*the function result:*

|            |                                                       |
|------------|-------------------------------------------------------|
| noErr      | all is well.                                          |
| memFullErr | there is not enough memory to allocated the grafPort. |
| badPort    | pPrGrafPort is corrupt.                               |
| badJobRec  | hJobRec is nil or of the wrong size.                  |

```
PrintError PrEndDoc( TPrGrafPort pPrGrafPort );
```

*what happens:*

This call replaces the PrCloseDoc call in the old Printing Manager. This call closes the grafPort. Note that in Ginsu, the application does not need to call PrPicFile to print the spool file. This is either handled in the background by PrintMonitor or handled by the printing software in the foreground.

*when to call:*

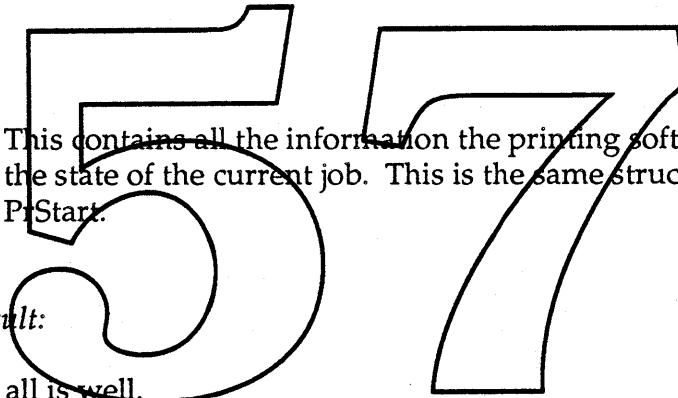
This call ends the print loop for the document. It should be called after all pages to be printed have been sent via PrStartPage/PrEndPage pairs.

*the parameters:*

pPrGrafPort This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

noErr all is well.  
badPort pPrGrafPort is corrupt.



```
PrintError PrStartPage(TPPrGrafPort pPrGrafPort,
                       Rect          *pPageFrame,
                       Handle         hFormatRec );
```

*what happens:*

This call replaces the PrOpenPage call in the old Print Manager. This call begins a new page.

In Ginsu, we offer support for by Page formatting. The Format Record parameter in this call designates which format applies to the current page. If the application does not support by page formatting or does support by page formatting but is sending a document with a single format, performance will be improved if the first call to PrStartPage contains a handle to that format and subsequent calls contain a nil handle for that format.

In Ginsu, we also offer support which allows the application to only send the Print Manager the pages it really wants to print. So each call to PrStartPage should be made with the intent that the page is to be printed.

*when to call:*

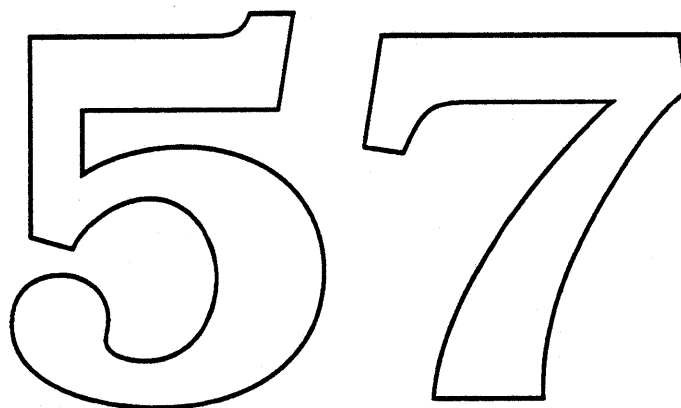
This call should be made after the call to PrStartDoc. Every call to PrStartPage must be balanced with a call to PrEndPage.

*the parameters:*

- pPrGrafPort This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.
- pPageFrame a pointer to a rectangle to scale the picture to. If no scaling is desired, this parameter should be nil.
- hFormatRec a handle to a Format record. This clues the printing software in on how to format the current page.

*the function result:*

|              |                                                                     |
|--------------|---------------------------------------------------------------------|
| noErr        | all is well.                                                        |
| badFormatRec | hFormatRec is never specified or of the wrong size or printer type. |
| badPort      | pPrGrafPort is corrupt.                                             |



**PrintError PrEndPage( TPrGrafPort pPrGrafPort );**

*what happens:*

This call replaces the PrClosePage call in the old Print Manager. PrEndPage finishes the spooling (and sometimes printing) of the current page.

PrEndPage returns a result code indicating the success of spooling the page. This is required to implement printing in a low disk space situation. If spooling was unsuccessful because of low disk space and/or low memory space, this result will signal the application to retransmit the page until spooling or printing of the page is successful. It is important that the application retransmit exactly the same page when retransmission is indicated. This is necessary because retransmission indicates that the page is being sent in bands to the printer. If, for example, the page contains a time variable, it is possible that one imaging pass will band half of this time variable representation and a subsequent pass will image the other half. If it is not possible to retransmit the exact same page, the application should abort at this point.

*when to call:*

This call should be made after the PrOpenPage call and all QuickDraw calls have been made to image the current page.

*the parameters:*

**pPrGrafPort** This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

|               |                                                                                                               |
|---------------|---------------------------------------------------------------------------------------------------------------|
| noErr         | all is well, ready for the next page.                                                                         |
| retransmit    | low disk space has caused the Print Manager to move into foreground printing. Send the page again.            |
| abortPrinting | a serious error has occurred or not enough memory is available to continue printing. Call PrEndDoc and PrEnd. |
| badPort       | pPrGrafPort is corrupt.                                                                                       |

The next set of calls deals with error handling.

```
PrintError PrErrorString( PrintError   iErr,
                          char         *theMessage ,
                          TPrGrafPort  pPrGrafPort );
```

*what happens:*

This call will return an error message suitable for presentation to the user.

*when to call:*

This call should be made after the Print Manager has been successfully initialized through a call to PrStart.

*the parameters:*

|             |                                                                                                                                                                   |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| iErr        | the print error code.                                                                                                                                             |
| theMessage  | the string corresponding to the passed error code.                                                                                                                |
| pPrGrafPort | This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart. |

*the function result:*

|         |                         |
|---------|-------------------------|
| noErr   | all is well.            |
| badErr  | unknown error code.     |
| badPort | pPrGrafPort is corrupt. |

```
PrintError PrSetErr( PrintError theErr, TPrGrafPort pPrGrafPort );
```

*what happens:*

This call replaces the PrSetError call in the old Printing Manager. PrSetErr allows the application to set or reset the current print error. This is useful in case the application wants to abort or if the application wants to continue from a print error that it considers non-fatal.

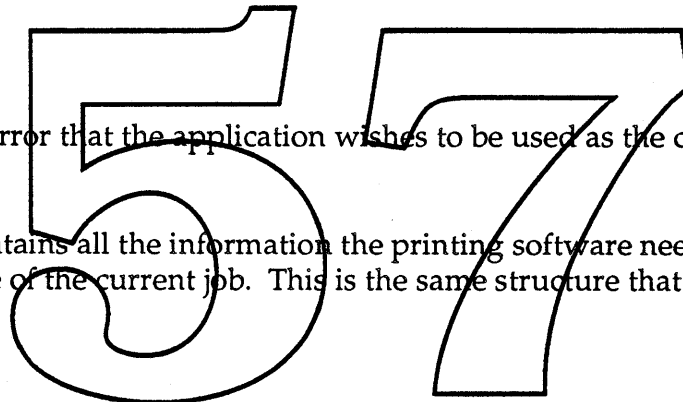
*when to call:*

This call should be made after the Print Manager has been successfully initialized through a call to PrStart.

*the parameters:*

theErr a PrintError that the application wishes to be used as the current print error.

pPrGrafPort This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.



*the function result:*

noErr all is well.  
 badPort pPrGrafPort is corrupt.  
 badErr unknown PrintError.

```
PrintError PrGetErr( PrintError *theErr, TPrGrafPort pPrGrafPort );
```

*what happens:*

This call replaces the PrError call in the old Printing Manager. PrGetErr allows the application to query for the current print error.

*when to call:*

This call should be made after the Print Manager has been successfully initialized through a call to PrStart.

*the parameters:*

theErr      the current print error.

pPrGrafPort      This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

noErr      all is well.  
 badStore      hPrintStore is corrupted.  
 badPort      pPrGrafPort is corrupt.



The next set of calls deals with memory considerations.

```
PrintError PrMemReserv( Integer      iBytesToLeave,
                        TPPrGrafPort pPrGrafPort );
```

*what happens:*

This call allows the application to request that a certain amount of memory be left in the application heap after the Print Manager does its buffer allocations.

*when to call:*

This call should be made after the Print Manager is initialized through a successful call to PrStart and before the application makes the PrStartDoc call. This is an optional call. This call will have no effect if made after the call to PrStartDoc.

*the parameters:*

**iBytesToLeave** An integer representing the number of bytes to leave free in the application heap after the Print Manager buffer allocations.

**pPrGrafPort** This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

**noErr** all is well.

**badPort** pPrGrafPort is corrupt.

**lbadNumber** iBytesToLeave is negative.

**PrintError PrGrowFriendly( TPrGrafPort pPrGrafPort );***what happens:*

This call informs the Print Manager that the application grow zone procedure will not abort when an allocation attempt fails. This type of grow zone procedure is considered friendly. A friendly grow zone procedure will result in better performance. If this call is not made, the Print Manager will assume that the application grow zone procedure will abort on allocation failure. Unless this call is made, the Print Manager will preflight all NewHandle or NewPtr calls that it intends to make out of the application heap and make sure that the application grow zone procedure is not invoked. If this call is made, allocations from the application heap will not be preflied.

*when to call:*

This call should be made after the Print Manager is initialized through a successful call to PrStart and be most effective if made before the application makes the PrStartDoc call. This is an optional call.

*the parameters:*

pPrGrafPort

This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

|         |                         |
|---------|-------------------------|
| noErr   | all is well.            |
| badPort | pPrGrafPort is corrupt. |

The next set of calls deal with imaging, etc. First the etc. calls.

```
PrintError PrGetVersion( Int16 *iVersion, TPrGrafPort pPrGrafPort );
```

*what happens:*

This call returns the current version of the print software.

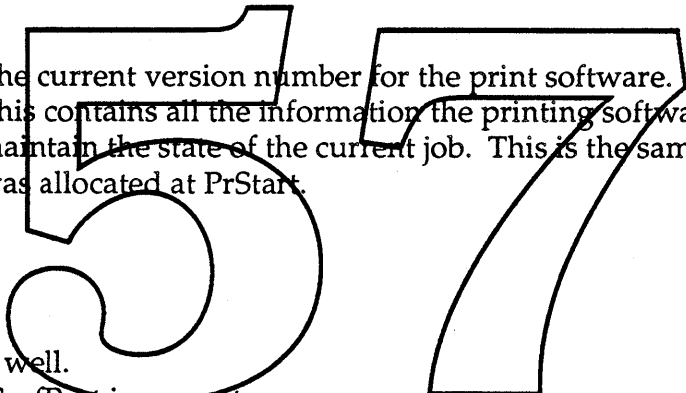
*when to call:*

This call can be made anytime after the Print Manager has been intialized through PrStart.

*the parameters:*

iVersion  
pPrGrafPort

The current version number for the print software.  
This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.



*the function result:*

noErr  
badPort

all is well.  
pPrGrafPort is corrupt.

```
PrintError PrSetIdleProc(Integer      IdleProc(),
                          TPrGrafPort pPrGrafPort );
```

*what happens:*

This call allows the application to set the idle proc to a procedure of its choosing. If an application does want to install its own idle procedure, this procedure must at least provide a mechanism which enables the user to abort the print job.

*when to call:*

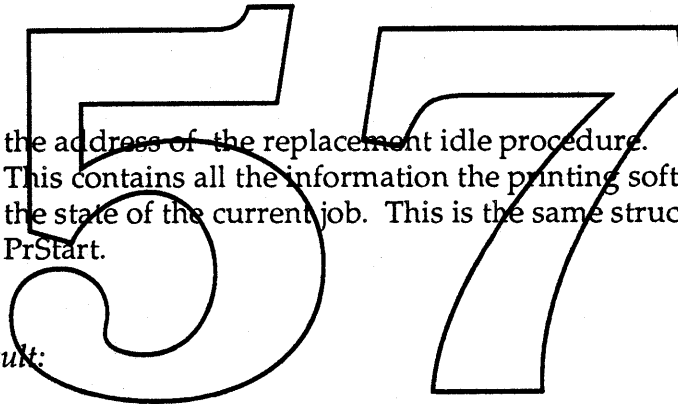
This call should be made after the Print Manager is initialized with a successful call to PrStart and before the PrStartDoc call. This is an optional call.

*the parameters:*

IdleProc      the address of the replacement idle procedure.  
 pPrGrafPort   This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

noErr          all is well.  
 badPort        pPrGrafPort is corrupt.



```
PrintError PrSetResolution( TResl      theResolution,
                           Handle     hJobRec,
                           TPrGrafPort pPrGrafPort );
```

*what happens:*

This call replaces the PrGeneral SetRsl function in the old Print Manager. This call allows the application to inform the Print Manager of the resolution it intends to image at. This call should be used in conjunction with the PrGetResolution call to determine which resolutions are most appropriate for the current printer type. If this call is not made, the Print Manager assumes that the application is imaging at 72 dpi. It is strongly recommended that an application image at 72 dpi to avoid strange scaling artifacts when the print software outputs to devices of different resolutions. Also, since users may now define custom paper types with arbitrarily large dimensions, it is conceivable that a large page size and large resolution could exceed QuickDraw space. In this case, either the user would need to reduce the page size or the application would need to image at a lower resolution. This condition can be avoided for all but the truly psychotic cases if the application chooses to image at 72 dpi.

*when to call:*

This call can be made anytime after the Print Manager has been initialized through PrStart and the application has a valid Job Record.

*the parameters:*

**theResolution** The resolution the application is imaging at. This is expressed in x and y space.

**hJobRec** This is a handle to a valid Job Record for the document. The resolution will be set here.

**pPrGrafPort** This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

*the function result:*

**noErr** all is well.

**badPort** pPrGrafPort is corrupt.

**badJobRec** hJobRec is nil or of the wrong size.

```
PrintError PrGetResolution(TResl      *iCurrentSetting,
                          Integer     *iNumChoices,
                          Resl        iChoices[iMaxChoices],
                          Handle      hJobRec,
                          TPPrGrafPort pPrGrafPort );
```

*what happens:*

This call replaces the PrGeneral GetRsl function in the old Print Manager. This call allows the application to query for the resolutions the current device type will image at and the current setting the Print Manager believes the application to be imaging at. Note that if the user has chosen to target all devices for formatting his document, the device resolution will be pegged at 72 dpi. It is important for an application that chooses to image at different resolutions depending on the output device, to check for the current resolutions available after every Page Setup call, as the user may now switch device types and page size in this dialog. If, for instance, the user chooses E size paper and targets a 4000 dpi film recorder for formatting, the Print Manager will restrict the resolution choices returned through PrGetResolution so as not to have a page size/resolution combination that will exceed QuickDraw space. In this situation, the application may be forced to reformat the document as a document may not contain pages imaged at different resolutions. To avoid the problem of exceeding QuickDraw space and to prevent strange scaling artifacts when going to different output devices, it is strongly recommended that applications image at 72 dpi.

*when to call:*

This call can be made anytime after the Print Manager has been initialized through PrStart and the application has a valid Job Record.

*the parameters:*

|                 |                                                                                                                                                                   |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| iCurrentSetting | The resolution the Print Manager believes the application is imaging at. This is expressed in x and y space.                                                      |
| iNumChoices     | The number of resolutions in the iChoices array.                                                                                                                  |
| iChoices        | An array listing the discrete resolutions that the output devices image at. These are expressed in x and y space.                                                 |
| JobRec          | This is a handle to a valid Job Record for the document. The resolution will be taken from here.                                                                  |
| pPrGrafPort     | This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart. |

*the function result:*

|           |                                      |
|-----------|--------------------------------------|
| noErr     | all is well.                         |
| badPort   | pPrGrafPort is corrupt.              |
| badJobRec | hJobRec is nil or of the wrong size. |

57

**PrintError PrPSToRaster( TPrGrafPort pPrGrafPort );**

*what happens:*

This call will enable transfer modes to work with PostScript devices. This can only be accomplished by rendering a bitmap and shipping this bitmap down to a PostScript device. Consequently, printing will be slower and more memory will be required. Also any direct PostScript sent will be ignored.

*when to call:*

This call can be made anytime after the Print Manager has been initialized through PrStart.

*the parameters:*

pPrGrafPort

This contains all the information the printing software needs to maintain the state of the current job. This is the same structure that was allocated at PrStart.

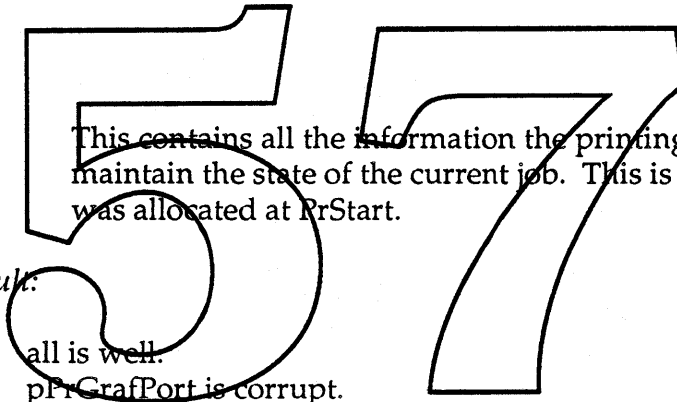
*the function result:*

noErr

all is well.

badPort

pPrGrafPort is corrupt.





Summary of Functions:Initialization and Termination of Print Jobs

```
PrintError   PrStart( TPrGrafPort pPrGrafPort );
PrintError   PrEnd( TPrGrafPort pPrGrafPort );
```

The New Print Records

```
PrintError   PrSizeOfJobRecord( Int16 *iJobRecSize,
                                TPrGrafPort pPrGrafPort );

PrintError   PrSizeOfFormatRecord( Int16 *iFormatRecSize,
                                    TPrGrafPort pPrGrafPort );

PrintError   PrUpdatePrintRecord( TPrint hOldPrintRecord,
                                   Handle hJobRec,
                                   Handle hFormatRec,
                                   TPrGrafPort pPrGrafPort );

PrintError   PrValidateJob(Handle hJobRec, TPrGrafPort pPrGrafPort);
PrintError   PrValidateFormat(Handle hFormatRec, TPrGrafPort pPrGrafPort);
PrintError   PrJobDefault( Handle hJobRec, TPrGrafPort pPrGrafPort);
PrintError   PrFormatDefault( Handle hFormatRec, TPrGrafPort pPrGrafPort);
```

The Dialogs

```
PrintError   PrDoStyleDialog(
    Handle      hFormatRec,
    TPrGrafPort pPrGrafPort,
    Boolean     fByPage,
    Int16       PrinterChange,
    Boolean     Formats(Handle hFormatRec, Integer iWhichFormat)
);

PrintError   PrDoJobDialog(
    Handle      hJobRec,
    TPrGrafPort pPrGrafPort,
    Int16       PrinterChange,
    Boolean     Formats(Handle hFormatRec, Integer iWhichFormat)
);
```

```
PrintError PrAddPanel(THPanelInfo hPanelInfo,
                    TPPrGrafPort pPrGrafPort );
```

### Customizing the Style Dialog

```
PrintError PrAddOrientation( Handle hOrientIcon,
                             Integer iOrientIconID,
                             char *orientName,
                             TPPrGrafPort pPrGrafPort );
```

```
PrintError PrSetOrientation( EOrientType eOrientation,
                             char *orientName,
                             Boolean kLockIt,
                             Handle hFormatRec,
                             TPPrGrafPort pPrGrafPort );
```

```
PrintError PrGetOrientation( EOrientType eOrientation,
                             char *orientName,
                             Boolean *kLocked,
                             Handle hFormatRec,
                             TPPrGrafPort pPrGrafPort );
```

```
PrintError PrAddPaperType( THPaperType hPaperType,
                           TPPrGrafPort pPrGrafPort );
```

```
PrintError PrSetPaperType( char *paperTypeName,
                           Boolean kLockit,
                           Handle hFormatRec,
                           TPPrGrafPort pPrGrafPort );
```

```
PrintError PrGetPaperType( THPaperType *hPaperType,
                           Boolean *kLocked,
                           Handle hFormatRec,
                           TPPrGrafPort pPrGrafPort );
```

```
PrintError PrSetScaling( Int16 iScale,
                          Boolean kLockIt,
                          Handle hFormatRec,
                          TPPrGrafPort pPrGrafPort );
```

```
PrintError PrDisableScaling( Handle hFormatRec,
                              TPPrGrafPort pPrGrafPort );
```

```
PrintError PrGetScaling( Int16      *iScale,
                        Boolean     *kLocked,
                        Handle      hFormatRec,
                        TPrGrafPort pPrGrafPort );
```

### Customizing the Print Dialog

```
PrintError PrSetCustomPageRange(
    char      *startPage,
    char      *endPage,
    Boolean   parsePageRange(char *first, char *last),
    Handle    hJobRec,
    TPrGrafPort pPrGrafPort );
```

```
PrintError PrGetCustomPageRange(
    char      *startPage,
    char      *endPage,
    Handle    hJobRec,
    TPrGrafPort pPrGrafPort );
```

```
PrintError PrDisablePageRange( char      *replacementString,
    Handle    hJobRec,
    TPrGrafPort pPrGrafPort );
```

```
PrintError PrGetPageRange( Int16      *iStartPage,
                           Int16      *iEndPage,
                           Handle      hJobRec,
                           TPrGrafPort pPrGrafPort );
```

```
PrintError PrSetReduction( Int16      iReduce,
                           Boolean     kLockIt,
                           Handle      hJobRec,
                           TPrGrafPort pPrGrafPort );
```

```
PrintError PrGetReduction( Int16      *iReduce,
                           Boolean     *kLocked,
                           Handle      hJobRec,
                           TPrGrafPort pPrGrafPort );
```

```

PrintError PrSetStackingOrder( Boolean      kFirstPageOnTop,
                               Boolean      kLockIt,
                               Handle       hJobRec,
                               TPPrGrafPort pPrGrafPort );

PrintError PrGetStackingOrder( Boolean      *kFirstPageOnTop,
                               Boolean      *kLocked,
                               Handle       hJobRec,
                               TPPrGrafPort pPrGrafPort );

PrintError PrSendingLastFirst( Handle      hJobRec,
                               TPPrGrafPort pPrGrafPort );

PrintError PrSetCollation( Boolean      kCollationOn,
                           Boolean      kLockIt,
                           Handle       hJobRec,
                           TPPrGrafPort pPrGrafPort );

PrintError PrGetCollation( Boolean      *kCollationOn,
                           Boolean      *kLocked,
                           Handle       hJobRec,
                           TPPrGrafPort pPrGrafPort );

PrintError PrStartDoc( TPPrGrafPort pPrGrafPort,
                      char          *documentName,
                      Handle       hJobRec );

PrintError PrEndDoc( TPPrGrafPort pPrGrafPort );

PrintError PrStartPage( TPPrGrafPort pPrGrafPort,
                       Rect          *pPageFrame,
                       Handle       hFormatRec );

PrintError PrEndPage( TPPrGrafPort pPrGrafPort );

```

the Print Loop

## Error Handling

```
PrintError PrErrorString( PrintError iErr, char *theMessage,
                          TPPrGrafPort pPrGrafPort );
PrintError PrSetErr( PrintError theErr, TPPrGrafPort pPrGrafPort );
PrintError PrGetErr( PrintError *theErr, TPPrGrafPort pPrGrafPort );
```

## Memory

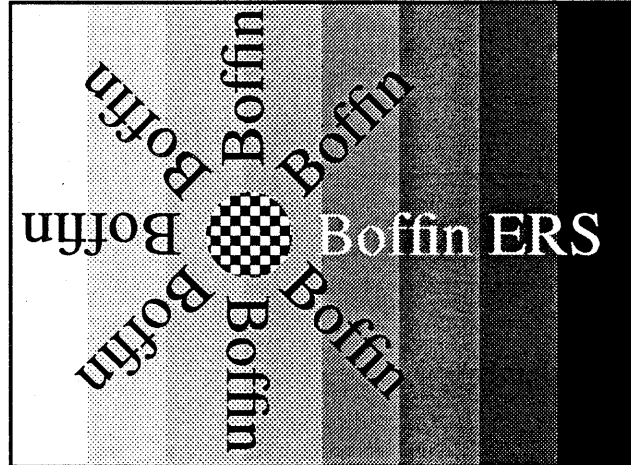
```
PrintError PrMemReserv( Integer iBytesToLeave, TPPrGrafPort pPrGrafPort );
PrintError PrGrowFriendly( TPPrGrafPort pPrGrafPort );
```

## Miscellaneous

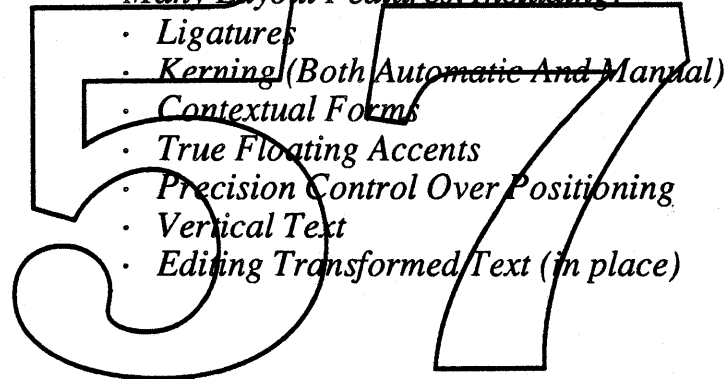
```
PrintError PrGetVersion( Int16 *iVersion, TPPrGrafPort pPrGrafPort );
PrintError PrSetIdleProc( Integer IdleProc, TPPrGrafPort pPrGrafPort );
PrintError PrSetResolution( TResl theResolution, Handle hJobRec,
                          TPPrGrafPort pPrGrafPort );
PrintError PrGetResolution( TResl *iCurrentSetting,
                          Integer *iNumChoices,
                          Resl iChoices[iMaxChoices],
                          Handle hJobRec,
                          TPPrGrafPort pPrGrafPort );
PrintError PrPSToRaster( TPPrGrafPort pPrGrafPort );
```

57

57



- *Device-independent Line Layout*
- *Fully Multilingual And International*
- *Many Layout Features, Including:*



- *Ligatures*
- *Kerning (Both Automatic And Manual)*
- *Contextual Forms*
- *True Floating Accents*
- *Precision Control Over Positioning*
- *Vertical Text*
- *Editing Transformed Text (in place)*

A Production of those Curious Folks in the  
International System Software group!

Friday, January 13, 1989

Contacts:

Mark Davis x2936

Peter Edberg x4275

Dave Opstad x6482



# Boffin ERS

## I. Introduction

### A. This ERS

Section I discusses what Boffin is and what it isn't. This is followed by a discussion of some fundamental data types and a glossary of terms used in this ERS.

Section II discusses Boffin's layout features and how they contribute to the creation of the non-positional layout. Then comes a lengthy discussion of how Boffin manages fine control of character positioning. The section finishes with a discussion of graphics features.

Section III discusses the classes of functionality that Boffin provides.

Section IV describes the architecture of Boffin, including the flow of control that occurs when a high-level layout request is made, as well as how Boffin connects to the other parts of the Macintosh system.

Section V is a description of the individual routines that clients may call, including their purposes, inputs and outputs, and dependencies (if any).

Section VI describes the tools that are required to support Boffin-style fonts.

Section VII is a listing of open issues.

Finally, a couple of things to keep in mind while reading this ERS:

(1) Because current tools on the Macintosh do not let us fully specify text layout, the illustrations in this ERS are only approximate; and

(2) Boffin supports vertical text in a manner analogous to its support for horizontal text. Any examples or descriptions that refer to horizontal positioning are also applicable to vertical, with the appropriate terms substituted (e.g. top for left, etc.) Note that the non-positional layout produces a set of glyphs in a normalized order: left to right or top to bottom order. This is called the *forward* order in the discussion below.

(3) A number of features in this ERS are listed as being not implemented in version 1.0. Note that provision for controlling those features is built into to the Boffin 1.0 interface and data structures.

(4) Weird and wacky line-spacing in this document is courtesy FullWrite™<sup>1</sup>.

### B. What Boffin is

The primary purpose of Boffin (also known as the Layout Manager) is to create and use *layouts*, which are data structures that describe the appearance of lines of text in a device-independent manner. Once such a device-independent layout is generated, it can be used as a template to perform consistent text actions on particular devices. Actions that Boffin supports include:

- Creating and destroying layouts;
- Drawing the text described by these layouts;
- Measuring the width of the line or some part of it;
- Determining the caret(s) for some location within the text;
- Highlighting the text or some part of it; and
- Hit-testing within the text.

---

1. ...and is further proof of the dire need for advanced text support routines (like Boffin!) in the real world.

## Boffin ERS

Layouts are divided into two types:

- *Non-positional* layouts; and
- *Positional* layouts.

A *non-positional layout* contains the output of the “first pass” of the layout process, which includes the processing of characters into ligature glyphs, contextual form glyphs, applied mark glyphs, glyph reorderings and the like, but which includes no information about the positioning of these glyphs in a medium. A *positional layout* is one of these non-positional layouts along with positioning information in relation to some “device”. The word “device” is in quotes here because Boffin can generate a positional layout for some imaginary device of infinite resolution. This is useful in several situations, such as determining line breaks, and providing base positioning numbers for rounding in case the user opts for a layout that is not device-dependent (see below for a discussion of device-dependent layout). In general, a positional layout can only exist in a world where knowledge of device characteristics is available, and so this type of layout is not normally directly available to clients. Rather, such layouts are created by Boffin and the graphics engine in response to some client request (such as “display the text contained in this layout” or “tell me where a mousedown occurred”).

To create a non-positional layout (which is the only kind a client can directly request), Boffin is supplied a description of the text to be laid out, consisting of a string of character codes, along with information about fonts, layout features (such as ligaturing or kerning), positional constraints (such as tab locations or pinned graphics locations), text styles and so on<sup>2</sup>. From these parameters, Boffin produces a non-positional layout that contains the consolidated style and glyph code information. This layout is then used by Boffin to create positional layouts as needed to render the text in a controlled and visually pleasing fashion on a particular device or set of devices, in a process called *rendering*. Boffin will make the text as legible as possible either for a specific device (if the client enables *device-dependent layout*), or it can simply round the values stored in an infinite-resolution positional layout.

In order to accomplish its job, Boffin interacts with the graphics package (e.g. QuickDraw) and the font system (e.g. the Font Manager). Many of Boffin’s capabilities depend on new data structures associated with Apple’s outline fonts (Bass); the design of some of these data structures is part of the Boffin project. The Boffin project also includes the creation of font tools that permit font manufacturers to create these data structures and to associate them with fonts.

One method Boffin uses to keep this whole process moving quickly is *layout caching*. Once a non-positional layout has been created, any positional layouts instantiated from it are associated with it in a cache. This speeds up the process of redrawing, for example, since Boffin can supply an instantiated positional layout to the graphics engine quickly from the cache if it has already created it.

### C. What Boffin isn’t

Boffin is, like the Script Manager, a set of support routines that client programs can call to implement certain functions. It is *not* a text editor or word processor, although it may certainly be used to help implement one of those. It has no “user interface” *per se*, since it’s not an application. It doesn’t understand anything about the organization of text at a level higher than a single line. In particular, paragraph properties such as “rivers of white space” or overall paragraph appearance are *not* in Boffin’s bailiwick. Boffin lays out text in one of two fundamental directions: horizontal or vertical. Any graphical transformation that might be applied to the end result of Boffin’s work (such as taking a line of text and wrapping it around a cylinder) is not Boffin’s responsibility. Boffin treats such transformations as “black boxes” about which it can obtain certain kinds of information from the graphics system (such as, for example, “does this style: ⚡⚡⚡ break ligatures?”), but about the internals of which Boffin knows nothing.

---

2. It might seem confusing that positional constraints are included in the parameters needed to create a non-positional layout. What Boffin does with these is store them in the non-positional layout as “hints” that later steps will use to create positional layouts.

## D. Fundamental concepts and data types

### 1. Characters

#### a) Character codes and fonts

In the current Macintosh paradigm, a *character code* is a value<sup>3</sup> associated with a particular character in a particular font or group of fonts (see section 1.2.2 for a more detailed discussion of the concept of a "font"). This linking of a character code to a font presents difficulties for the straightforward handling of multilingual text<sup>4</sup>. Boffin therefore uses a somewhat different internal paradigm, namely *every character gets a unique, font-independent, unsigned 16-bit character code*. In this sense, therefore, a *character code* is a 16-bit unsigned value that has, by convention, some semantic meaning associated with it (such as "upper-case A" or "comma"). In order to be useful with current Macintosh applications, Boffin will accept current Macintosh-style character codes; clients need not worry about this conversion, but instead should just use character codes in the same manner as they always have. Boffin will always refer to character offsets in the original text string by the original Macintosh byte offsets (i.e. 1-byte and 2-byte values).

#### b) Intrinsic character properties

Some properties of characters are determined solely from their character codes and don't require supplementary information like style or font. For example, an Arabic "lam" character is known to be Arabic by its code alone; furthermore, the fact that it is Arabic means that it is laid out by default from right to left, rather than from left to right. No supplementary control codes, styles, or font information are required to know this fact, the character code alone is sufficient. The only information that a font needs to supply is extra information above and beyond this, for example ligatures that include the "lam" character.

Once the 16-bit codes for the characters exist, the creation of a table which indexes these codes by these fundamental properties is an easy matter. Boffin will refer to such a table, which at a minimum will indicate the directionality of characters and whether they are applied marks. (Note that this is a Boffin dependency on the Script Manager)

#### c) Strings and ordering within strings

A *string* is an ordered array of character codes. Note that the order within this array might be different than the display order. A string is an example of *backing store*, which is a term that refers to some ordered repository of character codes. The term *backing store order* designates the order in which character codes are stored in, say, a document—i.e., before any rendering actions have been performed. Sometimes the term *phonetic order* is used as a synonym for backing store order.

It's important to remember that this ordering within a string is raw, simple character codes in phonetic order. In particular, Arabic strings are *not* presented to Boffin in already reversed order. If I have a string of mixed Arabic and English, say, those characters are present in the same order they would be spoken in. See figure 1 for an example of this.

3. This value may be one-byte (for most alphabetic scripts) or two-byte (for ideographic scripts and certain complex alphabetic scripts).

4. For a discussion of the value of a font-independent character code assignment, see Joseph Becker's article "Multilingual Word Processing" in the July 1984 issue of *Scientific American*, page 96.

## Boffin ERS

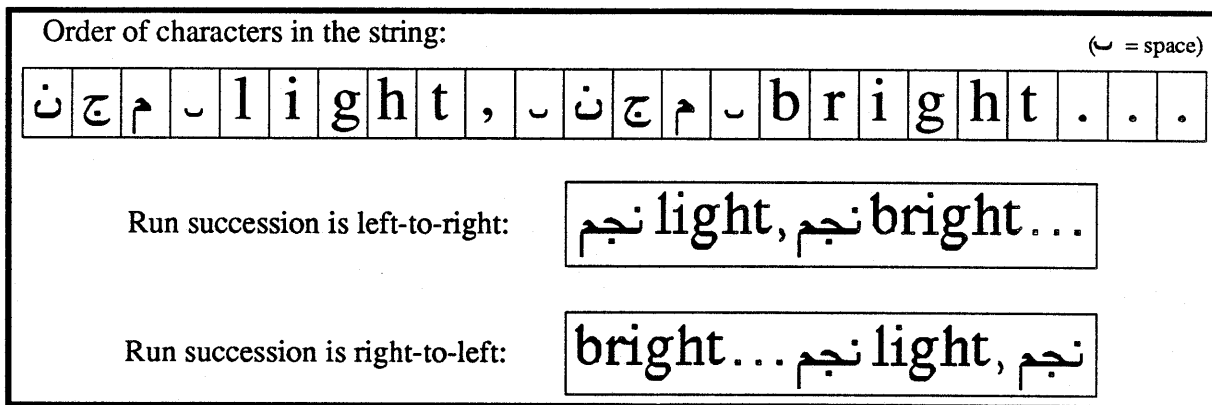


Figure 1—String order vs. Rendered Appearance

## 2. Fonts and glyphs

A *glyph* is a graphical depiction which usually represents a character, symbol, or other textual object. A *font* is a collection of glyphs which generally have some element of consistency in their appearances (e.g. serifs, or stroke thickness), along with other information such as which glyphs represent ligatures, or contextual forms, etc. Within a font, each glyph is associated with a 16-bit code called its *glyph ID*<sup>5</sup>.

A *complex font* is a font that contains information associating some glyph IDs with certain combinations of characters and rules. For example, there may be some information in a font that associates the glyph ID \$1A01 (which happens to have the appearance 'fi') with the combination of the two characters \$0066 ("lower-case f" semantic) followed by \$0069 ("lower-case i" semantic). Any font that contains no such associative information is called a *simple font*<sup>6</sup>. The glyphs in a complex font can be divided into two classes: *rendering forms*, for which combination rules appear; and *character glyphs*, which have a one-to-one correspondence with character codes<sup>7</sup>. Rendering forms include ligatures, applied marks, and contextual forms; these are discussed under *layout features*, below.

See the "Open Issues" section for a discussion of some of the problems involved in runtime font substitution.

## E. Glossary

(See section 1.2 for a discussion of the terms *character code*, *glyph*, *font* and other fundamental terms)

*Absolute space* is resolution-independent space. Measurements in absolute space are always in terms of distances. An *absolute point* is a point in absolute space.

An *Area* is some geometrical entity into which the text is to be laid out. Its implementation varies with different graphics engines. In QuickDraw, an area is synonymous with a region, while in Skia and Albert an area is synonymous with a path. Note that early Boffin implementations will only support simple areas (highlighting, for instance, is necessarily at least a polygon); more sophisticated areas will be supported later. Note also that an area can consist of multiple disjoint parts.

5. Note that the glyph ID associated with a particular glyph need not be the same across different fonts. However, by default, a glyph that is always associated with some particular character is assigned a glyph ID equal to that character's 16-bit value. For example, if we associate the 16-bit character \$0041 with the "upper-case A" semantic, then by convention, the glyph ID of the glyph ('A') that is associated with this character will also be \$0041 in any font containing it.
6. The glyphs in a simple font are accessed by the implicit association between the 16-bit value of the character and the 16-bit glyph ID.
7. There is no convention regarding glyph IDs for rendering forms: that is, even if the 'fi' rendering form exists in two or more complex fonts, it need not have the same glyph ID in those fonts.

## Boffin ERS

A *layout description* is a client-supplied description of some body of text that is to be laid out. It includes such things as the character codes themselves, styles, fonts, sizes, and so on.

The term *LayoutSpec* refers to the data structure that is returned and used by high-level Boffin routines. It encapsulates the description of the text in absolute space. Note also that the locations of glyphs encapsulated by the *LayoutSpec* are relative to some internal origin. Clients should never need (and will never get!) access to the internals of a *LayoutSpec*; it is used solely by the Boffin and graphics parts of the Toolbox.

An *absolute layout origin* is an absolute point which identifies the placement of the *LayoutSpec* in the graphics world. It locates the intersection of the baseline and the leftmost (or topmost) edge of the rectangle that encloses the area in which the text was laid out.

The *text source* (sometimes also known as the *backing store*) is the collection of characters that comprise the layout description. These characters are phonetically ordered as appropriate for the relevant language, and *not* in the order that they might appear in once rendered. Note also that Boffin assumes that all character codes are 16-bit Unicodes; in the Blue world, existing 8-bit code/script pairs are converted (invisibly to the client) into these 16-bit forms.

When referring to character positions within the text source, *Upstream* refers to a location closer to the start of the text source, while *Downstream* refers to a location further away from the start of the text source.

A *layout feature* is a kind of visual effect that Boffin supports. Layout features are divided into two kinds: *non-positional layout features* are those features that affect the glyph selection but not the glyph-to-glyph positioning within a layout. Non-positional layout features include ligaturing (which includes pre-rendered accented forms and accent ligatures), contextual forms, applied marks, reordering of glyphs, character directionality overrides, and the invisibility of ligatures to caret movements. *Positional layout features* are those features that affect the glyph-to-glyph positioning within a layout. Positional layout features include kerning (both manual and automatic), transcription, optical alignment, hanging punctuation, justification, centering, gridding, tab stops, and distribution spacing. These features are all described in more detail in following sections.

A *graphics feature* is a kind of visual effect that the graphics system supports, about which Boffin doesn't need to know anything (other than metrics). For example, color or boldness are text attributes that the graphics engine needs to implement, but Boffin doesn't need any specialized knowledge about these features.

## II. Introduction to Layout Features

This section describes the various layout features that Boffin can perform. The major part of Boffin's work is the construction of *layouts* which describe the appearance of a line of text. In order to construct a layout, Boffin needs not only a source of text, but also information that describes different styles and other options that affect the end appearance. The term *special effect* is used for these styles or options that have an effect on how the text gets laid out. There are two classes of special effect that act to transform the appearance of a line of laid-out text (as opposed to operations that adjust the positioning of things within a line): *layout features* are those effects that Boffin has direct control over, while *graphics features* are those effects for which the graphics engine is responsible<sup>1</sup>. The class of layout features is further divided into those features that do not affect inter-glyph positioning, and those features that do.

One important thing to keep in mind: the layout features described below are *required* for proper support of international text, which has many attributes that low-quality English text never needs. While these features are optional for English, they are vital to the correct rendering of text in many other languages.

Many layout features, both positional and non-positional, have different levels of effect that could reasonably be chosen from. For example, in English one might want to say "never substitute any ligatures", or "substitute only common ligatures like 'fi'", or "do as much ligaturing as possible, including things like 'æ'". In order to allow for this, these layout features can be individually enabled or disabled, for one character or a range of characters in the text source, and at five different levels: suppress, mandatory, normal, optional, and a meta-level of default<sup>2</sup>.

The *suppress* level means no support (i.e. inhibit the layout feature). Note that this might be harsher than expected: in Arabic, for instance, a value of *suppress* for the ligature feature would inhibit the formation of the usual lam-alif ligature. This setting should therefore *only* be used if the user wants to see something approaching the naked, unvarnished text source.

The *mandatory* level means to support only mandatory instances of the particular feature. For example, during rendering of Roman characters, Boffin might only use an 'é' glyph for a 'e' followed by a " in the text source if the "accent anchoring" feature level is at least *mandatory*. Roman kerning, on the other hand, might not ever be considered mandatory, and therefore a value of *normal* or higher might be needed to enable kerning.

The *normal* level means to support "normal" instances of the special effect. In Arabic ligaturing, for example, this level is required to get most of the normally used ligatures (such as "initial lam-mim"), assuming the font supports them. This is also the level that most font manufacturers will use for Roman kerning.

The *optional* level means to support any and all instances of the special effect that the font supports. For example, an "a" followed by an "e" in English text might cause an "æ" ligature to be used if the ligaturing level is *optional*. Similarly, variant appearances of certain Chinese characters might be selected with a forms feature level of *optional* (note that the font will have to contain adequate information to identify what happens to a particular glyph in this case—Boffin has no knowledge about this!)

The fifth option is *default*. Associated with each layout feature there exists a global default value which is set via a cdev, and which is used if the *default* level is chosen.

1. Boffin still needs to keep track of graphics features, of course, since it needs to pass the identities of those features to the Font Manager when it needs character metrics; however, that is all that Boffin needs to do with them.
2. Note that in all these cases, the determination of what constitutes mandatory, normal and optional effects is left up to the font manufacturer. This allows quite a bit of flexibility, but can cause confusion if the user is using two different fonts that have quite different interpretations of what constitutes normal vs. optional features. Apple should specify a series of guidelines to help font manufacturers make their decisions.

## A. Non-positional layout features

### 1. Ligatures, accented forms and accent ligatures

A *ligature* is a rendering form that represents a combination of two or more individual characters. The word “ligature” is also used as a verb: to ligature is to form a ligature. Some examples include the ‘fi’ ligature in English, and the “lam-alif” ligature in Arabic. For our purposes, an *accented form* is a special type of ligature: a rendering form that combines a letter with an accent mark. For example, the glyphs ‘ä’ and ‘ð’ are accented forms. Multiple accent marks themselves may be present in a font as *accent ligatures* (see figure 7 for an example of accent ligatures).

### 2. Applied marks

An *applied mark* is a glyph that is dynamically composited with some other glyph. The recipient glyph is called the *baseform*, and a baseform can have one or more marks applied to it. For example, Polish uses Roman script with some additional glyphs such as ‘ś’. If this glyph does not exist as a rendering form in the font, it could be created by dynamically composing the baseform ‘s’ and the applied mark ‘˙’. The composition process will ensure that marks are properly placed with respect to the baseform<sup>3</sup>.

The act of applying marks is done via *anchorpoints* which are attachment points within the baseform to which marks are applied. This could be done without anchorpoints, but the results are not as satisfactory. In figure 1, for instance, we show the results from three different algorithms for applying the tilde mark. In the first case, we simply center the mark over the center of the letter, but this breaks down for asymmetric letters like ‘L’. In the second case, we apply the mark at some fixed displacement into the width of the glyph, but this gives even worse results. The best results are shown in the third case, where each letter (and the mark) has anchoring information that allows the correct placement of marks, irrespective of the shape of the letter and mark.

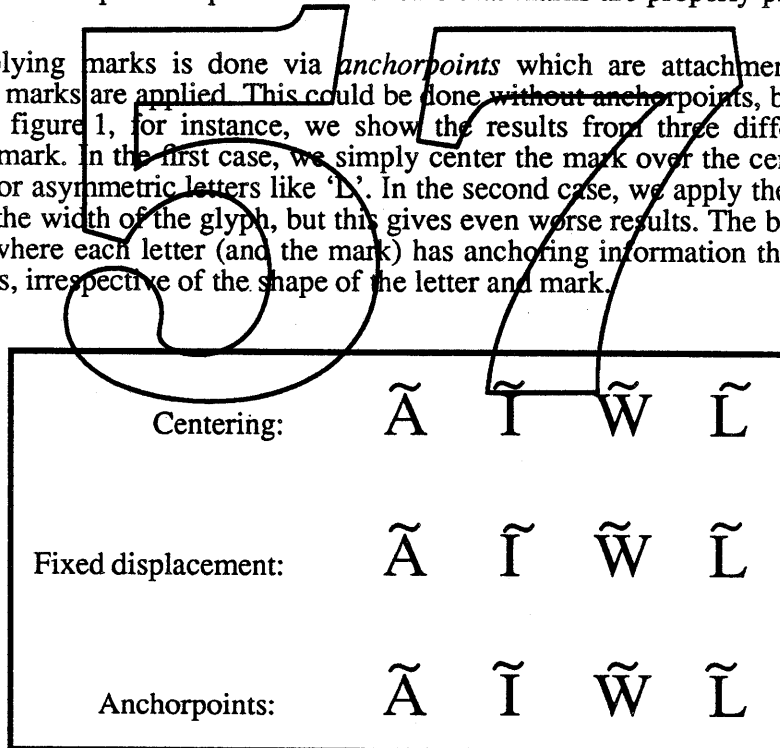


Figure 1—Applying Marks

3. Note that applying marks may force a change in the width of the base character, such as a wide circumflex accent being applied to a narrow character such as an ‘l’.

## Boffin ERS

### 3. Contextual forms

A *contextual form* is an alternate appearance of a glyph that is used in certain contexts. Arabic, for example, has different contextual forms of characters, depending on whether they are at the beginning, the middle, or the end of a word. Figure 2 illustrates this by showing the forms of the Arabic letter “ha” that appear alone, at the start, middle, or end of a word. The same character code is used for each case; it is Boffin’s responsibility to choose the correct glyph.

|                 |    |
|-----------------|----|
| Standalone "ha" | ه  |
| Initial "ha"    | هـ |
| Medial "ha"     | هـ |
| Final "ha"      | هـ |

Figure 2—Arabic contextual forms

In figure 3 we see how a Roman font manufacturer could choose to support either contextual forms or ligatures to present the appearance of a combined ‘fi’ glyph.

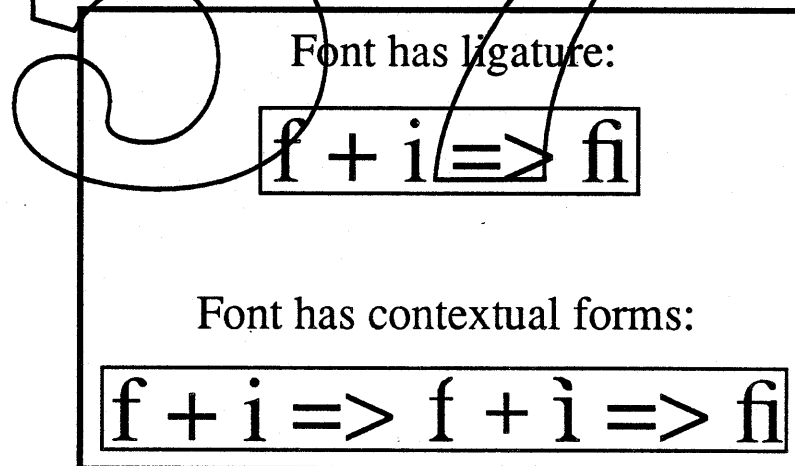


Figure 3 — When Boffin sees ‘f’ followed by ‘i’

Contextual forms can be used for subtle effects in English non-cursive text. It permits font vendors to maintain text color by changing the glyphs associated with a character. (Color is a term in lettering which refers to the overall blackness of a piece of laid-out text—the common method of helping to judge color is to squint or de-focusing the eyes. An example is provided by the following diagram.

LAYOUT  $\Rightarrow$  LAYOUT



## Boffin ERS

You will notice that kerning can be used to effect color in the second word. However in addition, the L has a shorter horizontal stroke in second example, to reduce the large amount of white space between the L and the A. This substitution of glyphs can be done via contextual forms.

### 4. Text Direction and Character Reordering

A given string may include substrings that are rendered in opposite directions. For example, Arabic letters are printed right to left, while Arabic numerals and Roman script are printed left to right. A *direction streak* is a substring consisting of characters which belong to classes that all have the same dominant rendering direction.

Even within a direction streak, some characters may be rendered in an order which is different than the backing store order. For example, in Devanagari script a short 'i', ष, is actually printed before the consonant that it logically follows: the word "Hindi," when printed in Devanagari, comes out something like "ihndi"<sup>4</sup> : हिन्दी . (Vowel markers in many Southeast Asian languages can occur on all four sides of the consonant that they modify.)

In the above cases, it is assumed that the characters in the string appear in "natural" (i.e. phonetic, semantic) order. Taking the example above, the backing store would contain character codes in the order "hindi," not in the order "ihndi." Boffin—not the string creator—is responsible for all reorderings. This point cannot be too highly emphasized: as far as Boffin's clients are concerned, they always present text to Boffin in phonetic backing store order, and leave the reordering up to Boffin.

Boffin is also capable of handling vertical text (Chinese, Japanese, Mongolian). Note, though, that Boffin will not permit mixing of horizontal and vertical text in the same string.

### 5. Character Directionality Overrides

The previous section described the "normal" direction processing that Boffin does—"normal" meaning that it follows the standards of the languages being rendered. Another feature that Boffin provides is the ability to override the direction that text normally takes. In figure 4, for instance, the top line represents text as it is normally displayed for the language (i.e. left-to-right for the English text and right-to-left for the Arabic text). The bottom row represents text with the directionality override set to reverse the order. The important thing to note here is that Boffin is still performing layout features on the reversed text. The Arabic text still undergoes contextual formation, although with the characters in reversed order. The English still uses ligatures, although in different parts of the word.

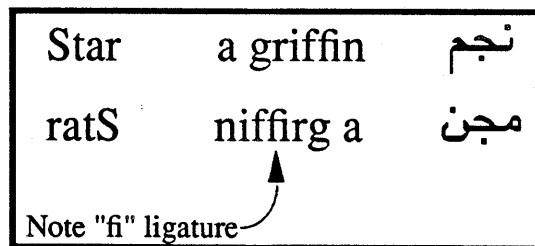


Figure 4—Direction Override Effects

Boffin has overrides to map forward characters to backward or backward to forward (as appears in figure 4). It also allows overrides to map neutral direction characters to either forward or backward.

4. Note that there is a distinction here between simple reorderings and reorderings that imply other transformations. For example, the Devanagari short-i character ष has a given width, but when it is reordered to be at the front of the letter it logically falls behind, it overhangs by a considerable amount.

## Boffin ERS

(Neutral characters include punctuation characters and others that adopt the direction of adjacent characters. If they are between characters of different direction, then they adopt the line run direction (see the discussion of line directions)).

### 6. Ligature Internal Visibility

In a particular layout, suppose that ligatures have been enabled, and that the 'fi' ligature is one of the glyphs that gets generated (from a separate 'f' and 'i' in the text source). Further, suppose that this layout is being used in a word processor, and that a flashing caret is positioned just to the left of this ligature. What visual effect should occur when the user presses the right-arrow key? There are two main approaches: the *indivisible* approach, which treats the ligature as an indivisible whole for purposes of caret location (though *not* for purposes of deletion; see below), and the *divisible* approach, which allows the caret to appear inside the ligature.

It's important to note that in both of these cases, editing still occurs one character at a time, and not one glyph at a time. Thus, if the caret were positioned to the right of the ligature, a single backspace would not delete the whole ligature, but rather would delete only the 'i', leaving an 'f' character (and glyph).

The choice of which of these approaches is used is left up to the user, via the specification of a layout feature level for this feature. If the user wishes to be able to position inside a ligature, the *divisible* level should be selected, and Boffin will make a mathematical division of the glyph into  $n$  equal sub-widths (where  $n$  is the number of characters that were absorbed into the ligature). If the user doesn't wish to make such positioning, the *indivisible* level should be selected.

### 7. User control of layout features

Most of the information needed to implement these features will be supplied by data structures in the fonts. Boffin clients will be able to specify a level of enabling for reordering, ligatures, applied marks, contextual forms, and transcription; disabling all of these features will display glyphs in essentially a one-for-one correspondence with character codes in the backing store—an 'exploded' mode. Callers will also be able to override font-supplied keying information. Figure 5 demonstrates a case in which a level specification for ligaturing could be useful—assuming the ligatures were actually present in a font, then the exact same sequence of characters in a string could be rendered in four different ways!

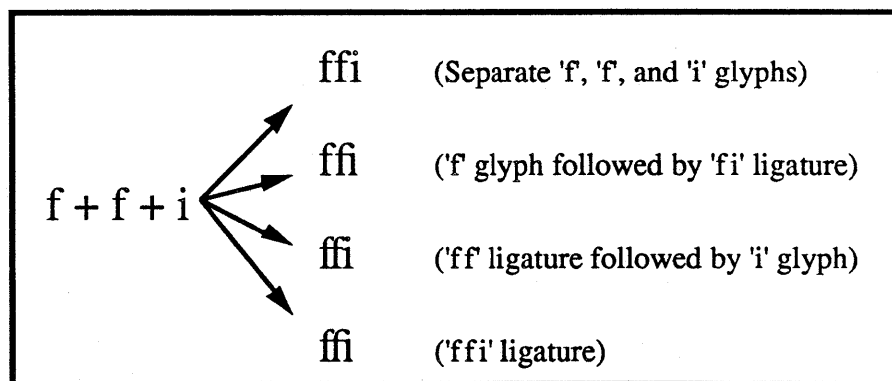


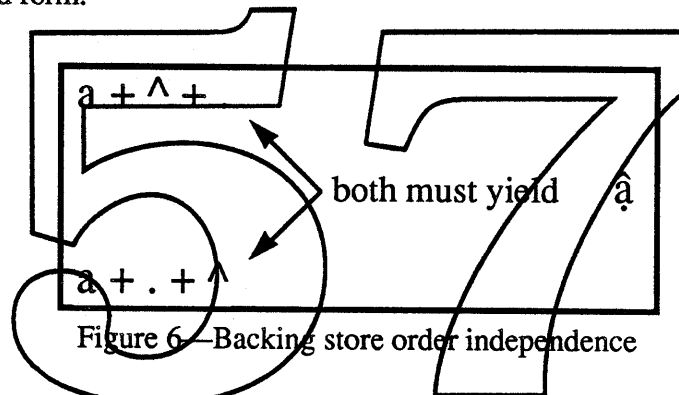
Figure 5 — Different ways to ligature the same text

## 8. Overlap of layout features

Note that different types of layout features can sometimes be used to obtain the same effects. As an example, accented characters can be handled either with applied marks or as accented forms. As another example, the 'fi' ligature can also be handled as a contextual form of 'f' followed by a contextual dotless-i 'i' form. [A third example: treating the Devanagari short-i as a reordering or an applied mark]. There are advantages and disadvantages to each approach, and font manufacturers must make the decision. A factor to consider in making this decision is the user's ability to separately enable or disable each feature.

## 9. Interaction among layout features

There are some interesting issues involved in the interaction of special effects that occur during layout. For example, if a "lower-case a" character is followed in the string by a series of accent characters, some of which map to accented forms (like "â"), and others of which are pure applied marks (such as the "under-dot accent" used in Vietnamese), then the software must make the best use of the font's characters, irrespective of the order in which the accent characters appear in the string. In figure 6, for instance, the end result in either case should be the application of an under-dot accent mark to the "â" accented form.



This is further illustrated in figure 7, where several accents that individually would cause the selection of an accented form instead combine to form an *accent ligature*<sup>5</sup>, which is then applied as a mark to the "a" baseform. When an applied mark (i.e. the under-dot) is then added to this, and considering the possible permutations of the accent positions in the string, you can see that the rendering software must be prepared to accept a variety of combinations so that users always see the optimal appearance, even if typing accents in the "wrong" order (if there is such a thing).

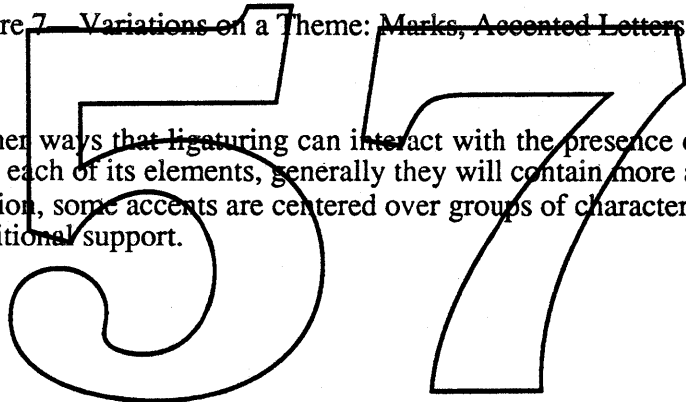
5. This combination of a grave + circumflex accent is used in Vietnamese. Note that in this example, it is assumed that this accent ligature is actually present in the font; if it were not, the same effect could be had by treating the various accents in figure 7 as simple marks to be applied to the "a" baseform.

## Boffin ERS

|               |   |   |                                                      |
|---------------|---|---|------------------------------------------------------|
| a + .         | → | à | Letter plus applied mark                             |
| a + `         | → | â | Accented letter                                      |
| a + ` + .     | → | â | Accented letter plus applied mark                    |
| a + ` + ^     | → | â | Letter with applied accent ligature                  |
| a + ^ + `     | → | â | Letter with applied accent ligature                  |
| a + ` + ^ + . | → | â | Letter with applied accent ligature and applied mark |
| a + ` + . + ^ | → | â | Letter with applied accent ligature and applied mark |
| and so on...  |   |   |                                                      |

Figure 7. Variations on a Theme: Marks, Accented Letters, and Accent Ligatures

There are other ways that ligaturing can interact with the presence of accents. Since a ligature can have accents on each of its elements, generally they will contain more anchor points than non-ligature glyphs. In addition, some accents are centered over groups of characters (e.g. **öö**), including ligatures, and require additional support.



## Boffin ERS

### B. Positional Layout Features

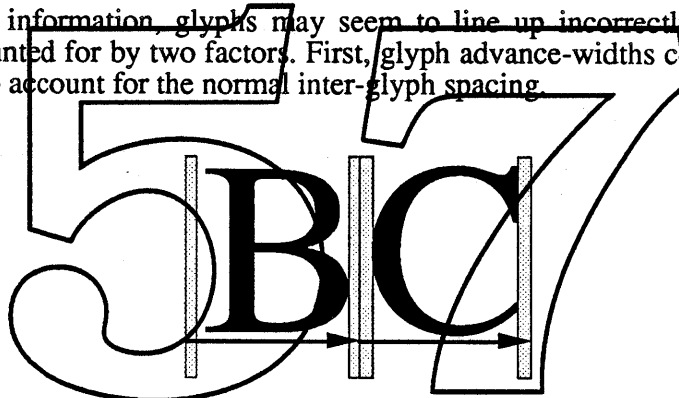
Positional layout is performed on the sequence of glyphs that result from the earlier glyph transformation phase performed by Boffin. The positional layout features include kerning, optical alignment, manual offsets, centering (right, left flush), justification, hanging punctuation, extending connected glyphs, device-sensitive spacing and tab stops.

Positional layout changes the relative positions of glyphs. Suppose that we define the *left side* of a glyph to be the vertical line that passes through the glyph origin, while the *right side* of a character is the vertical line that passes through the origin + advance width. An initial approach to text layout is to simply lay down the glyphs in order, with the left side of each succeeding character matching the right side of the previous character. Positional alignment produces modifications to this basic strategy in order to improve the fundamental appearance of the text, and to allow the end user to fine-tune the appearance to match his or her requirements.

#### 1. Spacing

##### a) Optical Alignment

Without additional information, glyphs may seem to line up incorrectly on the right and left margins. This is accounted for by two factors. First, glyph advance-widths contain a certain amount of extra white space to account for the normal inter-glyph spacing.



This produces certain anomalies, since this space varies with the font size. For example, if different sizes of the same font are left (or right) flush in QuickDraw, they will not generally line up correctly:



The second problem is that due to certain optical effects, curved lines do not appear to line up properly with straight lines. To make them appear to line up, some compensation must occur. For this reason, curved letters such as C or S are generally designed to extend slightly below the baseline, so that they appear to line up with straight letters such as H.

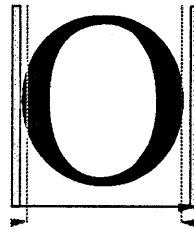
CASH

This same effect happens on the right and left edges of lines. In both of the following example, the center O is lined up with the H's, in the sense that the leftmost black edge of the H is even with the leftmost black edge of the O; this is shown by the line on the right. However, the letters do not appear to align correctly because of optical effects, as you see by looking at the words on the left.

Hotel... Hotel...  
Other... Other...  
Hotel... Hotel...

To compensate for these effects, Boffin can apply optical alignment information contained in the font. When determining the right and left edges of a line of text, whether at tab stops or line starts, Boffin will use the optical right and optical left edge. [Note that corresponding top and bottom optical edges are available for vertical text as well.]

These values are obtained from the font as a pair of offsets from the right and left sides of the glyph as defined above. In the following example, the black arrow represents the origin and advance width, the gray rectangles represent the glyph spacing, the short gray arrows represent the optical offsets, with the gray lines showing the resulting optical edges of the glyph.



## b) Hanging Punctuation

Hanging punctuation is punctuation that hangs outside of the bounds of the text. This commonly includes very light-weight punctuation such as periods or quotation marks:

## Boffin ERS

“The quick brown fox jumped over the lazy dog,” said the sage.

If the text style allows hanging punctuation, then Boffin queries the font for the edge glyphs to determine which ones hang outside of the bounds. If the font does not contain optical alignment information, then the glyph origin and advance width are assumed to define the glyph edges.

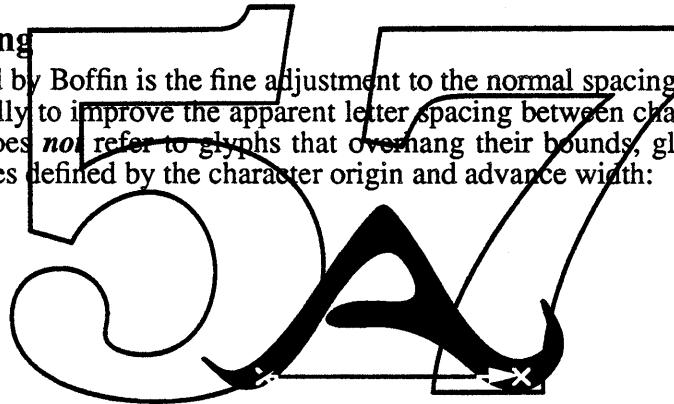
Glyphs can be marked as either left or right hanging glyphs, or both. (A different table is used internally for vertical glyphs, since glyphs may not behave the same in a vertical context.) Note that only base glyphs are considered: applied marks are not relevant to determining the optical text edges (version 1.0 only?).

When determining the optical left (resp., right, top, bottom) side of the text for the purposes of alignment (justification or centering), Boffin will use the optical left (right, ...) side of the leftmost (rightmost, ...) base glyph that does not hang on the left (right, ...).

*Open Issue: we could allow the client to specify a maximum overhang rectangle, that would limit the amount that text can overhang its bounds due to hanging punctuation, style effects (e.g. italics) or character overhang.*

### c) Automatic Kerning

Kerning as applied by Boffin is the fine adjustment to the normal spacing that occurs between two or more glyphs, usually to improve the apparent letter spacing between characters that “fit together” naturally. Kerning does *not* refer to glyphs that overhang their bounds, glyphs that extend beyond their right or left edges defined by the character origin and advance width:



Boffin uses information based in the font tables to determine how much to increase or decrease the space in between two glyphs. In the general case, this amount can depend on more than just the two adjacent glyphs: it may also depend on preceding or following glyphs. (Actually, the context is dependent on the context of glyph *objects*, but any glyphs with applied marks are not kerned in version 1.0.) For example, the following two pairs might kern, but the triple would not:

F.     $\Rightarrow$     F.                      .”     $\Rightarrow$     .”  
F.”     $\Rightarrow$     F.”

Based on the font tables, Boffin maps a pair of left and right glyph sequences  $\langle g_n, \dots, g_1, g_0 \rangle$ ;  $\langle h_0, h_1, \dots, h_m \rangle$  onto a kerning offset. [This is a formal description of how kerning values are associated. The internal structures of the font kerning tables permit a higher performance method of deriving the kerning values.] This kerning offset is used to decrease (or increase) the spacing between the two pairs of glyph sequences. When kerning, the offset is effectively split between the two characters. The example on the right shows where the caret would appear between the two characters.

To To

Clients can disable kerning on a style run or whole layout basis.

*In version 1.0, the kerning offset is a single value, which does not change with the font size, nor with the spacing distribution values (see below). In later versions these facilities may be added.*

**i) Cross-Stream Kerning**

Cross-stream kerning allows the automatic movement of characters perpendicular to the line orientation of the text. For horizontal text, this means vertically. For example, a hyphen between two capital letters should be raised to reflect the centers of those characters:



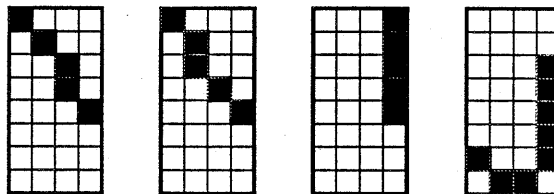
**ii) Shape-Based Kerning**

With shape-based kerning, the shapes of the characters are used directly to produce kerning values. This allows characters to be kerned across style runs of different fonts or size.

There are two basic methods of doing this. The computationally expensive method depends upon determining the actual white space between the outlines of the glyphs, and adjusting it towards a norm while maintaining a minimum distance between the outlines.



The second method uses support from the font to map glyphs onto a standard shape class (one for the right and one for the left half of the glyph), where a shape class represents the rough shape taken up by that type of character. For example, in a given font, the right side of A, left side of V (and W), right side of H and left side of j might map onto classes representing the following shapes, respectively. These classes would allow AV (or AW), Hj, and Aj to kern, but not HV.



*Note: shape-based kerning will not be implemented in version 1.0.*



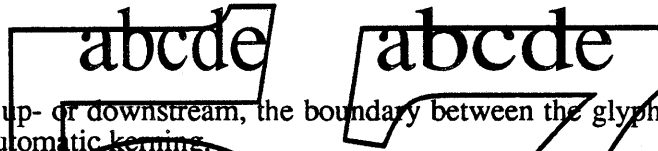
## Boffin ERS

### d) Manual Position Shifts

The client can include offset information in the style run for two types of positional shifts: cross-stream shifts, and with-stream shifts (anyone have a better name?). Cross-stream shifts will *raise* or *lower* the entire style run (and the corresponding horizontal movement for vertical text). With-stream shifts will move each glyph in the style run *UPstream* or *DOWNstream*, and can be used for manual kerning (in the first word, the *s* in *stream* is shifted upstream, while in the second, it is shifted downstream).

For cross-stream alignment, each glyph in the style is shifted up (or down) by the given offset. For with-stream, each glyph in the style run is shifted by the given offset closer to (or further from) the *previous* glyph. For example, in the following example there are two sets of four glyphs. The third and fourth glyphs of each pair (gray) are shifted with-stream, backwards in the first pair, and forwards in the second. As you can see, this is similar to marking the *second* through fourth characters as condensed or extended (although these styles alter the *appearance* of the glyphs, rather than just expand or compress the spacing around them.)

abcde abcde



When text is shifted up- or downstream, the boundary between the glyphs is adjusted to be half-way in between, as in automatic kerning.

### e) Embedded Object Placement

Boffin allows the placement of embedded objects such as pictures (Picture). These objects behave as though they were characters (with neutral direction), for the purpose of text flow, hit-testing, highlighting, etc. The client is called with a callback procedure to draw the object if it is not a picture, whenever the object is to be rendered. After hit-testing, the client has the responsibility of determining that the offset refers to an object rather than a picture, and handling additional feature interaction from there: including highlighting as a whole object, allowing resizing, etc.

The one additional feature available is the ability to set the virtual baseline for a picture, which allows the picture to be placed properly with reference to different sizes of text.

### f) Transcription

In printed Japanese, it is often necessary to place small kana characters adjacent to kanji characters in order to clarify the pronunciation of the kanji characters. Small kana used in this way are called *furigana*. A similar technique is often used to indicate transcriptions from one script to another: for example, "XOPO" in Cyrillic script can be transcribed "horo" in Roman script. In both of these cases, it is desirable that the transcription characters be attached in some way (for cut and paste operations, etc.) to the character(s) they are transcribing. All such uses will be designated *transcription*.

と し かん  
図書館

h o r o  
XOPO

Transcription (e.g. Kanji furigana) is positioned specially. When a style run is marked as transcription, then it is removed from the main body of the text, and centered above the preceding character. (There are actually two varieties of transcription, one which goes above (right for vertical), and one that goes below (left for vertical). If the previous style run is marked with the style `transcriptionBase`, then the transcribed text is centered above the entire previous style run.

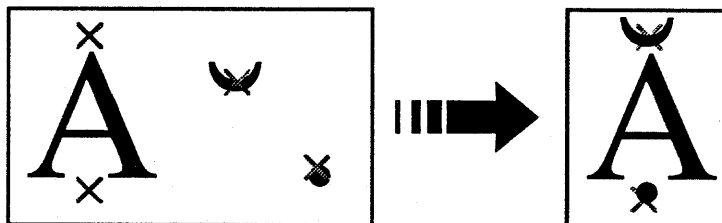
## Boffin ERS

*Transcription will not be implemented in version 1.0.*

### 2. Alignment

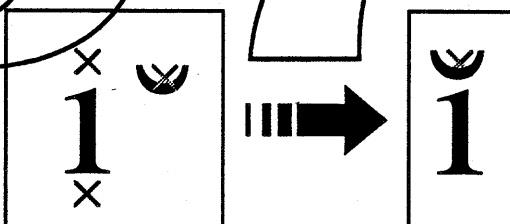
#### a) Applied Marks

Applied marks are usually aligned by means of anchor points. A glyph object associates a base glyph anchor point with each applied mark. Each applied mark has a single distinguished anchor point (point zero?). This anchor point is lined up with the associated base glyph anchor point for rendering. For example:



When the font manufacturer has not supplied anchor points, then Boffin will use heuristics to try to place the applied marks, based on the bounding boxes of the different glyphs. These heuristics will attempt to produce reasonable text, but will not in general match the quality available by using anchor points or separate rendering forms. The black bounding box and its relation to the character sides for the respective glyphs is used in these heuristics.

Notice that the positions of the anchor points may be hinted, and vary with the device. Also, notice that in some cases the advance width of the glyph object may be different that the advance width of the base glyph: the overhang produced by an applied mark might have to be taken into account.



#### b) Justification

Boffin supports segment justification based upon the justification parameter  $j$ . This allows applications to support various degrees of raggedness on a per-segment basis, from completely ragged to fully justified.

|                   |                     |
|-------------------|---------------------|
| The rain in Spain | justification = 0.0 |
| The rain in Spain | justification = 0.5 |
| The rain in Spain | justification = 1.0 |

## Boffin ERS

The distribution spread  $ds$  is the amount that the line length is to be increased. This value is determined as the product of the justification parameter with the difference between the bounds width and the line width:  $ds = j \cdot (bw - lw)$ . The justified line width is the sum of the line width and the distribution spread:  $jlw = lw + ds$ .

Optical alignment (kerning, hanging punctuation and optical spacing) and manual offsets are applied before justification, so the line width referred to here is the *adjusted* line width. The text edges that are justified with respect to the bounds are the optical glyph edges.

### c) Centering

The centering parameter  $c$  is independent of the justification parameter, and is applied after justification. The centering parameter determines where the line is positioned with respect to the line bounds after justification has been applied.

|                   |                 |
|-------------------|-----------------|
| The rain in Spain | centering = 0.0 |
| The rain in Spain | centering = 0.5 |
| The rain in Spain | centering = 1.0 |

A centering value of 0 is commonly known as left-flush, while a centering value of 1 is known as right-flush. The centering offset  $co$  is the distance from the left side of the bounds to the left edge of the text. It is determined by multiplying the centering parameter times the difference between the bounds width and the justified line width:  $co = c \cdot (bw - jlw)$ .

As in the case of justification, optical alignment is taken into account before centering.

### d) Monospace Gridding

Monospace gridding is a special style used in Chinese, where the Chinese characters are monospace, and forced to be on monospace boundaries; any intervening proportional spaced text is justified in between the Chinese characters. *[I am not making this up—MD]* For example, notice how all of the monospaced characters line up vertically, even though there is mixed proportional text.

This·is·a·line·of·monospaced...

The·second·line·has·mixed·text...

*Boffin will not support Monospace Gridding in version 1.0.*

## 3. Stops

Boffin lays out a line with respect to a list of *stop entries*. These entries include both the traditional notion of tab-stops, and the margins. There must be at least one stop entry with every layout, in order to determine the line margins. If the optional fitting area is supplied, then that information may override some of the stop information.

Boffin will break a line of text into a sequence of segments, where each segment is delimited by a tab (or the start or end of the text). Each segment is associated with a corresponding stop.

### a) Stop Entries

A stop entry specifies the starting and ending boundary (with respect to the line origin), associated centering and justification parameters, and optional information discussed below. The

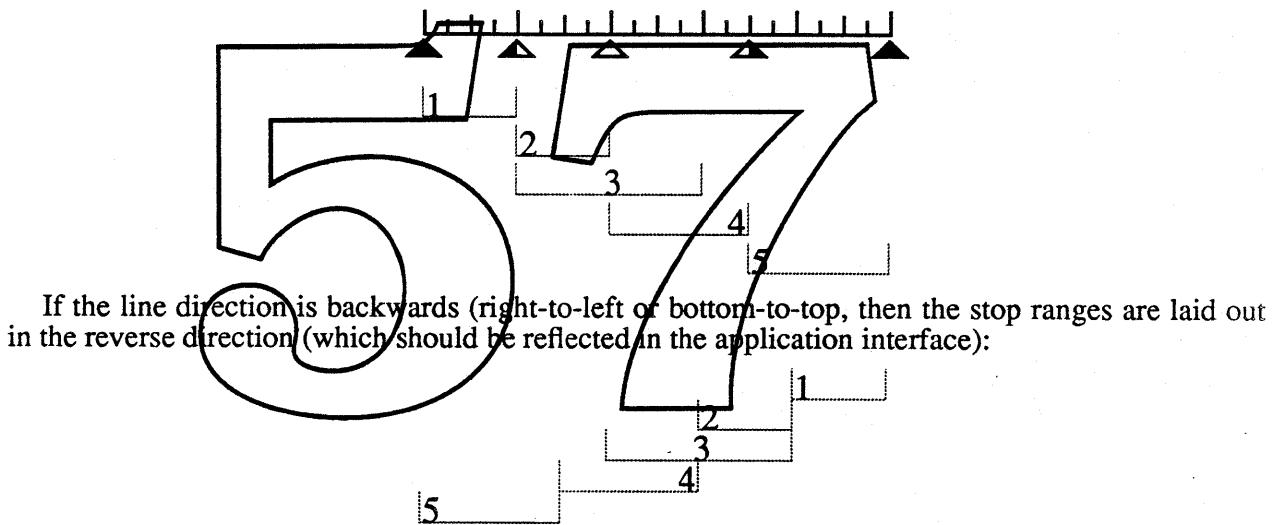
## Boffin ERS

client can use stop entries to implement a variety of tabulation schemes.

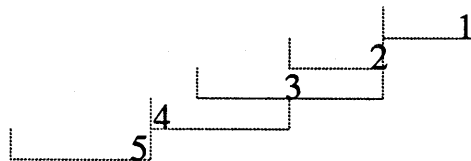
Boffin currently does not impose a particular model of tabulation; instead, it has a general notion of stop entries that allows applications to specify tabulation in different ways. The application can map its notion of tabulation on to Boffin stop entries in a variety of ways, in order to implement different detailed behaviors. For example, if a client narrows the paragraph margins, do the tab stops remain the same, or are they proportionately narrowed, or are they fixed relative to the left, center or right? How do tab keys behave if there are no tab-stops? What happens if the text in between tabs is too long? How does line justification affect individual tab-stops?

*Open Issue: should we try to impose a standard behavior on tabulation, so that we don't get the discrepancies between programs such as PageMaker, FullWrite, Ready-Set-Go, or Word? Sounds like a job for the HIG.*

The following diagram illustrates the stop entries that would result from one method of mapping the tabulation marks indicated above into Boffin's data structures. The client has tabulation marks with the following meanings: ▲ left margin, ▲ left tab, ▲ center tab, ▲ right tab. The gray numbered boxes indicate how the client might set Boffin's corresponding stop entry bounds, where the position of the number with respect to those bounds indicates the value of the centering parameter.



The `relativeCentering` flag causes the centering parameters to inverted for backwards lines. If this flag were on in the above example, then the stop ranges would have the following effect.



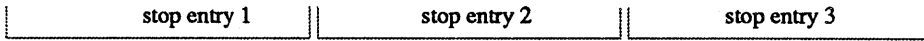
### b) Overflow Conditions

When the centering offset would cause the segment to overwrite the previous text, then the segment is handled specially, depending on the `stopOverflow` parameter set in the layout specification (refer to the following example)

If the value is `stopOverstrike`, then the text is simply overwritten ( $\beta$ ). If the value is `stopAdjacent`, then the segment is laid out adjacent to the previous segment, with the tab character

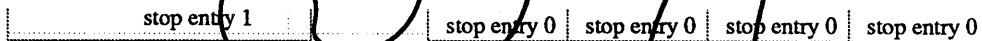
## Boffin ERS

in between ( $\gamma$ ). If the value is `stopSkip`, then the segment is associated with the next stop entry ( $\delta$ ). If the value is `stopBreak`, then a line break is returned at that point ( $\epsilon$ ). Since Boffin returns the number of the last stop when a line break occurs, other behavior can also be supported ( $\zeta$ ).



- ( $\alpha$ ) Before the tab, and after it.
- ( $\beta$ ) Before the tab but too long, and after it.
- ( $\gamma$ ) Before the tab but too long, and after it.
- ( $\delta$ ) Before the tab but too long, and after it.
- ( $\epsilon$ ) Before the tab but too long, and after it.
- ( $\zeta$ ) Before the tab but too long, and after it.

If the segment is too long to fit into the final stop entry, then a line-break occurs regardless of the `stopOverflow` value. If there are too many segments for the number of stop entries, then there are three possible behaviors, depending on the `excessSegments` parameter set in the layout specification. If the value is `segAdjacent`, then the segments are laid out adjacently. If the value is `segBreak`, then a line break is returned. If the value is `segRepeat`, then Boffin applies the `defaultStopEntry`. This entry is repeated across the span of the line bounds.



- ( $\eta$ ) Before the tab, and after, and after,

### c) Text Leads

When the optional text lead parameter is specified in the stop entry, then the gap between the previous segment and the current segment is replaced by repeating text.

..... 123.45

\*\*\*\*\* 123.45

\_\_\_\_\_abcde

### d) Centering Characters

When the optional centering text parameter is specified for the stop entry, then the first occurrence of a matching text is pinned to the stop. This is commonly used for decimal tabbing. If a matching sequence occurs in a segment, then the bounds of that occurrence of the text are centered according to an ancillary centering parameter. If it does not, then the whole segment is centered appropriately according to the standard centering parameter.

## Boffin ERS

For example, if the centering parameter is 1 (left-flush) and the centering text is a period, then the above behavior occurs (notice though that the left sides of the periods are aligned):

*Open issue: can we limit these cases to just one character (simpler storage)? Should we allow the client to specify the bounds of the text, rather than depend upon our matching of text strings?*

### 4. Line Break

When Boffin encounters a final stop-entry line-overflow condition, then it handles line break using the word-break and hyphenation procedures. If a carriage return caused the line break, then justification is not applied to that segment, unless the user sets the `justParaEnd` parameter in the layout.

#### a) Basic Line Break

First Boffin will call a word-break procedure to determine the word crossing the boundary. It will use the default Script Manager procedure unless the client has supplied one specifically. Note that the word-break procedure distinguishes between word-break for purposes of selection, and word-break for the purpose of line-break. The latter will handle the so-called taboo characters in East-Asian languages.

The `keepCharWithPrevious` style forces Boffin to keep characters together. That is, Boffin will ignore word breaks where the character at the start of the word has this style.



The quick brown fox jumped

Line break must be performed after all other formatting has been applied to the text: ligatures, contextual forms, kerning, offsets, stop alignment, etc. Logically speaking, Boffin starts with a length of zero, and completely formats increasing lengths of text, searching for a case where a word causes the length to exceed the last stop entry width. Of course, heuristics are used to avoid the performance drawbacks of this approach by making reasonable approximations as to the boundary word.

Boffin will try squeezing the last word in, or (depending on the justification) expanding the remaining text out. This is subject to the minimum and maximum space distribution parameters (see below) contained in each font present in the segment. If the results are unsatisfactory (exceeding the minimum and maximum, resp.), then Boffin will turn to hyphenation.

Hyphenation will not be attempted if the layout style for the word does not permit it, or if the client has indicated no hyphenation for the layout as a whole. Otherwise Boffin will call the user-supplied hyphenation routine. If there is no user-defined routine, then Boffin will check for an Everest-installed hyphenation service for the language of the text specified in the layout style. If there is no Everest service, then no hyphenation will be performed.

The hyphenation routine returns a weighted list of hyphenation points and (possible) spelling changes. (See Everest specification for examples. Hyphenation may have amusing complications; note that in German, *Zucker* hyphenates as *Zuk-ker!*)

a-but-ted  
↑    ↑  
3    15

Boffin will then compute which of the hyphenation points are allowable, given the space distribution parameters, and use the weights to determine the best hyphenation point.

## Boffin ERS

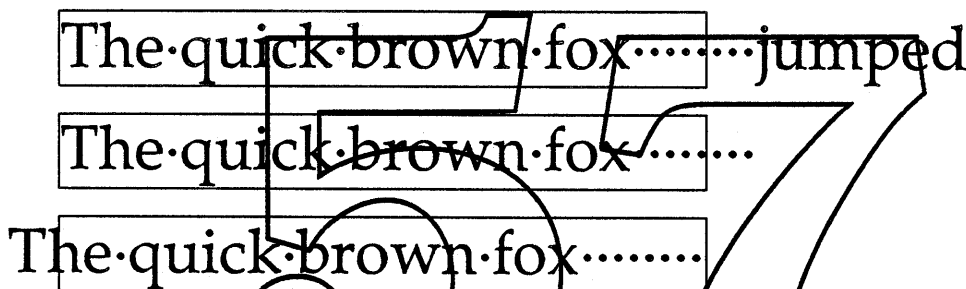
If there is no acceptable hyphenation point, then the line will be broken before the word. If there is only one word on the line, or if the `keepCharWithPrevious` prevents the use of normal word breaks, then the line will be broken at the nearest character.

### b) Knuthian Paragraph Breaks

*Knuthian paragraph breaks require additional support from Boffin. The form of this support is still under discussion. We are thinking along the lines of providing routines that provide sets of weighted word-break points and associated widths. Note that these widths are only approximate, since the surrounding context of the words could have effects on the widths. Similarly, sets of hyphenation points and widths can be supplied for particular words. As with all aspects of this ERS, feedback and ideas are welcome.*

### c) White Space Breaks

If the boundary word consists of sequence of white space glyphs, then it is handled specially. First, carriage returns are stripped by Boffin. If the white space consists of downstream glyphs, and the centering permits, then then line is broken at the end of the white space. Otherwise, it is broken at the last permissible glyph. Notice that line-break is sensitive to the centering:



Letting the center-dots represent white-space, then if the top line is broken at the end of the white-space, left-flush works correctly, since the white-space does not show up. However, right-flush has serious problems. When the centering does not permit, then Boffin will not necessarily include all the white space, in order to avoid this problem.

*Open Issue: The other alternative is to break the text the same way, but to offset the text so that the left edge does not go outside the boundaries.*

## 5. Distribution Spacing

Once the amount of distribution space  $s$  has been determined, Boffin must apportion different amounts of space to different positions within the line. The basic structure of the algorithm is fairly straightforward, although it requires some modification in due to boundary conditions.

### a) Basic Algorithm

From the font spacing table, Boffin derives a spacing factor  $f$  for each glyph. For example, this factor might be 15 for a space vs. 2 for other glyphs. Each spacing factor is multiplied by the point size of the glyph to produce the spacing weight  $w$ . The weighted spacing sum ( $\sum w_i$ ) is derived by summing the weights for each of the glyphs in the line. The proportion of space attached to the glyph  $g_j$  is proportional to the weight over the sum ( $s \cdot w_j / \sum w_i$ ). (When the sum of the weights is zero, then no adjustments can be made.)

The rain in Spain

## Boffin ERS

In the above example, there are (a) four glyphs of size 18, factor 2 (*r,a,i,n*); (b) ten glyphs of size 12, factor 2 (*T,h,e,i,n,S,p,a,i,n*); and (c) three glyphs of size 12, factor 15 ( , , ). The corresponding weights are 36, 24 and 180, so the weighted spacing sum is 924 ( $= 4 \cdot 36 + 10 \cdot 24 + 3 \cdot 180$ ). If the distribution space is 27 pts, then the amount apportioned to each of the glyphs according to category is:

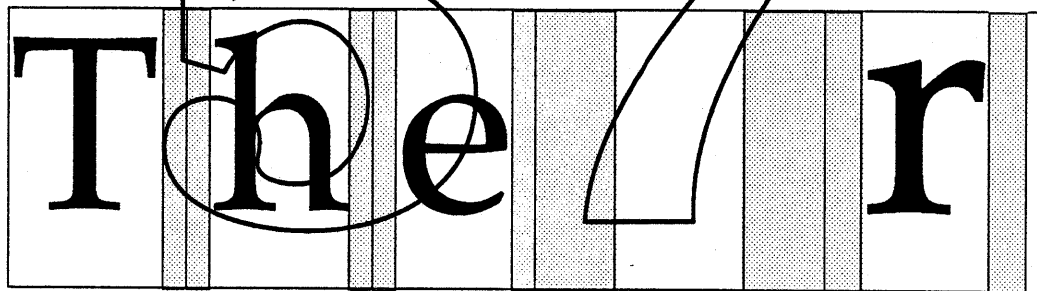
- (a) 1.05 pts ( $36 \cdot 27 / 924$ )
- (b) 0.70 pts ( $24 \cdot 27 / 924$ )
- (c) 5.26 pts ( $180 \cdot 27 / 924$ ).

A naïve application of this algorithm would involve a division per glyph. This can clearly be optimized in the typical cases.

### b) Initial Refinements

In the above the space apportioned to each glyph is split evenly between both side of the glyph. It neglects three factors: first, the glyphs on the boundaries should not be spaced in from the edges; second, for some languages the spacing should not be distributed on the basis of the glyph, but must be more finely allocated on the basis of the right or left side of the glyph; and third, proper caret placement may require shifting the proportion between different sides of the glyph. Modifying our algorithm appropriately, we return two factors for each glyph,  $f_r$  and  $f_l$ . *Editorial Note: it might be better just to give the full algorithm from the beginning.*

To change the above example, let us suppose that the right and left factors for spaces are 7 and 8, respectively, and that the right and left factors for other glyphs are both 1.



Then there are (a) eight sides of size 18, factor 1 (*r,a,i,n*); (b) eighteen sides of size 12, factor 1 (*T,h,e,i,n,S,p,a,i,n* [the left side of the *T* and the right side of the *n* are not included]); (c) three sides of size 12, factor 7 ( , , ); and (d) three sides of size 12, factor 8 ( , , ). The amount apportioned to each of the sides according to category is:

- (a) 0.54 pts ( $18 \cdot 27 / 900$ )
- (b) 0.36 pts ( $12 \cdot 27 / 900$ )
- (c) 2.52 pts ( $84 \cdot 27 / 900$ )
- (d) 2.88 pts ( $96 \cdot 27 / 900$ ).

The weights attached to each character are not constant: in particular, we need two sets of weights, one for positive distribution space and one set for negative. *It is an open issue whether we need any more categories than this, or whether the weights should a more general non-linear function of the distribution spacing. We may also want to allow the user to change the ratio of weights; in particular, the ratio of space vs. other character weight.*

### c) Distributing Over Connected Text



## Boffin ERS

In connected text such as Arabic or cursive English, the inter-glyph spacing can be performed by means of extension bars rather than by simply adding white-space. Even in non-cursive English, horizontal bars in characters such as H, L or T can be modified slightly to change character width. This is typically done in quality hand-lettering.

**TITLE**  $\Rightarrow$  **TITLE**

In Arabic, for example, there can be five different glyph side factors, two for white-space values such as space or unconnected glyphs, and three for different connected glyphs, since the height and shape of the glyph determines the length of the extension bar that should be apportioned to it. In Boffin 1.0, there is only provision for one extension bar per font, which is assumed to be horizontal (or vertical in the case of vertical text). Later versions may support improved functionality.

*This needs more explanation, especially as to the mechanisms for inserting extension bars properly, and the rôle of curved extension bars, as in cursive English.*

### d) Distribution Contrast

The proportion of distribution weights can be parametrically modified on a style run basis. For any weight  $W$ , a new weight  $W'$  can be produced using the distribution contrast parameter  $CP$ : This parameter has the following effects:

- $CP=2$  Maximum contrast: all weight given to the largest weight
- $CP>1$ : More contrast
- $CP=1$ : No effect
- $CP>0$ : Less contrast
- $CP=0$ : No contrast—all weights equal
- $CP<0$ : Less contrast, weights reversed in priority
- $CP=-1$ : Weights reversed in priority
- $CP<-1$ : More contrast, weights reversed in priority
- $CP=-2$  Maximum reversed contrast: all weight given to the smallest weight

This corresponds to the notion of inter-character vs. ~~inter-word~~ proportion for a simple bi-valued weighting scheme. For example, to have more inter-word space distribution, a high contrast would be chosen, or for more inter-character, a low contrast value will be chosen. For 50% to each weight, a contrast of zero can be used.

*Open Issue: We could give even more control to clients, by separating space-adjusted and extension-adjusted weights, and allowing parameters to control the weights within and between these two classes.*

### e) Distributing Over Hanging Punctuation

If kerning has resulted in hanging glyphs, then their weights are not taken into account in determining the distribution spacing. However, the client can specify that they receive a proportion of extra spacing. The glyph side weight is multiplied by the hanging space parameter when the glyph is outside the boundaries. For no extra hanging space (the middle of the following examples), the client supplies a parameter of zero.

## Boffin ERS

"The rain in Spain

"The rain in Spain

"The rain in Spain

*Open Issue: Does this need to be a number, or can it just be a flag?*

### 6. Device Dependent vs. Independent Positioning

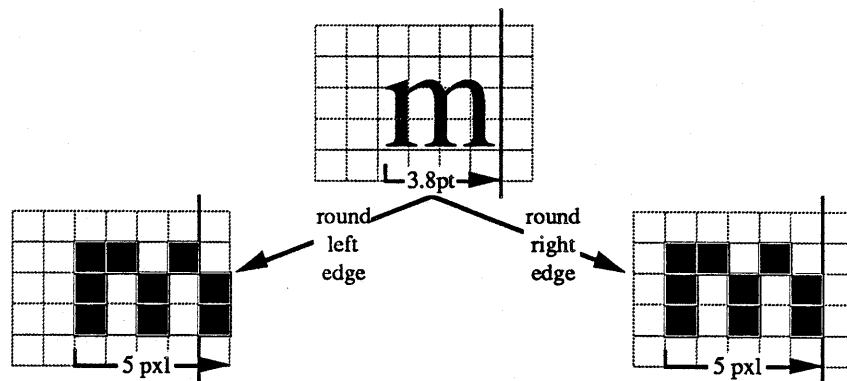
The device glyph widths may will generally fail to be exact multiples of pixel size on a given device. They may also differ significantly from the ideal glyph widths due to outline hinting. The inter-glyph spacing may also fail to be an exact multiple of pixels for the given device. To deal with these issues, the client can specify two different behaviors for Boffin to adopt. The behaviors are based on the `floatGlyphs` parameter available on a style-run basis. There is also a global parameter `fixLayout`, which overrides the `floatGlyphs` parameter to always produce device independent behavior.

*In version 1.0, floating is actually handled on a segment-by-segment basis: if any glyph within a segment has the `floatGlyphs` parameter off, then the entire segment is handled with `floatGlyphs` off.*

#### a) Device Independence

If the `floatGlyphs` parameter is off, then Boffin will round ideal glyph positions to the nearest pixel value. Notice that it makes a difference whether the left-side of the glyph is rounded, or the right side is rounded. To round the left edge, the glyph origin is set to  $\text{Round}(\text{position})$ . To round the right edge, it is set to  $\text{Round}(\text{position} + \text{idealWidth} - \text{deviceWidth})$ .

To show what a difference it can make, suppose that the ideal character width would be 3.5 pixels, but the device glyph width is 5 pixels. If the text is right-flush, we can get the following anomaly:



To account for that, thus making sure that right-flush text does line up on the right, Boffin will make minor adjustments, to round a particular edge based on the justification and centering parameters for the current stop.

## Boffin ERS

*Another option is to split the difference between the right and left rounded positions. This would show some anomalies at right and left bounds, but would reduce the error in caret positioning such as occurs in FractEnable under QuickDraw. This sets the glyph origin to be  $\text{Round}(\text{position} + (\text{idealWidth} - \text{deviceWidth}) / 2)$ .*

*This issue is only important at low resolution (e.g. < 150 dpi), where the hinting mechanisms can produce significant variance in glyph widths, and where a few-pixel difference is noticeable.*

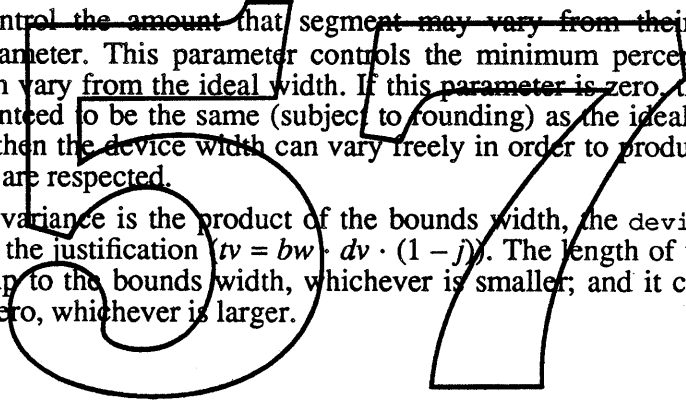
### **b) Device Dependence**

If the `floatGlyphs` parameter is on, then Boffin may position the text based on device characteristics. (See the Architecture section for a discussion of the flow of control.) This involves determining the difference between the device glyph widths and the ideal glyph widths, and distributing the extra (perhaps negative) space over the whole of the segment. This may result in a different segment width, but the stop bounds and centering parameter will be respected: the segment will not exceed the bounds (except perhaps a one character segment!), and will be centered properly.

The distribution of space over the segment is the same as the ideal space distribution, with special checks to make sure that the inter-glyph spacing is a uniform number of pixels, if possible. If the segment is not justified, and there is sufficient slop between the text end and the bounds, then minimal adjustments will be necessary.

The client can control the amount that segment may vary from their ideal width with the `deviceVariance` parameter. This parameter controls the minimum percentage of the stop entry width that the text can vary from the ideal width. If this parameter is zero, then the device width of the text will be guaranteed to be the same (subject to rounding) as the ideal width. If it is 1.0 (and justification is zero), then the device width can vary freely in order to produce the best appearance, so long as the bounds are respected.

Formally, the text variance is the product of the bounds width, the `deviceVariance` parameter value, and one minus the justification ( $tv = bw \cdot dv \cdot (1 - j)$ ). The length of the text can increase by the text variance or up to the bounds width, whichever is smaller; and it can decrease by the text variance or down to zero, whichever is larger.



## C. The Impact of graphics features

### 1. What are graphics features?

All of the above discussion has related to layout features; that is, those features for which Boffin has responsibility. There is another class of feature, *graphics features*, for which the graphics engine has responsibility. The most important thing to keep in mind about this distinction is that Boffin treats graphics features largely as black boxes: it assumes that it can get information about a particular graphics feature by querying the graphics system (or perhaps the font manager), but doesn't need to know about the internals of those features. It needs that information, for example, to determine if two consecutive graphics styles runs cause, say, ligaturing to break.

Certain graphics features, such as small caps or the use of swash variants (swash numbers look something like 123456789, although they cannot be represented well with this font) could be simply glyph substitutions within the current font. [Should variant forms—such as swash variants or Chinese character variant forms—be considered as graphics features?]. This is distinguished from the actual substitution of characters in the backing store string, which is *not* Boffin's responsibility.

Some graphics features may be applied to a set of glyphs taken as a group. This may be necessary, for example, to properly handle underlining or strikeout in a group of characters of different sizes.

### 2. Interaction among fonts, graphics features, and layout features

Some layout features will not work within certain graphics features. For example, a style that rotated each letter by 30° would break the formation of ligatures. In addition, some layout features will not work across certain graphics feature transitions (on/off transitions, or quantitative transitions such as degree of obliqueness). For example, having a red 'f' followed by a black 'i' would allow the formation of a contextual pair, but not a ligature.

Boffin must also be able to find out from the graphics system what effect the interaction of multiple graphics features will have on the various layout features. The way it determines this is to query the graphics system as to the degree of support various layout features are permitted within a particular style run (internally), and across a pair of style runs (externally). This means that all developers of new styles are responsible for supplying this information.

The problem of cross-style layout feature breaking is complex, since new styles may be added; a heuristic for the graphics system to use is the following: a layout feature breaks across two style runs just in case there is some style having different values between the two runs, and that style breaks the layout feature across style runs. For example, suppose style run A consists of italic and bold, where the weight of boldness is 1.5, and style run B consists of italic and bold, where the weight is 1.75. Since italic is on in both runs, and does not break ligatures internally, it does not break ligatures. However, since bold breaks ligatures externally, and the boldness parameters conflict in the two runs, ligatures are broken.

## Boffin ERS

### III. Functionality

This section briefly introduces the operations that can be performed using a layout, and some of the problems that Boffin solves for clients in doing so. For a more detailed description of the implementation of these operations, see section IV, and for a blow-by-blow account of the function calls, arguments, input and output, and special notes see section V.

#### A. Measuring and hit testing

Text measurement (and hit testing, which is related) can be quite complex in Boffin, including not only the measurement the total width of a string of text, but also the correlation of display widths with character code offsets within the original text string. This is necessary for text selection (mapping a mouse-down point to a position within the text) and for displaying character positions (with carets or highlighting).

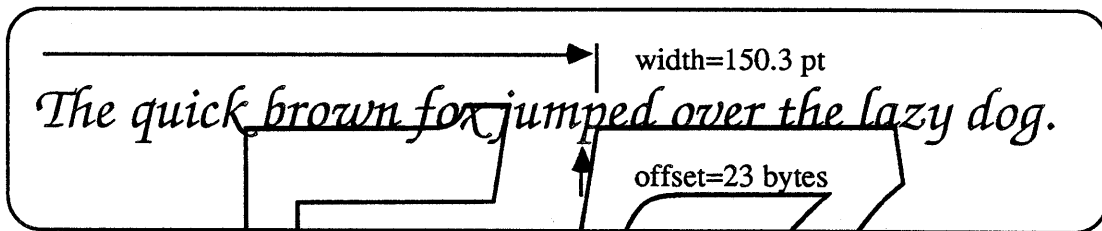


Figure 1—Pixel width and Character offset

Ligatures, applied marks, kerning, and reordering all complicate this task; the inverse mapping from display position to code offset is not trivial. For example, suppose that in a particular language, characters are written from right to left, *C* and *H* invert their order, and the characters *E* and *A* combine to form the character *Æ*. If we measured the word *BEACHES* after these transformations, we would get something like figure 2 (the two offsets are for leading (upper) and trailing (lower) edges of characters).

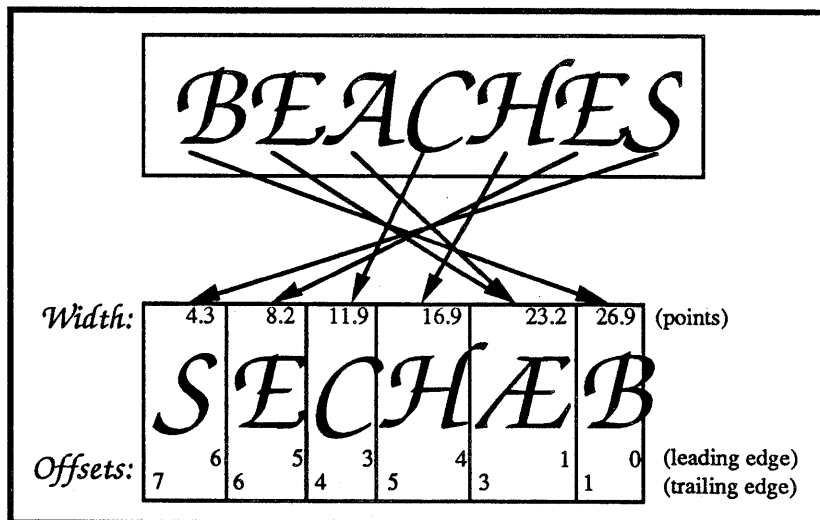


Figure 2—Offsets under layout features

## Boffin ERS

There are three general features brought out by this example. First, widths are not monotonically increasing: the width to the end of the last character is zero, while the width to the start of the first character is 26.9 points. Second, the context of the entire text (e.g. the start and total length) is necessary to convert between widths and offsets: the width to the start of the *C* is 11.9 pts, but only if the *H*, *E*, and *S* are included. Third, a text offset may correspond to two widths: the offset 3 corresponds both to 16.9 points (the leading edge of the *Æ*), and to 11.9 (the trailing edge of the *C*).

Another point is illustrated in figure 3. If the graphics system contains instructions to transform the appearance of a line of text that are independent of the layout features encapsulated in the layout for that line, this complicates the process of hit-testing.

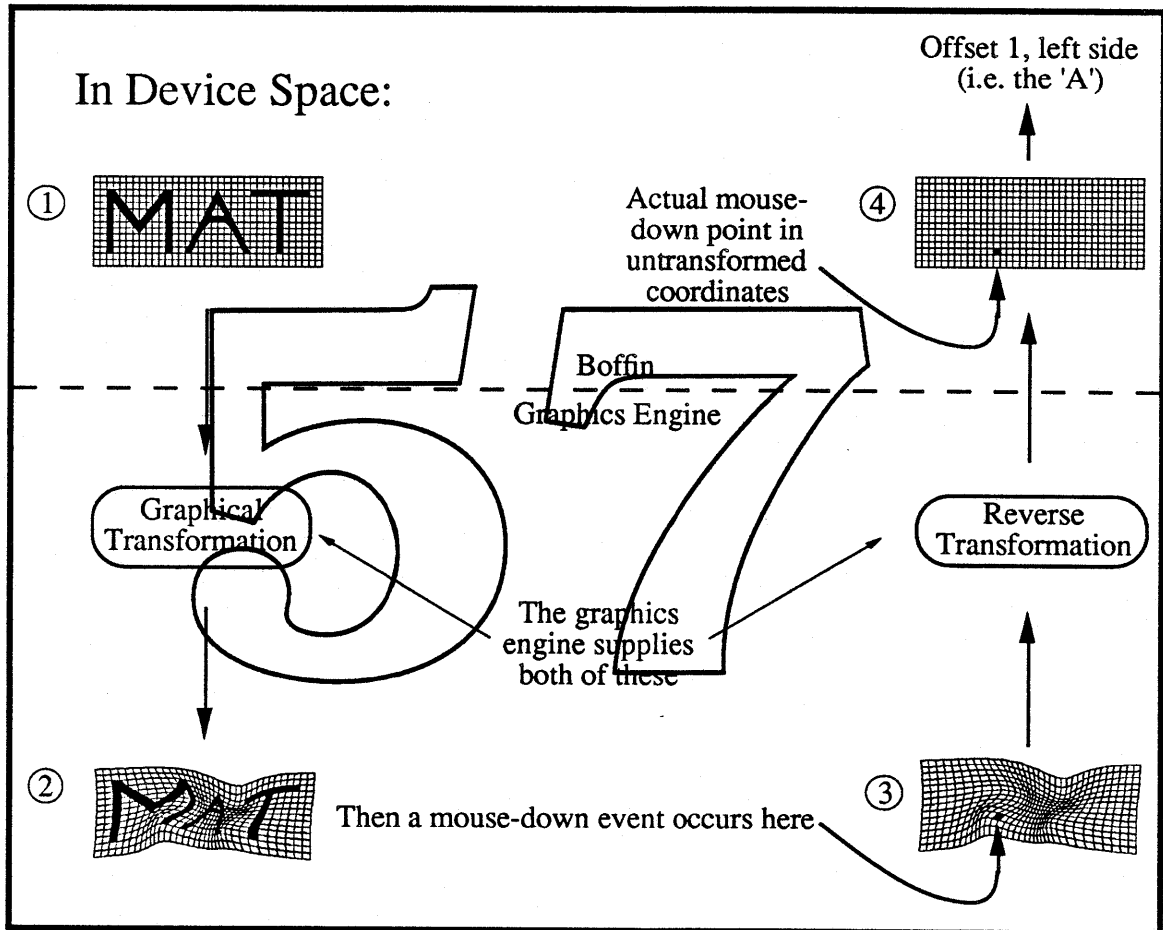


Figure 3—Hit-testing under graphics transformations

Boffin provides clients with a way to obtain three different "tightnesses" of bounding area for the text described in a layout. *Loose* means the single rectangle that encompasses all the text in the layout. *Snug* means an area that is the union of all the bounding rectangles for the individual glyphs. *Tight* means a true area that conforms to the shape of the glyphs.

This tightness of bounding area could be used by the graphics engine to assist Boffin in the determination of how close a particular hit occurred. For example, in figure 4, the *tight* bounding area would be useful in distinguishing the two different hits. This might be useful to advanced text editors that allowed a different interpretation of an exact hit as opposed to an approximate hit.

## Boffin ERS

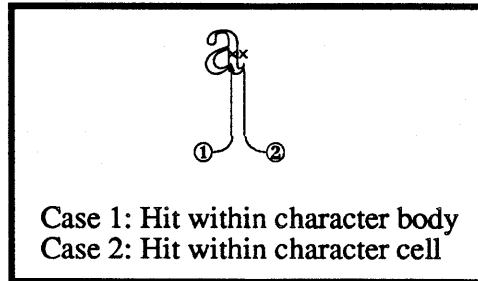


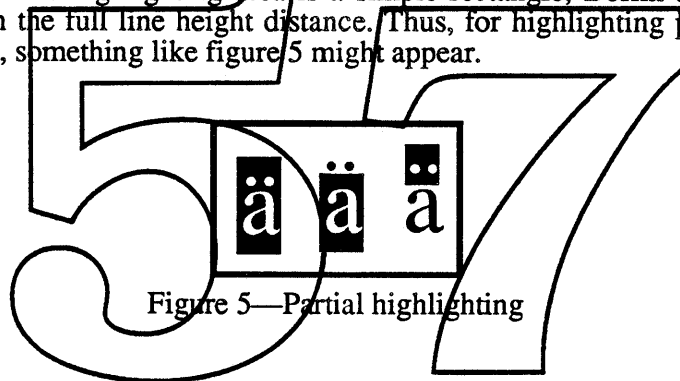
Figure 4—Tightness and hit testing

### **B. Highlighting**

Boffin provides a call that produces the highlighting area between two character offsets. Any graphics transformation that is in force (via the graphics engine) has an effect on the visual appearance of this highlighting area. The area need not be a simple rectangle, either; in particular:

- Non-rectangular areas are supported
- Non-contiguous areas are supported (and are actually required for languages like Arabic)

Even in the case where the highlighting area is a simple rectangle, Boffin allows a rectangle that doesn't necessarily stretch the full line height distance. Thus, for highlighting part of a baseform and applied mark combination, something like figure 5 might appear.



### **C. Caret handling**

In a manner similar to highlighting, Boffin provides a way to get an area that describes the caret. Carets may be modified by graphics transformations, just like highlighting areas. Boffin will eventually provide for carets that appear between baseforms and applied diacritical marks.

An example of the complications that can arise in caret handling is illustrated in figure 6. Location 'A' in the rendered text is actually associated with two different locations in the backing store: 0 (for a right-to-left insertion) and 3 (for a left-to-right insertion). Similarly, a given location in the backing store, say 3, can be associated with two different places in the rendered text, namely 'A' and 'D'. In order to accommodate this, the Boffin caret routine will return a split area representing a split caret; the convention is that the high part of a split caret represents the next character location in the current dominant line direction (which is left-to-right in our example). Note that a split caret is the usual case when dealing with mixed-direction text.

Boffin ERS

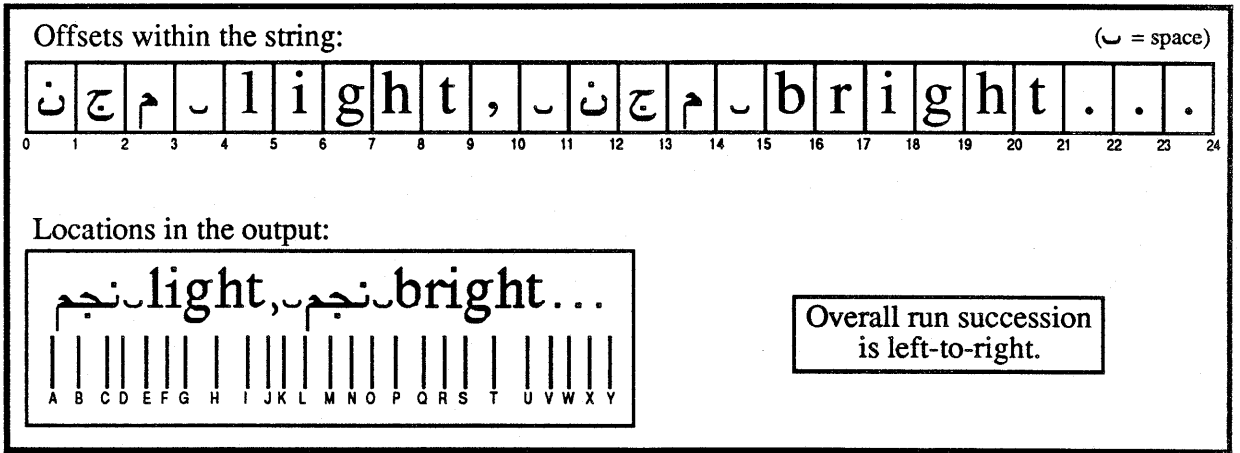
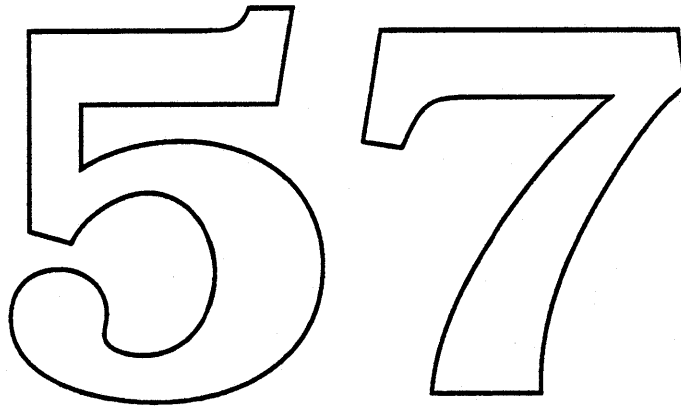


Figure 6—Correspondance between offset and caret location





## IV. Architecture and Data Structures

### A. The two-phase approach

Boffin is divided into two main parts (see figure 1). The high-level part accepts a specification of styled text (along with default and override information), and creates a non-positional LayoutSpec (i.e., a layout that contains the effects of all of the non-positional layout features such as ligatures and applied marks, along with information needed for later positioning operations). From the Font Manager, high-level Boffin obtains information about glyph availability as well as font-dependent tables specifying the operation of various layout features. From the graphics system, it obtains information about style interaction with layout features. From the Script Manager, it obtains information derived directly from the character codes. Information about graphic styles is not processed directly by Boffin, but is included in the LayoutSpec for later use by the graphics system and Font Manager.

The low-level part of Boffin is called by the graphics system to implement positioning for a particular device or set of devices, via a process called *instantiation*. An instantiation for an ideal, infinite-resolution device can be performed to determine line breaks; this lets us maintain WYSIWYG across different devices.

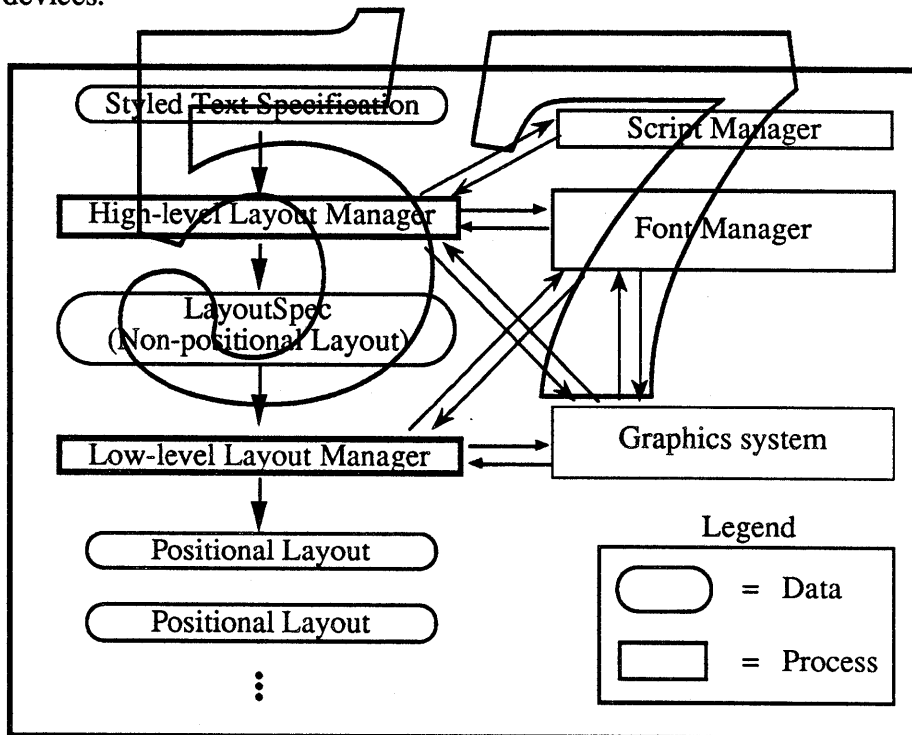


Figure 1—Overall Layout Actions

Note that applications do not have access to the internal structure of either the LayoutSpec or the positional layouts, although they will have a handle to the LayoutSpec.

#### 1. Generation of glyph objects for the non-positional layout

To go from a layout description to a non-positional layout, a sequence of character codes is converted into a sequence of *glyph objects* (or *globs* for short). The sequence of globs in the non-positional layout embodies all of the non-positional layout features such as reordering, ligatures, and

## Boffin ERS

grouping of applied marks with their baseforms. These globs have a uniform structure, and comprise the following components:

- A baseform, which is a glyph ID code from the font (*not* necessarily a character code)
- Zero or more applied marks, to be applied to the baseform's anchors
- An indication of the character offsets in the original string for all of the characters that were used to construct this glob.
- Other implementation data

Production of the non-positional layout will be handled by one or more state machines in Boffin; these state machines will make use of the reordering and forms/ligatures tables in the font.

### **B. General usage and summary of client-callable routines**

There are two high-level routines that an application can call in order to ask Boffin to create a layout. If the number of characters to be laid out has already been determined, `LayoutText` can be called to produce a `LayoutSpec`. If, however, the application needs to determine where a stream of text should be broken to fit in a specified area, the `LineBreak` routine will return the location in the text source where layout stopped. `LineBreak` accepts optional pointers to hyphenation and word break procedures, and can optionally produce a `LayoutSpec`. The break location returned by `LineBreak` can then be used as the starting location for the next layout call. This latter case will probably be the norm with word-processing type applications which are designed from scratch to take advantage of Boffin's capabilities, since they will not need to determine for themselves where lines break.

Both `LayoutText` and `LineBreak`, begin with a layout description supplied by the caller in the form of a `SingleLayoutDesc` data structure; this structure is described in the next section. `LayoutText` and (optionally) `LineBreak` return a handle to a `LayoutSpec`; however, the internal structure of this `LayoutSpec` is not available to the client. [Boffin may provide calls allowing clients access to some of the values in the `LayoutSpec`].

When a client is finished with a `LayoutSpec`, the space it occupies can be freed with a call to `DisposeLayout`.

Boffin supplies several routines for performing operations using a `LayoutSpec`. `LayoutArea` returns an area that bounds the text, with three tightness options. `LDrawText` is used to render a layout on a particular device (or set of devices). Given a point in ideal space, `LHitTest` identifies the character in the backing store that corresponds to that point. `LCursor` returns an area defining a cursor region for a particular side of a particular character in the backing store. `LHighlightText` returns a highlighting area that covers the text between two character offsets in the backing store (note that in bidirectional text, this may not be a single contiguous block). `LMeasureText` returns the size of the smallest rectangle that would bound the rendering area corresponding to the text between two character offsets in the backing store.

We anticipate no substantial degradation in performance for plain Roman text. Boffin 1.0 may support two additional routines that might be useful for speeding up certain layout operations with more complicated, the necessity for these routines will be determined in implementation. These routines, `LayoutInsert` and `LayoutDelete`, would allow insertions and deletions to layout without requiring recreation of the whole layout.

## Boffin ERS

### 1. Old Application Support

Some Boffin features are available in a semi-transparent manner to older applications that don't call `LayoutText` or `LineBreak`. The `SetLayoutState` routine sets the global state that Boffin will use to perform layout on direct calls to the graphics system made by these applications.

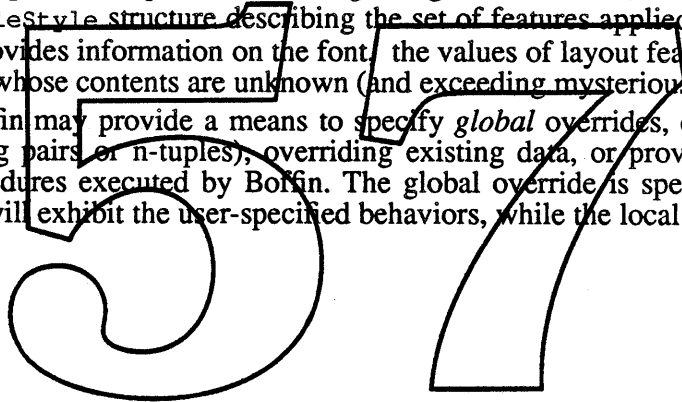
Boffin also contains several routines for use by the Print Shop, that allow them to use Boffin for heuristic layout of multiple `DrawText` calls. These routines allow for limited use of Boffin features: they provide for uniform character spacing and can also allow for printing of ligatures and application of kerning for old applications. See the discussion of routines for more information.

### C. Client data structures

The Layout description which the client supplies to Boffin is in the form of a `LayoutDesc` data structure; figure 2 describes its structure. The layout description breaks a string into *style runs*: substrings of text that have a particular set of graphic and layout features.

The `runList` field in the `SingleLayoutDesc` structure points to a `RunList` structure, which in turn references an array of `SingleRun` structures and an array of `SingleStyle` structures. For each style run, a `SingleRun` structure provides a pointer to the beginning of the text run, the length of the text run, and the index of the `SingleStyle` structure describing the set of features applied to the text run. Each `SingleStyle` structure provides information on the font, the values of layout features, and a pointer to a graphics feature record, whose contents are unknown (and exceeding mysterious) to Boffin.

Future versions of Boffin may provide a means to specify *global* overrides, either for adding new data (such as new kerning pairs or n-tuples), overriding existing data, or providing procedures that override the normal procedures executed by Boffin. The global override is specified such that *every* layout generated using it will exhibit the user-specified behaviors, while the local overrides happens for a single line.



# Boffin ERS

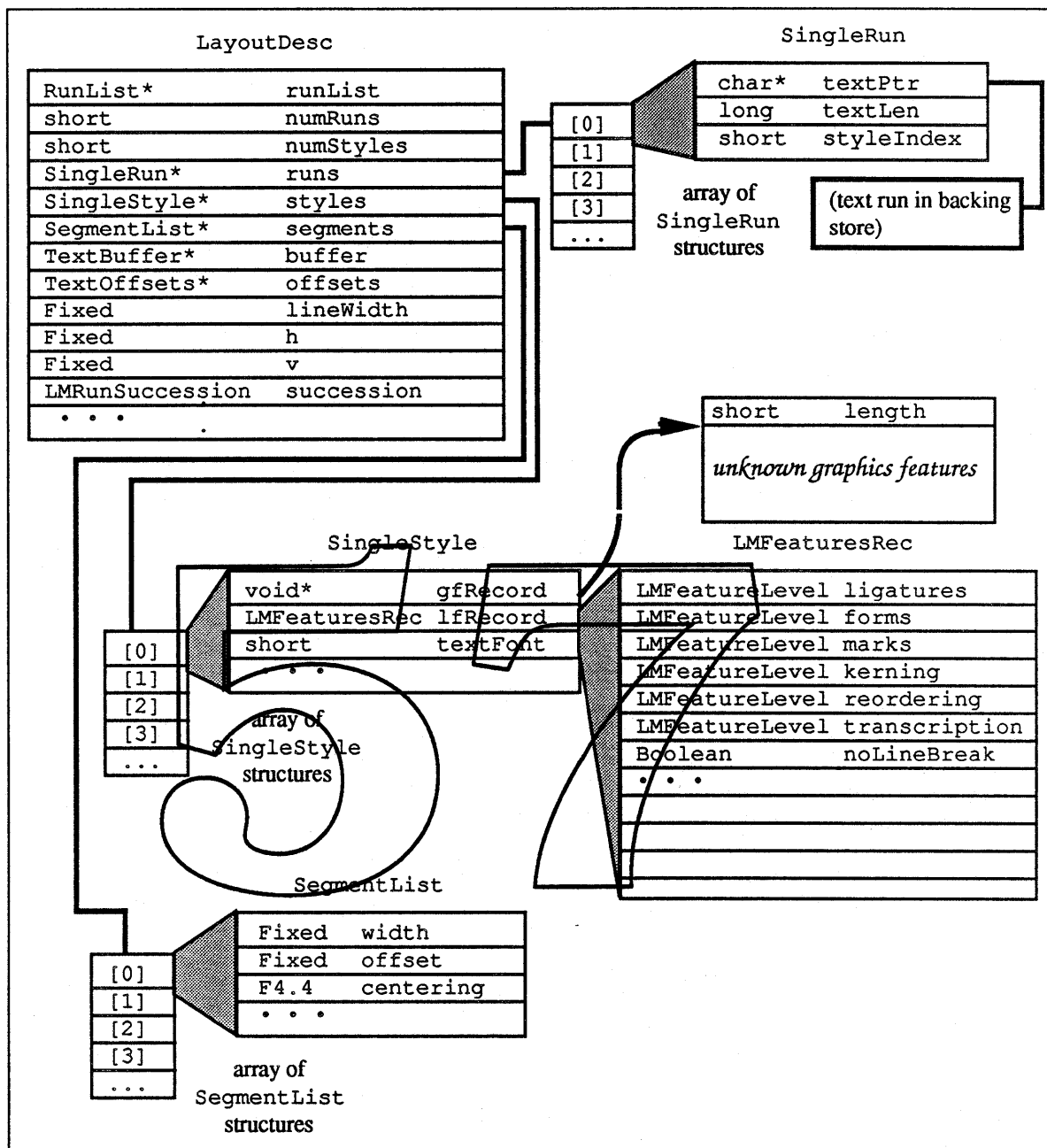


Figure 2—LayoutDesc data structure

## D. Flow of data

The discussions in this section assume a generic graphics engine with generic calls; the actual calling details will depend on the particular engine (such as QuickDraw or Skia). Also note that the discussions of device-dependent positioning in this section will tend to focus on somewhat pathological examples—small characters on relatively low-resolution devices—because these provide the best illustration of some of the problems of device-dependent layout. In most real cases, the rendered glyphs will look better than those in the examples here.

## Boffin ERS

To provide a context for the flow discussion, it will be useful to examine the sequence of operations required to render transformed and styled text, how much Boffin needs to know about these operations, and the requirements placed by Boffin on these operations. Note that values for coordinates, locations, and measurements are always specified using a device-independent measuring system, such as points or centimeters. When specifying device-dependent values in the system, the values will always be in multiples of some unit that depends on the device characteristics (and possibly the graphic transformation). Thus, a location two pixels from the origin on a 144 dpi device will have a location of 1.0 points (pt). Boffin never uses the concept of pixels per se; it performs device-dependent operations using fixed positional increments within a device-independent measuring system. The only difference between device-dependent and device-independent operations is that the positioning increment is very small for device-independent operations.

Suppose (as in the following example) we want to render the text "Eat your words" in a particular font, at a particular point size, on a particular device, justified into a certain area, with "Eat" in bold, and the whole text scaled by 2/3 and rotated 30°:

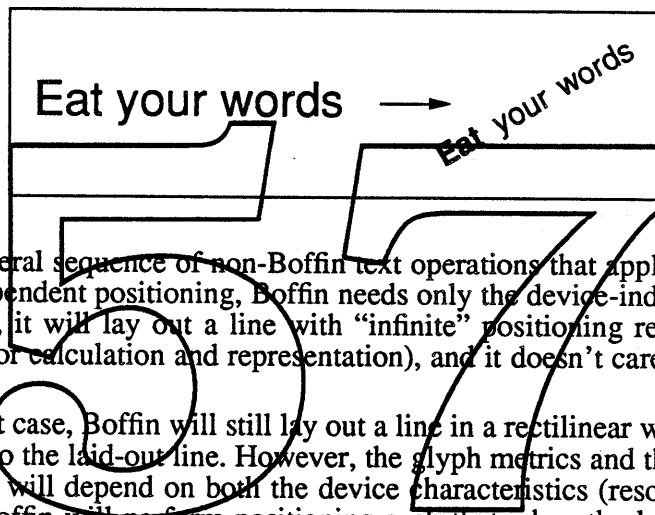


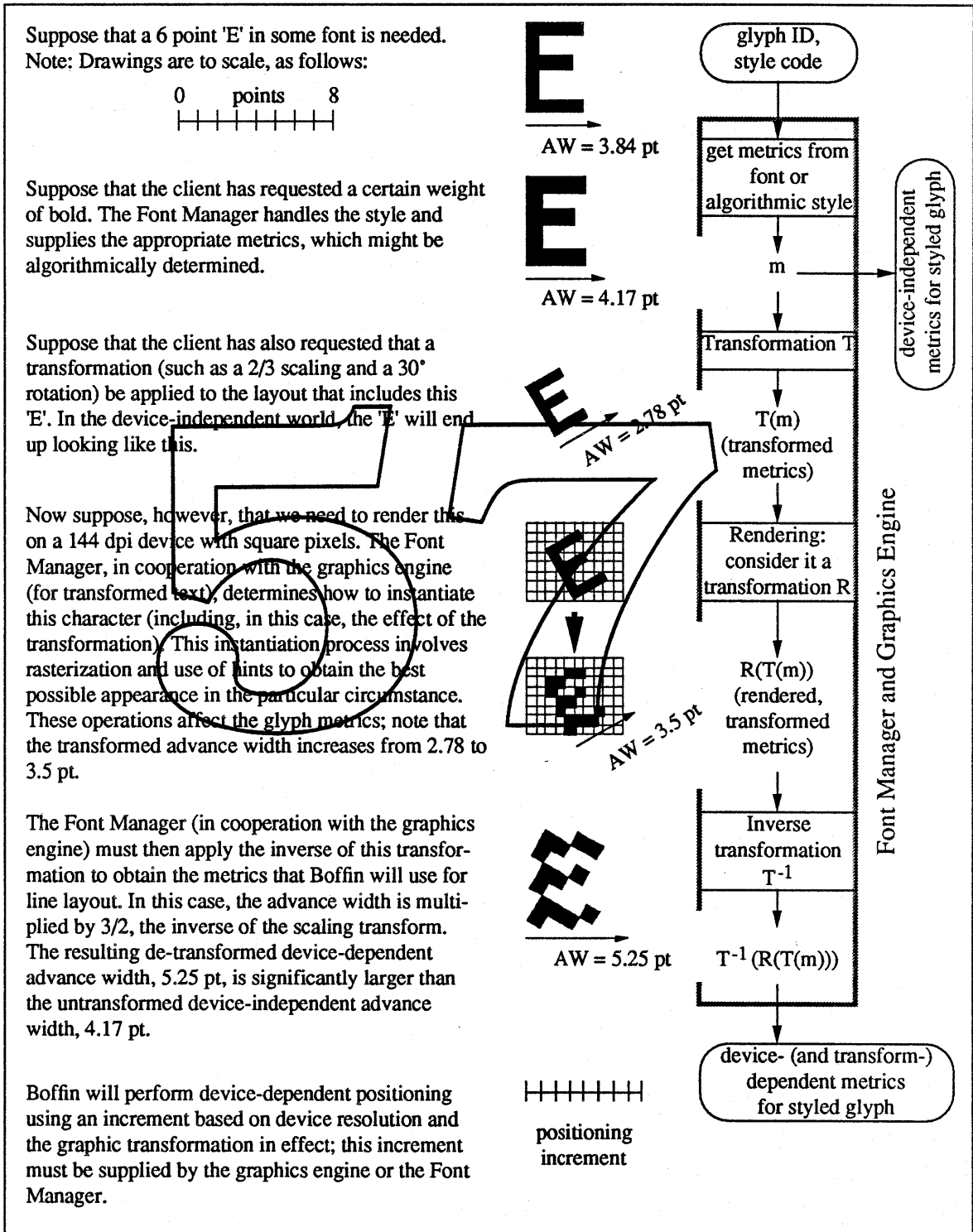
Figure 3 shows the general sequence of non-Boffin text operations that apply to the letter 'E' in this example. For device-independent positioning, Boffin needs only the device-independent metrics for the styled glyph. In this case, it will lay out a line with "infinite" positioning resolution (limited by the number of bits available for calculation and representation), and it doesn't care about the effects of the graphic transformation.

In the device-dependent case, Boffin will still lay out a line in a rectilinear world, before any graphic transformation is applied to the laid-out line. However, the glyph metrics and the positioning increment that it uses for this layout will depend on both the device characteristics (resolution, dot size) and the graphic transformation. Boffin will perform positioning such that when the layout is transformed and displayed on the device for which it was laid out, it will look as good as possible. The positioning increment should depend only on device characteristics and the graphic transformation. The device-dependent glyph metrics needed by Boffin should be determined by the Font Manager and graphics engine by computing the device-dependent metrics for the transformed glyph, and then applying the inverse of the graphic transformation to these metrics.

Note that this means that for every graphic transformation, there must be a way to run glyph metrics (and mouse-down points, for hit testing) backwards through the transformation, in order to find the inverse transformation of these metrics. In addition, the glyph metrics used by Boffin for laying out a line mustn't depend on the glyph position in the line. Even if the graphic transformation is such that a given glyph will have a different rendered shape or size at different places in the line—say, for example, with a perspective transformation—the Font Manager must return some kind of standardized metric. Even though this may result in suboptimal appearance under such transformations, it also means that positioning will only require a finite amount of time (an important consideration for certain business markets). Note that in the QuickDraw world, transformations are limited to scaling, which makes things much simpler.

# Boffin ERS

Figure 3



## Boffin ERS

Now—some further words on the positioning increment. When the graphics transformation involves only scaling or other rectilinear operations, the positioning increment is easy to determine. In the case of scaling, for example, the position increment needed by Boffin is the device resolution divided by the scaling factor. For other graphics transformations—such as rotation—we actually need two kinds of position increment.

Consider figure 4. In this example, a rotation transformation is used so that text is laid out along a path with a slope of  $1/3$ , and we want to position some small Helvetica 'l' glyphs on a low-resolution device. For consistent position of the glyphs with respect to the baseline, we are limited in the locations we can use; the positioning increment will be (pixel size)  $\times \sqrt{10}$ . If we are willing to tolerate some deviation of glyphs above and below the baseline, we have three times as many choices, with an average or normalized positioning increment that is  $1/3$  as big.

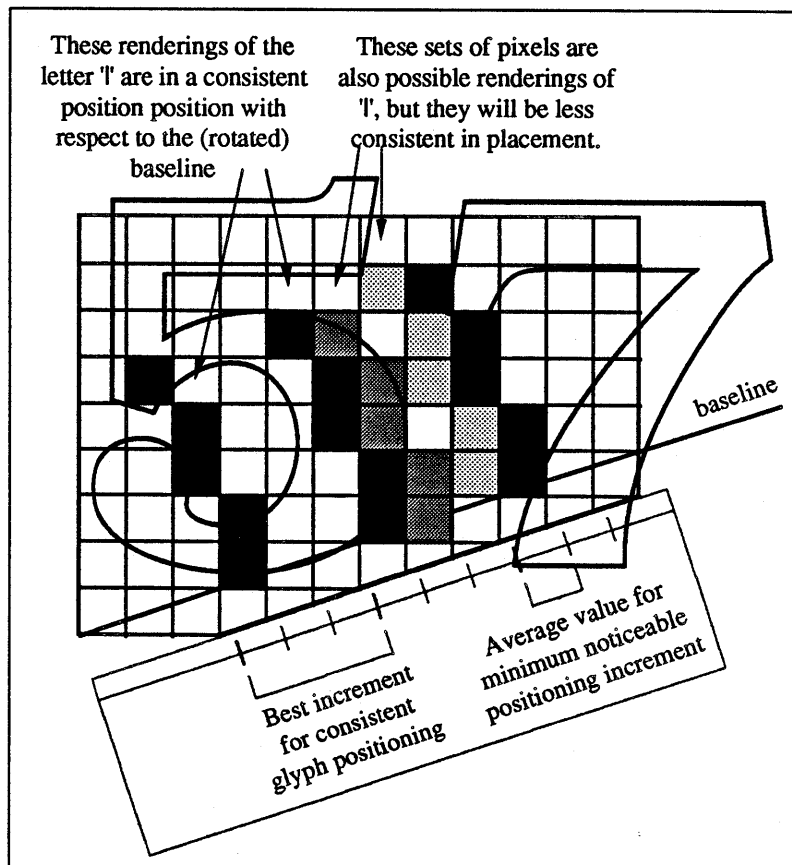


Figure 4

This illustrates a general problem for non-rectilinear transformations: the graphics engine will have to supply two positioning increments. One is a “good” positioning increment, the other is a “minimum noticeable” positioning increment. Of course, this particular example is somewhat contrived; in general, these values may be more difficult to determine. For example, a line with an irrational slope such as  $\pi$  will have no ideal “good” increment, although a “good” positional increment corresponding to a line with a slope of  $22/7$  would be quite close. The graphics engine will have to decide what values

## Boffin ERS

to return for each type of increment. Boffin will then make some determination of which value to use for positioning, based on the size and ratio of the two values.

### E. Points to remember

- Boffin always uses a device-independent measuring system.
- Boffin always works with an untransformed (i.e., rectilinear) layout.
- However, the positioning increments used by Boffin may reflect on both device characteristics and the graphic transformation, so that when a layout is transformed and displayed on a particular device, it will look its best.
- A device-independent layout is just one with a "very small" positioning increment.
- Boffin will layout either in device-independent or device-dependent spaces, according to the requests of the client and capabilities of the graphics systems.
- For each transformation, there must be some way to obtain the inverse transformation for points and glyph metrics.
- The graphics system has fundamental control: see below.

### F. Examples of Data Flow

The examples on the following pages illustrate the interactions between Boffin and the graphics system. An open issue in the following examples is the method by which information identifying the desired graphic transformation is supplied to the graphics system: whether implicit or explicit. Similarly undefined is the method by which the Font Manager receives information about device characteristics and the graphic transformation.

If the transformation is implicit, then Boffin is left out of the picture as much as possible: the client could call a graphics engine function `GESetTransformation` to specify the transformation before calling any of Boffin's positional layout functions, and the graphics engine could supply the graphics transformation and device characteristics to the Font Manager before calling `InstantiateLayout`. This is the method that is assumed in the following examples.

A possibly cleaner method is for the client to include some graphics transformation identification in calls to Boffin, for the graphics engine to include this identification as well as device characteristics in calls to `InstantiateLayout`, and for Boffin to include this information in calls to `GetCharMetrics`.

In any case, Boffin must have some way of identifying transformations and device characteristics if is going to handle caching of layouts.

#### 1. Drawing

The data flow for drawing is diagrammed in figure 5. In order to draw text, the client must first specify the graphic transformation to be used; it is the client's responsibility to keep track of which graphic transformation (if any) is used for each layout. The client then calls the Boffin routine `LDrawText` with a `LayoutSpec` and a location. Boffin just passes this information on to the graphics engine, which determines which device or set of devices is required to display the layout. For each device, it calls `InstantiateLayout` to get the device-specific layout for the device, transforms the layout if necessary, and then displays the layout. Note that the device-specific layout may have been cached from a previous operation, in which case no actual instantiation needs to be done.

#### 2. Highlighting

The data flow for highlighting text is diagrammed in figure 6. As above, the client first specifies the graphic transformation to be used; it then calls the Boffin routine `LHighlightText` with a `LayoutSpec` and a pair of offsets. Boffin just passes this information on to the graphics engine. For each device, the graphics engine gets the (possibly cached) device-specific layout. It passes this layout and the



## Boffin ERS

offsets to the Boffin routine `LDGetArea`, which will return an area (which may span multiple devices) . The graphics engine then determines the portion of this area that is on the current device.

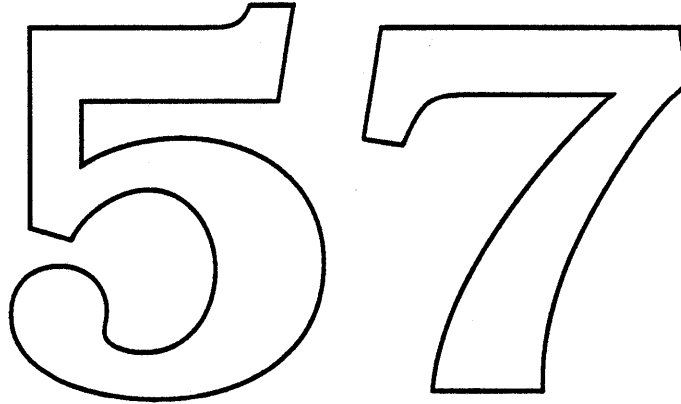
The union of these device-specific areas is then returned to the client.

### 3. Hit testing

The data flow for hit testing is diagrammed in figure 7. As above, the client first specifies the graphic transformation to be used; it then calls the Boffin routine `LHitTest` with a `LayoutSpec`, a location, and a mouse-down point; Boffin passes this information directly to the graphics engine.

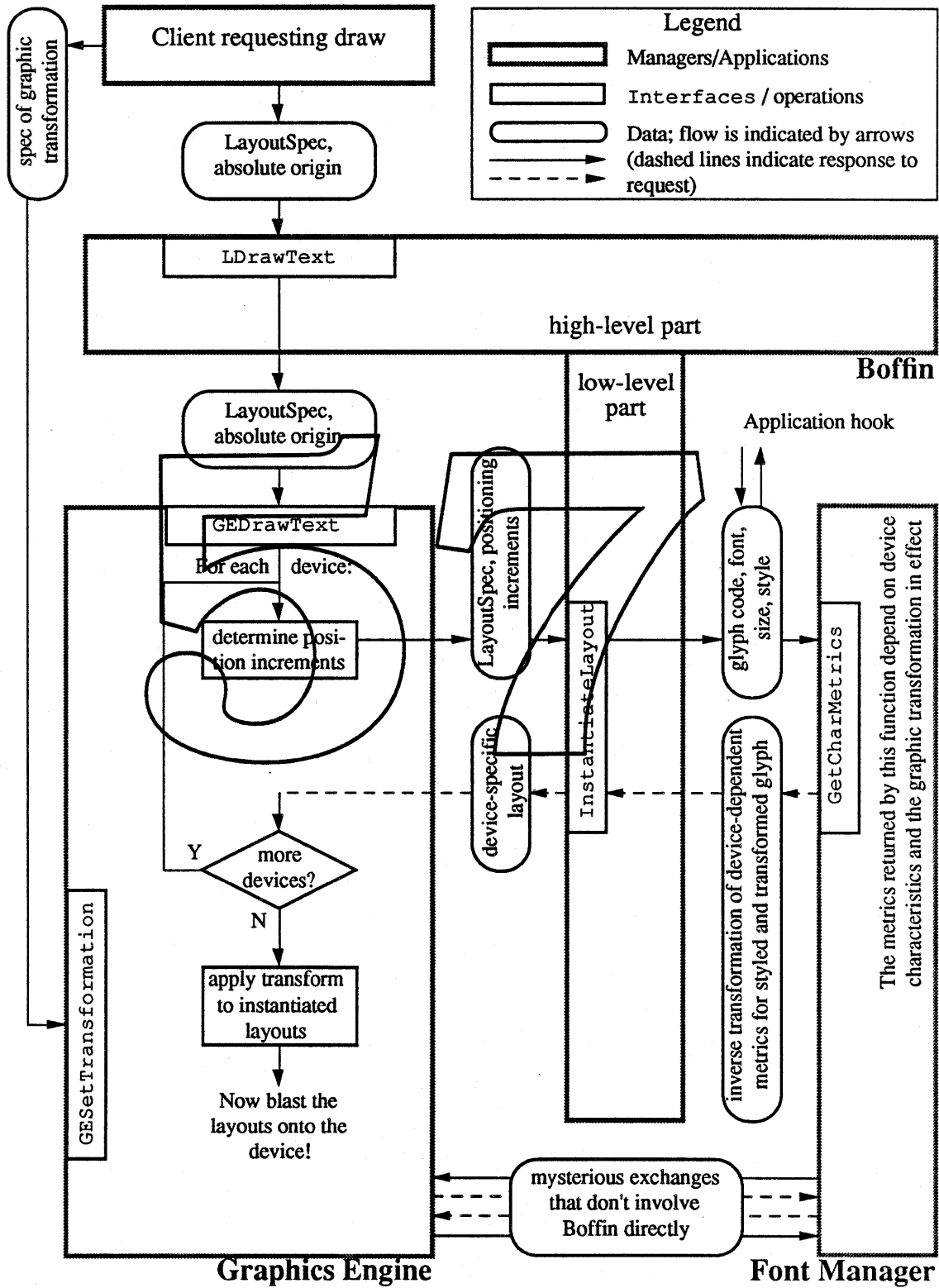
The graphics engine uses the mouse-down point to select a device and get a (possibly cached) device-specific layout. It must also apply the inverse of the graphic transformation to the mouse-down point. It then passes the device-specific layout and the “de-transformed” mouse-down point to the Boffin routine `LDHitTest`, which returns a text offset; this is passed on to the client. If this de-transformed mouse-down point is outside the device-specific layout, `LDHitTest` indicates this, and also indicates the “nearest” offset within the layout (according to Boffin’s determination of nearest).

An important point is that the client must supply the `LayoutSpec` and associated graphic transformation with the `LHitTest` call. This means that the client must either know in advance which (possibly transformed) areas correspond to which layouts, or it must walk through all of the layouts in some (possibly prioritized) order, calling `LHitTest` until it gets a hit inside a layout.



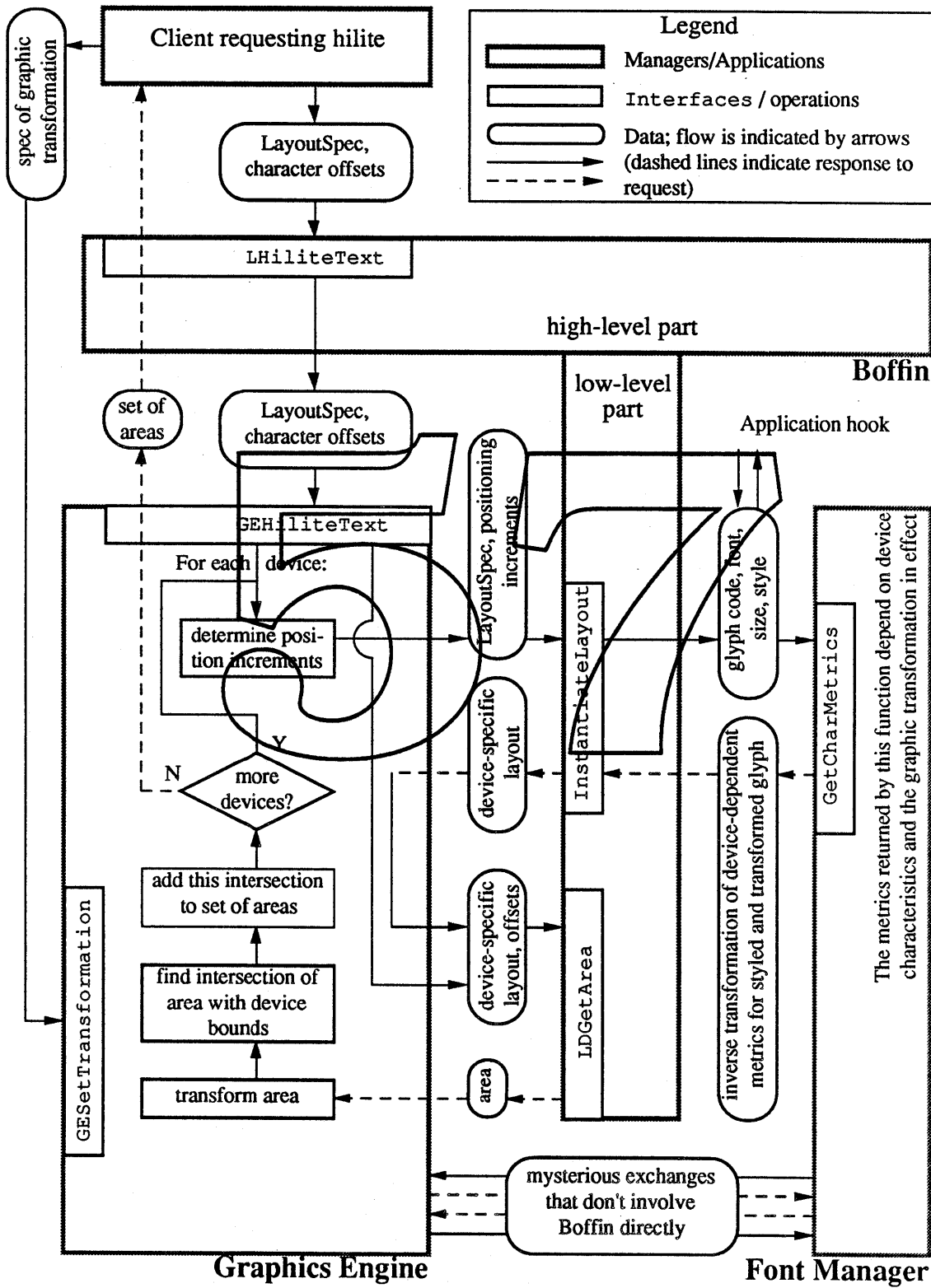
# Boffin ERS

Figure 5—Data flow for drawing text



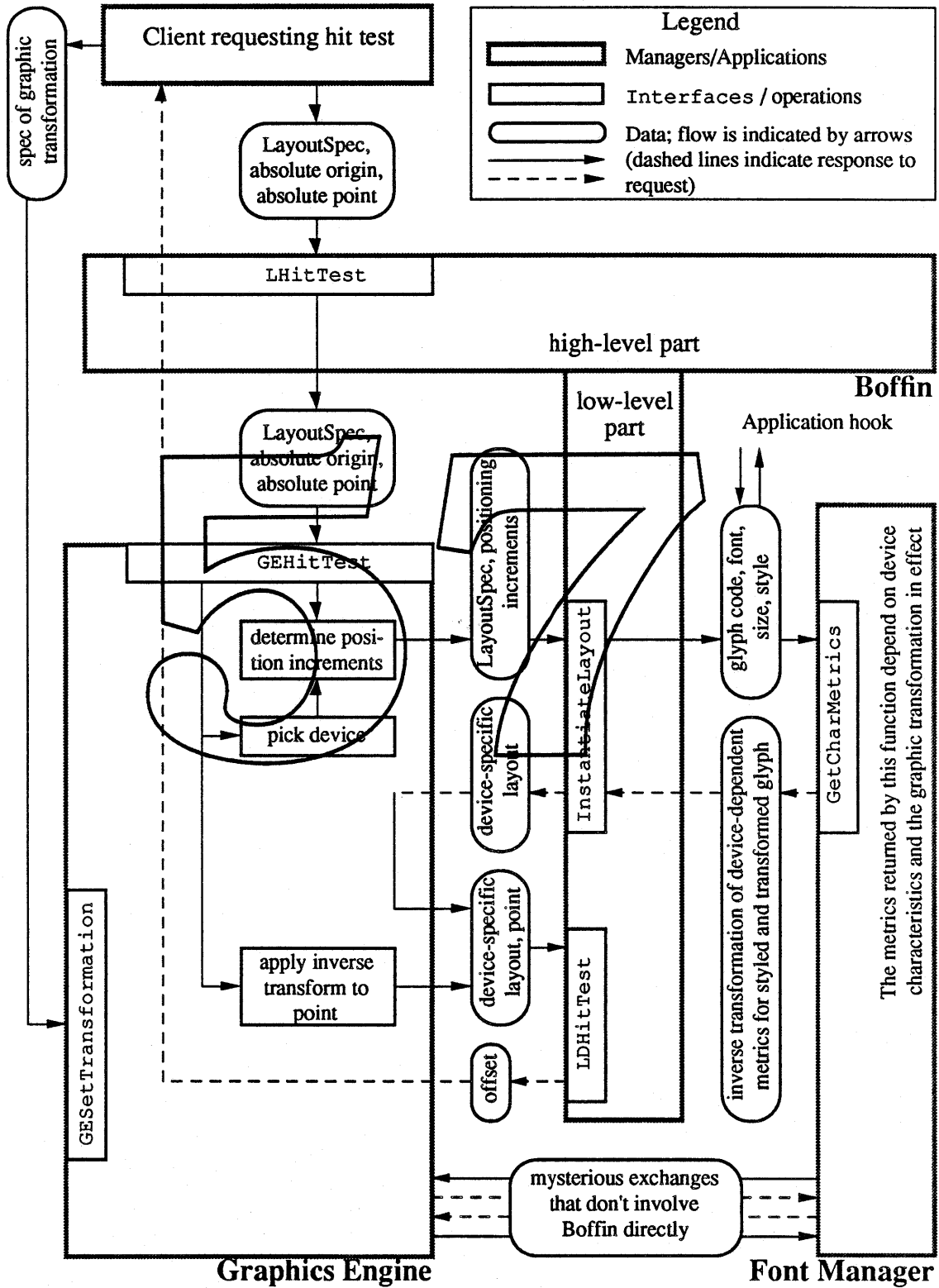
# Boffin ERS

## Figure 6—Data flow for highlighting text



Boffin ERS

Figure 7—Data flow for hit testing



## Boffin ERS

### V. Routines

This section describes in detail the various procedures and functions that Boffin provides to clients. It also describes those routines that Boffin requires to be provided from other parts of the system. Names of functions and procedures in this section are printed in *Courier* to be more noticeable. Note that for the present we omit any discussion of error codes returned by Boffin in these routines.

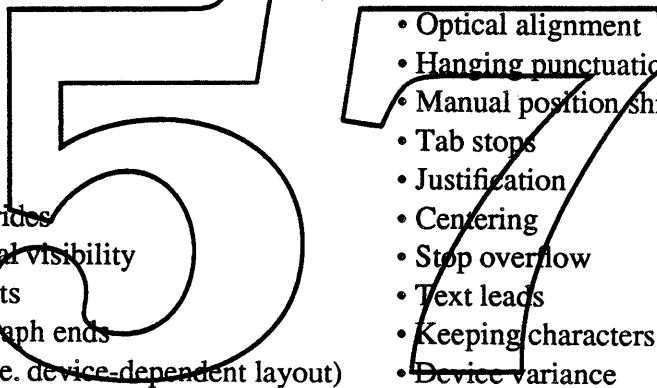
This section describes the routines that will be used with QuickDraw. Many of them will probably be used without change for Skia. For Albert, the routines will have to be converted into the object-oriented paradigm.

### A. Data Structures

#### 1. LayoutFeatureRec

The LayoutFeatureRec is the structure that contains values describing how much (if at all) the various layout features, both positional and non-positional, are to be performed by Boffin.

The layout features over which clients have control via this structure are as follows (see section II for a more detailed description of what these all do):

- 
- Ligatures
  - Forms
  - Applied marks
  - Kerning
  - Reordering
  - Direction overrides
  - Ligature internal visibility
  - Excess segments
  - Justified paragraph ends
  - Float glyphs (i.e. device-dependent layout)
  - Distribution contrast
  - Optical alignment
  - Hanging punctuation
  - Manual position shifts
  - Tab stops
  - Justification
  - Centering
  - Stop overflow
  - Text leads
  - Keeping characters together
  - Device variance
  - Relative centering

In future versions, this structure will expand to include the more advanced items discussed in section II, including:

- Cross-stream kerning
- Transcription
- Shape-based kerning
- Monospace gridding

#### 2. SingleLayoutDesc

The SingleLayoutDesc is the principal means whereby the client communicates the text and styles information to Boffin for the creation of a layout.

#### 3. LayoutSpec

The LayoutSpec is the object returned by Boffin that contains the encapsulated description of a line. Initially, after a LayoutText call, this structure only contains the non-positional layout, but after operations are performed using this layout, the positional instantiations (at various different resolutions) are also associated with it, via a cache mechanism. Note that the internals of this structure are invisible to the client; only Boffin has access to the data structure (although of course information about the line described by the layout is available via calls).

## Boffin ERS

### B. High-level Boffin routines

These are the only routines that a client will normally call. They deal with the creation and destruction of layouts, and the various things that can be done with those layout once they have been created.

#### 1. SetLayoutState

This routine allows clients to set the current global state that Boffin will use to perform layout on direct calls to the graphics system. It is intended mainly to support old applications that wish to take advantage of Boffin's text layout capabilities without recompilation. For a discussion of the different levels of layout feature support, see section 2.1.1. There will be a cdev (or equivalent) that allows the user to make this choice.

##### a) Input

LayoutFeaturesRec

##### b) Output

(none)

#### 2. LineBreak

When the client needs to find out where a stream of text should be broken to fit within some specified area, LineBreak is called. The client may opt to have a LayoutSpec returned with the break location, which spares the client the need to make a separate call to LayoutText after calling LineBreak. Note that it's always faster to make a single call to LineBreak than it is to separately make line break decisions and then call LayoutText (see below for a description of LayoutText).

##### a) Input

Layout description

Area

Hyphenation proc (optional)

Word break proc (optional)

Flag indicating whether to return a LayoutSpec

##### b) Output

Index of the last char that could be included in the LayoutSpec

LayoutSpec (optional)

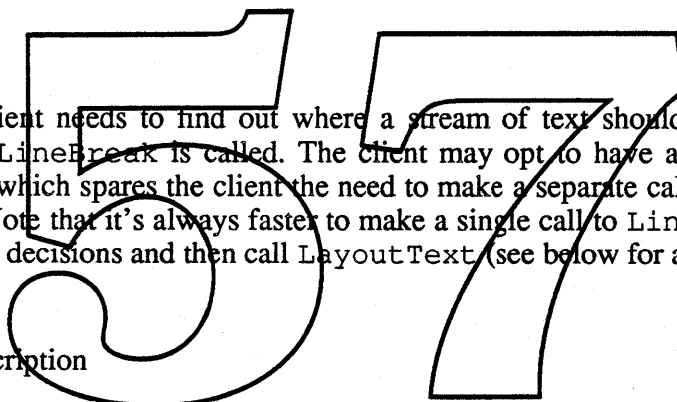
##### c) Font Manager & graphics assistance required

GetCharMetrics

#### 3. LayoutText

When the client has already decided on the number of characters it wants to deal with, LayoutText is called to generate a LayoutSpec describing the appearance of those characters in absolute space.

##### a) Input



## Boffin ERS

Layout description  
Area

b) Output

LayoutSpec

c) Font Manager & graphics assistance required

GetCharMetrics

4. DisposeLayout

Once the user is finished with a LayoutSpec, the space it occupies (which is *always* under Boffin's control) can be released with a call to DisposeLayout.

a) Input

LayoutSpec

b) Output

(none)

5. LayoutArea

This function determines the bounding area of the text. For a discussion of the meaning of tightness options, see section III.C

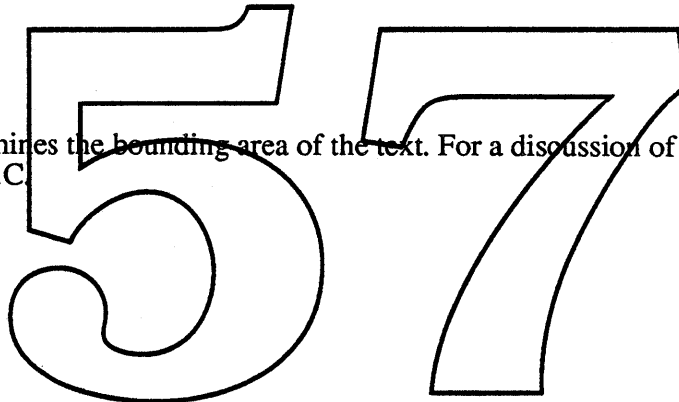
a) Input

LayoutSpec

Tightness option

b) Output

Bounding area of the specified tightness



6. LDrawText

Once a LayoutSpec has been created, the client may call LDrawText to show the text represented by the layout onto the current medium (or media). An absolute layout origin is also specified, which allows the text to appear wherever it is desired (remember that a LayoutSpec specifies glyph positions relative to some internally-specified origin, and not in relation to the global graphics space).

a) Input

LayoutSpec

Absolute layout origin

b) Output

(none)

c) Font Manager & graphics assistance required

Figure 7 in section IV describes the flow of control that occurs as the result of an LDrawText call. The graphics engine and font functions that Boffin requires are:

## Boffin ERS

- GEDrawText
- GetCharMetrics
- GESetTransformation (indirectly)

LDrawText calls must be encapsulated in PICTs in much the same way as DrawText calls are (this is currently an open issue). Clients might want to be able to recover elements of the picture, which would be hard to do in the case of the text that was laid out, since the actual layout doesn't store that text (rather, it stores the glyphs that are the results of the layout process). We could add information that allows an inverse conversion, but this would burden layouts that are not captures.

### 7. LHitTest

The LHitTest routine is similar in purpose to the older Pixel2Char routine. It takes a point in absolute space (perhaps corresponding to the location of some event like a mouse-down) and identifies the corresponding character offset within the text that drove the layout.

#### a) Input

LayoutSpec  
Absolute layout origin  
Absolute point

#### b) Output

Char offset of hit character  
Flag indicating directionality of the hit character  
Flag indicating whether hit occurred on left or right side of hit char  
Flag indicating whether the hit occurred outside the loose area of the layout

#### c) Font Manager & graphics assistance required

This is one of the most interesting functions to implement. Any transformation that was applied to the text by the graphics engine must be undoable, via a reverse transformation function supplied by the engine. Note that normally, the Boffin routines have no knowledge (nor any need of knowledge) about anything the graphics system does to the whole layout (whereas we *do* need to know about individual character transformations, since they might change advance widths).

The overall process is simplified somewhat by Boffin's implicit assumption that rectilinear text layout is all that ever occurs. Boffin lives in innocence of any graphics transformation that may exist for a particular layout.

### 8. LCaret

The LCaret routine is similar in purpose to the older Char2Pixel routine. It takes a character offset (within a layout), and returns an area defining the caret region for that character. In the early implementations of Boffin this area will likely be a simple rectangle (or two, for mixed-direction split carets), but future capabilities may increase this to a general area (for example, for italic slanted carets).

#### a) Input

LayoutSpec  
Absolute layout origin

---

1. See section III for a discussion of loose, snug and tight areas.



## Boffin ERS

b) Output

Area (which may be a disjoint collection of sub-areas, remember)

c) Font Manager & graphics assistance required

Any transformations applied to the whole layout (as well as individual character faces that involve special transformations) will have been made to the returned caret area. Note that it is up to the graphics engine to actually perform these transformations, as Boffin only knows the (perhaps split) caret area in a rectilinear sense.

### 9. LHiliteText

This routine takes two character offsets and returns a highlighting area that covers the text between them (inclusive). Note that mixed-directional text implies that there may be multiple disjoint areas (theoretical considerations suggest as many as 5 in the current Arabic model). Note also that for text which is inherently more two-dimensional (such as Thai with stacked vowel marks) we need to be able to highlight non-rectangular areas (or perhaps, areas that are rectangular but do not extend the full line height; see the figure in section III). In early Boffin implementations, however, rectangular areas will be the only ones supported. Also, note that disjoint character ranges always produce disjoint highlight areas; thus, clients need never worry about "double inversion" causing highlighting to disappear in this case.

a) Input

LayoutSpec

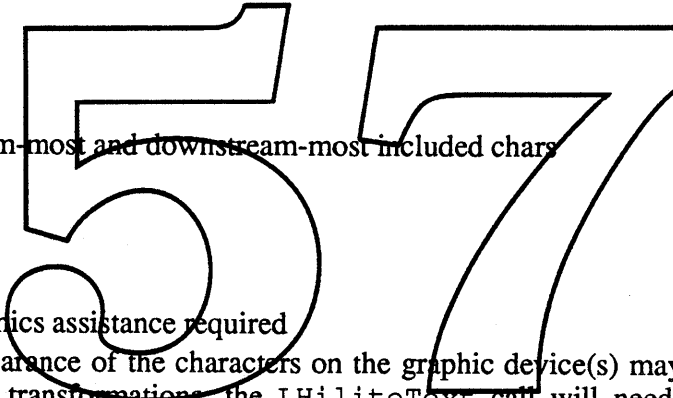
Offsets for upstream-most and downstream-most included chars

b) Output

Highlighted area

c) Font Manager & graphics assistance required

Since the end appearance of the characters on the graphic device(s) may have been distorted by one or more applied transformations, the LHiliteText call will need to make calls into the graphics system to obtain the transformed versions of the basically rectangular areas that make up the basic layout.



### 10. LMeasureText

This routine takes two character offsets and returns the width (or height, for vertical text) of the smallest rectangle that would bound the area of the text as it finally appears. The units returned are absolute distances. As a special case, if no offsets are provided, then the width of the entire layout is returned. In early Boffin implementations, only the width (height for vertical) is returned; future implementations will return the height (width for vertical) as well.

Note that most of the functionality of this function could be provided by LayoutArea; perhaps the two routines will be merged. Also note that the width of the text includes such things as hanging punctuation and optical centering effects.

a) Input

LayoutSpec

Indices for upstream-most and downstream-most included chars (optional)

b) Output

## Boffin ERS

### Width and Height

- c) Font Manager & graphics assistance required

The same kind of assistance is needed here as is required for `LHiliteText` (q.v.)

### 11. `RightArrowOffset` and `LeftArrowOffset`

In mixed-directional text, the work needed to make the caret move one position to the right or left on the screen (say, in response to a right or left arrow keypress) is non-trivial<sup>2</sup>. These two routines simplify this task for client programs. They take a character offset and return the character offset of the appropriate visually adjacent character. Note that in the “split caret” paradigm that is used for mixed-directional text, moving “right” means moving the *primary* (i.e. topmost) caret to the right; the *secondary* caret (if it remains split after the move) will move independently.

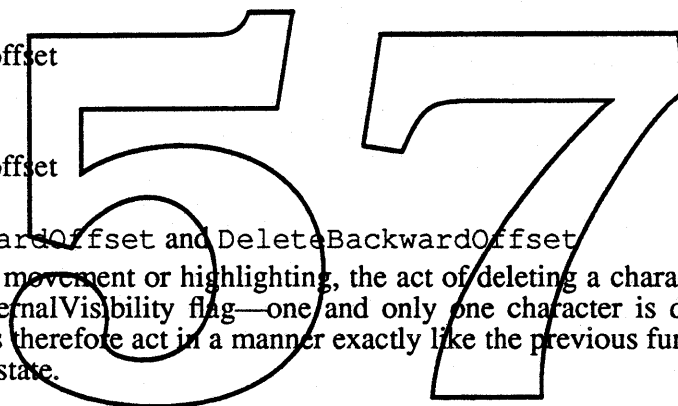
The `LigatureInternalVisibility` layout feature has a major effect on the behavior of these functions. If the setting is *indivisible*, then the caret must appear to jump entirely over a ligature (i.e. the offset of the character past the ligature is returned). If the setting is *divisible*, however, then the caret is permitted to move into the ligature (in general, the ligature will be split into  $1/n$  pieces, for an  $n$ -character ligature).

- a) Input

Character offset

- b) Output

Character offset



### 12. `DeleteForwardOffset` and `DeleteBackwardOffset`

Unlike caret movement or highlighting, the act of deleting a character never considers the state of the `LigatureInternalVisibility` flag—one and only one character is deleted from the backing store. These functions therefore act in a manner exactly like the previous functions with this flag hard-wired to the *divisible* state.

- a) Input

Character offset

- b) Output

Character offset

### 13. `LayoutInsert` and `LayoutDelete`

These routines are listed here, although they may not be in Boffin 1.0. It is intended that layout using Boffin will be very fast; in the event that goal is not easily attainable in 1.0, these routines may help. They allow insertions and deletions to an existing layout without the need of recreating the whole layout.

- a) Inputs, Outputs, etc. will be determined if the need arises.

---

2. Note that moving one position in the backing store is trivial, being more or less the simple increment or decrement of a pointer. What we're talking about here is moving the cursor one position on the *screen*.

## Boffin ERS

### C. Low-level Boffin routines

#### 1. InstantiateLayout

This routine takes a `LayoutSpec` and creates an instantiated version of it, appropriate for a particular resolution and dot size. It is called by the graphics engine (which in turn was presumably called via the `LDrawText` call).

##### a) Input

`LayoutSpec` (device-independent)

Resolution

Dot size

##### b) Output

`LayoutSpec` (device-dependent)

#### 2. LDGetArea

This is the low-level Boffin version of the high-level `LHitText` call. It is **only** called by the graphics engine; for a description of the flow of control, see the illustration in section IV.

#### 3. LDHitTest

This is the low-level Boffin version of the high-level `LHitTest` call. It is **only** called by the graphics engine; for a description of the flow of control, see the illustration in section IV.

### D. Boffin dependencies on other parts of the system

These routines are used by Boffin but supplied by other parts of the system.

#### 1. GetLayoutTables

This routine is central to Boffin's working. It retrieves the layout tables from the font in preparation to using them.

##### a) Input

Font

##### b) Output

Pointer to tables

#### 2. GetCharMetrics

This routine is called by the Boffin routines, both high-level and low-level, in order to get character metrics information from the Font Manager.

Any styles or faces present in the layout description need to be applied to the absolute characters in order for the correct `LayoutSpec` to be generated. Therefore, `LineBreak` needs to be able to get character metrics (including advance width, sidebearings, and outline bounding rectangle) on the styled versions of the characters. Note that Boffin assumes that any graphics features applied to the text will be taken into account in the values that get returned from the `GetCharMetrics` call, since Boffin doesn't understand the internals of graphics features.

## Boffin ERS

Boffin communicates the resolution and dot size it wishes the metrics to be returned in via the *resolution record*. Note that a dot size of zero is taken to mean that Boffin wants the resolution-independent metrics. Which metrics are specifically desired is indicated in the *selector mask*. This mask allows the return of any or all of the following:

- Advance widths
- Tight (“black-bit”) bounding metrics
- Anchorpoint metrics
- Optical bounds metrics

A *metrics record* is returned, containing the specified metrics.

a) Input

Glyph code  
SingleStyle  
Resolution record (described above)  
Selector mask (described above)

b) Output

Metrics record (containing the desired metrics indicated by the selector mask)

3. GetBreakInfo

This routine is called to allow the graphics system to inform Boffin about which layout features are supported within a given style, and between a pair of styles. It is only called by the high-level Boffin routines. Pointers to style descriptor records are passed, rather than the records themselves, since Boffin has no understanding of the contents or size of style records; that's up to the graphics engine. These pointers are therefore cast as pointers to the relevant structure.

If only the first pointer is given, the request is for the break information for two characters within that style; if the second pointer is also given (i.e. is not nil), then break information is also returned for two characters of the two different styles.

a) Input

Pointer to first style descriptor record  
Pointer to second style descriptor record (optional)

b) Output

Break table for first (single) style  
Break table for first and second styles (optional)

4. IsGlyphInFont

This utility routine allows Boffin to ascertain whether a particular glyph can be rendered from a given font. Note that the usual working assumption is that if the font's tables yield a particular glyph code in some circumstances (e.g. an 'fi' ligature for an 'f' followed by an 'i'), that glyph will be present in the font. The main use of this routine, therefore, is in those cases where font substitution might be necessary.

a) Input

Font id  
Glyph code

## Boffin ERS

### b) Output

Flag indicating whether the glyph is actually in the font

### 5. AbsoluteToDevice and DeviceToAbsolute

These routines allow conversion of absolute points into device-dependent points, and vice versa. They are important, for example, in handling mouse hit events. Note that in QuickDraw, the graphics model dictates that resolution is fixed at 1/72", and so this conversion is very simple.

### 6. GEDrawText

### 7. GEHiliteText

### 8. GEHitTest

These three functions are called by the high-level Boffin routines, and represent essentially the pass-through points into the graphics engine that start the low-level parts of the respective calls. See the illustrations in section IV for more details.

### 9. GESetTransformation

If a pointer to a graphic transformation is passed along by Boffin, this routine is called in order to establish that transformation. [There's still some last-minute designing being done around the issue of whether the transformation is passed around, or whether Boffin simply allows the graphics engine to maintain it as some sort of global state].

## E. Printer special-purpose routines

In addition to the standard routines described above, there are certain special-purpose routines that have been added for the convenience of printing. These routines allow the serial cumulation of pieces of a line, rather than the usual Boffin method of presenting the whole line at once.

### 1. OpenLayout

This routine is called to start the process of cumulating pieces into a layout. It is similar to the QuickDraw OpenPicture call in that respect. It takes no parameters, and returns a handle, called a CumulHandle, which is used as inputs to further calls.

#### a) Input

(none)

#### b) Output

CumulHandle

### 2. AddToLayout

Once the OpenLayout call has been issued, AddToLayout is called with each new run. Note that the runs have to already be in left-to-right display order. This routine is passed the cumulation handle and the new run, and returns nothing.

#### a) Input

CumulHandle

#### b) Output

## Boffin ERS

(none)

### 3. CloseLayout

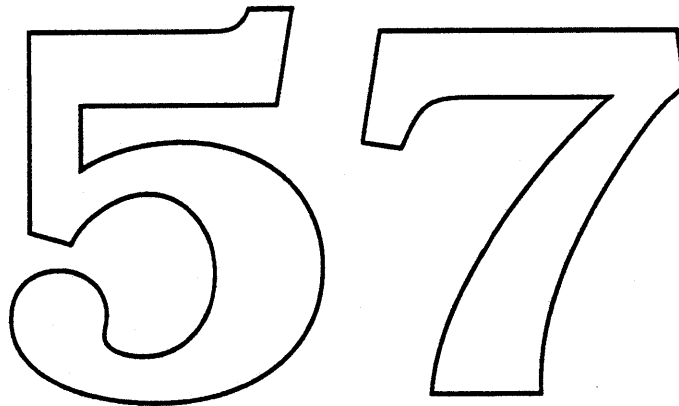
After cumulating all the pieces of the line (via multiple AddToLayout calls), the client calls CloseLayout to obtain a high-level layout for the line. This routine also frees the storage associated with the CumulHandle.

#### a) Input

CumulHandle

#### b) Output

LayoutSpec



## VI. Support tools

### A. SFNT resource editor

This tool is intended for use mainly by font manufacturers. It adds the international pieces to an SFNT so that SFNTs may be used with Boffin. In the Boffin 1.0 time frame, this tool will probably work something like Rez does, taking a text file of information and “compiling” it into the relevant SFNT pieces. A later version will likely have a more friendly user-interface.

### B. FONT/FOND converter

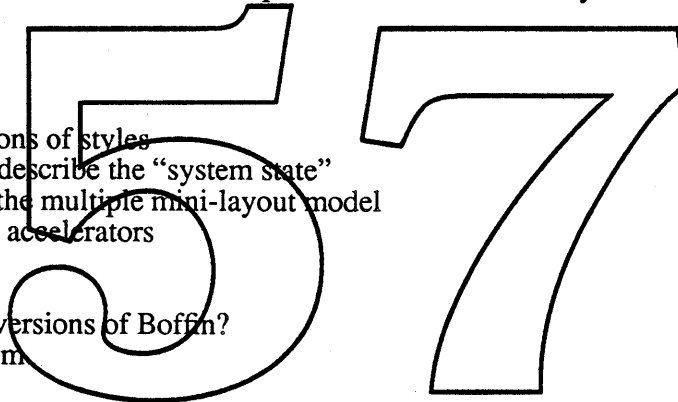
This tool is needed to add a form of the international resources to older fonts, so they might take advantage of the new Boffin functionality.

### C. Layout Features cdev

This cdev (or equivalent in different systems) will allow the user to designate the layout feature levels that are to be used as defaults, in case no explicit choice is made at layout time.

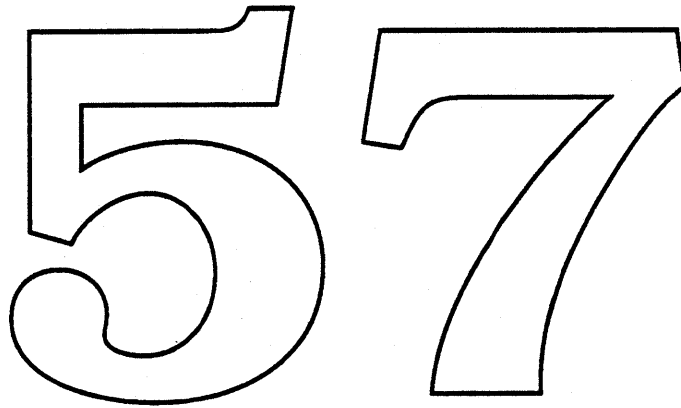
## VII. Open Issues

- A. QuickDraw parametrizations of styles
- B. The idea of *signatures* to describe the “system state”
- C. The tabulation model vs. the multiple mini-layout model
- D. Performance and possible accelerators
- E. Font substitution
- F. Style substitution
- G. What’s going into future versions of Boffin?
- H. PICTs and support for them.



# Everest ERS (Draft)

|                         |    |
|-------------------------|----|
| Introduction.....       | 1  |
| Design Constraints..... | 2  |
| General Design.....     | 3  |
| Services.....           | 6  |
| File Organization.....  | 12 |
| Everest Routines.....   | 13 |





## INTRODUCTION

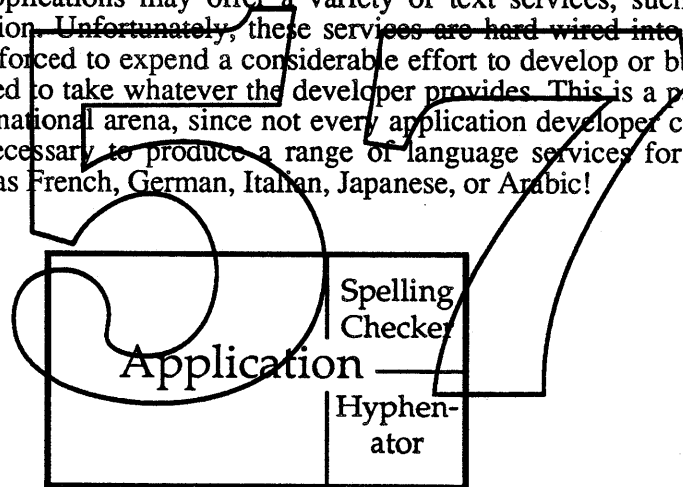
Little attempt has been made to eliminate inconsistencies between sections, since this draft is more aimed at getting ideas on the overall structure. The main attempt at this point has been to include all the ideas somewhere in this document. The organization will be cleaned up once the ideas are all worked out.

Everest is also known as the Language Manager.

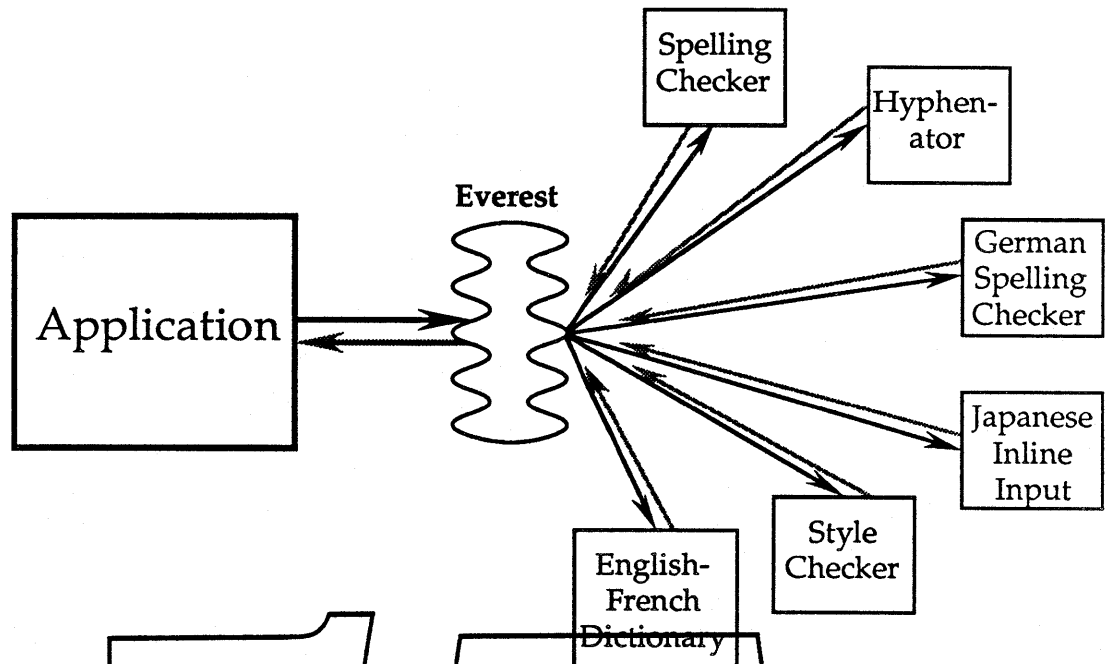
*Open issues or comments are indicated by italics.*

Lack of paragraph spacing, and peculiar line spacing at the end of paragraphs preceding outline items is courtesy FullWrite™.

Current Macintosh applications may offer a variety of text services, such as spelling checking or hyphenation. Unfortunately, these services are hard-wired into the application; the developer is forced to expend a considerable effort to develop or buy these services; the user is forced to take whatever the developer provides. This is a particular disadvantage in the international arena, since not every application developer can easily acquire the expertise necessary to produce a range of language services for English, let alone languages such as French, German, Italian, Japanese, or Arabic!



Everest provides support for a standard interface to a variety of text analysis services, allowing any application to make use of any 3<sup>rd</sup> party spelling checker, thesaurus, or other text analysis service. This allows applications to avoid the necessity of building in sophisticated text analysis services and gives users the flexibility to use a variety of different services. This also allows text services to be used with a wide variety of applications that do not currently offer them: e.g. a spelling checker with AppleLink, hyphenation with presentation graphics, in-line Kanji in most applications, &c. Moreover, users can buy the best quality text services for their needs, instead of being stuck with the built-in service (naming no names here).



Language services provided for by Everest include spelling checkers, hyphenators, regular expression search & replace, in-line East Asian input methods (e.g. KanjiTalk), syntax checkers, style checkers, translation aids, and perhaps help or tutoring services.

By supporting Everest, applications allow the use of a wide range of additional services, significantly enhancing their actual and perceived functionality. In addition, they are not locked in to a particular language or script, nor are they required to support equivalent services in every language: they can leverage off of the efforts of third-party language service suppliers.

Language services also have a great deal of freedom in terms of the functionality they can provide, and in terms of the organization of their data. It is even possible for large modules or data bases to be server-based, as in grammar checkers or translators.

## DESIGN CONSTRAINTS

Everest will meet the following constraints:

*Minimal application coding.* Applications will need little code change to incorporate Everest, and thereby use any of the supported services.

*Independence.* The functionality offered by an individual service is completely independent of the application: services can add new functionality without any change to the application.

*Service extensibility.* Completely new types of services (grammar checking, etc.) can be developed: Everest will automatically allow the user to use those services in any application supporting Everest.

*Public Interface.* The programmer interface for producing services will be public: any company can produce a language service that works with Everest.

*Open Market.* Apple will not produce text services, but rather encourage third parties to develop them. In support of that aim, Apple will provide sample code for several prototype services, and a sample application that uses Everest.

*Future Architectures.* For text service products, the code migration to future architectures will be straightforward.

*Text-Based.* Services are primarily concerned with text: they may depend on the font information associated with text in order to distinguish overloaded character codes (e.g. "abcde" vs. "αβχδε" vs. "⦿⦿\*\*\*"), but do not otherwise concern themselves with format information.

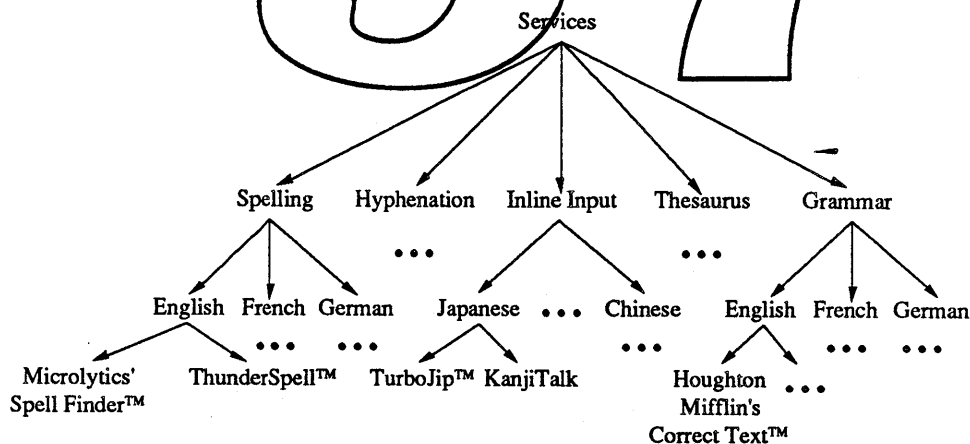
## GENERAL DESIGN

Everest supports different types of user interface. The interfaces allow four sorts of actions: configuration, invocation, control, and termination. All of these actions are available within any supporting application, and users can add new services to their systems as easily as dragging the required files into their system folder.

Configuration involves selecting among the different available services. For example, different companies offer spelling checkers, so the user can select the ones that he wants to use. This configuration should be saved on a per-document or per-application basis.

[Note that configurations are typically dependent on both language and script: German hyphenation will differ from French hyphenation, as well as from Arabic hyphenation. Sophisticated applications can use Everest to support simultaneous multiple languages. For example, an application can allow users to mark text of different languages (flagging terms like *faux pas* as French), allowing Everest to invoke different spelling checkers based upon that information. Applications can also allow users to flag text as having no language, for text that is not to be processed by a given class of service: e.g. tables or code samples.]

*We currently have code numbers for different languages. DTS needs to register new ones on request.*



Invocation involves requesting the desired service: as when the user selects the Check Spelling menu command. Once the user has invoked the service, he is then typically presented with an interface, such as a dialog. The content and operation of this interface is entirely up to the service. This allows the user to control the function of the service. Finally the user will finish using the service, and quit the control portion of the interaction.

Some services offered by Everest can also be used for static documents. For example, a stand-alone spelling checker may use Everest to process known document formats to find

misspelled words, and allow the user to then alter just those words without necessarily having general editing capability. This is not generally as useful as built-in, interactive services, but allows processing of documents whose applications do not support Everest.

### Service Types

A services can have several attributes. The input service attribute indicates that the service can process text as it is typed in by the user. The scanning service attribute indicates that the service can process large chunks of text by scanning through the document.

*What are other interesting attributes that are not included here?*

Input services are characterized as services that handle certain events in the applications window. For example, an inline Japanese input method may change the text 'to' to the kana equivalent immediately after the 'o' is typed in.

When using a scanning service, applications are encouraged to provide a mechanism for users to specify the text to be processed by the service, and a mechanism for marking a span of text to not be processed. This is useful, for example, to exclude code fragments from spelling checking.

A particular service can have neither of these attributes, only one of them, or both. For example, a spelling checker can be both a scanning and an input service, an in-line East Asian input ~~service is an input service and not a scanning service~~, a grammar checker is (probably) a scanning service but not an input service, and a thesaurus service will be neither a scanning service nor an input service.

### Interface.

At launch time, the application gets a list of the available services from Everest. It also gets the (localized) name of each service class (e.g. 'Spelling', 'Buchstabieren',...). Service classes include spelling checkers ('SPEL'), hyphenators ('HYPH'), regular expression search&replace ('GREP'(??)), East Asian Input Methods ('INPM'), Syntax checkers, etc. [The service classes are extensible, and can be registered with DTS just as file type & creators are registered.]

*Human Interface needs to be brought in on for guidelines on selecting and configuring services.*

The application then makes up a menu listing the service classes (e.g. Spelling, Thesaurus,...). A separate menu item (Configure...) lets the user choose the default service for each service class (e.g., which spelling checker to use when the user chooses spelling, if there are multiple spelling checkers installed).

The following is only one example of how the application can support the srvice selection interface.

### Events.

When a language service is active, then each event received in the main event loop is passed on to the service. The service may pass that event right back to the application, or it may choose to handle the event itself. A service will handle a mouse, update or active event if it occurs in the service's window, or a keyboard event if the service's window is frontmost. Input services may also handle certain events in the frontmost window, even if that window does not belong to the service.

*If the application does not call Everest, then it will use the old pop-up window method of Kanji input.*

*Note that if there is a tablet attached to the system, the input event may consist of both a*

character and a screen co-ordinate. Applications' use of this information can be integrated better on future systems.

The most important feature of Everest is callback routines. These routines perform the actions requested by Everest. It is the application's responsibility to supply these routines if it works under Everest.

*This is where the guts of the work has to be done, so that we get a list of routines that any application can support, and that are sufficient to do the work we need. The MacApp people should have some good feedback on how we can manage the flow of control nicely.*

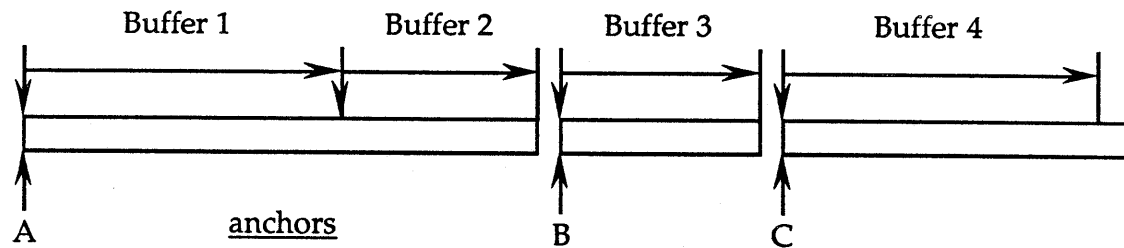
Applications can also make direct calls on the service. For example, the application can request the hyphenation for a given piece of text, supplying a pointer to a block of memory in the text parameter fields.

*It may not make much sense to have a generic direct call mechanism. The application is going to need to know a lot about the service to be able to direct call it, so maybe we should just provide specific calls for those services, like hyphenation, that are most likely to be direct called.*

File Marks.

Everest makes very few assumptions about the internal structure of an application's text. When a service communicates text information to and from an application, the text is always plain text specified by pointer and length. Position information is communicated by means of file marks. A file mark consists of an anchor point and an offset. The anchor point is specific to the application, while the offset is a zero-based displacement from that anchor point. The offset refers to the storage location (in bytes), not a character position (in the case of double-byte scripts on the current Mac the number of bytes is not directly related to the number of characters).

For example, if an application structures a document as a series of paragraphs, the anchor point might refer to a particular paragraph, and the offset would be a position within that paragraph. When a service queries the current selection, it receives back the start and end marks for the text. When it asks for a buffer full of text from one of those marks, then the application passes back as much text as it can relative to that anchor point. If it runs out of text in that paragraph, then it changes the anchor point that it passes back to the service the next time it is called.

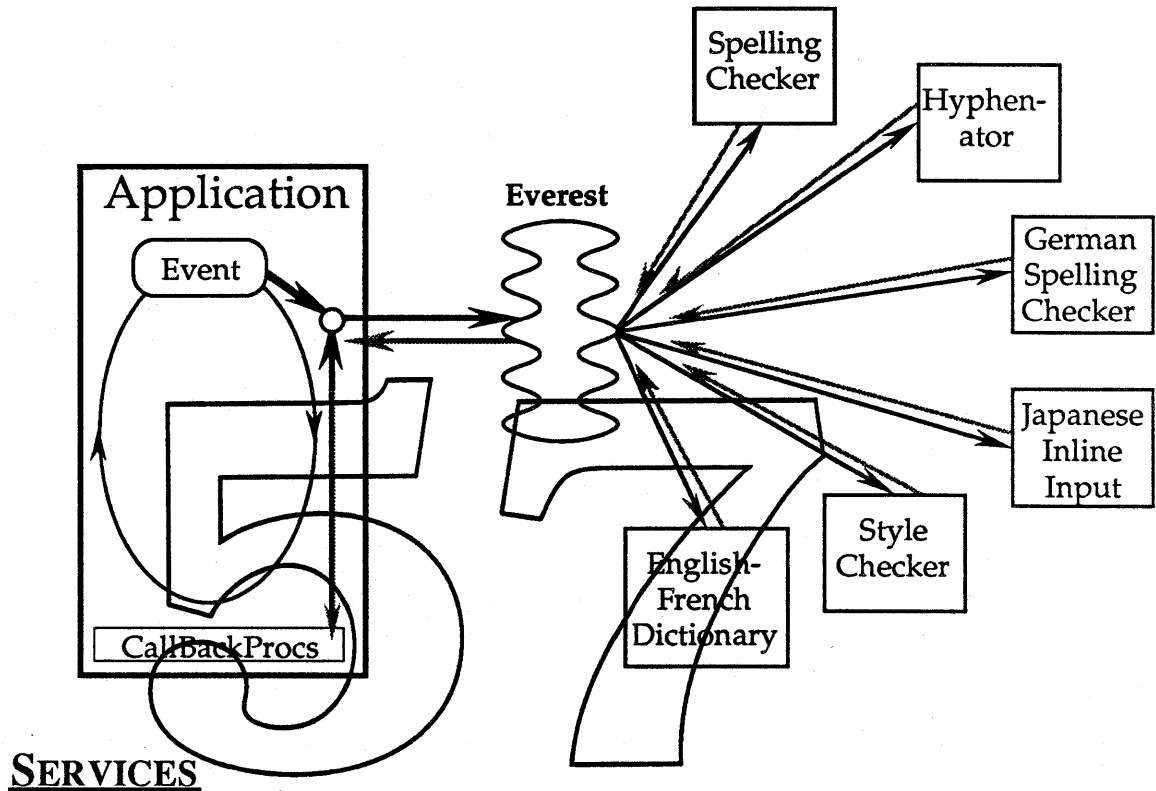


The service always refers to the application text by means of marks. So, when it wants to replace the text between two positions, it will pass the first mark (anchor point and offset), and the second mark (anchor point and offset) to the application.

The anchor points can even be more finely distributed, as when each style run starts an anchor point. For example, a spelling checker might be checking the word 'FISH!'. The first time it asks for more text, it gets back the character 'F' with anchor point A. The next time, it receives the characters 'ISH' with anchor point B. The next time it asks it gets an

exclamation point, and so knows that the previous word is complete, and can thus be checked against the dictionary.

At certain times, the application will want to indicate that the boundary between anchor points is a discontinuity. For example, suppose the application has supplied all of the text in the body of the document, and is ready to supply text in the headers to a style checker. It can communicate this information back to the service so that the service does not treat the text as a continuous flow.



## SERVICES

The following describes a set of typical services. For each service a scenario is given for a typical user session. After each scenario, the set of actions the service needs to perform on the application text is given. These lists are used to develop the Everest interface, which is given in the next section.

*Have we really considered a wide enough set of services to be sure we've got the bases covered for at least the next five years or so?*

### Dictionaries

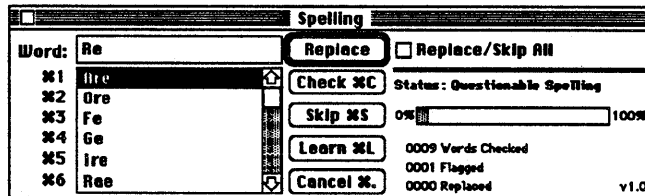
Most services will use dictionaries. There are two basic types of dictionaries; read-only dictionaries, which will be supplied with the service or purchased separately, and user editable dictionaries, whose contents will be determined by the user.

In general, the user should have the ability to control which dictionaries are used by the service. Dictionaries which are being used are said to be *active* and dictionaries not being used are said to be *inactive*. Some dictionaries, such as the read-only dictionaries, will typically be active during the entire session. Other dictionaries will be activated and deactivated by the user during the session. A third class of dictionaries will be associated with a given document and will be active only when that document is being processed.

*Note: none of the dictionary functions require any interaction with the application except the ability to store document-specific dictionaries.*

### Spelling

One of the application's menus contains an item which invokes the Spelling service. The user chooses this item, and the spelling checker registered with Everest is activated. A window belonging to the Spelling service comes up:



When the user chooses the Check Button, the Spelling service scans the document text until it locates a misspelled word. The document window scrolls to the misspelled word, if it wasn't already visible, and highlights it. The misspelling is also put in the Word: item, and suggested corrections are placed in the list item. The spelling service waits for the user to decide what to do about this misspelling.

The user can choose one of the suggested misspellings in the list by selecting it with the mouse. If the user doesn't like any of the suggestions, another spelling can be typed into the Word: item. Once the correct spelling has been selected or typed in, the user can click on the Replace button to replace the misspelling in the document window with the correction. If the user clicks the Skip button, the Spelling service will skip this misspelling. If the Replace/Skip All box is checked, the spelling service will automatically correct or skip further instances of this misspelling without stopping. If the misspelling is actually correct, the user can click the Learn button to instruct the Spelling service to remember this word.

Other possible spelling checker functions not shown in the window above are:

- The ability to check an entire document and add all misspellings to a dictionary
- The ability to scan backwards instead of forwards for misspellings
- The ability to check spelling as the user types
- The ability to mark portions of text, such as code examples, to not be checked

Here are the functions the Spelling service needs from the application:

- Scan through text
- Highlight text
- Replace text
- Access dictionary associated with document

### Syntax and Style Checking

The interface for Syntax and Style checking is related to the Spelling interface. It does present more information about the kinds of errors. For example, in the sentence "There's less people here than before," it would flag "There's less people.." as being invalid subject-verb agreement, and "There's less people.." as being an invalid mass/count adjective-noun agreement. It would also suggest the alternative "There are fewer people..." and supply information about the two grammatical rules if requested.

However, in terms of the interaction with the application, actions required by Syntax and Style checking are very similar to Spelling: scanning text, highlighting, replacing,...

### Hyphenation

Not all languages are hyphenated in the same way. In German, the spelling of a word can change if it is hyphenated. (e.g.: Zucker  $\Rightarrow$  Zuk-ker) Other languages, like Thai, don't insert a hyphen. Chinese, Japanese, and Korean words can be broken after any character with the exception of a few "taboo" characters which cannot start or end a line.

*It is not clear to me if the concept of hyphenation should be extended to a generalized line break calculation. This is tempting because Everest seems like a good place to put all language dependent text details. On the other hand, adding too much to the definition of hyphenation may overload the mechanism..*

Normally, the application will call the hyphenation service directly during text layout. The user will need to interact with the hyphenation service to fine-tune the hyphenation for a particular document. The sorts of things the user will need to do are:

Show hyphenation points for this word - Display the hyphenation points for the given word. (This could operate on the selection, or on a word that the user types into a separate text box.)

Add this hyphenation to a user dictionary - This could be a generalized user dictionary or it could be one associated with the document.

Override a particular hyphenation point - There needs to be a way for a user to specify a hyphenation point different than the one chosen by the service for a particular instance of a word. This is probably best done by allowing the user to enter a special hyphenation character to indicate the desired hyphenation. (Placing this character at the beginning of a word would cause it to not be hyphenated at all. This can also be done in a more general way by selecting a range of words and indicating No Hyphenation to the application.)

*We need to standardize this character because all the hyphenation services need to agree on what it is (note that ISO character sets contain a soft hyphen: SHY). If the hyphenation services all recognize the character then the application doesn't need to bother with it beyond providing a way for the user to type it.*

*On the other hand, maybe the service doesn't need to see this character at all... the application could look for this character in the word it wants to hyphenate and skip the service call if the override character is present. In general multi-lingual text, finding the override character might be tricky, though...*

Don't ever hyphenate this word - This is just a special case of adding a word with no hyphenation points to an editable dictionary.

Automatically insert soft hyphenation points into words as they are typed - This could be the same character used to override the service's hyphenation points.

So, hyphenation services don't require much from the application except:

- Minimal support for an override character
- The ability to interrogate the selection
- The ability to replace text

### Thesaurus and Translation Aids

These services are similar and quite straightforward from the Everest point of view. All that's required is a way to generate translations or synonyms for the selected word and a way to replace the selected word with a synonym or translation. This requires the following functions from the application:

- Interrogate selection



- Replace selection

### East Asian Input

This discussion is in terms of the Japanese language, which has the most complex writing system used in Asia today. The issues for Chinese and Korean are mostly the same. See The appendix for a description of the Japanese writing system computer input of Japanese.

Most Japanese words can be followed by a grammatical inflection and/or auxiliary words which are written in *kana*. A word followed by an inflection and/or other auxiliary words is called a *phrase*. If conversion is done a phrase at a time, the extra information can be used to reduce the number of homophones. For example, the homophone of “kanji” meaning “feeling” can be followed by the inflection “ta”, indicating passed tense, giving “kanjita”, meaning “felt”. This phrase is unambiguous, having no homophones.

Entering text a phrase at a time is still somewhat slow and tedious. It’s easier to enter a whole sentence or more at once. Since the actual conversion of *kana* to *kanji* must still happen a phrase at a time, this introduces a whole new problem. Remember that Japanese is written without spaces, so there’s no easy way to tell where the phrases are. Dividing a sentence up into phrases is called *segmentation*, and each phrase is called a *segment*.

Two traditional segmentation algorithms are called *fewest segments*, and *two longest segments*. The *fewest segments* algorithm divides the sentence into segments in the way that produces the fewest number of segments. The *two longest segments* algorithm divides the sentence into segments such that successive pairs of segments are as long as possible. In typical implementations, these algorithms do not require the user to choose a particular homophone during the conversion process; one is chosen automatically, usually based on frequency of use statistics, or the user’s previous choices. It is possible to improve these algorithms by using the part of speech of each segment to decide if the sequence of segments makes grammatical sense.

Since homophones are chosen automatically during the conversion process, there must be a way for the user to select alternate homophones for each segment. Sometimes the algorithm will not segment correctly, so there must also be a way for the user to change the segmentation.

Currently on the Macintosh, Japanese input is done in a separate “floating” input window which is maintained by the input method, independently from the application. As the user types, the input method intercepts the events and routes them to this window rather than to the application window. When *kana-kanji* conversion is complete, the user hits the return key, and the input method sends the application a series of events containing the converted text.

This method places no burden on the application, since it never sees the text until after *kana-kanji* conversion is done. It is also modeless, as far as the application is concerned. However, it has the following disadvantages:

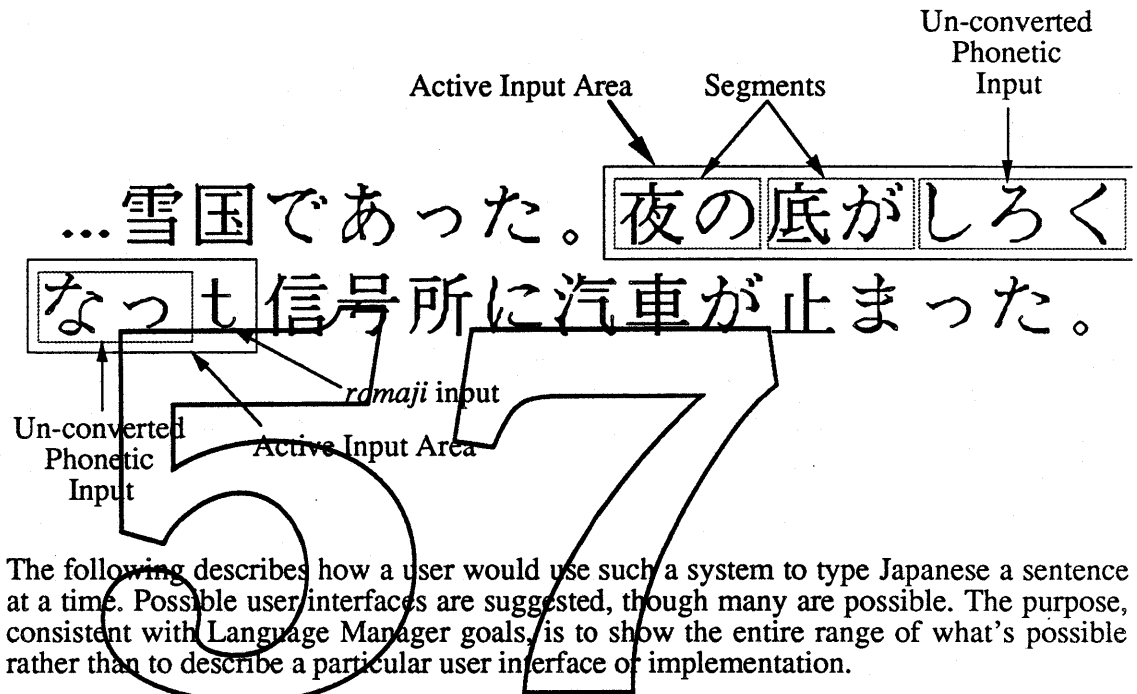
- Users must keep shifting their visual focus between windows
- The input window interrupts all other processes while it’s active
- The input window is not WYSIWYG: users can’t see what their text will look like until they enter it.

Inline input solves these problems by sending the input text directly to the application window, with the following advantages:

- Users can focus their attention on the application window
- No special window, so other processing not affected

- All input is WYSIWYG

The figure shows how inline input might look while in progress. The sentence being input, called the *active input area*, is outlined with a solid line. Notice that it starts in the middle of the first line and ends in the middle of the second line. The text outside of the active input area does not participate in the conversion process. The segments which have already been converted, and the un-converted input are outlined with dotted lines. The un-converted phonetic input is followed by a single *romaji* letter. The insertion point is after the *romaji* letter.



The following describes how a user would use such a system to type Japanese a sentence at a time. Possible user interfaces are suggested, though many are possible. The purpose, consistent with Language Manager goals, is to show the entire range of what's possible rather than to describe a particular user interface or implementation.

As the user types text it is processed by the input method and displayed in the application window. The active input area should be visually distinct from the rest of the application text so that the user can see which text is being processed. As text is processed, the contents of the active input area may change; for example, as *romaji* input is transcribed into *kana*, or as new segments are identified by the input method.

When the input method generates segments, it may not generate the correct segment boundaries, or it may not choose the correct homophone for a given segment. In these cases the user must be able to make adjustments. First, the user must indicate which segment is to be adjusted. This could be done by pointing at the segment with the mouse, or by typing some sequence of commands, such as the left and right arrow keys. However the segment is indicated by the user, it must be given a visual appearance different from the rest of the document text, and the rest of the text in the active input area.

The user can change the segmentation by indicating that the indicated segment should be made one (phonetic) character longer or shorter. (Note that this makes the following segment one character shorter or longer, and is thus the same as moving the segment boundary.) This could be done by dragging the segment boundary with the mouse, or by typing some sequence of commands, such as the up and down arrow keys. Depending upon the particular input method, changing the length of a segment may cause all the segments following it to change as well.

The user can display alternate homophones for a segment by double-clicking on the seg-

ment, or by typing some sequence of commands, like the enter key. Alternatives can be selected either by changing to the next homophone in-place, or by bringing up a floating window displaying all homophones, from which the user would then choose the correct homophone.

Text editing within the active input area cannot be the same as text editing elsewhere in the application window because of the special actions the user needs to perform on the active text. The distinctive visual appearance of the active input area indicates to the user that this is the case.

Different levels of functionality could be supported by different input methods. For example if the user were to delete a character from the middle of a segment, the information that an input method maintains to process the active input area could become invalid. On the other hand, most input methods will probably allow the user to freely edit the un-converted input. Less sophisticated input methods might wait until the whole sentence is typed before doing any segmentation, while other input methods might generate segments "on the fly". Some input methods might decide to "drop" segments off the front of the active input area as the user types in order to limit the amount of internal storage needed.

The user can request that the un-converted input be converted by some keyboard command. It probably makes sense to have the command be a modification of the command that requests alternate homophones for a segment.

The user can indicate that conversion is complete by some command like return or enter. When this is done the distinctive visual appearance of the active input area and the active segment would be removed and the caret would be placed after the newly converted text.

In summary, users need to be able to perform the following actions on the active input area:

- Type in phonetic text
- Change transcription method
- Change the length of a segment
- Choose another homophone for a segment
- Edit text in active input area
- Convert un-converted input
- Indicate that conversion is complete

Methods for performing these actions include pointing and clicking with the mouse, and typing cursor keys and command keys. Note that most of these actions will have different meanings outside of the active input area.

In the case of inline input, the application and the input service must cooperate closely. The input service must see all events, though it will still let the application handle most of them. Only the input service knows where the active input area and the segment boundaries are; only the application knows where characters are on the screen and how to alter their visual appearance. When the user uses the mouse to indicate a particular segment, only the application can do the mapping between mouse position and character positions; only the input service knows which characters are in which segments.

Given this division of knowledge, the input service must be able to request the following functions from the application:

- Highlight the active input area
- Highlight a particular segment
- Replace the text in the active input area
- Map a mouse position to a character

This set of functions seems sufficient to permit the input service to direct the application to perform the user interface functions required for inline input if the application passes

all events to the input service before processing them. Still, the following issues remain:

While the input service must examine all input events, it cannot be expected to know what events like selecting a menu item, or typing a command key mean, since these are going to be application specific. Some of these events might change the document in such a way that any in-progress inline input cannot be completed. How does the input service find this out? The best solution seems to be to depend on the application to tell it. Can the application always know?

An important user interface goal on the Macintosh is that applications be modeless. Inline input is far from modeless. Moreover, the text in the active input area will behave differently than the rest of the text in the same window! Is this OK? Will users see this as a natural consequence of typing Japanese, or will they get confused?

How does the application know when to pass events to the inline input service and when not to? For example, if the user's typing English, there's no need to pass all the events to the input service which will just pass them back without looking at them. If there are more than one inline input service loaded, how does the application know which one to call? *I guess that the Language Manager should be responsible for deciding which input service has control; the application just talks to "the input service" without knowing how many are there... This leaves the question of how the Language Manager decides which service to use...*

## BLUE FILE ORGANIZATION

Active service files are present in the system folder, and have the type 'lmgr'. As with INIT or CDEV files, each file is opened at boot time, and any 'INIT' resources are executed. Each service resource fork contains an information resource of type 'linf' and two distinguished strings: the service class name, and the service name. For example, 'Spelling' would be the service class name, and 'MacSpeller™' the service name. The 'linf' resource contains, among other things, the language code and the script code for the service.

Everest will compose a list of the active services at INIT time. The service class name selected will be the first name found that is in the system language (stored in the itlc resource in the system file). That is, if there is a German spelling checker, and an English spelling checker, then the German spelling checker would specify the service class name 'Buchstabieren', while the English spelling checker would specify 'Spelling'. On a German system, the service name would be the former; on an English system, the latter.

*This implies that when a new service is added the system will need to be restarted before the service is available for use. We need to think about init and its relationship to sharing under Multi-Finder.*

The lmgr files will also contain a resource of type 'lmgr' (0, Purgeable). This resource contains the definition code for the service. (There may be other segments of code, but as in the case of desk accessories or cdev's, there is no jump table, so the burden of loading and calling that other code falls on the lmgr code.)

*Once we have IPC and multitasking, the services can be in separate processes. We can even fake these two under the Multi-Finder (or leave it for the service to implement).*

The lmgr resource contains a dispatch table to handle the different calls routed to it by Everest. The calling sequence is as follows:

Everest is called by the application immediately after GetNextEvent or WaitNextEvent. It then calls each appropriate service until the event is handled.

Everest calls a service by loading the lmgr resource from the proper file (if purged). [If the resource cannot be loaded, then the arguments are stripped, and an error code is returned to the application.] The LM then jumps to the first instruction in the lmgr resource. The lmgr code has the responsibility of locking and unlocking itself.

The lmgr code has complete control. It can open files (e.g. dictionary files), pass information to and from the application, request actions by the application, respond to requests by the application, and even perform the bulk of its function by requesting computing resources from remote sources (e.g. a syntax checker—which is very CPU and data intensive—may reside on a remote server, and analyze text for many different workstations).

Memory can currently be allocated in the system heap (*ugh!*), but must be cleaned up upon deactivation or termination. The lmgr code is also loaded into the system heap for execution.

## EVEREST ROUTINES

The following is a C header file for a prototype version of Everest. "TSM" means "Text Service Manager", an old name for Everest.

### Header File

```

/*
  Modification History
  <d1 erm>      Alternate version with call-back vector
  <d2 erm>      Change TSMAppTextMark to TSMTextAnchor
  <d3 erm>      Add CompareMarks
  <d4 erm>      Added comments (real programmers don't use comments!)
*/

#ifdef _TSM_
#define _TSM_

#ifdef _EVENTS_
#include <Events.h>
#endif

/* Possible results of call-backs: */
typedef enum {
    tsmNoError,                /* no errors */
    tsmBlockEnd,              // the text returned is at the end of a contiguous
    tsmNoMoreText,           // no text returned because the mark was at the end of the
                              // document
    tsmUnknownError          /* any other errors I didn't think of yet... */
} TSMResult;

/* Results of comparing two text marks: */
typedef enum {tsmBefore, tsmEqual, tsmAfter, tsmDisjoint} TSMRelation;

/* Highlighting levels. The exact appearance of these isn't specified other than

```

```

to note that segment highlighting will happen to text which is also inline input
highlighted, which suggests that the three levels had better be different and
compatible. */
typedef enum {
    tsmSelection,          /* highlight as current selection */
    tsmInlineInput,       /* highlight as current inline input */
    tsmSegment            /* highlight as inline input segment */
} TSMHighlightLevel;

/* Highlighting actions: */
typedef enum {
    tsmHighlight,         /* highlight specified character range */
    tsmDeHighlight,       /* de-highlight specified character range */
    tsmDeActivate         /* highlight with inactive highlighting */
} TSMHighlightAction;

/* An application specific anchor point in the document text. TSM will not interpret
these.
*/
typedef struct {long one; long two; long three;} TSMTextAnchor;

/* A reference to a particular document character relative to an anchor.
TSM only looks at the offset. */
typedef struct {
    TSMTextAnchor anchor;
    long offset;
} TSMTextMark;

/* Used for pointers to application specific context data. TSM never looks at these. */
typedef Ptr TSMAppLcTxt;

/*
The following types define the call-back procedures which TSM uses to operate on the
application text in an application-independent way.
*/

/* Sets startMark and endMark to point at the first and last characters in the selection.
*/
typedef pascal TSMResult (TSMGetSelection) (
    TSMTextMark *startMark,
    TSMTextMark *endMark,
    TSMAppLcTxt context);

/* Compares two marks and returns their relationship. Returns tsmDisjoint if
the marks aren't in the same "text flow" (e.g. one is in main body text,
and one is in a foot note)
*/
typedef pascal TSMRelation (TSMCompareMarks) (
    TSMTextMark m1,
    TSMTextMark m2,
    TSMAppLcTxt context);

/*
Get text into buffer, starting at the character pointed to by startMark, for
up to textLength characters. Set textLength to the actual number of characters
stored. Set textFont and textLanguage. Set endMark to point to the first character
not stored. The application should return only as many characters as is conve-
nient,

```

for example, all in one style run, or one "piece" in a piece table. All the text returned should be logically contiguous. If the last character stored is at the end of a logically contiguous run of text, return tsmBlockEnd. If startMark points at or past the end of the text, return tsmNoMoreText. Otherwise, return tsmNoError.

```
*/
typedef pascal TSMResult (TSMGetText) (
    TSMTextMark startMark,
    TSMTextMark *endMark,
    Ptr textBuffer,
    long *textLength,
    short *textFont,
    short *textLanguage,
    TSMApplCtxt context);
```

```
/* Highlight the text starting with the character pointed to by startMark up to and including the character pointed to by endMark. level specifies what level of highlighting to use. action specifies whether the text should be highlighted, de-highlighted, or shown as inactive. If level is selection, the selection should be set to the highlighted text. */
```

```
typedef pascal TSMResult (TSMHighlightText) (
    TSMTextMark startMark,
    TSMTextMark endMark,
    TSMHighlightLevel level,
    TSMHighlightAction action,
    TSMApplCtxt context);
```

```
/* Replace the text between startMark and endMark with the textLength characters at text. Change startMark and endMark to point to the characters just inserted. */
```

```
typedef pascal TSMResult (TSMReplaceText) (
    TSMTextMark *startMark,
    TSMTextMark *endMark,
    Ptr text,
    long textLength,
    TSMApplCtxt context);
```

```
/* Translate the mouse co-ordinate in point to a text mark. */
```

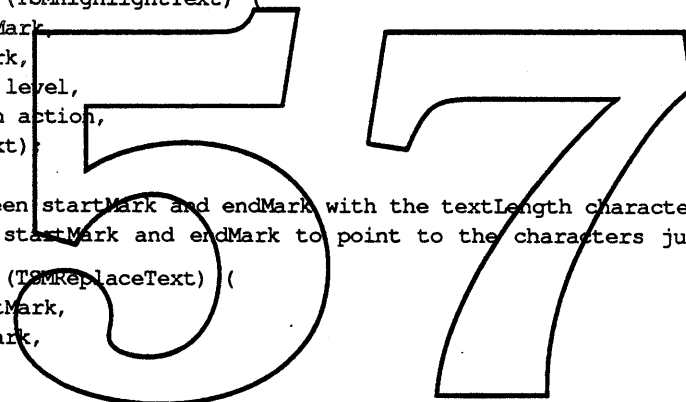
```
typedef pascal TSMTextMark (TSMPointToMark) (
    Point point,
    TSMApplCtxt context);
```

```
/* These aren't defined yet... */
```

```
typedef pascal TSMResult (TSMRetrieveDict) (void); /* fill the arguments in later */
typedef pascal TSMResult (TSMStoreDict) (void); /* figure out the arguments later */
```

```
/* This is the call-back vector supplied to the TSM by the application. */
```

```
typedef struct {
    TSMGetSelection *getSelection;
    TSMCompareMarks *compareMarks;
    TSMGetText *getText;
    TSMHighlightText *highlightText;
    TSMReplaceText *replaceText;
    TSMPointToMark *pointToMark;
    TSMRetrieveDict *retrieveDict;
    TSMStoreDict *storeDict;
} TSMCallbacks, *TSMCBPtr;
```



```

/* Each TSMHandle represents a specific incarnation of the TSM */
typedef Handle TSMHandle;

/* This describes the attributes of a particular service */
typedef struct {
    unsigned scanning : 1;      /* service scans document text */
    unsigned input : 1;        /* service processes keyboard input as it is typed */
    unsigned : 30; /* pad out to a long. Can we do this without counting bits? */
} TSMAttrs;

/* This describes a particular service. Class might be "spelling checker"
   and instance might be "FooBar's SpelGud" */
typedef struct {
    long class;
    long instance;
    TSMAttrs attributes;
} TSMService, *TSMServicePtr;

/*
These routines are called by the application to interact with the TSM and
the various services.
*/

/* Begin an interaction with the TSM. The application provides a pointer to the
   call-back vector, and a pointer to any application specific context.
   The TSMHandle returned must be supplied to all subsequent calls. */
pascal TSMHandle TSMInitialize (TSMCBPtr callBacks, TSMAppLcTtxt context, TSMResult
*result);

/* Terminate an interaction with the TSM. The TSMHandle will be invalid on return. */
pascal TSMResult TSMTerminate (TSMHandle handle);

/* Used to obtain a list of all registered services. */
/* Probably not the right arguments yet. Might want to use a call-back instead... */
pascal TSMResult TSMIndexService (TSMHandle handle, unsigned index, Ptr name, long
*nameLen);

/* Activate the specified service */
pascal TSMResult TSMActivateService (TSMHandle handle, TSMService service);

/* Deactivate the specified service */
pascal TSMResult TSMDeactivateService (TSMHandle handle, TSMService service);

/* Not worked out yet... */
pascal TSMResult TSMOpenDoc (void); /* figure out paramaters later */
pascal TSMResult TSMCloseDoc (void); /* ditto */

/* Applications call this to give the TSM a chance to handle events they're not
   interested in dealing with... */
pascal Boolean TSMFilterEvent (TSMHandle handle, EventRecord *event);

/*
Used to call a specific service directly.
Since the application will have to know details of the individual
services to make direct calls, it may not make much sense for the TSM
provide a generic call like this. It might be better to provide a specific
call for each class of service for which it makes sense...

```



**Rough Draft**

Everest ERS

```
*/  
pascal Ptr TSMDirectCall (TSMHandle handle, TSMService service, Ptr args); /* who owns  
result? */  
  
#endif
```

57

## FUTURE IMPLEMENTATION

*This section is not included for outsiders!*

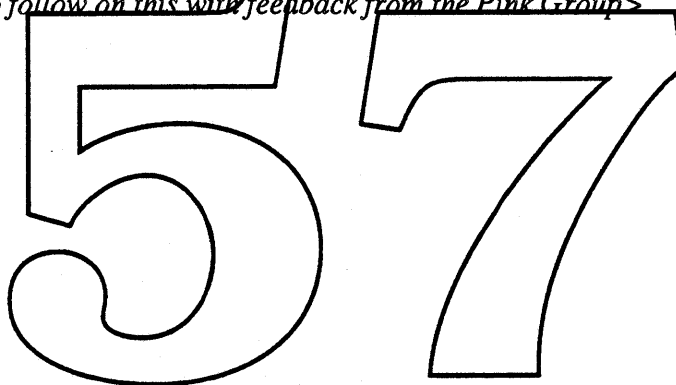
### Integration

On Pink, we can do a far better job of integrating Everest. The Pink features that are especially interesting include multitasking, IPC, and overriding.

Multitasking allows services to use separate common processes, and be swapped out when not active. Interprocess communication allows these separate processes to communicate in a standard fashion with the base applications. Overriding allows more flexibility than the use of call-back procs.

The Pink architecture can also make use of the standard interface to text services that Everest provides. Information binding between files, for example, can make use of a Root Extraction service which is sensitive to the language of the text. (Root extraction maps variant words onto a paradigm root word: e.g., "calling", "calls", "called" ⇒ "call").

*<More to follow on this with feedback from the Pink Group>*



## The Japanese Writing System

Japanese is written using three different kinds of characters: Chinese ideograms, called *kanji*, phonetic syllabic characters called *kana*, and Roman letters, called *romaji*. Words are not separated by spaces, except when *romaji* is used. Lines can be broken after any character, with the exception of a few characters which cannot start or end a line. Text can be written vertically or horizontally. For purposes of this discussion, I'll ignore vertical text.

Each *kanji* represents a single concept, like "walking", "above", "big", or "tree". A single *kanji* can have more than one pronunciation, depending on how it is used, and more than one *kanji* can have the same pronunciation. *Kanji* are usually used to write the roots of nouns, verbs, and adjectives. There are about 1,900 *kanji* in common use and some 50,000 listed in large dictionaries.

Each *kana* represents a single phonetic syllable, like "su", "shi", "fu", or "ji". With a few simple exceptions, each *kana* has only one pronunciation. *Kana* are usually used to write grammatical inflections and connectors, and to write adverbs and pronouns. Japanese can be written exclusively in *kana* though this is not common since it is actually harder to read. There are two types of *kana*; *hiragana*, and *katakana*. *hiragana* is most commonly used. *Katakana* is used for emphasis, and to write foreign loan words, in much the same way that we use italics. There are 50 basic *kana*, some of which can be modified by the addition of a mark, and some of which can be written smaller. All together, there are about 90 different *kana*, including some special *katakana* used only for transcribing foreign sounds that don't occur in Japanese.

*Romaji* is used to write things like "cm", "mm", and "km" and sometimes for English words mixed in with the Japanese. It is also possible to write each *kana* syllable in *romaji*, so that Japanese can be written using only *romaji*. (When this is done, spaces are written between the words.)

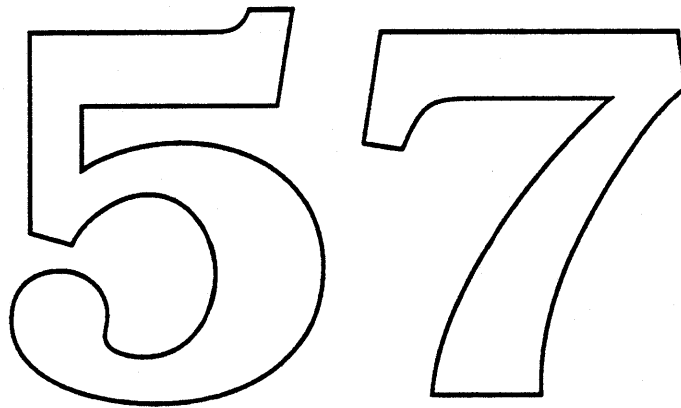
Since there are thousands of *kanji*, it isn't practical to make a keyboard which contains each character. (Some manufactures, like IBM, have actually done this. However, the keyboards are almost impossible to use!) So, a way of specifying thousands of characters with a small number of input symbols is required. Many such systems have been proposed. Some are based on a completely arbitrary assignment of symbol combinations to *kanji*, and some are based on attempts to describe the shapes of the characters. Most of these systems are difficult to learn and/or difficult to use.

As I explained above, Japanese can be written using only *kana*. So, it is possible to implement an input method, called *kana-kanji conversion*, where the text is entered phonetically in *kana* and converted to *kanji*. Since it is also possible to write Japanese entirely in *romaji*, the *kana* syllables can be spelled in *romaji* and automatically converted to *kana* "on the fly". (This method is popular because many people in Japan are familiar with an English typewriter, while a *kana* typewriter doesn't exist since it wouldn't be particularly useful.) Let's consider how to input the word "kanji", meaning "Chinese ideogram". This word is written with two *kanji*: "kan" meaning "Chinese", and "ji" meaning "character".

The simplest method is to do the conversion one *kanji* at a time. While this can be done, it is fairly cumbersome, since many *kanji* have the same pronunciation but different meanings. Words or *kanji* with the same pronunciation but different meanings are called *homophones*. In our example, there are 143 homophones for "kan" and 46 homophones for "ji". Since the input method cannot tell which homophone the user wants, it must display them all and let the user choose.

The number of homophones is greatly reduced by doing the conversion one word at a time. In our example, the word "kanji" has four homophones meaning "Chinese ideo-

gram”, “feeling”, “manager”, and “inspector”. Each homophone is written with a different set of *kanji*, which makes the intended meaning clear to the reader.



57

# Script Manager 3.0 - Preliminary ERS

draft version 0.6, Friday, January 13, 1989

Peter Edberg

## I. Introduction

### A. Goals

The Script Manager 3.0 Project provides incremental enhancements to the Script Manager, in the following ways:

- Greater user control over the operation of Script Manager functions (Map CDEV, new International CDEV).
- Improved user interface for Script Manager functions (color keyboard icons, Map & Keyboard CDEVs, Key Caps).
- Support for the full Roman character set (character codes \$D9-\$FF).
- Support for new keyboards (e.g. Odin).
- New functions needed by the Finder group, the File System group, and Boffin.
- Functionality improvements in several areas: revamped numerics support; parsing and generation of international outline item designators (e.g. I., A., b., ii.); additional functions to provide feedback about the current keyboard script; a faster and more general mechanism for finding word-select boundaries and line break locations; and KeyTrans modifications for multiple character generation and multiple dead keys.
- Internal use of Boffin routines where appropriate.
- A new itlr resource which contains word break/word select table, character type tables, and character case tables. This makes it easier to modify these tables, which is especially important for the ROM version of the Script Manager kernel.
- Modifications to permit the Script Manager kernel to be in ROM.
- Improved initialization.
- Modifying the ScriptUtil dispatcher to support Boffin calls.
- Efficiency improvements, bug fixes, and cleanups.
- Updating of related development tools.
- Improved external and internal documentation.

### B. What is included in the Script Manager

The following pieces are currently under the ægis of the Script Manager, and will be affected by this project:

- The Script Manager kernel code, which includes the Roman Script System. These are part of the PTCH resource on current system disks, and will also be included in ROM beginning with Harpo (these are supported by itlc, itlb, itl0/1/4, itlr, KCHR, and KSWP)
- The International Utilities in PACK6 (supported by itl0/1/2)
- Keyboard handling (OS modifications), Key Caps DA, and Keyboard CDEV (the first two are supported by KMAP/KCHR and KCAP, respectively)
- The Map CDEV and the new International CDEV
- The international resources (itlc, itlb, itl0/1/2/4, KCAPs, KMAPs, KCHR, KSWPs, and the new itlr)

Note: Modification of TextEdit for Script Manager compatibility is an associated project, and is covered by a separate ERS.

## Script Manager 3.0 ERS

Note: Script Manager 3.0 includes, as a subset, changes that were originally planned for the 2.17/Altair/Betelgeuse versions of the Script Manager. Many of these changes are already complete. Others are relatively low priority, and may be dropped if necessary—for example, changes to the Map CDEV, or revamped numerics support.

### C. What is not included

Script Manager 3.0 does not include the following ISS projects, although they are related:

- InfoWorld (MultiScript System): Provides simultaneous handling of an arbitrary number of scripts.
- Script System Core: A set of documented routines that provide the core of a script system.
- Unicodes: Uniform character codes covering all scripts, with associated character class and script information. Script Manager 3.0 will include tables and procedures based on this information.

### D. Schedule

Some of the Script Manager 3.0 improvements—in particular, changes to itlc, itlb, KCAPs, KMAPs, KCHR, and the addition of itlr—will be included in System 6.0.4 and the international versions of System 6.0.3. These are required for support of the Odin keyboard, among other things.

ROMification of the Script Manager kernel will be completed in time for the Harpo and Cobra II/Four Square ROMs.

Other than the above constraints, Script Manager 3.0 is scheduled for alpha in April '89 and beta in July '89 (i.e., Big Bang time frame).

### E. Dependencies

Script Manager 3.0 depends on:

- the Unicodes project, which will provide a code convertor and a character class table required by the Script Manager.
- some modifications to the INIT 31 procedure to permit cleaner initialization of script systems.
- facilities provided by Boffin (this is a weak dependence).

Script Manager 3.0 features are required by:

- Boffin
- the Finder group [which project?]
- the File System group [which project?]
- InfoWorld (possibly)

## II. Description of Improvements

### A. New International CDEV

This CDEV gives the user some control over measurement units, date/time/currency/number formats, collation sequences, and so on. It allows the user limited editing of the resources that specify these items.

Referring to figure 1: With the top box the user can select the current script from a pop-up menu showing the available choices. The rest of the dialog depends on this choice, since the resources are organized by script. [We could use the keyscript to make this determination; this is less obvious to the user on non-Roman-only systems, but saves real estate.]

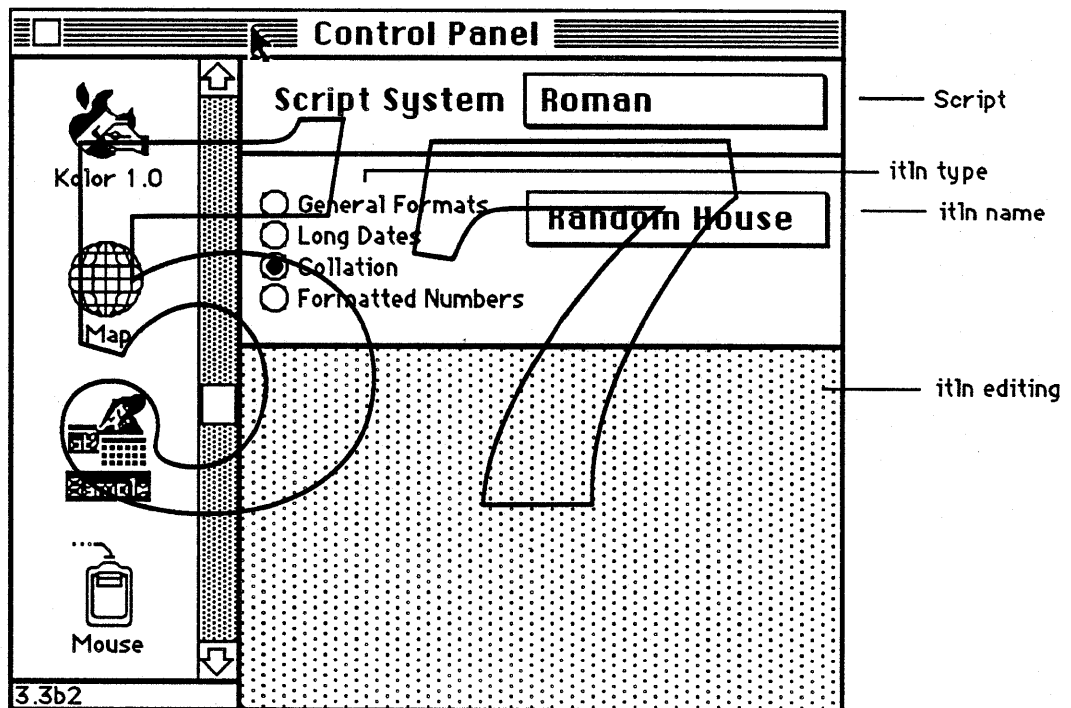


Figure 1

The second box specifies the particular resource to be edited. The radio buttons select the resource type, as follows:

- General Formats refers to the itI0 resource, containing formats for short dates, simple numbers, currency, measurement units (e.g. Metric), and list separators.
- Long Dates refers to the itI1 resource, containing formats for long and abbreviated dates.
- Collation refers to the itI2 resource, containing code resources for determining the collation sequence.
- Formatted Numbers refers to the itI4 resource, containing formats for extended number formats, Macro tokenizing, and upper/lower/type character tables.



## Script Manager 3.0 ERS

For each type, the name of the currently-active resource of that type is displayed. The name display is a pop-up menu, so that users can select alternate resources of the same type.

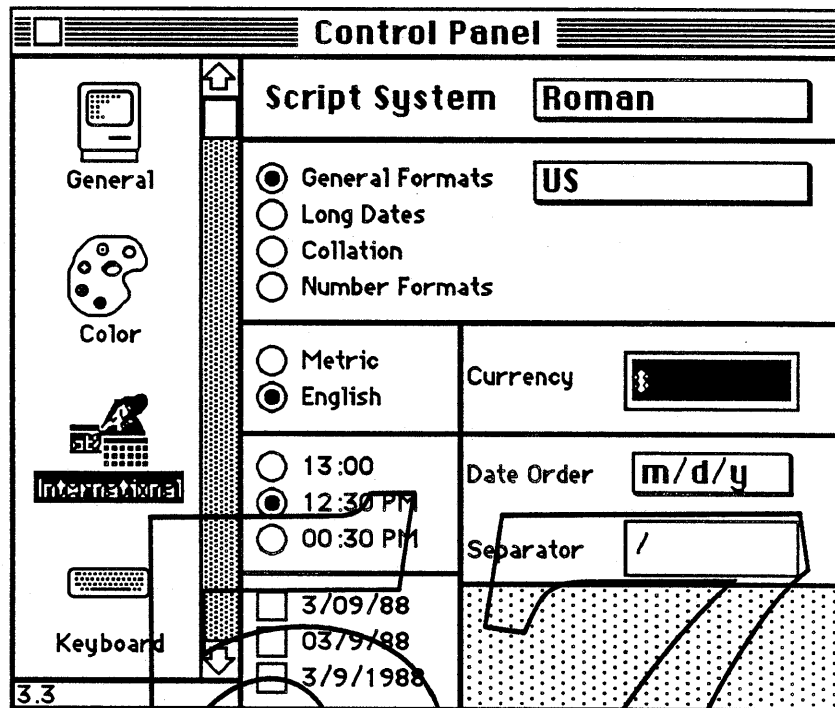


Figure 2

Some resources allow editing. For these resources, an editable portion of the window appears, as shown in figure 2 (Some countries may not want to offer all the editable features, so individual items can be disabled). Pop-ups will be used in cases where the number of items may vary dynamically, or there are a relatively large number of options. For example, the date order may be: m/d/y, m/y/d, d/m/y, d/y/m, y/m/d, or y/d/m; a mouse-down in the date order field will display all of the options.

A prototype is currently implemented that permits editing of itl0 and itl1. The user interface design is only a first pass. Testing of different implementations is required, as well as feedback from Human Interface, International Engineering, etc.

### *Open issues:*

- *Do we want a button—say, 'New'—to allow the user to create a new resource with a different name, which can then be edited?*
- *Do we want some sort of confirmation dialog when editing?*
- *Should the user's editing be changed on reboot, or take effect immediately?*
- *Should we add fontForce and teSysJust switches, since they are common to many scripts? (leaning towards yes)*
- *How about letting the user set the showIcon bit (in itlc) from the International CDEV? (This bit forces display of the keyboard script icon even if only one script is installed)*

## Script Manager 3.0 ERS

### B. Improved Map CDEV

The MAP CDEV adds:

- Daylight savings support
- Support for irregular time zones
- Several fixes and extensions to the data base
- MacDraw-style zooming controls, replacing the presently-undocumented keyboard-activated zoom feature
- Documentation of two hidden options—selection of miles, kilometers, or bearing, and selection of time difference or time zone

It will also have a Color Map, with the colors chosen so that they appear black and white on a non-color system (We have such color bitmaps now, but we will need time from the Human Interface or Creative Services group to make them pretty).

Finally, the general human interface will change to incorporate many of the recommendations made by the Human Interface Group.

### C. Color keyboard icons

The keyboard script icons used in the menu bar and in the Key Layout section of the Keyboard CDEV will support color.

### D. Better user interface for dead keys in Key Caps

Key Caps will indicate that a given key is a dead-key. After a dead key has been pressed, the Key Caps keyboard shows what characters would result from pressing another key. A compound key will be shown properly (e.g. ü, ñ, ...) and highlighted, while an invalid dead-key combination will not be highlighted.

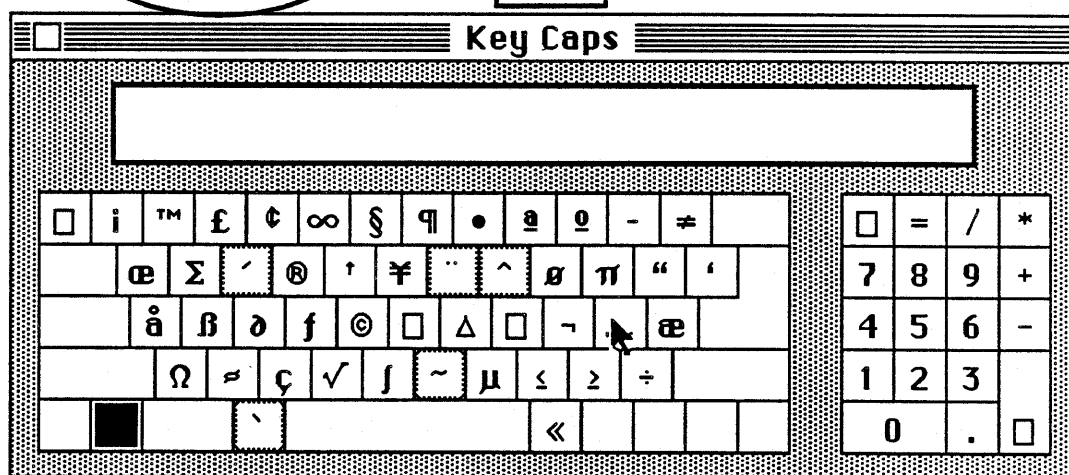


Figure 3

## Script Manager 3.0 ERS

### **E. Support for full Roman character set**

The current Roman character set on the Macintosh is inconsistent: we distribute LaserWriter fonts that use characters from \$D9 to \$FF, but do not support these characters in system software.

Full use of the extended Roman character set (codes \$D9-\$FF) requires the following:

1. All of the KCHR resources (which map virtual key codes to characters) must also be updated to support these characters. These resources are used by KeyTrans, the Script Manager kernel, and Key Caps.
2. The word break and word select tables (used by FindWord), type tables (used by CharType), and case tables (used by Transliterate) must all be updated to reflect these characters. These tables have all been moved to the new itlr resource.
3. The string comparison tables in itl2 and the tokenizer tables in itl4 must also be updated.

### **F. Support for new keyboards**

Each new keyboard requires a KCAP resource for use in Key Caps; this describes the shape and location of each key. New keyboards may also require a new KMAP resource, which indicates the mapping of raw key codes to virtual key codes.

Script Manager 3.0 will include new KCAPs for the Odin keyboard, the Harpo keyboard, and the Harpo ISO keyboard; it will also include a KMAP for the Odin keyboard (these resources will all be included in System 6.0.4 and the international versions of System 6.0.5).

### **G. KeyTrans handling of multiple characters**

In the current KeyTrans model, one or two characters result from a sequence of one or two keys. Script Manager 3.0 will modify KeyTrans to provide a more general model, in which an arbitrary sequence of keys can produce an arbitrary number of characters. This is necessary to meet ISO keyboard standards.

This will require modifications to KCHR, KeyCaps, and KeyTrans.

### **H. Character code convertor and access to new character type tables**

In the current Macintosh model, the association of character codes with characters depends on the font; code \$B6 in an Arabic font represents a different character than code \$B6 in a Japanese font. This presents a number of difficulties for the handling of international text. The Unicodes project is developing a set of 16-bit codes that represent characters in a font-independent way; these codes will be used internally by Boffin, by Apple File Exchange, by the ClearingHouse software, and for accessing certain tables in Bass fonts.

Script Manager 3.0 will provide a function that converts current Macintosh character codes in a particular font to Unicodes. It will also provide a function that can return character class information for a given Unicode, including such attributes as letter/punctuation, intrinsic direction, and script.

### **I. New font hooks**

Script Manager 3.0 will provide functions that can return the following for a given script: a monospaced font, the small size for the application font, and the default font for high-quality printing.

## Script Manager 3.0 ERS

These functions have been requested by the Finder group.

### **J. Method for obtaining canonical strings for use by HFS**

This will be used to make filename string comparison internationally sensitive; it is needed by the File System folks. This feature will depend on a set of modular, installable tables associated with each script. Script Manager 3.0 will include a function that retrieves these tables for a particular script and a function that, using these tables, can convert a text string to some script-dependent canonical form. In Roman script, for example, this might be all uppercase.

This canonical form can then be used for a simple and fast byte-by-byte comparison, which tests for "weak" identity (e.g., 'A' and 'a' are weakly equal). However, this canonical form does not provide enough information to sort filenames in correct collating sequence.

### **K. Additional KeyScript functionality**

The following two functions will be added:

- ShowIcon, which could be called from a CDEV (probably the International CDEV) to force display of the current script icon.
- SwitchKeyboard, which could be activated with a KSWP-defined keyboard command. It will switch the current keyscript with the last keyscript.

Both of these will probably be implemented as calls to KeyScript with new verbs. These functions are needed to provide more feedback about the current keyboard.

### **L. New FindWord**

Each script system presently provides its own version of FindWord, which locates word boundaries for word selection and line breaking. A new, state-table-driven version of FindWord has been written which is faster and more general than the current one. If supplied with appropriate tables, this new version should be able to replace the script-dependent versions and provide additional functionality. This will become part of the Script Manager kernel.

### **M. Revamped numerics routines and tokenizer**

These will be rewritten to reduce size and improve speed. Parsing and generation of international outline item designators (e.g. II., A., b., ii.) will be added to the Tokenizer.

### **N. Resources**

A new itlr resource has been created, which contains the word select and break tables for use with FindWord, as well as the character type and case tables for use with CharType and Transliterate. This provides the capability to easily modify these tables.

Any system that will use the ROM version of the Script Manager (or Script Manager 3.0) must include this resource. It will be part of System 6.0.4.

### **O. ROMification**

The following modifications permit the Script Manager kernel to be put in ROM. The changes listed below will be conditionalized, so that the same source code can be used to build either a ROM version or a system disk version of the Script Manager.

## Script Manager 3.0 ERS

- Roll in Script Manager patches to Toolbox and OS routines: InitResources, InitFonts, InitMenus, InitWindows, InitApplZone, TEInit, DrawMenuBar, GetOSEvent, LwrString, UprString, KeyTrans.
- Put the \_ScriptUtil trap permanently in the system dispatch table
- Allocate new RAM structures for the Script Manager and Roman script system dispatch tables; allocate another RAM area for vectorization of Script Manager internal routines.
- Move modifiable fields out of the kernel!
- Modify Key Caps to check for KCHR in ROM.

### **P. Initialization improvements**

- INIT 31 handling needs to be modified to guarantee that script systems get initialized before any other INITs. This could be done by testing for a special character in the creator name, or something similar. We need help from Brian McGhie on this.
- Always initialize ExpandMem if it is not there.
- Make sure all relevant values from itlc are obtained at boot and switch launch times.

### **Q. Sharing the ScriptUtil trap with Boffin**

Because traps are so hard to come by, it may be necessary for Boffin to share the Script Manager trap. In that case, the ScriptUtil trap dispatcher would have to be modified to support Boffin calls.

### **R. Updating development tools**

We will ensure that the MPW resource types and ResEdit templates for the international resources are up to date.

### **S. Documentation**

Script Manager external documentation will be corrected and improved, and internal documents will be prepared describing Script Manager data structures, resources, and aspects of operation.

# TextEdit 3.0 ERS

## Functionality

New functionality has been added to TextEdit to make it Script Manager compatible. The current implementation of TextEdit does not incorporate the changes that were made to the original TextEdit to work with different script systems. The majority of this new functionality will only be apparent on non-Roman script systems. These new features are on top of the 6.0 TextEdit, and support styled TextEdit.

TextEdit will now handle:

- double-byte characters
- right-to-left directional scripts
- highlighting of mixed-directional scripts
- split carets for mixed-directional scripts
- keyboard-font synchronization
- word and line breaks
- measuring the visible length of text
- background outline highlighting
- text buffering for performance improvements
- cursor movement across mixed-directional scripts

TextEdit 3.0 will be included in the new MacII family ROM and in the Big Bang version of system software.

## Double-Byte Characters

If a two-byte character is input to TextEdit, the first byte is buffered until the second byte is processed, at which time the character will be displayed.

When TEKey is called with a right or left cursor key or a backspace at a double-byte character, then the selection point is updated beyond the second byte in order for the caret to move once over the entire character. TextEdit also makes calls to the Script Manager for hit test, caret display, highlighting, key buffering, etc. for double-byte characters.

## Right-to-Left Directional Scripts

Style runs are now used in display order rather than backing-store order for a right-to-left directional script. Style runs on a line are clustered into script runs. Text flows and style runs are ordered in the script's line direction. Hence, whenever TEDelete, TEInsert, TEDoText or any other TextEdit routine is called requiring line adjustment or redraw, hit-testing, or measurement of mixed-directional text, TextEdit orders the script runs so that they can be used in display order and not backing-store order.

## Highlighting of Mixed-Directional Scripts

Whenever a routine is called that causes TextEdit to highlight or unhighlight a selection encompassing mixed-directional text, the style runs included between the selection points, in backing-store order, are highlighted. The text highlighted may appear discontinuous on the display line. The routines specifically needing highlighting of a selection or as a result of their functions include TEClick, TEActivate,

TEDeactivate, TEsSetSelect, TECut, TEDelete, TEInsert, TEPaste, TEsStylInsert, TEsSetStyle, TEReplaceStyle, and TEDoText.

### Split Carets for Mixed-Directional Scripts

If TEDoText is called to position the pen to draw the caret, and a right-to-left script system is being used, then split carets will appear under certain conditions. The conditions for this to occur are that the cursor is on a cluster boundary<sup>1</sup> or on a block boundary within a native-script style run<sup>2</sup>. The split carets are displayed from the top and bottom of the line, and are only half the line's height. They are displayed at the current character offset with the *high* caret displayed with the style in the dominant line direction and the *low* caret displayed with the style in the opposing line direction.

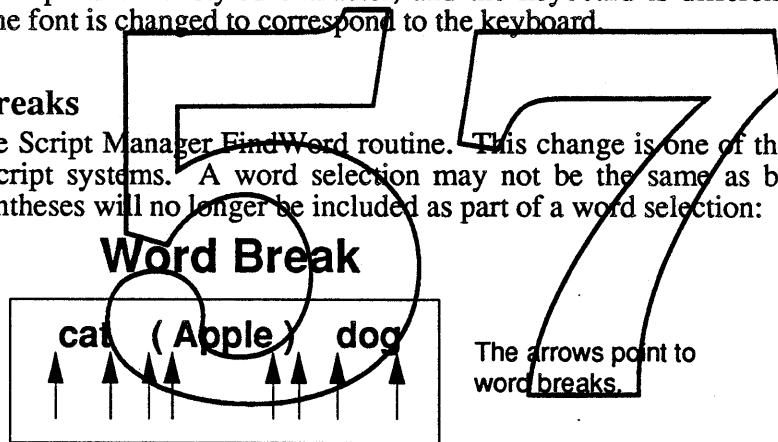
### Keyboard-Font Synchronization

If a TEsSetStyle, TEsSetSelect, or TEClick call is made to change the font style or process a hit-down in text either as an insertion point or selection, the keyboard is changed to correspond to the font at the selection point.

If TEKey gets called to paste an unstyled character, and the keyboard is different from the font at the selection point, then the font is changed to correspond to the keyboard.

### Word and Line Breaks

TextEdit now calls the Script Manager FindWord routine. This change is one of the few that will also be apparent to Roman script systems. A word selection may not be the same as before. E.g. A known difference is that parentheses will no longer be included as part of a word selection:



TextEdit also line breaks correctly for single-script text or multiple-scripted text since it calls the Script Manager LineBreak routine.

### Line Break

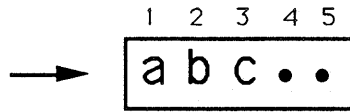
...order reorder order re-  
order order reorder ...

### Visible Length

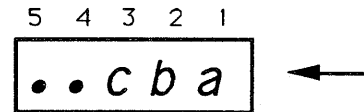
1. A cluster is a script run of one or more style runs in that script.
2. A block is a run of either all Roman or all Native-script text. For historical compatibility with plain-text applications and system software, each script allows Roman text to be mixed in. This mixed text (Roman and Native) can be broken up into blocks. See the Script Manager 2.0 ERS for information on the FindScriptRun routine for more information.

In order to accurately measure a line, it is necessary to remove any trailing white space from the end of the line, with the line direction taken into account. This means that downstream white spaces at the end of the backing-store are stripped. Downstream means that the character direction is the same as the line direction.

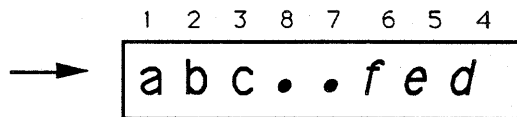
## Visible Length



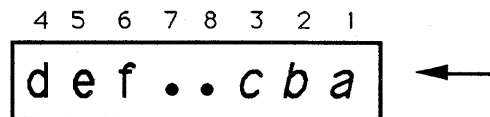
VisibleLength of this Left-Right example = 3.



VisibleLength of this Right-Left example = 5.



VisibleLength of this Left-Right example = 8.



VisibleLength of this Right-Left example = 8.

## Outline Background Highlighting

There is a flag for highlighting that allows outline highlighting when a window is in the background. This will be of primary interest to Everest.

## Text Buffering

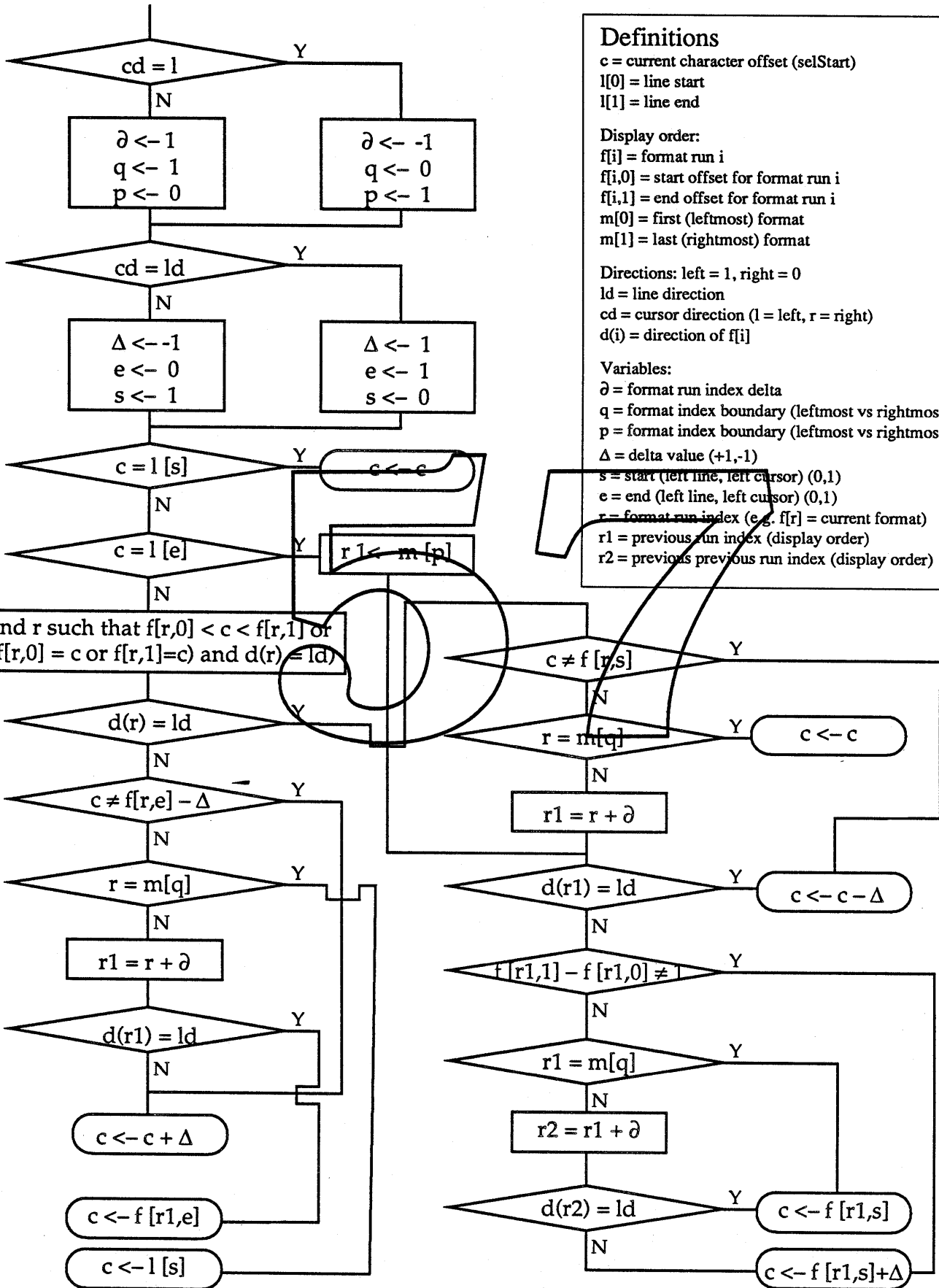
Text buffering occurs for Asian performance improvements. We buffer each TeKey input until a condition occurs that requires us to dump the buffer. This condition could be a teIdle call; a Tekey call for a backspace or control character; any routine that does cut, copy, or paste; or the buffer space is full. The buffer is dumped before the condition is handled.

## Cursor Movement Across Mixed-Directional Scripts

When using the arrow keys to move left and right across characters on a line, the cursor should move in the direction of the arrow. This is not a difficult implementation when cursoring in the middle of any style run. However, the model becomes very complicated on cluster boundaries or any character position directly to either side of the cluster boundary. The high caret should always display in the next position (at the character offset) in the direction of the cursor key. The algorithm to solve this problem is demonstrated below.



# Curs(or)ing thru Mixed-Directional Text



## Definitions

c = current character offset (selStart)  
 l[0] = line start  
 l[1] = line end

### Display order:

f[i] = format run i  
 f[i,0] = start offset for format run i  
 f[i,1] = end offset for format run i  
 m[0] = first (leftmost) format  
 m[1] = last (rightmost) format

Directions: left = 1, right = 0

ld = line direction

cd = cursor direction (l = left, r = right)

d(i) = direction of f[i]

### Variables:

∂ = format run index delta

q = format index boundary (leftmost vs rightmost)

p = format index boundary (leftmost vs rightmost)

Δ = delta value (+1, -1)

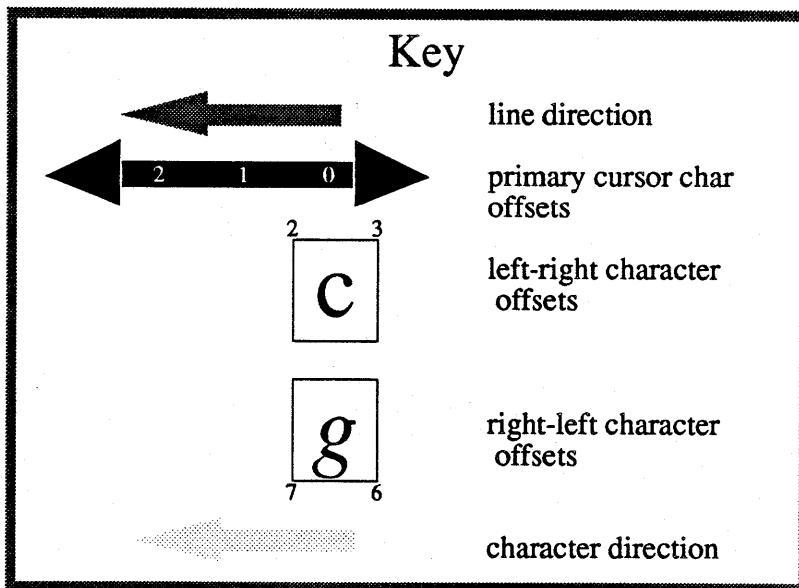
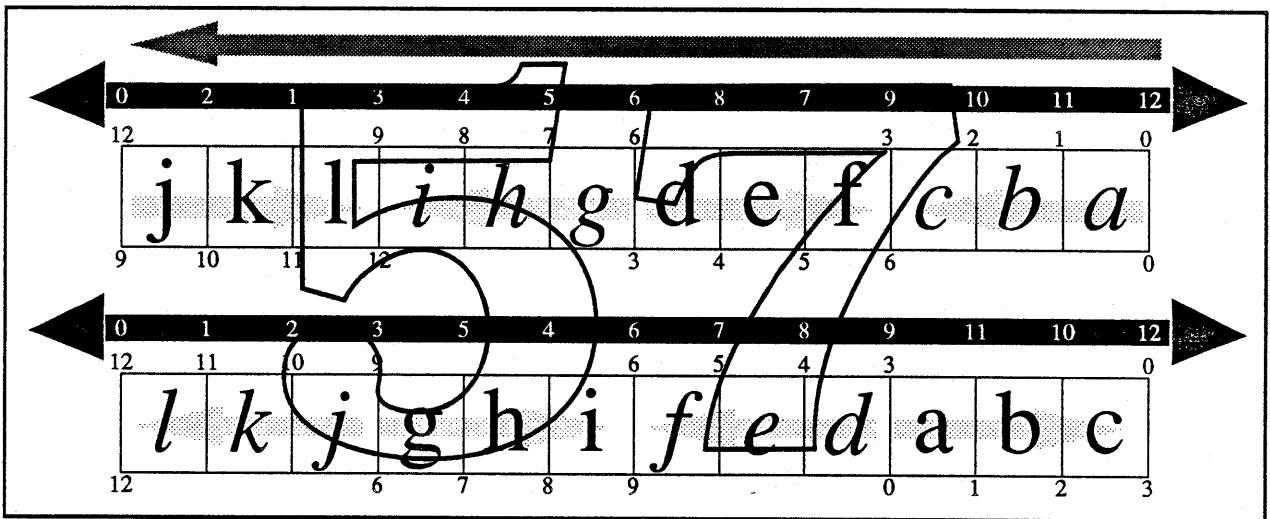
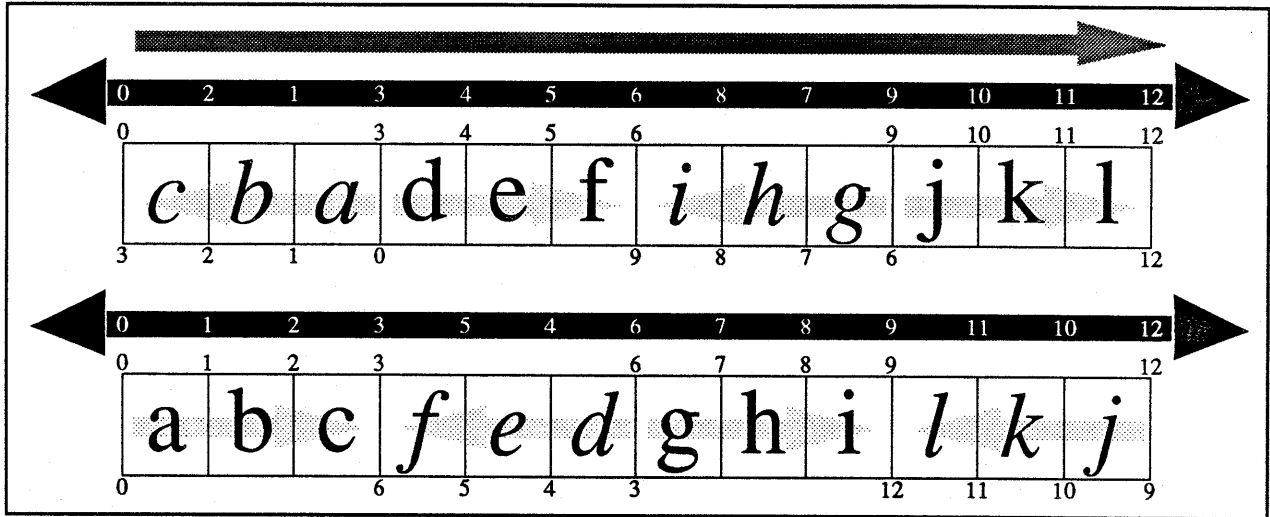
s = start (left line, left cursor) (0,1)

e = end (left line, left cursor) (0,1)

r = format run index (e.g. f[r] = current format)

r1 = previous run index (display order)

r2 = previous previous run index (display order)



57

# InfoWorld ERS

## Introduction

The Multi-Script system will provide any generic (English) system with the capability of handling many scripts, Roman and non-Roman, simultaneously. Currently, the main limitation in terms of multilingual text handling on the Macintosh is that we supply at most two scripts at a time. However, the Script Manager internal architecture has always supported up to 64 script systems at the same time. The Multi-Script system allows users to install an arbitrary number of scripts to be handled simultaneously.

True simultaneous multilingual capability is necessary for a wide variety of clients including government sales, multinational corporations, and many foreign companies.

## Functionality

The goal of this project is to produce a generic system with multiple script systems installed. This will allow users to use text of different scripts, such as Japanese, Chinese, Korean, Hebrew, Arabic, Thai, and Greek all within the same document.

The main effort involved in this project is to identify other parts of the system that need to be changed for a more seamless interaction. ~~For example, file names need to have associated fonts, which requires modification to the Finder and Standard file.~~ Dependencies include: the File System, the Finder, Utilities (Control Panel, Font DA Mover (or equivalent)), Script Systems.

## Installer Scripts

Installer scripts are needed to install each of the script systems onto an English system. These take a folder or diskette for each script system and install the necessary files into the system folder, and resources into the system file.

## Script Systems

Necessary modifications include the de-localization of the script systems, and correction of any bugs that would prevent the script system from working properly in a multi-script environment. For example, Kanji-Talk currently displays Japanese text in dialogs in the system font, expecting that system font to be Japanese. When it is not, garbage appears. All of the script systems will also need to support MultiFinder.

## System Software Dependencies

The other system software that has been identified as requiring changes include the following:

### 1. File/Folder names with scripts.

Character encodings are determined by script, which is determined by font. Unless we maintain font or script information with file names, this information is lost, and display and sorting of file names will be incorrect.

This feature involves:

- Modifying the Finder and Standard File to show fonted file/folder names.
- Adding the editing of file/folder names to the Finder. (If arbitrary fonts are supported, a menu needs to be added for changing a filename's fonts; if scripts alone are supported, the Script Manager KeyScript can be used.)
- Adding Get/PutFileFont (or FileScript) calls for applications.

### 2. Filename case-sensitivity.

There is a problem with non-Roman scripts, in that filenames are incorrectly viewed as equivalent. This causes, for example, the incorrect file to be deleted with a "Replace existing file" alert.

This feature involves:

- Producing case-sensitive volumes.
- Adding a button to the disk-init package to allow case-sensitive volumes.

However, this is a fall-back solution for the file system. The correct solution is to provide the ability to do file name identification by means of tables that are script sensitive. A table would be identified as belonging to a particular script, and would be used to differentiate file names. Two file names would be different if they have different script numbers or if they have the same script number but different table mappings. Otherwise, they are the same.

### *3. Mover for Keyboards, International resources*

Currently users can choose among different keyboards (KCHR, SICN, itlk combination) and different international resources (itl0, itl1, itl2, itl4, itlr) but cannot move them in and out of the system (without using ResEdit). The Font/DA Mover must have this capability.

### *4. International Clock Setting (Alarm Clock/ Control Panel)*

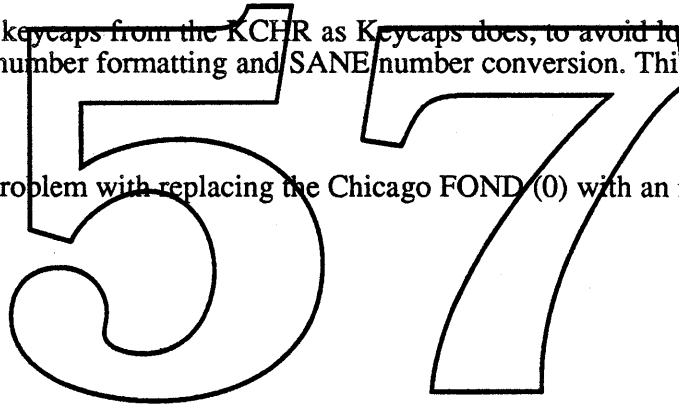
All relevant applications and Desk Accessories should use the Clock Manager. (see Scott Douglas)

### *5. Calculator*

The calculator should get the keycaps from the KCHR as Keycaps does, to avoid localization problems. It should also use international number formatting and SANE number conversion. This should also represent a code savings.

### *6. Installer*

The installer currently has a problem with replacing the Chicago FOND (0) with an identical FOND which has id \$3FFF.



# Script System Core ERS

## Introduction

The Script System Core will provide a shell enabling developers to produce a script system with a minimum of work. It should serve as a basis for new script systems, as well as old script systems that are being revised. We will abstract the common basic features from all Script Interface Systems.

Apple cannot attempt to develop systems for all possible scripts. However, for many languages, script systems are absolutely required for use of computers. By allowing third parties to supply script systems, we leverage off of their efforts, while maintaining quality script systems.

## Functionality

The Script System Core will be based upon a union of the core of the Middle Eastern and East Asian systems. It will handle initialization, dispatch, resource support and usage, as well as provide traps for the main text routines. It will also contain hooks for all of the standard Script Manager procedures and trapped routines. It will be formatted as a library and written in C or C++, with some assembly glue as necessary. There are two phases of this project; it has yet to be determined whether Phase Two will be done.

### Phase One

- **Initialization**

We will provide a standard method of handling error alerts during initialization, preferably using icons rather than text. Presently on the Japanese Interface System, different writing methods for alerts are available at different phases during initialization of the system. Using icons for alerts would greatly simplify this task.

- **Data Structures for Script Systems**

The data structures will be for the low-level script globals.

- **Interface to simple routines**

- **Interface to Boffin**

### Phase Two

- **Asian input methods**

The Script System Core is most useful in the context of the Multi-Script System. Even with good production tools, good documentation will be necessary to produce a useful system. We will document how to use the Core to implement a Script Interface System by creating a Cyrillic SIS as our test case.

## Packaging

The Script System Core will be distributed to developers but we will require developers to register their character sets with Apple due to data compatibility issues.

57

# Unicode ERS

## I. Macintosh to Unicode Character Code Conversion

Character coding in the Macintosh currently relies on the association of one or more eight-bit bytes (two are required for Chinese, Japanese, and Korean) with font information. For efficiency reasons, Boffin will internally use pure sixteen-bit characters (UniCode), requiring code conversion from the current Macintosh format. This conversion can be understood as a mapping of pairs consisting of font ID and single or double byte characters to one or more sixteen bit values.

$$f(\text{font, bytes}) = \text{UniCodes}$$

Multiple UniCode characters result from breaking composites that exist as single characters in the Macintosh set (e.g. é) into base character plus diacritic components (e + ´).

Boffin will require that the conversion be done so that it can recover the offset of the characters in the original text. We might return an array of offsets or work on a character-by-character basis.

The current ScriptManager does not provide sufficient granularity to identify "Roman" script fonts such as the Adobe Symbol font that in fact contain characters that will map to non-Roman segments of the sixteen bit character set (for example, the Greek letters). Hence, we need to introduce an interface in the ScriptManager to allow developers of such overloaded fonts to provide their own unique mapping table.

## II. Unicode Character Properties

Boffin requires character property information for correct, high quality layout of complex, multi-directional scripts. At a minimum, two properties are associated with each character, *directionality* and *quality*.

The directionality of a character is *forward*, *backward*, or *neutral*. Forward means the default layout direction is left-to-right or top-to-bottom. Backward means the default layout direction is right-to-left or bottom-to-top. Neutral means the default layout direction depends on the surrounding characters.

Quality refers to whether a character is a *base character* or an *applied mark* (see the Boffin ERS). Quality determines how the character interacts with other characters in the layout process. As the names suggest, characters which have the quality of applied-mark are applied to base form letters. For example, consider the character é.

Exactly how properties will be associated with each character is still under investigation. We do know that they will be table-based and that these tables must be either small or modular for storage efficiency.



57

57

57

---

# Blue Toolbox Overview

Phac Le Tuan

The "Blue Toolbox" has historically been a convenient home for various System Software projects when they did not seem to strongly belong to any other groups of the Blue Macintosh team. Some of the various projects described hereafter have started as independent investigations, orphan projects, or remnants of reorganization waves.

So, is there any chance to see a common vision serendipitously be unearthed from those seemingly unrelated projects which just happen to be sharing a common home from some time?

Identifying and resolving the dependencies as well as crafting the common thread between the various projects has been an interesting task, aimed at ensuring user interface consistency and providing directions for new developments like the Database Access Tools (SnarfMan), or DDIF (Esperanto).

## Blue Toolbox Guidelines

Four buzzwords are attached to the areas currently addressed by the Blue Toolbox:

- User Interface
- Inter-Application Communication
- Database
- Sound

The areas in question are in fact quite large and it is clear that many other groups inside Apple are also addressing them, from a different perspective though. What the Blue Toolbox is set up to achieve is to provide solutions which are:

- general enough to be useful for a majority of developers,
- powerful yet simple enough to be compelling to use, reinforcing the traditional Macintosh ease-of-use and user interface consistency,
- available in Blue, with the current MacOs environment.

A Macintosh is easy to use. The Blue Toolbox will make it easy to share data on a Macintosh.

Exchanging data with a spreadsheet or a word processor application (with Diet-Coke) is not inherently different from accessing a remote database and obtaining data as a result of a query (with SnarfMan). The user experience in both cases should not be fundamentally different.

Also, sharing data involves not only having tools to establish communication between different applications, but also to agree on the type of data being exchanged. This translates into the need to have standard interchange formats supported by System Software, and being able to handle text, graphics, database, and sound. Under these circumstances, the Esperanto project, although limited today to interchanging data with DEC's Vaxen only, appears to be a step in the right direction.

Sound is a particular example of an area where a lot has to be done, since the software is lagging well behind what the Macintosh hardware is capable of today. On top of providing tools which will bring as much power and flexibility in managing sound as Quickdraw brings to graphics (with D.J. and the Sound System), the sound efforts will also give the Mac the capability to exchange sounds between applications, with as much direct manipulation as possible (Visible Sounds).

In addition, the Toolbox, as the traditional guardian of the user interface, has to continuously provide tools which make it easier to develop friendly applications, especially when the environment becomes more complex, with all the new features introduced by Big Bang, and the capabilities of the future CPUs.

Some of the projects above already require extensions to the user interface (Diet-Coke and SnarfMan must share a new Menu item in the Menu Bar, On-Line Help introduces its own omnipresent menu item), so it was necessary for Glass-Plus, the new generation of Menu and Window managers, to revisit some basic concepts, and to go well beyond the simple benefits of tear-off menus.

### Diet Coke

One tool to share data between Applications is provided by Diet Coke, which defines how the user selects a section of data to share, and how he will dynamically have it included in an other document.

Diet Coke implements the concept of sharing data between applications by focusing on the user experience and providing one simple new user paradigm to resolve multiple user situations: the Publication.

A section of shared data is called a Publication, documents including shared data are Subscribers to the Publication. When a Publication is updated, Subscribers are automatically notified, and updated if desired. Publications can be shared between several Subscribers. Users can move, delete, copy files with shared data without ever creating confusing situations...

The emphasis given to the data to be shared - Publications are visible on the desktop under the Finder and can be directly manipulated - has led to minimize the navigational aspects associated with the links created between documents sharing the same data. We believe the duality Shared Data/Links is an orthogonal one and that addressing only the Shared Data aspect in Blue is an acceptable compromise between complexity and time to market. Also, the Link Navigation aspect is extensively addressed right now by Pink.

Being able to define which section of data is shared is insufficient if the data which is being imported is not understandable by the receiving application. Currently, the Macintosh Scrap Manager, which is used to access the Clipboard, only supports two data types, the PICT format, and the Styled TextEdit format. The first version of Diet Coke will also support these two data types only. Clearly, this will be insufficient in the future (more sophisticated Text Layout is being developed by International System Software, spreadsheet data are commonly candidates for sharing, SKIA will introduce new graphics features...), and extensions to the supported data types are to be considered.

Another important aspect in Inter-Application Communication is Network support. Clearly, Network support would make it very compelling for developers like Aldus to adopt Diet-Coke, since they cannot provide such a capability by their own development. Diet-Coke has been designed in such a way that such an extension should not be a problem, and it will work unchanged on any Appleshare volume. In order to provide a higher level of Network support, more investigation is needed, but this will be covered only by the next release of Diet-Coke (Jolt).

### Glass-Plus

Started initially as a quick implementation of tear-off menus to support large and/or multiple monitor environments, this project has evolved into a reassessment of the capability of the Menu and Window managers to support a newer generation of applications with ever-increasing requirements on the user interface.

The menu bar itself has been under close scrutiny, to extend its capabilities yet keep it simple. The whole gamut of possibilities, from scrollable menu bars to menu bars within application windows, to multiple-line menu bars, has been investigated, without much success so far. A simple idea is being investigated, which is to replace system level menu items, such as **File**, **Edit**, **Help**, and **Exchange** (a new one introduced by Diet-Coke and SnarfMan), by icons on the menu bar. Their ubiquity will make them non ambiguous by nature, and they do not occupy more real estate on the menu bar than the original **File Edit** ones. Further user testing is needed, and the availability to applications of icons on the menu bar will also be investigated.

Glass-Plus extends the concept of Windows with the unifying concept of a Hierarchical Window System. Such an architectural foundation makes it easier to implement Tear-offs and Floating Windows, with additional benefits to MultiFinder among other things. However, it is not clear yet whether the availability of Hierarchical Windows per se will be a boon to developers or a curse to end-users because of the potential complexity they may bring, if implemented carelessly. Again, there is a tremendous need for human interface guidelines in this area.

Another critical feature of Glass-Plus is the support of Customized Menus, thus allowing the user to further simplify his work environment. The major hurdle here is also the user interface, since the implementation will be relatively straightforward thanks to the architectural foundation mentioned above.

Overall, for Glass-Plus, user interface is the most critical and the most innovative part as well. Direct manipulation is a must, and several solutions are being investigated in order to satisfy the needs of naive users as well as power users. For example, mouse pattern movements may be used to tear off individual menu items, and combined with command keys, they could provide short cuts to power users. On the other hand, the idea of "Stick Down" menus has the potential to provide a simple solution to tear offs, customized menus as well as to On-Line Help.

### SnarfMan

The Database area is a very difficult one, not only because it is a central issue for a lot of applications as soon as they grow in complexity, but also because it is convoluted with the strategies of a large number of influential developers and even computer manufacturers. There is also the need to provide a consistent growth path for CL/1, which is the visible part of our database strategy iceberg today.

The components of the proposed Database system fall into three parts:

- SnarfBoy, which provides a mechanism to execute queries by sending them to the right Database server,
- SnarfTools, which are interactive tools to create queries and store them in a way compatible with SnarfBoy,
- SnarfBase, a lightweight database engine.

Concerning SnarfBoy, a major concern is to make sure the user interface is consistent between SnarfMan and Diet Coke, since the concept of importing data is the same whether the data come from a database or from another document. A classic application is one where SnarfBoy provides access to a remote database of stock quotes, and makes it available to an application which would display the results each time they change. The beauty of the combination becomes even more attractive with the possibility to make asynchronous queries which automatically create a Publication for various Subscribers, be they active (then Diet-Coke notifies them right away) or inactive (then they get updated the next time they open).

SnarfBase is a concept which still needs a lot of investigation, and at this time, there is no plan to make it available with Big Bang. SnarfBase will probably not be as pervasive as the Object-oriented database intended for Pink, but it should provide a much needed tool for lots of application developers who do not need full-fledged engine like the ones available on the market today, or it could even be used by database vendors as a foundation to port their database server on the Macintosh.

## Esperanto

Better known as the DDIF project, Esperanto is developed as part of the Apple/DEC alliance, and will provide means to exchange as transparently as possible documents between a Macintosh and a Vax.

The Digital Document Interchange Format (DDIF) is the most visible part of DEC's Compound Document Architecture, and is the preferred vehicle to exchange documents between VAX applications. On the Macintosh, the closest such thing is the format used by MacWrite 4.5 or 5.0, which is the *de facto* standard for almost all word processor oriented applications. Esperanto will provide converters between these formats, running initially on the VAX for performance reasons, and automatically triggered by direct manipulation from the Macintosh desktop or from within compliant applications.

This solution will ensure immediate effectiveness for most users since it will work with their existing applications. The conversion process will leave out some peculiarities of DDIF which have no counterpart in MacWrite (such as footnotes, multiple columns...), but most advanced word processors usually provide simple tools to cope with the problem.

Another critical part here is, as usual, the user interface, which will be designed to leverage on the Grinder concept introduced by NuFinder, which becomes the Invoker for Esperanto. An API will also be provided in order to allow applications to invoke the converters directly. Esperanto could then be the second step in the direction of a revised Apple File Exchange desktop service (the first one being the Grinder itself).

Although not directly in the mainstream of System Software development as such, this project is interesting in the sense that it addresses a real problem, as unveiled by the Inter-Application Communication tools provided by Diet-Coke: what will support the interchange of documents between applications on the Macintosh, or between the Mac and other computers ?

Compound Document investigations in ATG will not provide an answer to this question in Big Bang time frame. Clearly, DDIF cannot be such a candidate because it is overly complex in some areas (page layout for instance) and still incomplete in others (graphics, page layout...), practically inappropriate for Macintosh Applications. SKIA, for example, will outperform DDIF's graphics capabilities, and we would have to consider the various possibilities and consequences of helping DDIF graduate to our level.

Defining an Apple supported Interchange Format will also make the transition between Blue and Pink more palatable, freeing Blue projects such as SKIA from some restrictive Pink compatibility constraints, and still ensuring a smooth transition for users (it is more difficult to ensure easy transition for developers, who will have to learn a whole new programming model).

### Sound Manager et al.

The Sound capabilities on the Mac today already outclass by far most PCs on the market, however, it is also true that the Sound Manager does not yet satisfy all basic developers' needs. So far, what is currently cooking is intended to provide tools which will make full use of the hardware capabilities available. It will be very easy to add good quality sound to every application, thanks to numerous improvements to the Sound Manager (multiple channels of sound, automatic Sound Expansion...) and to three new additions:

- the Sequencer (Project D.J.) will allow applications to fully control the generation of long sequences of sounds,
- the Sonic System will provide applications with hooks for attaching sounds to particular user actions,
- "Visible Sounds" will bring direct manipulation of sound resources to the user, with a wealth of future possibilities still to be investigated (should a sound be played when double-clicked on, or dragged to RecordPlayer desktop service ?...).

Sound is an integral part of the user interface, and as such, necessitates a lot of user testing not only on the way it behaves, but also on what sounds are shipped with System Software.

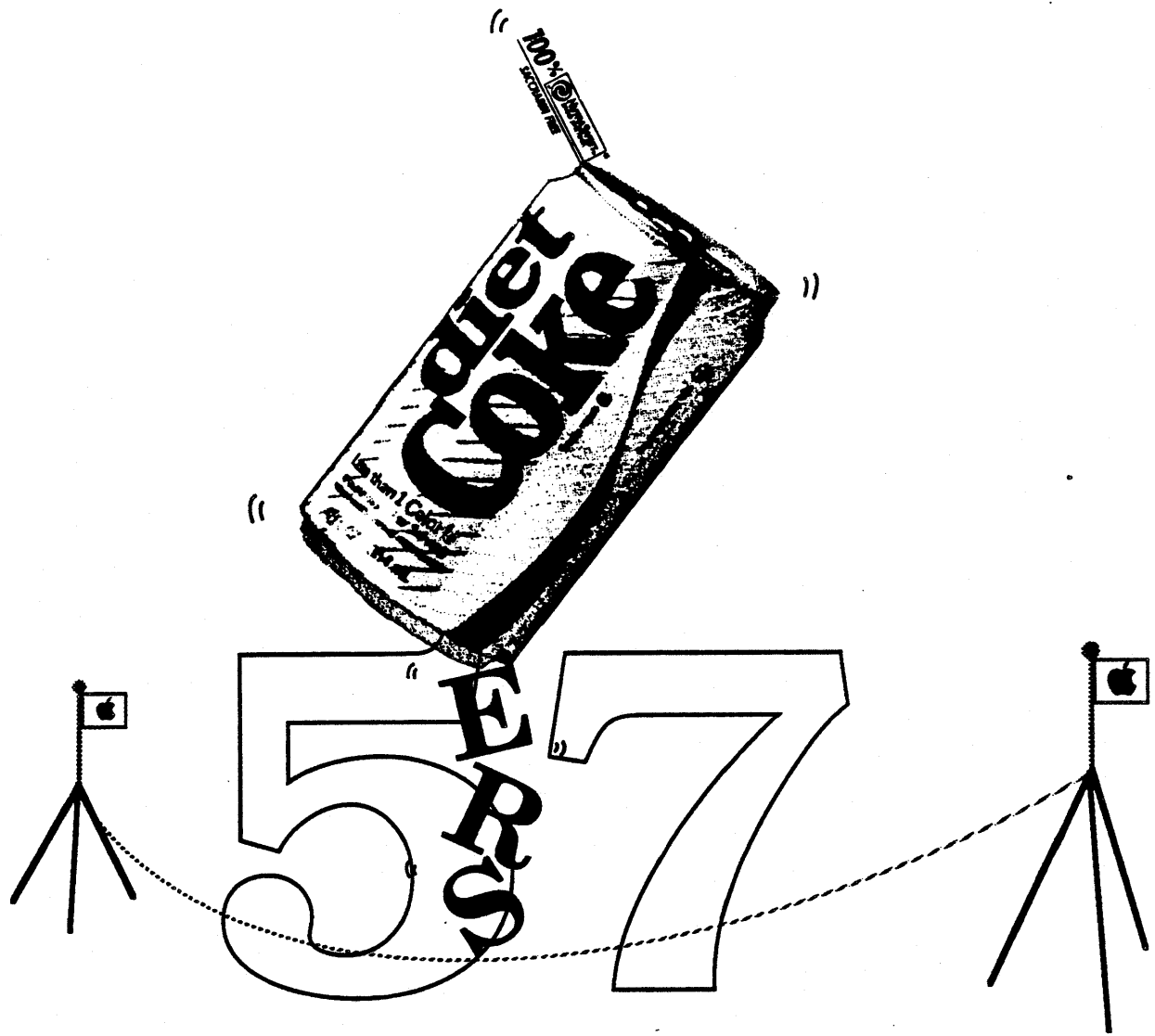
### On-Line Help

This important aspect of the ease-of-use of Macintosh applications is also one of the most controversial ones due to the potential abuses it may induce from careless applications. NuFinder will come out with an implementation of On-Line Help which hopefully will be inspiring and enlightening for other applications.

Providing On-Line Help integrated in the Toolbox is an issue which has not been specifically addressed yet, and it is not clear whether there are enough resources available today to provide a high quality tool within the Big Bang time frame. The critical aspect is to provide large flexibility to the user and a lot of guidance to the developer, and this is a complementary effort to providing support for just one Apple application, even if it is NuFinder, the spearhead of our user interface concepts.



57



## Feature Description ERS

Alan Lee  
Nick Kledzik

Version 0.5

November 22, 1988

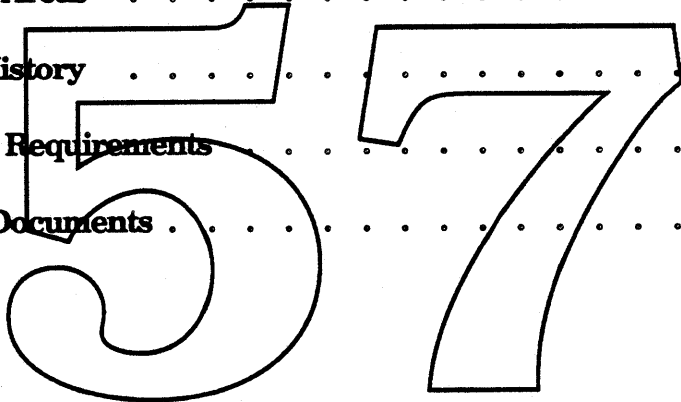
Apple Confidential

57



# Table of Contents

|      |                                           |    |
|------|-------------------------------------------|----|
| 1.0  | About This Document . . . . .             | 1  |
| 2.0  | About Diet Coke . . . . .                 | 1  |
| 3.0  | Fundamental User Model Concepts . . . . . | 2  |
| 4.0  | User Scenarios . . . . .                  | 3  |
| 5.0  | Diet Coke Limitations . . . . .           | 5  |
| 6.0  | System Requirements . . . . .             | 6  |
| 7.0  | Problem Areas . . . . .                   | 6  |
| 8.0  | Design History . . . . .                  | 7  |
| 9.0  | Software Requirements . . . . .           | 13 |
| 10.0 | Related Documents . . . . .               | 14 |



57



## 1.0 About This Document

This document describes the features of Diet Coke. It gives a brief description of what Diet Coke is, and what it is not. The document also defines the targeted user system and describes the software requirements.

## 2.0 About Diet Coke

*Diet Coke* is a project to provide a facility for automated data exchange between applications/documents in the Macintosh environment. The project originally grew out of Arn Schaeffer's Zirconium Data Exchange (ZDE) proposal. It is also known outside of Apple R&D as Inter-Application Communication (IAC). Currently, the project team members are Alan Lee & Nick Kledzik (Macintosh Toolbox), Tom Erickson & Scott Jenson (Human Interface), and Steve Goldberg (Product Management).

Diet Coke takes a high-level approach to inter-application communications. The facility is not intended to be used as an IPC mechanism, but rather it is a "top-down" approach to data exchange - the high-level application services are defined first without regard to the lower levels. In fact, the lower levels are purposefully hidden from application developers and might not ever become available for their direct use.

The Diet Coke facility is meant as a "first step" along the path towards a "full-featured" data exchange facility, and as such, it is limited in certain areas. For instance, there is no support for data exchange between applications on a network, nor is there support for "chain-reaction" data exchange through a string of documents. Diet Coke does, however, provide a very useful subset of functionality for users, and sets up a framework which allows enhancements to be added in the future.



### 3.0 Fundamental User Model Concepts

#### Data Sharing

The Diet Coke facility is based on the notion that different documents (possibly owned by different applications) can share a common part. For example, a certain picture can appear in several different documents at the same time. Today, this can be achieved by copying the picture and pasting it into the other documents. With Diet Coke, however, the picture can be converted into a shared object, so that changing the shared object changes the picture in all the other documents as well.

#### The Publication Metaphor

A shared document part is stored in a **publication**. A publication is a file which contains the latest edition of the shared part. ~~Publications appear in the Finder just like other files – they have an icon and a name, and they can be moved around and placed in different folders. Publications are similar to documents in that they can be opened from within an application, edited, and saved.~~

The user can include the data from a publication in any number of other documents by setting up subscriptions to the publication from those documents. The area of a document which contains an included publication is said to be a **subscriber** to that publication. A subscriber always has a copy of the publication data, but not necessarily the latest edition. If a publication is revised, the subscribers get the new edition at their earliest convenience, or when explicitly directed to do so by the user. Fig. 3-1 illustrates how data flows from a publication to its subscribers.

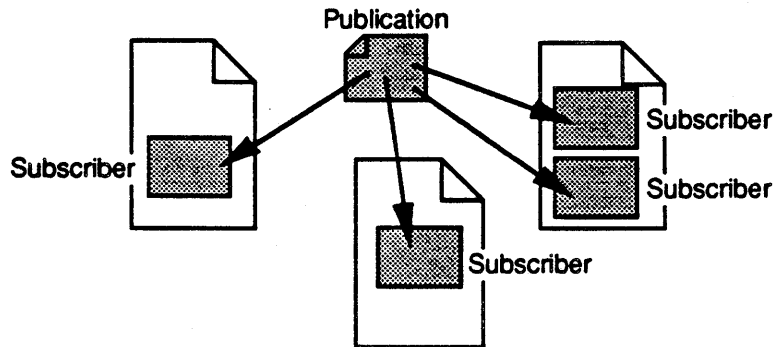


Fig. 3-1: A Publication and Three Documents with Subscribers



The user creates a new edition of a publication by opening the publication, changing the data which made up the previous edition, and then saving the changes as the new edition. The user can open a publication in three different ways: from the Finder by double-clicking on its icon, from within the creating application through the Open menu item, or by selecting a subscriber and choosing the Revise Publication menu item. In the last case, the system automatically finds the selected subscriber's publication and opens it, launching the publication's application if necessary.

Once a publication is opened, the user can make changes to the data in the publication window. All of the application's tools are available to edit the data. The user may also Cut, Copy, and Paste data into or out of the window.

The changes in an open publication are not made available to the subscribers until the user saves the publication – saving the publication declares the new edition complete. ~~Once an edition is saved, it becomes available to the subscribers.~~ Subscribers in opened documents are notified of the new edition so that they can get the new publication data (usually in a matter of seconds). Subscribers in closed documents are ~~not notified of the new edition until the next time their document is opened.~~

#### 4.0 User Scenarios

A simple example of Diet Coke is the scenario where you want a bar graph, which is created in a document with a charting application, to also appear in a report created with a word processing application. In the current Macintosh environment, you would Copy the bar graph from the charting document and Paste it into the report. However, every time you changed the bar graph, you would have to go to the trouble of re-Copying the graph and re-Pasting it into the report – if you remembered to do it at all.

To automate this process, you could use the Diet Coke facilities of the charting application to create the bar graph in a publication, and subscribe to that publication from the word processing document. Then whenever you revised the graph in the publication, the changes to the graph would also be automatically reflected in the report.





The next example involves the FlyByNite Corporation, a fast-paced, hot, new company. In fact, it is so hot that it frequently has to pack up everything and get out of town – often just steps ahead of the local police. Upon settling in a new location, the company usually decides to change its name and logo. Knowing that they work in a third-wave company where "the only constant is change", the employees have become heavy Diet Coke users. They have a Diet Coke publication which contains the company name and another which contains the company logo. Every official company document subscribes to one or both of these publications, whether it is a word processing, spreadsheet, drawing, or page layout document. Then, whenever the company changes its identity, the employees just publish new editions of those two publications and all of their documents are updated with the latest name and logo.

In the next example, suppose you are creating a newsletter using a page layout application and a word processor. You use the word processor to create the text in a Diet Coke publication and subscribe to the result in the page layout document. Now suppose that you are looking at the text in the page layout document and you decide that you want to modify the text. You select the subscriber (the text) and choose the Revise Publication menu item. This causes the publication to open (and the word processor to launch if it isn't already running). You make the changes to the text in the publication window and save them as a new edition. The new edition immediately flows to the page layout document replacing the previous edition.

Another example involves the sharing of some data with several different parts of the same document. Suppose you are updating some presentation slides that you created with a drawing application. The date appears in the bottom right corner of each slide. Anticipating future changes, you cleverly used Diet Coke to put the date in a publication and subscribed to that publication in each of the slides. Now, all you have to do to update the date in all of the slides is go to the publication and publish a new edition which has the new date.

Diet Coke can also be used to display "real-time" data. A stock quote application connected to a modem could periodically get certain current stock prices and display them in a publication. If the publication is automatically saved (either periodically, or every time a price changes), then new editions are automatically generated. A subscribing document (e.g. a spreadsheet or graph) could then receive up-to-date stock quotes, process the data, and display the result (e.g. a spreadsheet could multiply the current price times the number of shares owned).



## 5.0 Diet Coke Limitations

### No support for transaction processing

Note that the Diet Coke model, as illustrated in the examples above, is different from a transaction model. In a transaction model each transaction (i.e. new edition) is important; it is crucial that the publication knows whether an edition has been successfully received by all of the subscribers or not. In the refresh (Diet Coke) model it is not critical that a publication knows, since the subscribers will eventually be refreshed by some later edition. Diet Coke does not support the transaction model and is not intended to be used in such a manner.

### Not suited for fast real-time data exchange

Diet Coke is not designed to be used in time-critical applications, or applications which require new editions to be published at a "fast" rate.

### Not a general-purpose IPC

Diet Coke is not intended to be a general purpose IPC mechanism. Instead, it is designed to be an application-level, user controlled mechanism for automating repeated copy and pastes.

### No support for networked data exchange (yet)

Diet Coke does not implement networked data exchange to support collaborative work. However, its architecture has been purposefully shaped so that this enhancement can be added in the future. It is easy to imagine that a publication file could be placed on a networked file system such as AppleShare or TOPS and how documents on different machines could subscribe to that publication.

### No support for nested publications

Diet Coke does not allow publications to be nested. Another way of saying this is that a publication may not subscribe to another publication. This means that the user can not create chain-reaction data exchange through a string of publications. Although this seems like it would be a useful feature, the technical difficulties in implementing it are beyond the scope of Diet Coke.



## 6.0 System Requirements

The first Diet Coke implementation is targeted for the single-user Macintosh system running on a machine equipped with:

- MultiFinder
- a hard disk
- at least 2MB of memory.

{ When we look more closely at exactly how to implement Diet Coke, we may be able to relax some or all of the requirements: Unifinder vs. Multifinder, Floppy Disks vs. a hard disk, 1MB vs. 2MB }

## 7.0 Problem Areas

### Data Exchange Formats

Often times, a user copies some data from one application and pastes it into another application. Sometimes the pasted data doesn't look quite right – it needs to be tweaked a bit. This happens when the data exchange format standards (TEXT and PICT) aren't rich enough to capture all of the nuances of the data (e.g. ruler or layout information may be lost during the exchange). So the user has to manually adjust the data after pasting it into the second application.

This inadequacy in the current data exchange formats became more apparent as users began to use MultiFinder to copy and paste data between different applications. Unfortunately, this problem will only get worse when users begin to use Diet Coke. With Diet Coke, data is exchanged every time a new edition is published, and possibly updated in several different subscribing documents at once. For the times when information is lost due to a weakness in the data exchange format, the user will want to tweak each of the subscribers. However, he will not be able to do that since a subscriber can not be edited.

In general, a problem can occur whenever there is an imbalance in data format "richness". The data being exchanged always degrades to the lowest common denominator of format features, whether the format is an application's internal format, or the common exchange format being used. The problem occurs because applications generally deal with data which is richer than TEXT (including 'styl') and PICT, but those are currently the lowest common denominators supported by all applications.



The obvious solution to this problem is to raise the lowest common denominator by enriching the standard set of data formats to something beyond just TEXT, PICT, and styl. Unfortunately, this problem is large enough and important enough to be a separate project by itself – it falls outside of the scope of the Diet Coke project. Thus, unless a separate project solves this problem, Diet Coke will assume that TEXT, PICT, and styl comprise the standard set of data formats.

## 8.0 Design History

This section briefly describes the evolution of the Diet Coke design. The reader may skip this section without fear of missing any vital information. However, for the curious, it does give some insight as to why the user model is shaped the way it is.

Three other user models were first proposed before arriving at the current design (ERS version 0.5). The basic idea of ERS version 0.3 was to have Publisher sections in documents which sent data directly to Subscriber sections in other documents. ERS version 0.4 introduced the notion of a Publication as a central point in the data flow. Publisher sections in documents could write data to the Publication, Subscriber sections could read data from it, and Co-Publisher sections could both read and write. Finally, there was a short-lived ERS version 0.45 which tried to simplify the 0.4 model by replacing Co-Publishers with Subscribers that could "kibitz".

### Maintaining Persistent Links

Underlying each of the user model proposals is the requirement to maintain persistent links (i.e. links which remain intact between user sessions). In Diet Coke 0.5 terms, subscribers are "linked" to their associated publication (in 0.3 terms, Publishers were linked with Subscribers; in 0.4 terms, Publishers, Subscribers, and Co-Publishers were linked to Publications). Any Diet Coke design has to make sure that these "links" are not lost even under adverse conditions.

For example:

- when a Diet Coke document or a publication is renamed or moved within a volume.
- when a Diet Coke document or a publication is copied to another volume (either with or without its counterpart).



- when a Diet Coke document or a publication is backed up (either with or without its counterpart) and then restored (either with or without its counterpart).
- when a Diet Coke document or a publication is Finder-duplicated.
- when the user does Save, Save As, Save a Copy In, and Revert to Saved on Diet Coke documents.
- when subscribers are cut/copied and pasted (especially between different documents of different applications).
- when subscribers are deleted and then the delete is un-done.

Much of the difficulty in coming up with a suitable Diet Coke design stems from the requirement to maintain persistent links.

Publisher & Subscriber User Model (ERS version 0.3)

In the Publisher & Subscriber user model of ERS 0.3, a section of a document can be converted into a Publisher. Subscribers to a Publisher are created by copying and pasting that Publisher. Whenever the user changes the data in the Publisher section, the contents of that section are sent to all of the Subscribers.

The user can also "Follow Link" from any Publisher or Subscriber and cause the section at the other end of the link to appear (launching the application, opening the document, bringing the window to the front, and scrolling the section into view, as necessary). Another feature is the ability to open a document and "auto-launch" its associated documents, so that the web of documents are opened as a set.

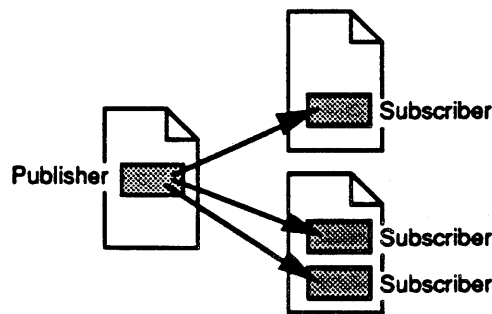


Fig. 8-1: Publisher & Subscriber User Model (ERS 0.3)



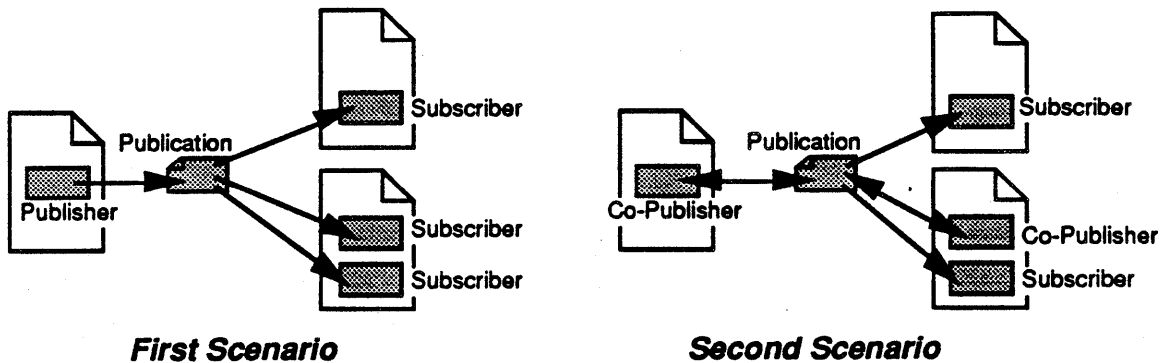
The major difficulties with this model are as follows:

- a) It requires a global naming scheme for uniquely identifying document sections on different volumes (and eventually, different machines).
- b) It requires a "database of links" which can tell you all of the sections that are linked to any given section.
- c) It is hard to keep the "database of links" up-to-date during the conditions described above in the section on "Maintaining Persistent Links", especially when a link spans across volumes, and one of the volumes is a removable floppy disk. For example, how do you take a document home, work on it, and then bring it back to work and merge all the links back together?
- d) There is no clean way to buffer data destined for Subscribers. What if the Publisher and Subscriber documents are never (cannot be) opened at the same time?
- e) The model does not extend easily into a networked environment. This would require that the "database of links" become network-wide. Also, the copy/paste method for creating links does not extend across the network because there is no way to paste into a window on another machine.
- f) Old applications can, and will inadvertently corrupt the "database of links".

#### Publisher, Subscriber & Co-Publisher User Model (ERS version 0.4)

The Publisher, Subscriber & Co-Publisher user model of ERS 0.4 introduces the notion of a Publication as a central point in the data flow where the latest edition is buffered. The user cannot edit the Publication directly, but rather, revises it through the Publisher or one of the Co-Publishers. There are two possible scenarios: In the first scenario, a Publication can have a Publisher and multiple Subscribers. In the second scenario, a Publication can have multiple Co-Publishers and multiple Subscribers.

In the first scenario, changes made to the Publisher section are also updated in the Publication; Subscribers get the current Publication edition at their leisure. In the second scenario, changes made to a Co-Publisher section are also updated in the Publication; Subscribers and the other Co-Publishers get the change at their leisure. To maintain consistency, a Co-Publisher must have the latest edition before revising the Publication, and only one Co-Publisher may revise a Publication at a time.



**Fig. 8-2: Publisher, Subscriber & Co-Publisher User Model (ERS 0.4)**

This user model solves a number of the problems inherent in the Publisher & Subscriber model:

- a) A "database of links" is no longer used. Instead, each section is responsible for remembering its associated publication. Publications don't have to know how many, or which sections are linked to it.
- b) Document sections are no longer named or tracked. Publications are the only thing the system keeps track of.
- c) There is a place to buffer data: in Publications. Furthermore, because Publications are user-visible, users are aware of, and understand why the buffering takes space on their disk.
- d) The model extends to a networked environment by simply allowing Publications to be placed on file servers.
- e) The probability of old applications corrupting the links is reduced. First, because there is no "database of links" to maintain. Second, because old applications would have to mess with Publications, but since they don't know about Publications, they probably won't touch them.



The major disadvantages of this model are as follows:

- a) The model is overly complex for the user. There are four different components that can be configured into two different scenarios.
- b) In the first scenario, you can create multiple Publishers by duplicating a document which contains a Publisher. This breaks the first scenario user model.
- c) Publications can become orphaned if all of its Publishers and Co-Publishers are deleted. If this happens then there is no way to ever revise that Publication again.
- d) The user can change a Publication by publishing a new edition from a Publisher section in a document and then not save that document. The result is that the Publisher section and Publication become "out-of-synch"; the Publisher section contains an older edition than the Publication. This also breaks the user model.
- e) Publications cannot be opened to see what's in them (often, a user wants to see a publication before subscribing to it). At best, a "viewer" application could be written which could "subscribe" to any Publication in order to display its contents.
- f) "Follow Link" and "Auto-Launching a set of documents" can no longer be supported because there is no longer a "database of links".

#### Publisher & Kibitzing Subscriber User Model (ERS version 0.45)

The Publisher & Kibitzing Subscriber user model of ERS 0.45 attempts to simplify the previous model by merging the two scenarios into one. It takes the first scenario and additionally gives Subscribers the ability to "kibitz" (i.e. they can also publish new editions if the Publisher has given them permission). This basically replaces the functionality provided by Co-Publishers, and so the second scenario is no longer needed.

While this simplifies things for the user somewhat, it does not solve the bulk of the problems present in the 0.4 user model.





Publication / Subscriber User Model (ERS version 0.5)

The user model of ERS 0.5 is a radical simplification of ERS 0.4. The only objects that the user has to know about are Publications and Subscribers. Publications can now be opened, like documents, and edited. This moves the role of where to edit the data to the Publication, eliminating the need for Publishers, Co-Publishers, and kibitzing Subscribers.

This simplified user model solves a number of the problems inherent in the ERS 0.4 and 0.45 models:

- a) The model is much simpler for users and developers.
- b) There is no duplicate Publisher problem, because there are no Publishers!
- c) Since a Publication is now the source of the data, it cannot become orphaned as it could in the previous models when its Publisher or all of its Co-Publishers were deleted.
- d) In order to "publish" a new edition, the user saves the Publication. This eliminates the "change the publication without saving the publishing document" problem.
- e) Users can "look" at any publication by opening it. This is part of the model, not an add-on.
- f) The user can open a Subscriber's Publication by selecting the Subscriber and issuing the Revise Publication command. The system will launch the Publication's application and open the Publication, if necessary. While this is by no means a generalized navigational "Follow Link", it does provide that style of functionality for the cases which commonly arise in a data sharing environment.

However, there is no such thing as a free lunch. The difficulties with this model are as follows:

- a) There is currently no need to distinguish "copying" as being either "cloning" or "backing up". However, with Diet Coke, users will want to sometimes "clone" a set of inter-linked documents and publications and have the links clone also. Other times users will want to "back up" a set of inter-linked documents and publications, where the links are not cloned – the links refer back to the originals. A new Finder verb is needed to



differentiate between these two operations. Note that this problem really exists in *all* of the other user model proposals as well.

- b) Applications which rely heavily on "derived data" for their document models, have to do a little more work. For example, in a spreadsheet you can no longer convert a cell "Total Sales" into a Publisher. What you must do instead is create a spreadsheet publication which consists of a cell with a formula that references the cell "Total Sales" in the original spreadsheet. Of course, the application can simplify this for the user by providing a command which performs the entire sequence of actions in one fell swoop.

## 9.0 Software Requirements

### Toolbox

- Implement the functionality behind the application programmer interface (as defined in the document "Diet Coke Programmer Interface ERS").
- Invent a way for multiple subscribers to be represented in the clipboard, along with normal data.
- Display the Publication mini-icon in the window title of Publication windows.

### MultiFinder

- Add new events for telling applications to open documents and for notifying them of new publication editions.

### Finder

- Maintain a Publication List which keeps track of where all the publications are located on a volume (similar to the Application List in the Desktop file).
- Warn the user when throwing away publications.
- Standard File displays the Publication mini-icon in the scrolling list.
- Add a verb to allow users to clone a set of documents and publications.
- It would be nice to have a new SFGetFile which returns cNodeID or DirID/Name instead of Working Directory. The old SFGetFile should then be converted to filter out Publications so that old applications won't see them.

### File System

- Add a publication bit in the file information to identify a file as a publication.
- It would be nice to have two half-copy calls – one which converts a file into a bitstream, and the other which converts the bitstream into a file.
- It would be nice to have calls which refer to files by cNodeID.



## Human Interface

- Develop Human Interface Guidelines for Diet Coke application developers.

## AppleShare

- Future extensions to Diet Coke will allow Publications to be placed on an AppleShare volume and shared over the network. We will need to be able to (somehow) keep a Publication List for these volumes as well. Subscribing machines will also (somehow) need to know when the Publication has a new edition.

## Applications

- New applications must be explicitly written to be Diet Coke-aware.
- Alter existing applications to become Diet Coke-aware (a non-trivial change).
- Applications which can copy Publications (i.e. files) must be altered to update the Publication List maintained by the Finder (by using the two half-copy calls described above?).

## 10.0 Related Documents

### Diet Coke Human Interface ERS

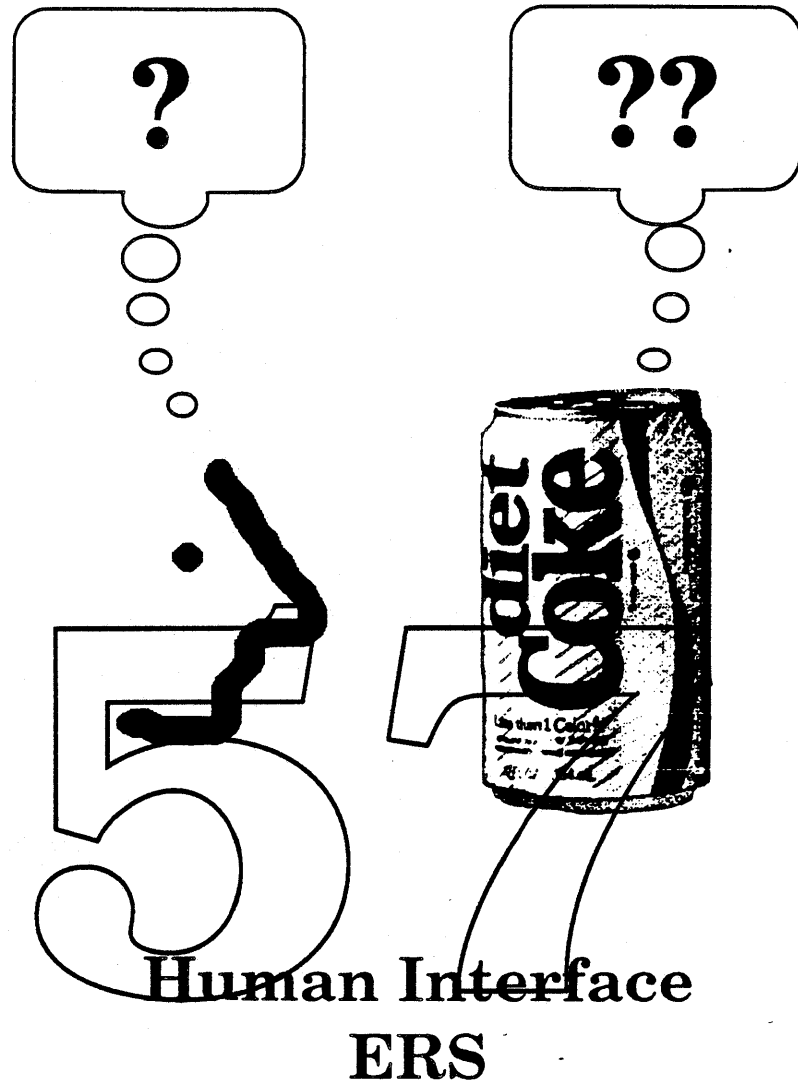
This document defines the User Model (the way that the user conceptualizes Diet Coke) and the User Interface (the way that those concepts are presented to the user via concrete mechanisms such as menus, buttons, dialogs, and icons).

### Diet Coke Programmer Interface ERS

This document defines the Programmer Model (the way that the application programmer conceptualizes Diet Coke) and the Toolbox Interface (the interface that the programmer uses to implement Diet Coke in an application).

### Diet Coke Software Guts IRS

This document describes the internal workings of the system software. It defines the schemes and methods used to implement the Toolbox Interface.



Mac Toolbox:

Alan Lee  
Nick Kledzik

Human Interface Group:

Tom Erickson  
Scott Jenson

Version 0.5

November 17, 1988

🍏 Apple Confidential

57



## Table of Contents

|            |                                                             |           |
|------------|-------------------------------------------------------------|-----------|
| <b>1.0</b> | <b>About This Document</b>                                  | <b>1</b>  |
| <b>2.0</b> | <b>Introduction to the User Model</b>                       | <b>1</b>  |
| 2.1        | The Publication Metaphor                                    | 1         |
| 2.2        | Notes on the Metaphor                                       | 4         |
| <b>3.0</b> | <b>Introduction to the User Interface</b>                   | <b>6</b>  |
| <b>4.0</b> | <b>Publications and Subscribers</b>                         | <b>8</b>  |
| 4.1        | Creating Publications                                       | 8         |
| 4.2        | Creating Subscribers                                        | 10        |
| 4.3        | Showing and Hiding Subscriber Boundaries                    | 11        |
| 4.4        | Selecting Subscribers                                       | 11        |
| 4.5        | Revising a Publication                                      | 12        |
| 4.6        | Updating Subscribers with the Latest Edition                | 15        |
| 4.7        | Subscription Info                                           | 16        |
| 4.8        | Cut, Copy, & Paste of Subscribers                           | 17        |
| 4.9        | Removing Subscribers                                        | 18        |
| 4.10       | Undo                                                        | 18        |
| <b>5.0</b> | <b>Diet Coke Documents</b>                                  | <b>19</b> |
| 5.1        | Opening a Document                                          | 19        |
| 5.2        | Closing a Document                                          | 20        |
| 5.3        | Saving a Document                                           | 21        |
| 5.4        | Reverting a Document to Saved                               | 21        |
| <b>6.0</b> | <b>Publications &amp; Diet Coke Documents in the Finder</b> | <b>22</b> |
| 6.1        | Appearance of Publications                                  | 22        |
| 6.2        | Moving Publications & Documents                             | 22        |
| 6.3        | Copying & Duplicating Publications & Documents              | 22        |
| 6.4        | Backing Up / Restoring Publications & Documents             | 25        |
| 6.5        | Renaming Publications & Documents                           | 25        |
| 6.6        | Deleting Publications & Documents                           | 26        |
| <b>7.0</b> | <b>Related Documents</b>                                    | <b>27</b> |

57



## 1.0 About This Document

This document describes the Diet Coke human interface. It defines the User Model (the way that the user conceptualizes Diet Coke) and the User Interface (the way that those concepts are presented to the user via concrete mechanisms such as menus, buttons, dialogs, and icons).

This document assumes that the reader is already familiar with the "Diet Coke Feature Description ERS" document.

## 2.0 Introduction to the User Model

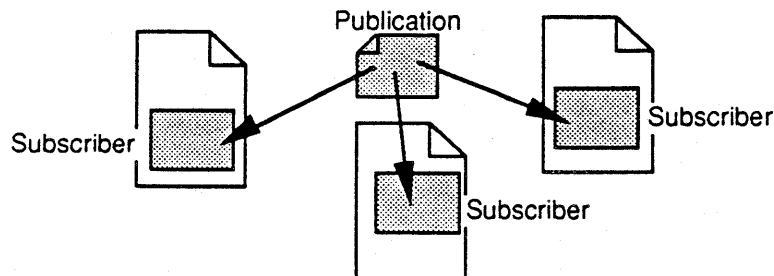
### 2.1 The Publication Metaphor

#### Publications and Subscribers

Diet Coke introduces two new concepts: Publications and Subscribers. A **publication** is a file which contains data that can be shared with several documents. The data in the file represents the "latest edition" of the publication. Publications appear in the Finder just like other files – they have an icon and a name, and they can be moved around and placed in different folders. Publications are similar to documents in that they can be opened from within an application, edited, and saved.

A **subscriber** is an area of a document which contains a copy of the data from some publication and which is periodically updated with that publication's latest edition. Subscribers are said to have a "subscription" to the publication.

Fig. 2.1-1 depicts a publication and three documents with subscribers to that publication. The arrows represent the data flow of new editions from the publication to the subscribing documents.



**Fig 2.1-1: A Publication and three Documents with Subscribers**





## Creating Publications

There are two anticipated scenarios of how users will want to create new publications. In the first scenario, the user knows ahead of time that he wants to have some shared data and so he creates a new publication and fills it with the data to be shared. In the second scenario, the user does not know ahead of time that he wants to share data; instead, the user sees some data in an existing document and then decides that he would like to share it with other documents.

In the first scenario, the user opens an application and issues the New Publication command to create a new publication of that application type. A new (blank) publication window appears and the user fills in the window with the data to be shared. The user Saves the publication thereby creating the publication's first edition.

In the second scenario, the user sees some data in an existing document that he would like to share and so he Copies the data into the clipboard. Next, the user issues the New Publication command and creates a new publication as above. The new (blank) publication window appears and the user Pastes the data from the clipboard into the window. The user Saves the publication to create the publication's first edition. Often times, the user will also want to subscribe to the new publication at the area of the document which was used to create the publication. To do so, the user issues the Subscribe command on the area of the document (still) selected.

The second scenario requires a lot of user actions. As a shortcut, the user may streamline the process by selecting the document data to share, and issuing the Publish & Subscribe command. This command creates a new publication, fills the publication with the selected document data, saves the data in the publication as the first edition, and then converts the original selection into a subscriber.

## Subscribing to Publications from Other Documents

The user can make any document subscribe to a publication. To subscribe, the user marks a point in the document with the mouse and issues the Subscribe command. This lets the user choose a publication, and specify whether or not the subscriber should automatically get new editions. The Subscribe command creates a new subscriber area in the document and fills it with the latest edition of the chosen publication. Note that a document may subscribe to several publications, and may even subscribe to the same publication multiple times.



Once a subscriber is created, the user is not allowed to edit the data inside that area. The rationale is that a subscriber contains an edition which is an entity controlled by the publication. Thus, if the user wants to make a new edition, he must do it at the publication.

### Revising Publications

To create a new publication edition, the user opens the publication, revises the data which made up the previous edition, and then saves the changes as the new edition. The user may open the publication by double clicking on it from the Finder, or by choosing the Open command from within an application. The user may also open a publication from a subscribing document by selecting one of the subscribers and issuing the Revise Publication command; this command opens the publication associated with the selected subscriber, launching the publication's application if necessary.

The user changes the data in the publication using the application's tools. When the user Saves the publication, the publication file is updated with the changes – a new edition is published. The new edition is then available to the subscribing documents.

Subscribers in opened documents get the new edition immediately; the application is notified of the new edition so that it knows to get the new publication data. Subscribers in closed documents are updated with the new edition the next time their document is opened.



## 2.2 Notes on the Metaphor

Diet Coke introduces some radical changes to the Macintosh user model. Documents, formerly independent entities, may now have "links" to publications. As a result, their "behavior" changes. For a document "linked" to a publication, its contents may change as a result of manipulations made to the publication. Furthermore, in such a document, the received data may not be directly edited, even though it appears identical to editable adjacent data. For a publication on the sending end of a link, changes to its contents may ultimately affect many documents.

In addition to these new behaviors which come with the concept of linking, there is another new kind of behavior that comes as a result of performance limitations: when data at the source of a link changes, we cannot guarantee that the corresponding data at the receiving end of the link will be immediately updated. Thus, even when users understand that changing data in one place will cause data in other places to also change, they also have to understand that the other changes may not take place immediately.

All this is new to Macintosh users, and, more importantly, this new behavior does not have a counterpart in the concrete world that the Macintosh Desktop tries to mirror. In the concrete world documents can't be linked: all data is editable; all changes affect only the document being manipulated; all change is instantaneous. So not only do Macintosh users have to learn new things, but they have to learn things which have no parallel in the concrete world of documents with which they're familiar.

Clearly, Diet Coke requires an addition to or extension of the basic Desktop metaphor. Though a number of metaphors were considered and rejected, an explanation of the reasons we rejected the initial metaphor – that of "links" – will further your understanding of the Diet Coke human interface.

The notion of representing Diet Coke's data sharing facilities as "links" between files was an obvious one, because the link metaphor is the one typically used by programmers. However, it has several disadvantages for end users. First, most end users don't know what links are: non-programmers at Apple tended to think of the AppleLink meaning of links; those outside of the Apple culture are more likely to think of physical links (e.g. links in a chain; a chain link fence; cuff links) than the abstract meaning of "link". Second, even if users can be counted on to understand that the abstract meaning of links is the appropriate one, the link metaphor doesn't carry much meaning. It suggests that the files are somehow connected to one another, but it implies nothing about data being



shared, or about data "flowing" from one file to another. Third, it's not clear how you would visually represent the link metaphor. One could show links in the Finder, although if there were a lot of links it would get ugly quickly. And if links were shown, then wouldn't users expect dragging a file with links to other files to pull them along too? There are many other problems associated with visually representing links. Yet, not providing a visual representation of the Diet Coke metaphor seems like a bad thing to do, too.

As a result of these, and other, problems, we began looking at metaphors which focused on the endpoints of the links – data being shared – rather than on the links themselves. We reasoned that users are ultimately interested in data; they don't care about links, except as they provide information about the nature and properties of the data that they see in their documents. Thus, we began exploring a number of real world instances of "links" involving data transmission (e.g. radio, television, telephone, publishing) and settled upon a publication metaphor. The publishing metaphor is appropriate because 1) it is safe to say that most of our users subscribe to at least some publications, and thus have a general model of publishing; 2) publishing embodies a number of concepts that users need to understand. To wit:

- a) Publications are expected to change over time (i.e. there are editions).
- b) A single publication is seen by many subscribers at a time.
- c) Subscribers don't change a publication; publishers (in this case, a user manipulating a publication) can change a publication – directionality is inherent in the metaphor.
- d) Time lags between the publishing and receipt of an edition are understandable.
- e) Publications have names, and are subscribed to by name.



### 3.0 Introduction to the User Interface

There are a number of commands for controlling different aspects of Diet Coke functionality; they are available in the Publications menu which appears in each Diet Coke application. The presence (or absence) of the menu lets the user know at a glance whether the application supports Diet Coke (or not). The menu is depicted below.

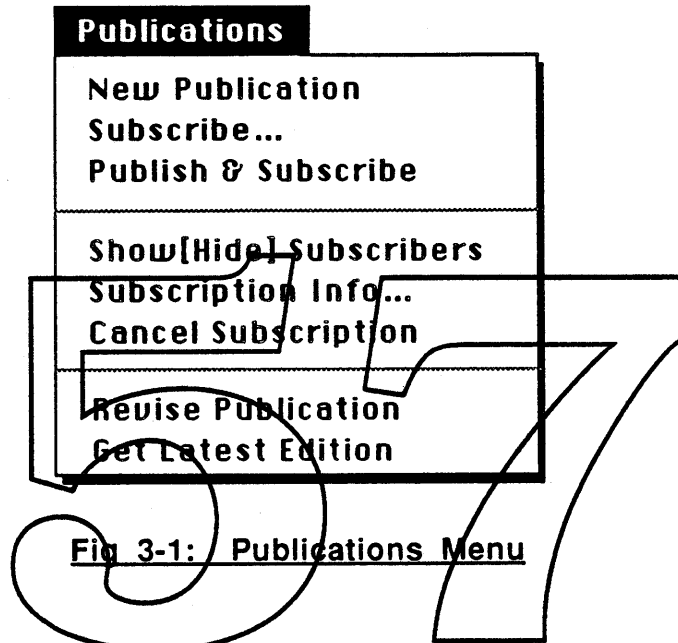


Fig 3-1: Publications Menu

The purpose of each menu item is briefly described below:

**New Publication:**

Creates a new, untitled, publication window.

**Subscribe:**

Creates a new subscriber to a publication at the current insertion point and includes the publication's current edition at that point.

**Publish & Subscribe:**

Creates a new publication using the selected data as the first edition, and converts the selected area into a subscriber to the new publication.

**Show[Hide] Subscribers:**

Toggles displaying (hiding) of the visual borders around subscribers throughout the document.

**Subscription Info:**

Displays a dialog which contains subscription information for the selected subscriber and allows the user to specify certain subscription options.

**Cancel Subscription:**

Reverts the selected subscriber(s) to normal – the data from the publication remains in the document, but the area(s) is no longer a subscriber(s). The area will not receive any new editions.

**Revise Publication:**

Opens the selected subscriber's publication. Launches the publication's application first, if necessary.

**Get Latest Edition:**

Updates the selected subscriber(s) with the latest edition of the publication(s). This is useful when automatic updates are turned off.

In addition, some of the standard File menu items operate on Publication windows in the same way that they operate on document windows:

**Open:**

Allows the user to open a publication. An open publication appears in its own window in the same way that an open document appears in its own window.

**Close:**

Closes the open publication window. If there are unsaved changes, then the application warns the user and asks if the new edition should be published before closing.

**Save:**

Saves the changes made in the publication window to the publication file; this constitutes the publishing of a new edition.

**Revert to Saved:**

Reverts the contents of the publication window to the last saved edition.

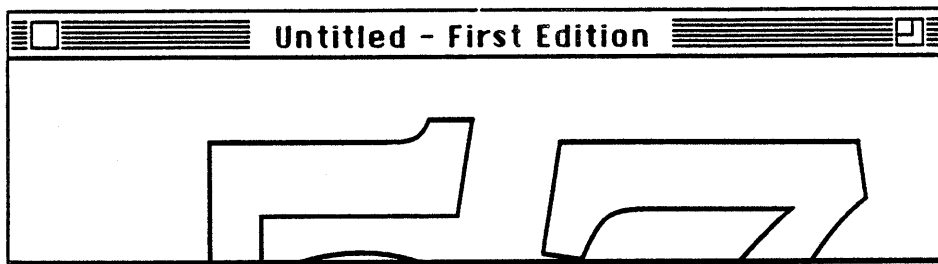
Note that Save As is not supported because it is uncertain whether the user wants subscribers to subscribe to the original publication or the new one.



## 4.0 Publications and Subscribers

### 4.1 Creating Publications

To share a piece of data with several documents, the user creates a new publication which contains the data to share. From within an application, the user chooses the New Publication command from the Publications menu. This creates a new, untitled, publication window of that application type (fig. 4.1-1). Note that a new publication file is not created until the first time the user Saves the contents of the window.

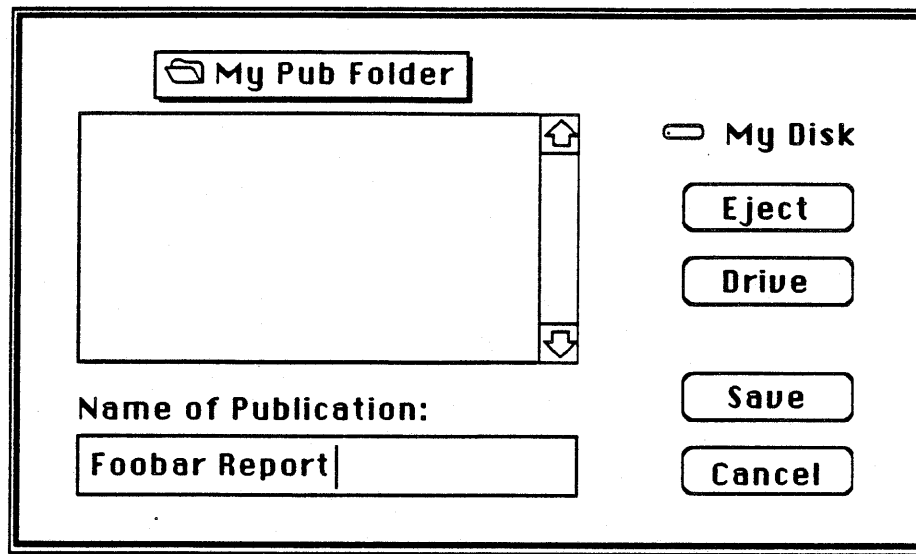


**Fig 4.1-1: New Publication Window**

*{ Should we put precede the window title with the "publication icon" if the window is a publication? This would help differentiate publication windows from document windows. One possible way to do this would be to add a "publication icon character" to Chicago font and add this character to the front of the window title (But what about international fonts?). }*

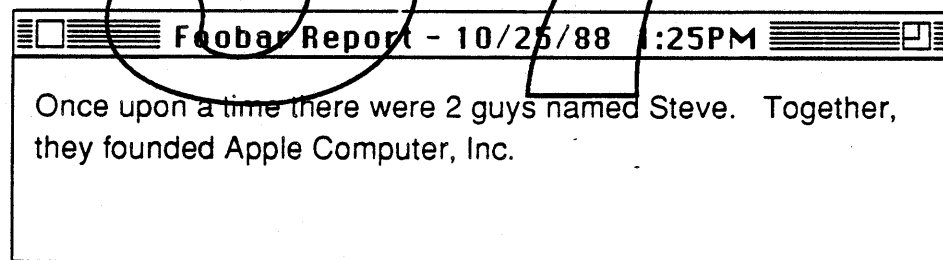
The user creates the first edition by entering data directly into the window, or by copying data from other documents or publications and pasting it into the window. When the user decides that the data in the window constitutes the first edition he saves the window using the Save command in the File menu. This causes a dialog to appear which lets the user name the publication and place it in any desired folder (fig. 4.1-2).

When a publication is saved, the data is stored in one or more well-known data formats (e.g. TEXT and PICT) so that any application knows how to interpret it. Other "not so well-known, but richer" data formats may also be stored along with the well-known ones so that those applications which understand them can make use of them. This is similar in philosophy to the way the clipboard works.



**Fig 4.1-2: Saving a New Publication**

After the publication is saved for the first time, the window title changes to the name of the publication along with the date and time of the first edition (fig. 4.1-3). Other documents can subscribe to the publication as soon as it is saved.



**Fig 4.1-3: Saved Publication Window**

As a shortcut, the user can use the Publish & Subscribe command to create a publication out of a document selection and convert the selection into a subscriber. The user selects the document data to be shared and then chooses the Publish & Subscribe command from the Publications menu. This command creates a new publication, fills the publication with the selected document data, and saves the data in the publication as the first edition. The user is presented with the dialog (fig. 4.1-2) to let him name and place the publication in a folder. Finally, it converts the original document selection into a subscriber to the new publication.

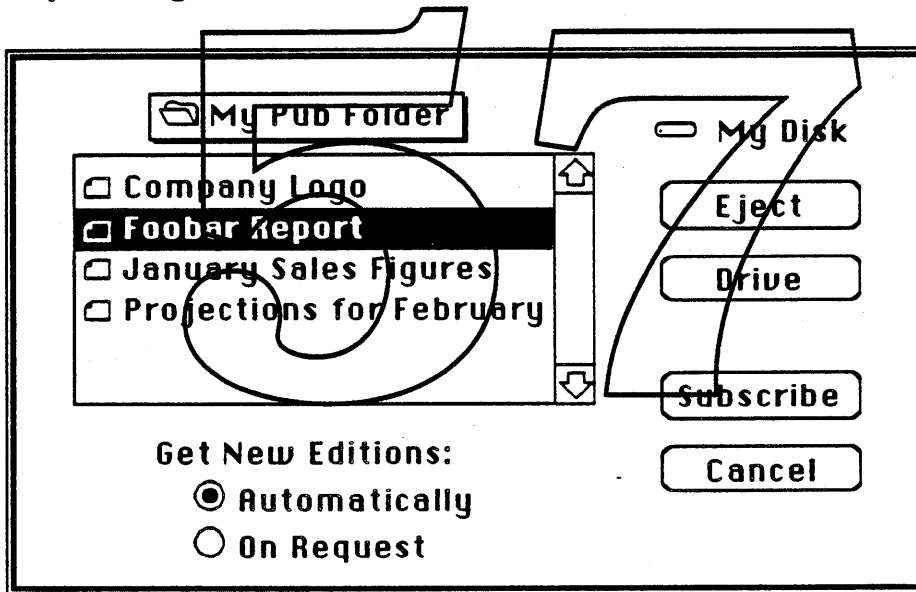




## 4.2 Creating Subscribers

Any (Diet Coke compatible) document can subscribe to a publication. In fact, a document can subscribe to any number of publications, and may even subscribe to the same publication multiple times.

To create a new subscriber, the user clicks the mouse in the document at the point where the subscriber is to be created, and then chooses the Subscribe command from the Publications menu. This brings up a dialog which lets the user choose the desired publication (fig. 4.2-1). By default, the last publication created, or the last publication subscribed to (whichever was done last), is pre-selected in the dialog. This (hopefully) reduces the number of actions the user must perform to subscribe, in the majority of cases. Of course, the user may choose a different publication by clicking on it with the mouse.



**Fig 4.2-1: Subscribe dialog**

When the user clicks on the Subscribe button (or equivalently, double clicks on a publication in the list), the latest edition of the chosen publication is placed in the document at the specified point. If the user selects a portion of the document instead of clicking a simple insertion point, then the publication edition replaces the data in the selection.

By default, an application automatically updates a subscriber whenever it is notified that the publication has been revised. The user has the option of creating the subscriber with automatic updates turned off. In this case, the user can

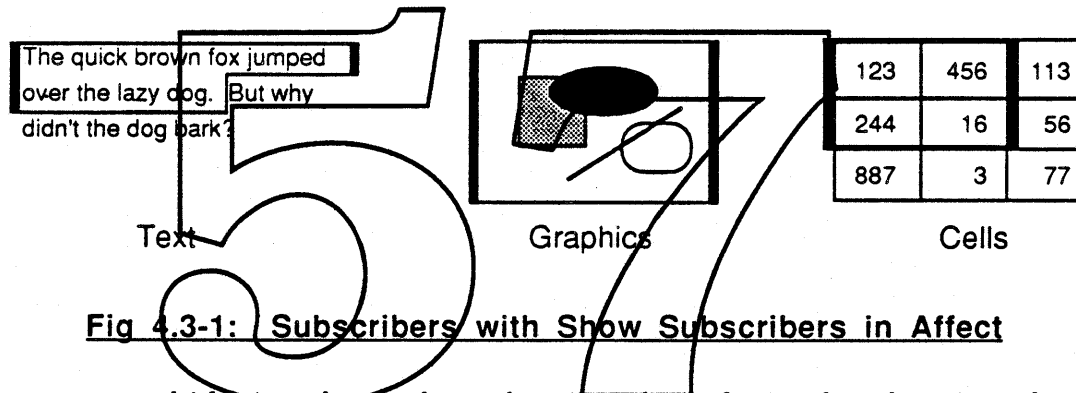


manually update the subscriber via the Get Latest Edition command in the Publications menu. The user can turn automatic updates back on from the Subscription Info dialog (see section 4.7: Subscription Info).

Another way of creating a subscriber is to copy and paste an existing subscriber. The duplicate subscriber subscribes to the same publication as the original, and inherits all of its subscription options. The same thing happens if a subscriber is indirectly duplicated by duplicating its document in the Finder.

### 4.3 Showing and Hiding Subscriber Boundaries

Subscribers appear in documents with a frame around the boundary. The boundary is both distinctive and consistent so that users can immediately recognize a Diet Coke subscriber regardless of what application is running.



**Fig 4.3-1: Subscribers with Show Subscribers in Affect**

The user may hide (or show) the subscriber boundaries by choosing the Hide Subscribers (or Show Subscribers) item in the Publications menu; the menu item toggles between Hide and Show. The default state is for subscriber boundaries to show. The rationale is that if new users just start to try things, they should be able to clearly see the results of their experimentation.

*{ Should there be different variations of the boundaries to distinguish between subscribers that receive new editions automatically vs. on request? }*

### 4.4 Selecting Subscribers

Many Diet Coke operations require that the user first specify a subscriber by selecting it. When the user clicks in, or drags into, any part of a subscriber, all of the data in the subscriber highlights – the user is not allowed to select only part of the data (fig. 4.4-1). The boundary also highlights whenever a subscriber is selected, even if Show Subscribers is not in affect.



By dragging or shift-clicking, the user may select more than one subscriber at a time, and may intersperse normal data in the selection. If Show Subscribers is in affect, the user may also select a subscriber by clicking directly on the boundary.

The quick brown fox jumped  
over the lazy dog. But why  
didn't the dog bark?

Text



Graphics

|     |     |     |
|-----|-----|-----|
| 123 | 456 | 113 |
| 244 | 16  | 56  |
| 887 | 3   | 77  |

Cells

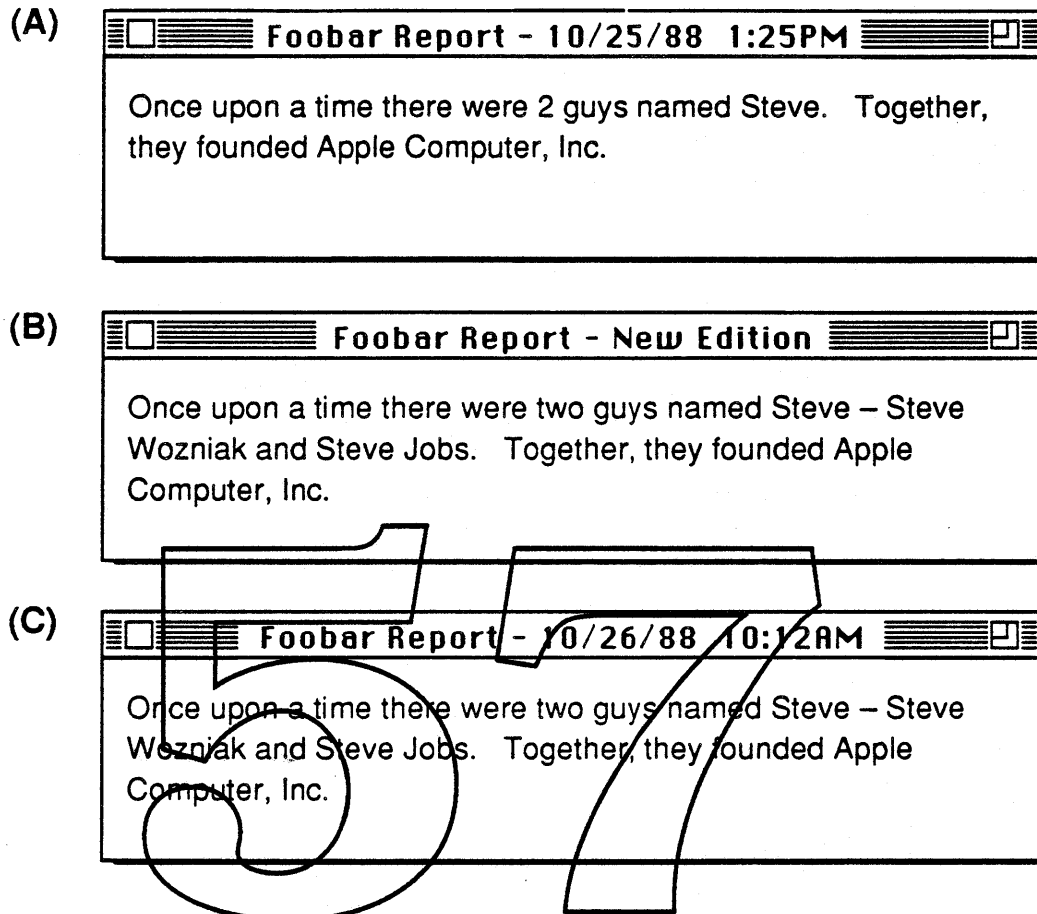
**Fig 4.4-1: Selected Subscribers**

### 4.5 Revising a Publication

To revise a publication, the user opens the publication, makes changes to it, and then saves the publication. There are several ways to open a publication. The user may open the publication from the Finder via the Open command or by double clicking on it. The user may open the publication from within an application via the Open command. Or, the user may select a subscriber to the publication (from some document) and choose the Revise Publication command from the Publications menu - this opens the associated publication, launching the publication's application if necessary, and brings the publication window to the front. As a shortcut, the user may double click on a *selected* subscriber to invoke the Revise Publication command.

The publication window title displays the name of the publication, and the date and time of the latest edition (fig. 4.5-1A). As soon as the user makes a change to the data in the publication window, the date and time in the window title changes to "New Edition" (fig. 4.5-1B). When the user saves the publication, the "New Edition" is replaced by the current date and time (fig. 4.5-1C).

After saving a publication, the user may start another new edition by just changing the data in the publication window. If this happens, the window title changes to reflect that a new edition is in progress (as in fig. 4.5-1B). The user may then publish the second new edition by saving the publication again.



**Fig 4.5-1: Revising a Publication**

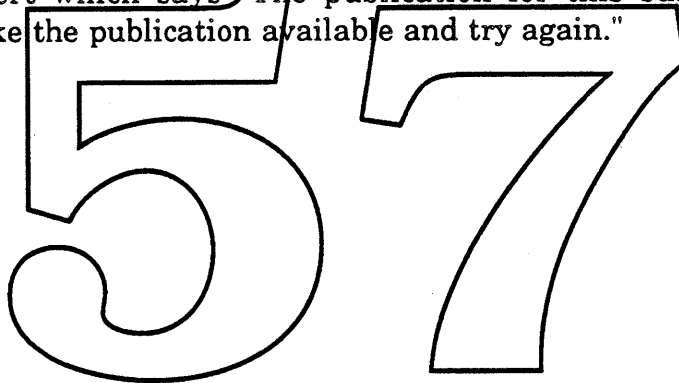
If the user tries to close the publication window after starting a new edition (as in fig. 4.5-1B), the application warns the user with an alert which says "Save new edition before closing?" along with the buttons SAVE, No, and Cancel. Clicking on SAVE executes the Save command and then closes; clicking on No throws away the changes made in the publication window and then closes (the publication file remains unchanged); clicking on Cancel stops the close – the window remains open. This is similar in spirit to the dialog which asks the user whether to save or not before closing a document.

The user is not allowed to create nested publications. That is, it is illegal, and thus impossible, for the user to choose the Subscribe command from within a publication window. This functionality can not be allowed until there is a standard compound data format which allows different data types to be mixed together in an arbitrary manner.



When the user chooses the Revise Publication command, there are a number of conditions which can occur. They are handled as follows:

- *The user selects more than one subscriber:*  
Open all of the associated publications, one at a time.
- *The publication for the selected subscriber is currently open:*  
If the publication is open on the same computer, then just bring the publication window to the front. Otherwise (this implies networking and so it might not be applicable for Diet Coke), display an alert which says "This publication is already in use. You may not open the publication until it becomes free."
- *The publication for the selected subscriber is not on-line:*  
Display an alert which says "The publication for this subscriber is not available. Make the publication available and try again."





#### 4.6 Updating Subscribers with the Latest Edition

When a publication is saved, the new edition becomes available to the subscribers. The subscribers get the new edition at their earliest convenience. Subscribers in opened documents get the new edition immediately; the application is notified of the new edition so that it knows to get the new publication data. Subscribers in closed documents are updated with the new edition the next time their document is opened.

Whenever a subscriber is updated, the subscriber area is briefly "highlighted" to give the user visual feedback. If the subscriber boundary is currently visible, then the highlighting consists of dimming the boundary, drawing the new edition, and then un-dimming the boundary. If the boundary is not currently visible, then the highlighting consists of making the boundary visible (but dimmed), drawing the new edition, and then making the boundary invisible.

If a subscriber is configured with automatic updates turned off, then it is not automatically updated whenever a new edition is published. Instead, it is updated whenever the user selects the subscriber and chooses the Get Latest Edition item from the Publications menu. The subscriber "highlights" in the same way as described above.

When the user chooses the Get Latest Edition command, there are a number of conditions which can occur. They are handled as follows:

- *The user selects more than one subscriber:*  
Get the latest edition for each subscriber.
- *The selection contains no subscribers:*  
Disable the Get Latest Edition menu item to prevent the user from operating on the selection.
- *The selected subscriber is already up-to-date:*  
"Highlight" the subscriber as feedback that the Get Latest Edition command was performed (even though the data does not change).
- *The selected subscriber receives updates automatically:*  
If, for some reason, the subscriber doesn't have the latest edition, get it.
- *The publication for the selected subscriber is not on-line:*  
Display an alert which says "The publication is not available. Make the publication available and try again."

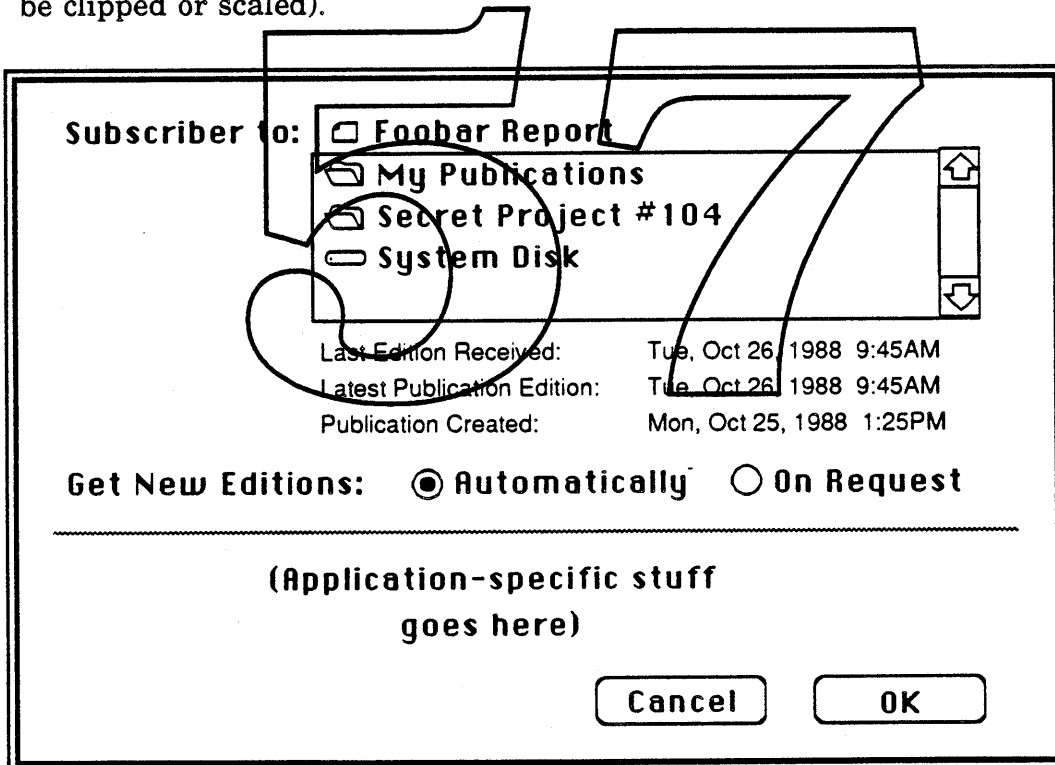


#### 4.7 Subscription Info

The user can select a subscriber and choose the Subscription Info command from the Publications menu to see information about that subscriber and configure certain subscription options.

A dialog appears which displays:

- a) the publication name.
- b) the last edition received by the subscriber (the date of the edition).
- c) the latest edition of the publication (the date the publication was last saved).
- d) the date the publication was first created.
- e) the controls which determine whether new editions are updated automatically or manually.
- f) application-specific attributes and controls (e.g. whether graphics should be clipped or scaled).



**Fig 4.7-1: Subscription Info dialog**



#### 4.8 Cut, Copy, & Paste of Subscribers

One way the user can move a subscriber within a document, or between documents, is by Cutting and Pasting. The subscriber subscribes to the same publication that it did before being moved. (Of course, another way of moving a subscriber is to drag it, as in a graphics application, for example).

The user can duplicate any subscriber within a document, or between documents, by Copying and Pasting (note that multiple Pastes after a Cut can also duplicate). The duplicated subscriber initially has the same edition as the original and subscribes to the same publication.

#### Cut, Copy, & Paste in a Publication Window

While creating a new edition, the user may cut, copy, and paste between the publication window and any other document or publication window. The only restriction is that it is illegal to paste a subscriber into a publication window since nested publications are not allowed. If the user tries to paste a subscriber into a publication window, then a dialog appears which informs the user that the action is not allowed. It then lets the user either paste just the data in the subscriber, or cancel the paste.

#### Copy & Paste of Subscriber Data Only

There are two ways that the user can copy and paste just the data in a subscriber, without creating a duplicate subscriber. With the first method, the user opens the associated publication, copies the data out of the publication window, and then pastes it. Using the second method, the user copies the subscriber, pastes it (creating a duplicate subscriber), and then cancels the new subscription (removing the duplicate subscriber).





## 4.9 Removing Subscribers

### Deleting Subscribers

The user can delete a *selected* subscriber by:

- a) typing the delete key
- b) replacing it with data entry
- c) replacing it with a paste
- d) replacing it by choosing Subscribe from the Publications menu
- e) choosing Cut from the Edit menu

### Cancelling Subscriptions

To cancel a subscription the user selects the subscriber and chooses the Cancel Subscription item from the Publications menu. This reverts the subscriber area to normal data. If Show Subscribers is in affect, the boundary disappears. The data that was in the subscriber remains in the document and becomes just like any other document data – the data is no longer associated with any publication.

## 4.10 Undo

In general, any operation which changes data should be undo-able via the Undo command in the Edit menu. For Diet Coke operations specifically, the following actions are undo-able:

- a) Subscribe
- b) deleting a subscriber
- c) Cancel Subscription
- d) Cut, Copy, & Paste

Some operations are not undo-able. This is because they either involve publishing a new edition (like saving a document, saving a publication is not undo-able) or they involve a dialog. The following can not be undone:

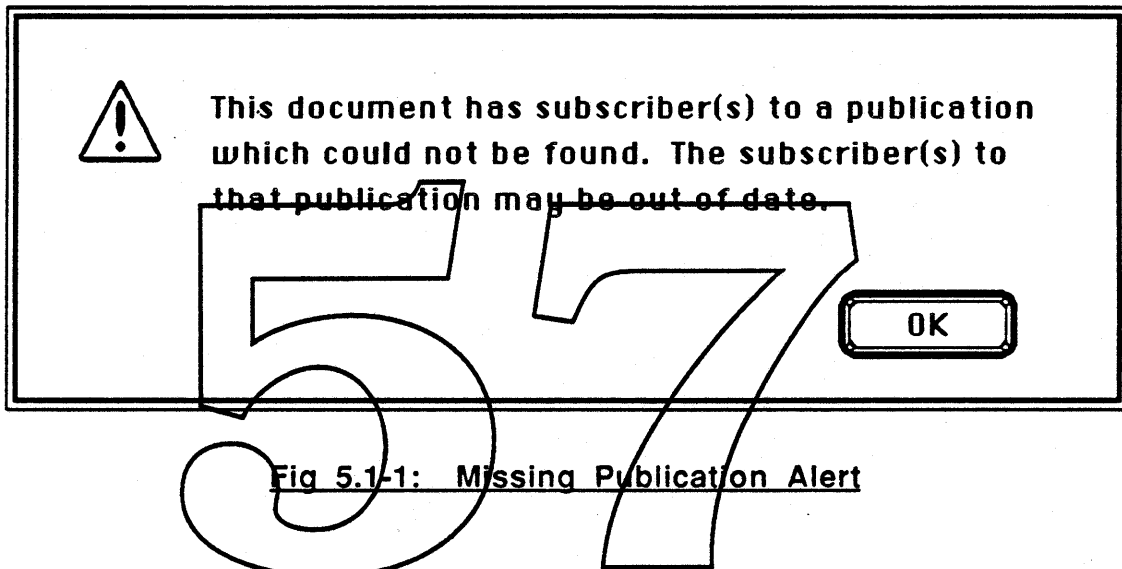
- a) New Publication (although you can close without ever saving the window)
- b) changing the configuration in the Subscription Info window
- c) Get Latest Edition



## 5.0 Diet Coke Documents

### 5.1 Opening a Document

When an application opens a Diet Coke document it looks for all of the publications that it subscribes to in all of the on-line volumes. If the application can not find a publication, then it warns the user that the subscriber(s) may not contain the latest edition (fig. 5.1-1).



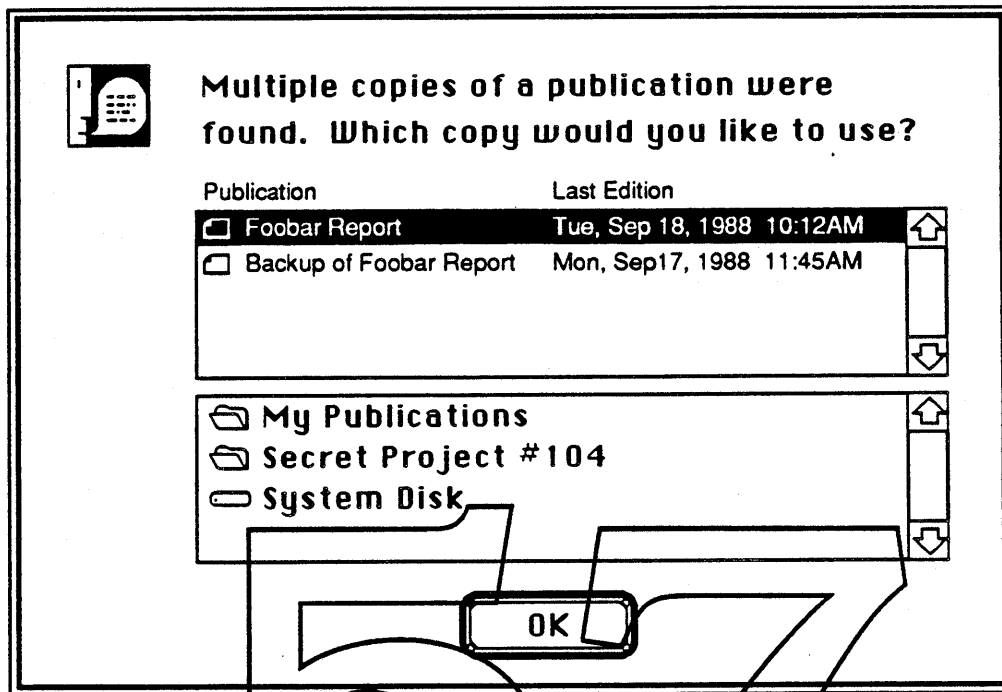
The user may place additional volumes on-line and the application may find previously missing publications on those volumes. If the user issues a Get Latest Edition command, the application can execute the command if the publication has come on-line. Conversely, the application won't be able to execute the command if the publication has not come on-line, or if it has gone off-line after the document was opened.

Note that if the user issues a Subscription Info command on a subscriber whose publication is off-line, then the dialog will display, but all information about the publication will be left blank.

If the application finds more than one copy of a publication (for example, this can happen if the user duplicates the publication via the Finder), then the application asks the user to choose the publication to use. The publications are presented sorted by modification date, with the newest publication displayed first. The chosen publication is used (for subscribers in all documents) until the next time the computer restarts – the user must then choose again. This discourages the



user from keeping multiple copies of a publication on-line.



**Fig 5-1-2: Multiple Publication Dialog**

When a document is opened, some of its subscribers may contain old editions (i.e. a new edition may have been published while the document was closed). If the subscriber updates automatically, the application is told to update the subscriber with the latest edition. The end-result is that the user sees an up-to-date version of the document very soon after it opens. If the subscriber updates manually (i.e. on request) then nothing happens; the subscriber has the old edition until it is replaced when the user does a Get Latest Edition.

## **5.2 Closing a Document**

Whenever the user closes a document, the application does its normal check to see if the document needs to be saved. A document needs to be saved if any changes have occurred since the last save, including any changes made to the document's subscribers (e.g. as the result of receiving the latest edition, or changing the settings in the subscription info dialog).



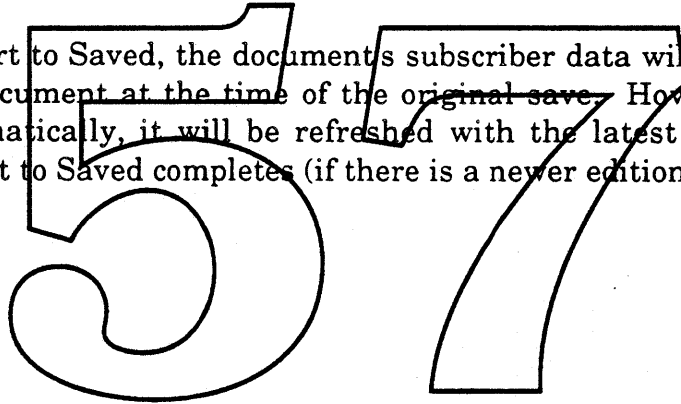
### 5.3 Saving a Document

Saving a document saves the data which currently appears in the document window. This includes the editions currently in the document's subscribers. The idea is that the user should be able to take a document to another machine without bringing along the associated publications and still see an edition in each subscriber.

### 5.4 Reverting a Document to Saved

Revert to Saved is functionally equivalent to closing the document (without saving), and then re-opening the document. These actions happen automatically; the application does not interact with the user during the Revert to Saved operation.

After the Revert to Saved, the document's subscriber data will be the editions that were in the document at the time of the original save. However, if a subscriber updates automatically, it will be refreshed with the latest edition immediately after the Revert to Saved completes (if there is a newer edition).





## 6.0 Publications & Diet Coke Documents in the Finder

### 6.1 Appearance of Publications

When the user creates a new publication it may be placed in any folder on any volume. The publication appears as a file in the Finder; it has a name and an icon. Publication icons have a special "look" to differentiate them from document and application icons. Each specific publication icon is filled in differently depending on which application created it (or last saved it?). Fig. 6.1-1 shows a Publication icon along with a Document icon.



Fig 6.1-1: Publication Icon vs. Document Icon in the Finder

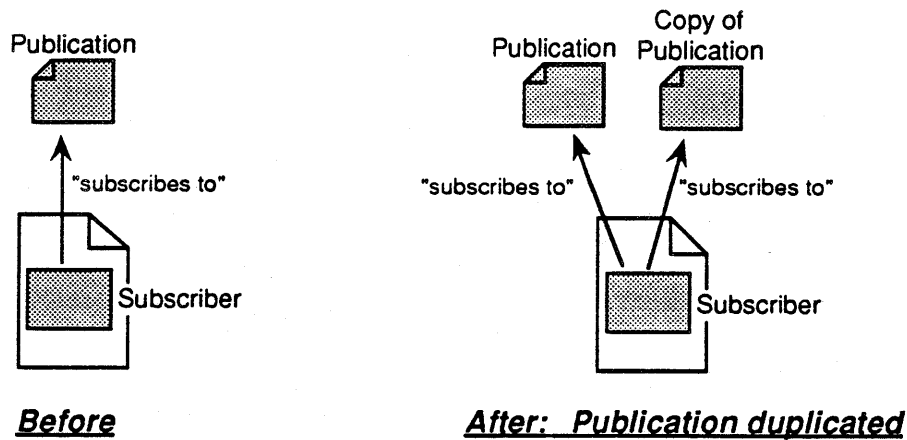
### 6.2 Moving Publications & Documents

The user can move (using the Finder) publications and Diet Coke documents anywhere within a volume without any adverse side effects. However, note that moving a publication with something other than the Finder may cause applications to not be able to find the moved publication the next time a subscribing document is opened.

### 6.3 Copying & Duplicating Publications & Documents

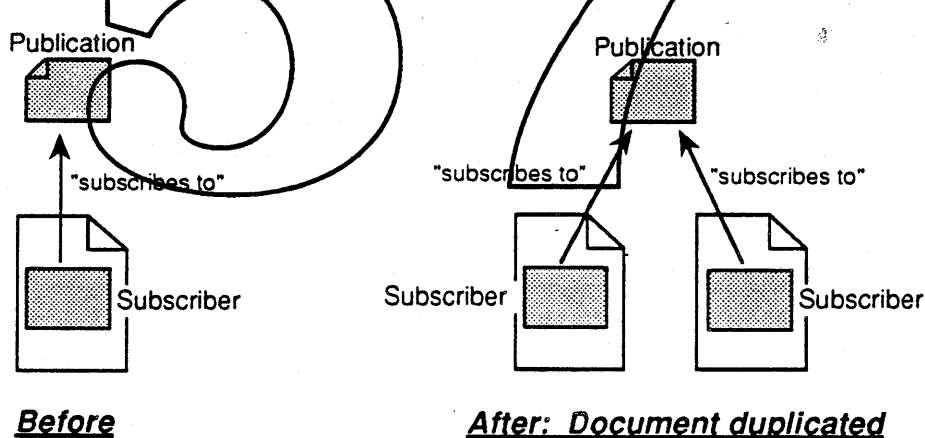
When the user duplicates a *publication* on the same volume, or copies a publication to another volume, the copy is treated as a backup of the publication, rather than as a different publication that initially looks like the original. The system considers the two publications to be interchangeable. Conceptually, a subscriber to the original publication also has a subscription to the copy! This is required in order to allow users to back up a publication without losing the associated subscriptions when the publication is restored. See fig. 6.3-1.

If both the original and copy are on-line when the user opens a subscribing document, then the application will find both publications and ask the user to choose one of them (see section 5.1: Opening a Document).



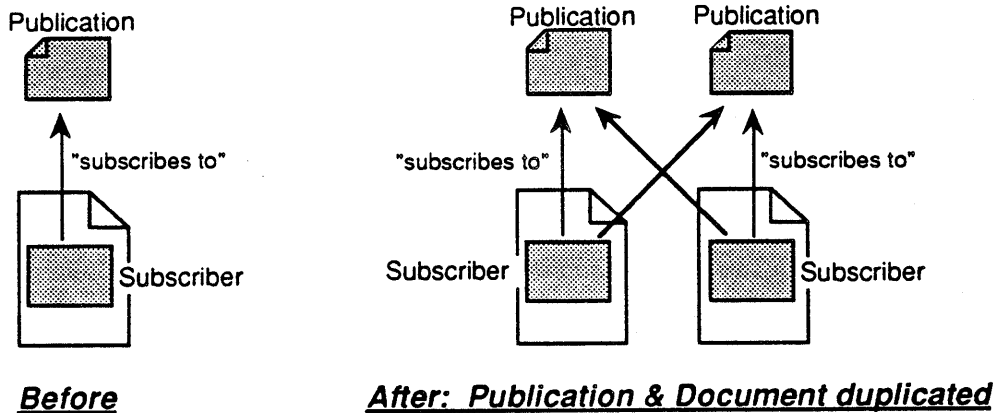
**Fig 6.3-1: Duplicating a Publication**

When the user duplicates a *document* on the same volume, or copies a document to another volume, the copy contains all of the subscribers that the original has. The subscribers in the copy subscribe to the same publications as the corresponding subscribers in the original. See fig 6.3-2.



**Fig 6.3-2: Duplicating a Document**

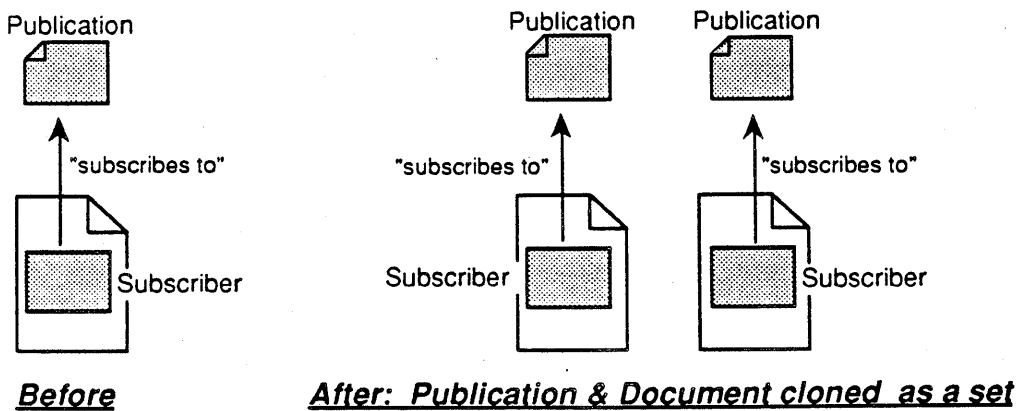
When the user duplicates a *publication and subscribing document* (together at the same time) on the same volume, or copies a publication and subscribing document to another volume, then both the original and duplicate documents each subscribe to both the original and duplicate publications (See fig. 6.3-3). This can happen if the user selects both objects at the same time and then duplicates or copies, or if both of the objects are in the same folder and the folder is duplicated or copied.



**Fig 6.3-3: Duplicating a Publication & Document**

Note that this is not what the user always (usually?) wants to have happen. If the user duplicates the publication and subscribing document for the purpose of backing up, then it is exactly what he wants. However, if the user duplicates the publication and subscribing document in order to *clone* the set, then it is exactly what he does not want.

For example, suppose the user has version 1.0 of a specification that consists of a document which subscribes to a publication. Now suppose that the user wishes to make an experimental version 1.1, and so he does not want to destroy version 1.0. The natural thing to do would be to clone version 1.0 and call it version 1.1 and then make the changes to the clone. The user would like for the cloned publication to be different than the original, so that the cloned document subscribes to the cloned publication, but the original document does not. Fig. 6.3-4 illustrates the *desired* result.



**Fig 6.3-4: Cloning a Publication & Document as a set**



To allow the user to clone a document and publication as a set requires a new Finder verb (e.g. "Clone" ?). Without such a new verb, there is no way for the user to accomplish this.

#### **6.4 Backing Up/Restoring Publications & Documents**

Backing up a *publication* is done by copying it (see section 6.3: Copying & Duplicating Publications). Restoring a publication causes any subscribing documents to be updated with the restored edition the next time those documents are opened (if automatic updating is turned on; otherwise, the next time the user issues the Get Latest Edition command).

Backing up a *document* is done by copying it (see section 6.4: Copying & Duplicating Documents). A restored document is updated with the latest editions the next time it is opened. Restoring a document may cause subscribers which were deleted to reappear, or cause subscribers which were created to disappear. If a subscriber reappears and the publication no longer exists, then it is called an "abandoned subscriber" (abandoned subscribers are discussed in section 6.7: Deleting Publications & Documents).

With this scheme, the user can bring home a "back up" copy of some or all of his documents and/or publications, alter the copies at home, and then "restore" them by replacing the originals with the altered copies. Any changes made to the documents at home will be consistent with the documents and publications not brought home, and any new editions made at home will be updated to the documents not brought home the next time those documents are opened.

#### **6.5 Renaming Publications & Documents**

The user can rename publications and Diet Coke documents (using the Finder) without any adverse side effects. However, note that renaming a publication with something other than the Finder may cause applications to not be able to find the renamed publication the next time a subscribing document is opened.





## 6.6 Deleting Publications & Documents

The user can delete a publication by throwing it in the trash can and emptying the trash. If the publication is currently in use by an open document then the Finder prevents the publication from being thrown away. Otherwise, if there are no other copies of the publication on-line, then the Finder asks the user if the subscriptions to that publication should be automatically cancelled the next time those subscribing documents are opened. Presumably, if the publication being thrown away is a copy (the original is off-line) then the user will not 'automatically cancel subscriptions'; if the publication being thrown away is the last copy then the user will 'automatically cancel subscriptions'. The user may also cancel the deletion.

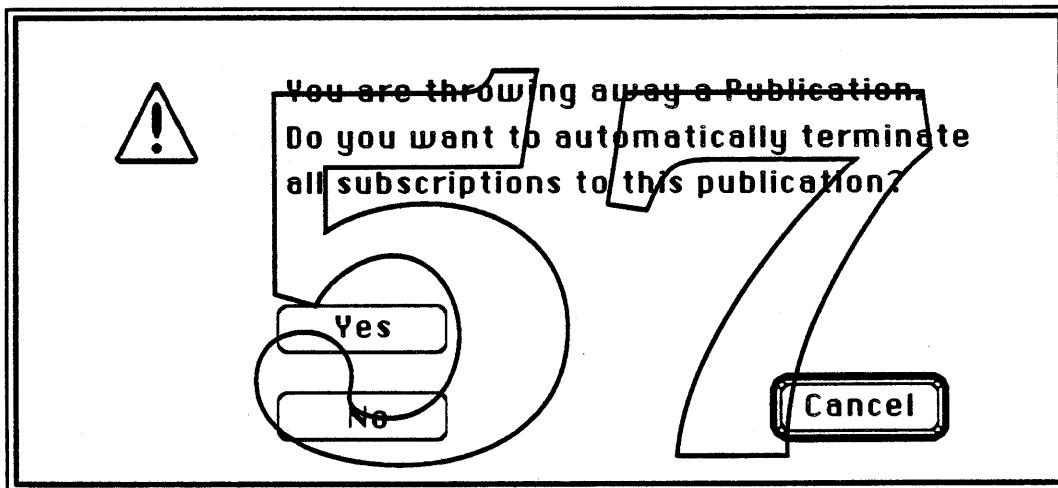


Fig 6.7-1: Delete Publication Dialog

If a publication is deleted and there are no other copies of it, then any remaining subscribers associated with that publication become "abandoned". An **abandoned subscriber** is a subscriber which has no associated publication. When a document containing an abandoned subscriber is opened, the application warns the user that it can't find the publication (as discussed in section 5.1: Opening a Document).



## 7.0 Related Documents

### Diet Coke Programmer Interface ERS

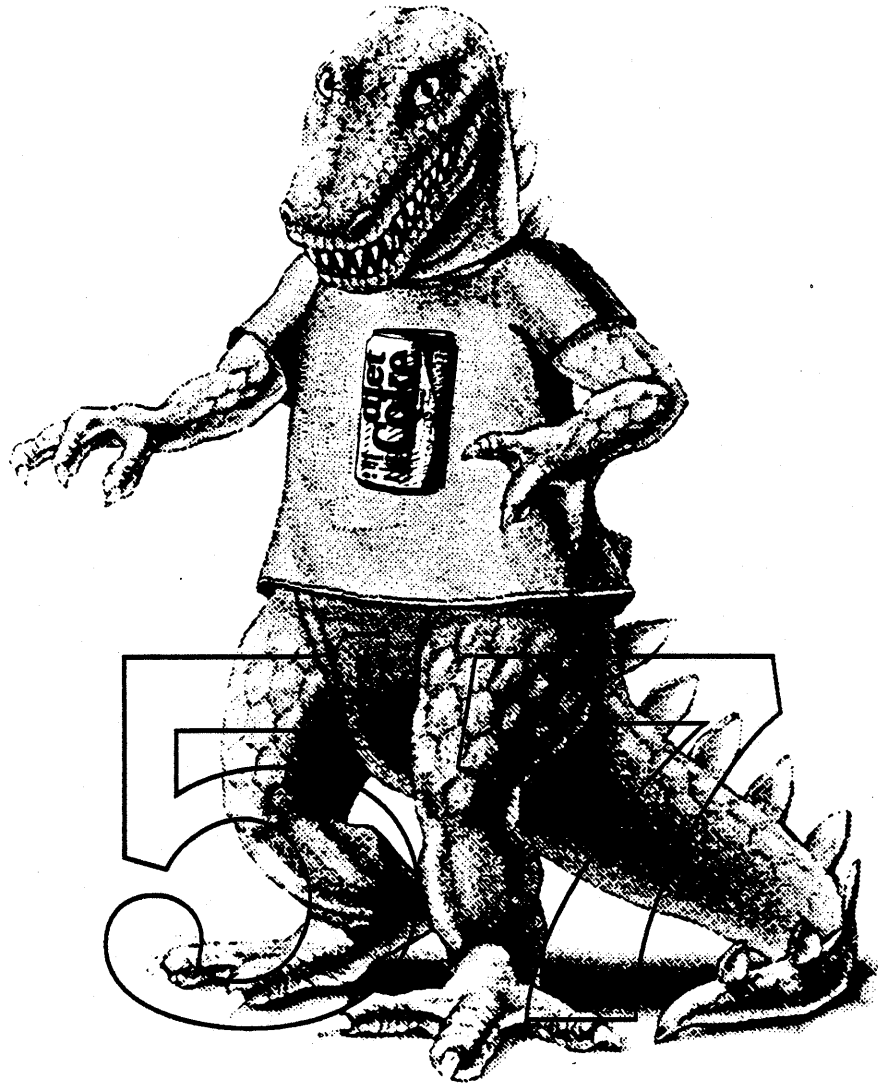
This document defines the Programmer Model (the way that the application programmer conceptualizes Diet Coke) and the Toolbox Interface (the interface that the programmer uses to implement Diet Coke in an application).

### Diet Coke Software Guts IRS

This document describes the internal workings of the system software. It defines the schemes and methods used to implement the Toolbox Interface.

57

57



# Programmer Interface ERS

Alan Lee  
Nick Kledzik

Version 0.5

November 23, 1988

🍏 Apple Confidential

57



# Table of Contents

|            |                                           |           |
|------------|-------------------------------------------|-----------|
| <b>1.0</b> | <b>About This Document</b>                | <b>1</b>  |
| <b>2.0</b> | <b>Programmer Model</b>                   | <b>2</b>  |
| 2.1        | Model Overview                            | 2         |
| 2.2        | SubscriptionRecords                       | 4         |
| 2.3        | Data Exchange Formats                     | 6         |
| 2.4        | Publication fdType & fdCreator            | 7         |
| 2.5        | Data Flow                                 | 8         |
| 2.6        | Documents                                 | 10        |
| <b>3.0</b> | <b>ToolBox Interface</b>                  | <b>11</b> |
| 3.1        | Subscriber Routines                       | 12        |
| 3.2        | Publication Routines                      | 16        |
| 3.3        | Data Exchange Routines                    | 19        |
| 3.4        | Events                                    | 22        |
| <b>4.0</b> | <b>Developing a Diet Coke Application</b> | <b>24</b> |
| 4.1        | Subscriber Creation and Deletion          | 24        |
| 4.2        | Handling New Editions                     | 25        |
| 4.3        | Handling Subscriber Updates               | 26        |
| 4.4        | Documents on Disk                         | 27        |
| 4.5        | Using the Clipboard                       | 28        |
| 4.6        | Derived Data Application Models           | 30        |

{ add copying Publications (reading and registering), Rules for Text and PICT usage, }

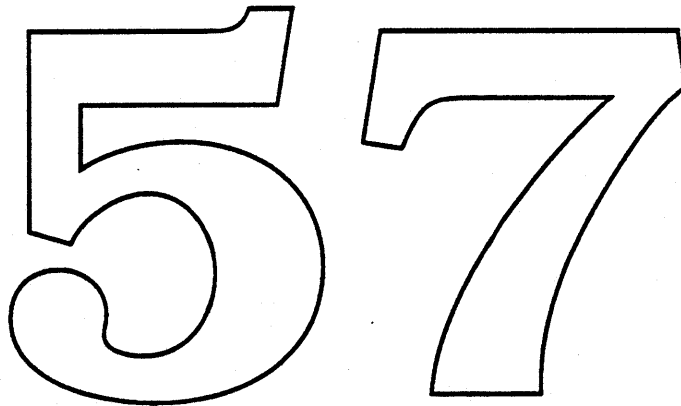
57



## 1.0 About this Document

This document describes the Diet Coke programmer interface. It defines the **Programmer Model** (the way that the programmer conceptualizes Diet Coke) and the **ToolBox Interface** (the way that those concepts are presented to the programmer via record structures and routines). The document also contains a section called **Developing a Diet Coke Application** which provides guidelines for developers on how to include Diet Coke in an application.

This document assumes that the reader is already familiar with the "Diet Coke Feature Description ERS" and "Diet Coke Human Interface ERS" documents.







## 2.0 Programmer Model

Diet Coke provides a facility for automated data exchange between documents. It has also been hyped as "inter-application communication" and "inter-application data exchange". However, from a programmer's point of view, it is better thought of as controlled data sharing. It is not an IPC mechanism. Data is exchanged by writing it to a special file (called a Publication) which is then read by another application. The shared access to the file is controlled by the Diet Coke (DC) manager.

### 2.1 Model Overview

#### Publications

The Diet Coke Programmer Model builds upon the user model in a straight-forward manner. There is a new kind of object known as a **Publication**. A publication can be thought of as a special type of document which contains data that can be shared. The current contents of the publication is referred to as the publication's "latest edition".

A Publication is similar to a document for the following reasons:

- A Publication shows up in the Finder via an icon which can be moved or renamed.
- A Publication can be opened (called "revising") by the creating application.

A Publication is different from a document for the following reasons:

- All access to all Publications is controlled through the DC manager. You should not use normal file system calls to access a Publication. In the future, Publications may not be implemented as files. *{How do you copy a Publication? E-mail, etc}*
- A Publication contains the data for its latest edition in many formats, including some standard formats so that any other document can subscribe to it and understand the data.
- An application may only open a Publication for revision if the application can write the new edition back out in the same formats; usually this limits it to the creating application.
- The data in a Publication tends to be simpler than the data in a document. This has to do with limitations in universal data formats. For example, a Publication edition can not contain a Subscriber.



Every Publication contains a unique identifier (called a PubID), which is assigned when the Publication is created. For BackUp and Restore purposes, a copy of a Publication contains the same PubID as the original. For efficiency, the Finder (or the system) keeps a list of all Publications and their location on a per volume basis. This is done in a manner similar to the way that the Finder maintains the list of all applications on-line in the desktop file of each volume (in order to quickly launch an application after the user double-clicks on one of its documents).

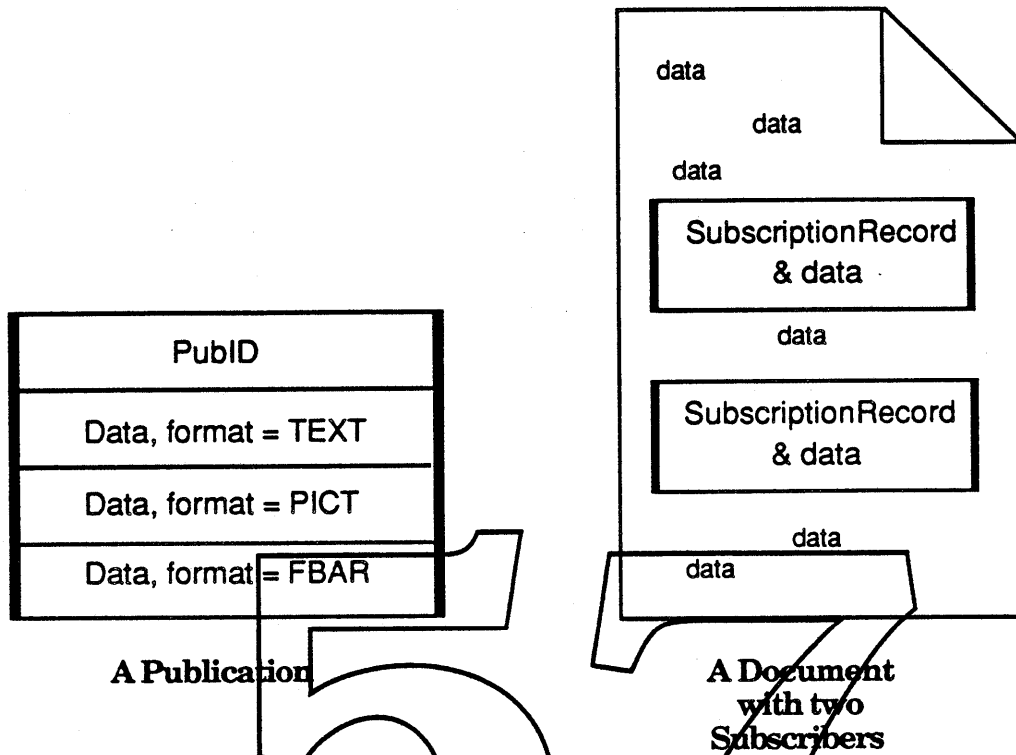
The PubID is a very large random number (~8 bytes); the intent is to make it *globally unique*. One way it might be constructed is to concatenate the clock time, the tick count, and a large random number. The PubID can always be made large enough to reduce the probability of collision to be less than the probability of you getting hit by lightning. (How do we handle the accidental duplicate PubID? Sue Mother Nature?) Like file names, a PubID can be unwieldy to work with. So, a PubRefNum is assigned to every Publication on an as-needed basis.

### Subscribers

The other new kind of object that Diet Coke introduces is a **Subscriber**. A Subscriber is an area of a document that is used to "include" a Publication. That is, the Subscriber area holds an edition (usually the latest) of its Publication. A Subscriber can also be thought of as a local cache of the contents of a Publication. A Subscriber always has a subscription to one particular Publication.

To maintain a Subscriber, an application keeps the following items with the document: 1) the location of the Subscriber in the document, 2) a copy (or cache) of the Publication data for that Subscriber, 3) a data structure called a "SubscriptionRecord", and 4) a structure containing any application-specific Subscriber attributes. The Subscriber's location and copy of the edition data can be stored and maintained in any way a developer likes. The SubscriptionRecord is used by both the application and the DC manager and so there are restrictions on how applications manage SubscriptionRecords.

The purpose of a SubscriptionRecord is very similar to a WindowRecord. It is a data structure which contains state information and is accessible by both the application and the DC manager. Unlike the window manager, the DC manager requires that the SubscriptionRecord be saved to disk with its document when the document is closed, rather than being disposed of (this makes subscriptions persistent). Also, SubscriptionRecords are variable length and are always referenced by a handle, rather than a pointer.



The above picture may help to summarize Publications and Subscribers. A Publication is a type of "file" that contains a PubID, which is used to identify it, and a latest edition, which is stored in multiple data formats. A Subscriber is an area of a document. It is comprised of a SubscriptionRecord, a copy of the associated Publication's edition data, and any application specific attributes.

## 2.2 SubscriptionRecord

A SubscriptionRecord is the data structure that the DC manager and an application use to internally represent a subscriber in a document. It is the application's responsibility to keep a SubscriptionRecord with each Subscriber in the document. The DC manager provides basic routines to create, dispose, and manage SubscriptionRecords. It also takes the responsibility of maintaining the association of each SubscriptionRecord to its Publication.

The DC manager "binds" each SubscriptionRecord to its Publication by appending the PubID to the end of the SubscriptionRecord. The PubID is not a SubscriptionRecord field; it is purposefully hidden from applications to allow for future expansion. The DC manager also fills in the PubRef field of each



SubscriptionRecord with a RefNum to its Publication. This gives applications a way to reference a subscriber's publication, and a way for the DC manager to find a Publication in an efficient manner.

|         |      |        |        |       |
|---------|------|--------|--------|-------|
| Edition | Mode | RefCon | PubRef | PubID |
|---------|------|--------|--------|-------|

**SubscriptionRecord**

The three other parts of a SubscriptionRecord are the Edition, Mode, and RefCon. The Edition field identifies the Publication edition of the data that the Subscriber now has. To maintain this, whenever an application reads a newer edition from a Publication, it should set the Edition field in the SubscriptionRecord, appropriately. The Mode is how the user expects to get new editions – "automatically" or "on request". The RefCon field is provided for application use. As an example, an application that can have multiple documents open, might use the RefCon field to hold a reference to the owning document. This "backwards" pointer could help to structure the internals of a document.

Because applications must save SubscriptionRecords to disk with a document when it closes, and restore them when it opens, the concept of "registration" is needed. The problem stems from the fact that it is the application that reads a SubscriptionRecord in from disk and not the DC manager. Because of this, there is a state during which the DC manager does not "know" about a SubscriptionRecord. This state is called "unregistered". To inform the DC manager about such a SubscriptionRecord, you "register" it. Another way to think of registration is that it turns an ordinary handle in to a handle to a SubscriptionRecord. This is very similar to the way AddResource changes an ordinary handle in to a handle to a loaded resource.

As a bonus feature, you can unregister the SubscriptionRecord of Subscribers that are in your clipboard or undo-buffer. This assures that you won't receive NewPubData events for those Subscribers; and allows you to throw away the contents of your clipboard or undo-buffer, without having to check to see if they contain registered SubscriptionRecords.

The routine to register a SubscriptionRecord also stuffs the PubRef field of a SubscriptionRecord with the appropriate PubRefNum. If this cannot be done, because more than one Publication is found with the required PubID; the user is alerted with a dialog asking to choose which one he wants to use and the appropriate PubRefNum is used. If no Publications are found with the required PubID, the user is alerted with a dialog warning that a Publication is not on-line



and the PubRef field is zeroed.

*{ One idea on the saving of SubscriptionRecords would be to require them to be stored as well-known resources. This has significant advantages for future capabilities (e.g. link maps, intelligent PubID renumbering, etc). The only cost is requiring developers to use the resource.fork. }*

### **2.3 Data Exchange Formats**

The basic formatting of data in Publications is the same as in the clipboard. That is, multiple formats of the data can co-exist in a Publication. Each format is labeled with a ResType and is a different representation of the same thing. Applications are encouraged to read and write as many different formats as possible, to ensure that Publications can be shared with as many different applications as possible. *(What formats will Apple require? support? just TEXT and PICT?)*

You may recall the rule that you can not have nested Publications (i.e. a Publication cannot contain a Subscriber). This rule vastly simplifies the data formats needed in a Publication. But, since Subscribers can be placed in the clipboard (necessary for the user who uses Cut/Paste to rearrange a document's contents), something needs to be done when the private clipboard is coerced to the global scrap.

There are a couple things that could be done: 1) only put a "static" copy of the data in the global scrap, 2) develop standard data formats that include the notion of Subscribers, or 3) develop incremental formats (like 'styl') that add the concept of Subscribers to existing data formats. Of the three, the latter seems the best. It would allow two Diet Coke aware applications to exchange Subscribers through the clipboard, and allow a DC aware application and non-DC aware application to exchange "static" data without an explosion of data types.

This implies a need for a data format type to describe the existence and location of Subscribers within other data types (e.g. within a TEXT chunk, position 45 through 76 is a Subscriber). *{This has yet to be defined. How do we do this for PICT, etc. ?}*

One of the data formats in each Publication is specially marked. It is the "minimal literacy needed to change the publication". That is, only applications that can read and write that ResType are allowed to revise the Publication. The Publication's Finder file type is used to specify that special ResType.



The reason for the minimal literacy ResType is that often times, the universal formats are rich enough to display the data, but not rich enough to edit it. For example, a MacDraw II document could be saved as a PICT, but (PICT comments aside) all grouping and layer information would be lost. This extra information is often critical to the correct meaning of the data in a Publication.

A more concrete example of this is a spreadsheet Publication. It may contain formulas that reference other spreadsheets. This Publication could contain formats 'TEXT' and 'calc', where 'calc' is some format that understands the inter-document formulas. If we did not have the "minimal literacy rule", any application that could edit TEXT would be able to revise the Publication. This would result in the 'calc' format being lost when the application wrote the new edition to the Publication. The meaning of the spreadsheet would be lost along with the 'calc' format. Thus, an application that creates these spreadsheet Publications should set the Publication file type to 'calc' to keep other applications out.

*(As you can see it is important to register format types with Apple?!?!?)*

#### 2.4 Publication fdType & fdCreator

As described above, a Publication's file type has a slightly different meaning from that of a document. In a document, the file type is used to "name" the format and layout of data in the file. In a Publication, the file type is used to choose which of the multiple formats in the Publication is the "richest" and must be understood in order to revise the Publication.

An additional note: Many applications can read a document that has a "foreign" type. The document is then converted to the application's preferred type and is saved in another document with that preferred type. This is not done with Publications because the new Publication would have a different PubID, which is probably not what is wanted. Instead, after appropriate warnings, the application saves the data back to the original Publication in the new format, overwriting the old format. The application also sets the Publication's file creator to be the application's signature.

The other use for the fdType and fdCreator attributes of a document is to bind an icon to the document in the Finder. This is also true for Publications. Applications need to have icons for their Publications and use the existing mechanism to bind the icons to Publications through use of fdType and fdCreator.



## 2.5 Data Flow

The basic model of data flow is: 1) User opens a Publication for revision, 2) User make changes to the Publication data, 3) User saves changes as a new edition, 4) Application with subscribing document receives a "NewPubData Event", 5) Application reads new edition from Publication, 6) Application displays new edition in Subscriber area. The following goes into further detail on how this is coordinated.

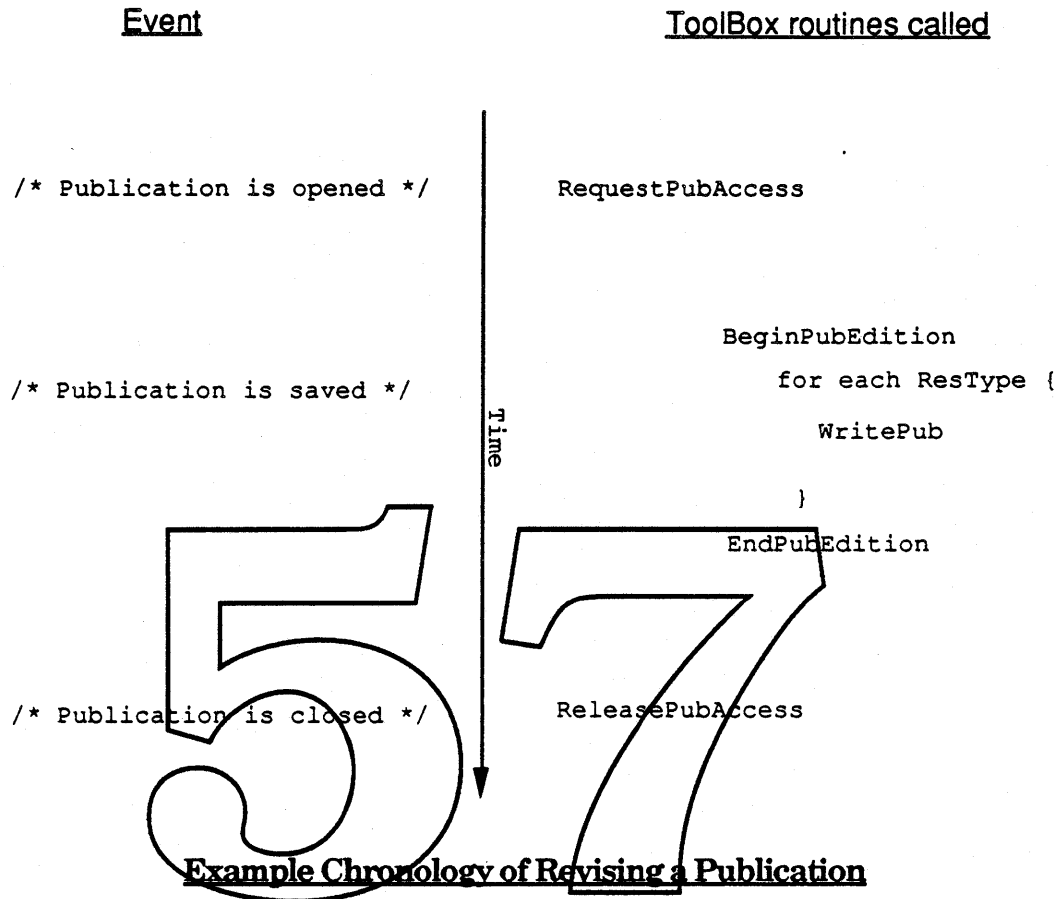
### Opening a Publication for Revision

There are three paths a user can take to cause a Publication to be opened for revision: 1) open (or double-click on) the Publication in the Finder, 2) select a Subscriber in a document and issue the Revise Publication command, 3) select a Publication from SFGGetFile after an Open... command. In cases 1) and 2), the creating application will be launched if necessary and fed an "OpenPublication Event". In case 3), the application is already running and "in control". The application should have set up the filters in SFGGetFile so that Publications which it cannot revise are grayed out. In all cases, before opening the revision window, the application needs to request exclusive revision access to the Publication (via RequestPubAccess). It, of course, releases the access (via ReleasePubAccess) when closing the Publication.

The getting and releasing of exclusive revision access enforces the rule that a Publication can only be revised in one place at a time (this becomes important in the future when Diet Coke is extended to work in a networked environment). Note that while a Publication is under revision, any application can still read the contents of the Publication. They just get the last saved (or published) edition. And, if a new edition is saved while an application is reading the Publication, the first read after the new edition will return an error code. The reading application would then start reading the Publication all over again.

### Saving a Publication

Saving a Publication is different than saving a document. The "write" calls are through the DC manager instead of through the file manager. And since editions are an important concept in DC, the "writes" are bracketed with BeginPubEdition and EndPubEdition calls. In other words, you must 1) declare a new edition, 2) write all the data in all formats for that edition, 3) declare the edition complete and available to Subscribers. You do not get to "append to the end of the file" or "change a few bits in the middle of the file". You must do a "safe save".



NewPubData Event Generation

Each Publication contains an edition number. It is the date and time that the edition was saved. Each SubscriptionRecord also has an edition number. It reflects the revision of the Publication which the document has cached. The DC manager keeps track of all registered SubscriptionRecords and compares the edition number in them against the edition number in each respective Publication. The DC manager only compares the edition field of a SubscriptionRecord with its Publication's edition field at certain well-known times. They are 1) when its Publication gets a new edition, 2) when the SubscriptionRecord is registered, and 3) when a volume is mounted causing its Publication to "come on-line". If a Subscriber is "out of date", a NewPubData Event is sent to its application informing it of the need to update.





## Reading an Edition

After receiving a NewPubData Event or if the user issues a Get Latest Edition, the application should read the edition data from the Publication. To do this, it does not need to "open" the Publication or "request access" as it did when revising a Publication. Instead, the application asks for the data formats available (via GetPublicationInfo) and then reads the formats it wants (via ReadPub). (For you curious folks, the low level file opening and closing of Publications is done on an as needed basis for registered SubscriptionRecords.)

### Event

```
/* New Edition is read*/
```

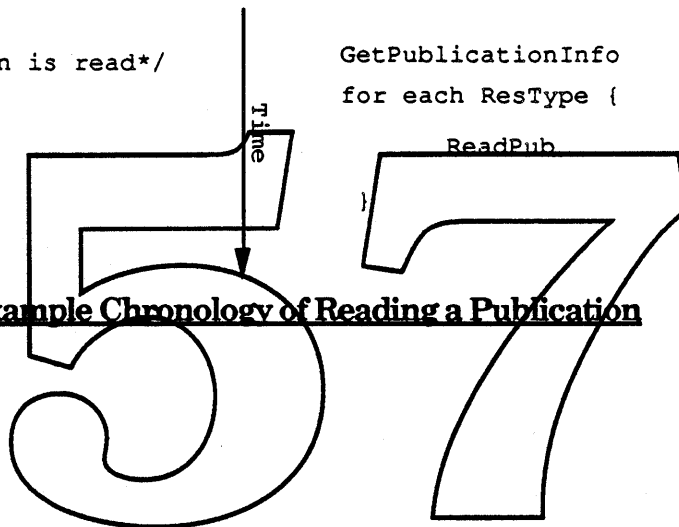
### ToolBox routines called

```
GetPublicationInfo
```

```
for each ResType {
```

```
  ReadPub
```

### Example Chronology of Reading a Publication



## 2.6 Documents

The data structures of a document are still application-specific. The only document structure change that DC dictates, is the addition of Subscribers. Again, for each subscriber, the document must hold a SubscriptionRecord and the edition data. In memory, an application is free to structure this in any way it desires, providing that the SubscriptionRecord is stored in a handle and is registered. On disk, applications may store documents in any way they wish.

*{ Do we require SubscriptionRecords be put in its document's resource fork? This gives other programs the ability to find subscribers in any document (e.g. to build a map of links). However, this precludes multi-user documents since the Resource Manager is single-user. }*



### 3.0 Toolbox Interface

As described in the Programmer Model, the basic toolbox structure maintained for each Subscriber is the SubscriptionRecord. Each SubscriptionRecord in use is stored in the application's heap as a relocatable block and referenced via a SubscriberHandle. It is important to remember that the SubscriptionRecord is a shared data structure. It is shared between an application and the DC manger. The only field which applications may arbitrarily set is the RefCon field, which is specifically intended for application use only. The record format is shown below in Pascal:

```

TYPE
  UpdateMode      = ( umAutomatic, umManual );
  Timestamp       = LongInt;  {?}
  PubRefNum       = Integer;  {?}
  SubscriberHandle = ^^SubscriptionRecord;
  SubscriptionRecord = RECORD
    Version: Integer;  { for future expansion }
    Edition: Timestamp; { edition last read }
    Mode: UpdateMode;  { receive automatically? }
    PubRef: PubRefNum; { only valid when registered }
    RefCon: longint;   { application specific }
    { hidden stuff of variable length for use by DC manger only }
  END
  
```

Edition: The Edition field is used to identify the edition of the data the document currently contains for this Subscriber.

Mode: The Mode field is used to specify whether the Subscriber should receive NewPubData events (umAutomatic) or not receive them (umManual).

PubRef: The PubRef field is filled in and maintained by the DC manger while a SubscriptionRecord remains registered. It contains a RefNum to the Publication to which the Subscriber subscribes. When the Record is on disk or is unregistered, the PubRef field is meaningless.

RefCon: The RefCon field is left for application-specific use only.

SubscriptionRecords are of variable length. Applications should not depend on the size of SubscriptionRecords being any fixed length. Since their length can change at any time, they are always stored as relocatable blocks in the application's heap and are only referenced via their handle. The DC manager adds its own internal variable length structure to the end of the SubscriptionRecord. This extra info contains the PubID and must be saved with the SubscriptionRecord. Whenever an application needs to know the size of a SubscriptionRecord, it should call GetHandleSize.



### 3.1 Subscriber Routines

NewSubscriber is the standard way to create a new Subscriber in a document and associate it with an existing Publication. The Publication with which to associate it is specified by the PubVRefNum, PubDirID, and PubName fields. The initial mode of the Subscriber is set by the mode parameter.

**FUNCTION NewSubscriber(PubVRefNum: INTEGER; PubDirID: LongInt; PubName: Str31; mode: UpdateMode; VAR SectH: SubscriberHandle): OSErr;**

PubVRefNum: The PubVRefNum parameter is the volume (or working directory?) RefNum of the volume containing the Publication file with which the new Subscriber is to be associated.

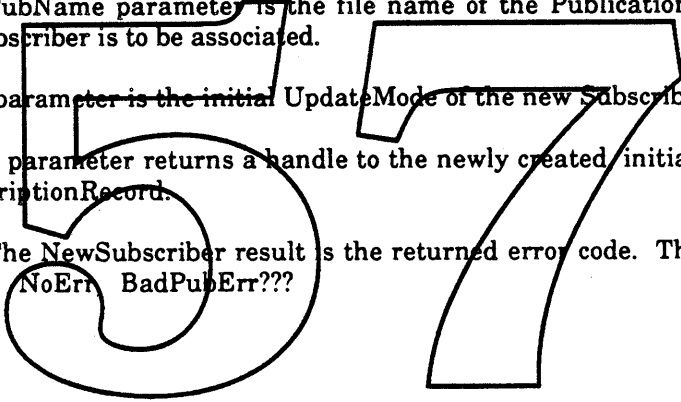
PubDirID: The PubDirID parameter is the parent dirID of the Publication file with which the new Subscriber is to be associated.

PubName: The PubName parameter is the file name of the Publication file with which the new Subscriber is to be associated.

mode: The mode parameter is the initial UpdateMode of the new Subscriber.

SectH: The SectH parameter returns a handle to the newly created, initialized, and registered SubscriptionRecord.

NewSubscriber: The NewSubscriber result is the returned error code. The possible return values are: NoErr, BadPubErr???



When a Subscriber is deleted from your document, call DisposeSubscriber to properly dispose of the handle and inform the DC manager that it is gone. This flushes any events for that Subscriber that are still in your event queue and removes the SubscriberHandle from the DC manager's list of registered SubscriberHandles. Also, when closing a document, you should iterate through all Subscribers in the closing document and call DisposeSubscriber on each.

**FUNCTION DisposeSubscriber(theSect: SubscriberHandle): OSErr;**

theSect: The theSect parameter is the handle of the Subscriber to dispose.

DisposeSubscriber: The DisposeSubscriber result is the returned error code. The possible return values are: NoErr, BadSectHndl, ???



`SubscriberInfoDialog` is the standard way to allow the user to see status information about a Subscriber and possibly change its attributes. The dialog is expandable. Applications that have additional Subscriber options may add items to the dialog to control those options. The strategy for these additions is that the DC manager controls the general attributes (currently only `UpdateMode`) and the application controls its specific attributes.

**FUNCTION** `SubscriberInfoDialog(theSect: SubscriberHandle; DITLItems: Integer; dlgHook: ProcPtr; RefParam: univ longint ): OSErr;`

theSect: The `theSect` parameter is the handle to the Subscriber of which the dialog is about.

DITLItems: The `DITLItems` parameter is the resource ID of a DITL which contains a list of application-specific dialog items only. The coordinates of the items are taken to be relative to the application area, not to the dialog window. If no additional items are needed, pass in 0.

dlgHook: The `dlgHook` parameter is a pointer to a procedure to handle item hits of application-specific items. Item hits in the DC area will not be passed to this procedure. If no additional items are needed, pass in NIL.

RefParam: The `RefParam` parameter is used to pass a value to your `dlgHook`. This can eliminate the need for a global variable. Usually, you will put a pointer/handle to your own data structure for the subscriber.

SubscriberInfoDialog: The `SubscriberInfoDialog` result is the returned error code. The possible return values are: `NoErr`, `CancelErr`, `BadSectHndl`, ???

When calling `subscriberInfoDialog`, the application should pass in the `SubscriberHandle` of the Subscriber in mind, the DITL resource ID of the list of application-specific items to add to the dialog, a pointer to a procedure to handle the controlling of those application-specific items, and an additional parameter to pass to that procedure.

Unlike `SFGetFile`, the item list does not include the standard items, only the application-specific ones. `SubscriberInfoDialog` will append your items to the end of its items. Because they are appended, the item number of your specific items will be different at run-time than their position in your DITL resource. To remedy this, your `dlgHook` will be passed the `itemOffset` parameter. Add it to your constant item numbers to get their position in the dialog item list at run-time.

Your specific items will also be repositioned to fit below the general controls. The rule of thumb is that Apple reserves the right to use the upper half of a MacPlus sized screen for its general controls. All application-specific controls keep their relative position. They are moved as a group and centered in the lower half of the dialog.



The following is an example of a dlgHook for use with SubscriberInfoDialog.

**FUNCTION MyDialogHook(theDialog: DialogPtr; theSect: SubscriberHandle;  
ItemHit: INTEGER; ItemOffset: INTEGER; RefParam: univ longint ): ??;**

theDialog: The theDialog parameter is a pointer to the dialog record that SubscriberInfoDialog built. Usually, you will use this only as a parameter to other Dialog manager routines, such as GetDitem().

theSect: The theSect parameter is the handle to the Subscriber of which the dialog is about. This is passed to you so that you can inspect the fields to see if changes will cause you to change your items. Any time a change is made to a general attribute, the dummy itemHit = 0 is passed to you.

itemHit: The itemHit parameter is the item number of your item that was hit. To calculate your relative item number, subtract the value of itemOffset.

itemOffset: The itemOffset parameter is the difference between the item numbering in your DITLitems and those in itemHit. It is equal to the number of general items in the item list.

RefParam: The RefParam parameter will contain whatever you put in the RefParam field of SubscriberInfoDialog.

MyDialogHook: The MyDialogHook is ???

*{ should there be a programmatic way to change the UpdateMode? }*

RegisterSubscriber is used to inform the DC manager when a new SubscriberHandle has been created or re-created by an application. When a document is opened, the DC manager needs to be informed about each Subscriber before any routines can be called which refer to them. A simple approach for an application opening a document would be: read in all the SubscriptionRecords stored in the document, store them in memory as SubscriberHandles, and then register each. If RegisterSubscriber returns RevertErr, the application should cancel the subscription. This routine may bring up a dialog if more than one Publication is found with a matching PubID.

**FUNCTION RegisterSubscriber(theSect: SubscriberHandle): OSErr;**

theSect: The theSect parameter is a handle to the SubscriptionRecord to be registered with the DC manager.

RegisterSubscriber: The RegisterSubscriber result is the returned error code. The possible return values are: NoErr, BadSubscriberErr, RevertErr, ???



When cutting a Subscriber into the clipboard, or putting it into the undo-buffer as the result of a delete, an application can `UnRegister` the Subscriber. In general, the application can `UnRegister` a Subscriber whenever it does not make sense to receive `NewPubData` events for that Subscriber. Note: `DisposeSubscriber` is equivalent to `UnRegisterSubscriber` and then `DisposHandle`, so all of the Subscribers in a document should be indirectly unregistered via `DisposeSubscriber` when the document is closed.

**FUNCTION `UnRegisterSubscriber(theSect: SubscriberHandle): OSErr;`**

`theSect`: The `theSect` parameter is a handle to the `SubscriptionRecord` to unregister with the DC manager.

`UnRegisterSubscriber`: The `UnRegisterSubscriber` result is the returned error code. The possible return values are: `NoErr`, `BadSectHndlErr`, ???

The image shows two large, hollow outline numbers, '5' and '7', positioned side-by-side. The '5' is on the left and the '7' is on the right. Both numbers are drawn with a simple black outline and no fill.



### 3.2 Publication Routines

`NewPublication` is the standard way to create a Publication. It is used when the user saves an "untitled" Publication. First, call a variant of `SFPutFile` to allow the user to select the place and name of the Publication. `NewPublication` automatically calls `RequestPubAccess` internally, to prevent a possible race condition which could occur if `RequestPubAccess` were called afterwards. Be sure to call `ReleasePubAccess` when closing the Publication window.

**FUNCTION** `NewPublication`(`PubVRefNum`: INTEGER; `PubDirID`: LongInt; `PubName`: Str31;  
`fdType`, `fdCreator`: OSType; VAR `thePubRef`: PubRefNum): OSErr;

`PubVRefNum`: The `PubVRefNum` parameter is the volume `RefNum` of the volume on which to create the new Publication file.

`PubDirID`: The `PubDirID` parameter is the `DirID` of the folder in which to create the new Publication file. This can be 0 if you are using working directories.

`PubName`: The `PubName` parameter is the file name of the Publication file to be created on the `PubVRefNum` volume.

`fdType`: The `fdType` parameter is the initial Finder file type of the Publication file to be created.

`fdCreator`: The `fdCreator` parameter is the initial Finder file creator of the Publication file to be created.

`thePubRef`: The `thePubRef` parameter is filled in with a `PubRefNum` to the created Publication.

`NewPublication`: The `NewPublication` result is the returned error code. The possible return values are: `NoErr`, `DupFNErr`, ???



`DeletePublication` is the standard way to delete a Publication file. In general, Publications are deleted through the Finder, which calls this same routine, and not from within applications. ( *should this be visible to App developers?* )

**FUNCTION** `DeletePublication(PubVRefNum: INTEGER; PubDirID: LongInt; PubName: Str31; TerminateSubscribers: Boolean): OSErr;`

PubVRefNum: The `PubVRefNum` parameter is the volume (or working directory) `RefNum` of the volume which contains the Publication file to delete.

PubDirID: The `PubDirID` parameter is the `DirID` of the folder which contains the Publication file to delete. This can be 0 if you are using working directories.

PubName: The `PubName` parameter is the file name of the Publication file to be deleted on the `PubVRefNum` volume.

TerminateSubscribers: The `TerminateSubscribers` parameter is used to tell whether or not the DC manager should leave a stub in place of the Publication which causes Subscribers to "cancel their subscription" the next time they try to register.

DeletePublication: The `DeletePublication` result is the returned error code. The possible return values are: `NoErr`, `BadPubErr`, `PubInUseErr`, ???

`GetPubRefNum` is the standard way to get the `PubRefNum` for an existing Publication, given its HFS location. This is used when you receive an `OpenPublication` event, since all routines used to revise a Publication require a `PubRefNum`.

**FUNCTION** `GetPubRefNum(PubVRefNum: INTEGER; PubDirID: LongInt; PubName: Str31; VAR thePubRef: PubRefNum): OSErr;`

PubVRefNum: The `PubVRefNum` parameter is the volume `RefNum` of the volume which has the interesting Publication file.

PubDirID: The `PubDirID` parameter is the `DirID` of the folder which has the interesting Publication file. This can be 0 if you are using working directories.

PubName: The `PubName` parameter is the file name of the interesting Publication file.

thePubRef: The `thePubRef` parameter is filled in with the `PubRefNum` of the interesting Publication file.

DeletePublication: The `DeletePublication` result is the returned error code. The possible return values are: `NoErr`, `BadPubErr`, `PubInUseErr`, ???





GetPublicationInfo is used to get some attribute information about a Publication.

**FUNCTION GetPublicationInfo(thePubRef: PubRefNum; VAR PubInfo: PubInfoRecord): OSErr;**

thePubRef: The thePubRef parameter is the PubRefNum of the Publication for which information is wanted.

PubInfo: The PubInfo parameter is a record to be filled in with the Publication information.

GetPublicationInfo: The GetPublicationInfo result is the returned error code. The possible return values are: NoErr, BadPubRefErr, ???

The PubInfoRecord structure is described below.

```

TYPE
  PubInfoRecord = RECORD
    CrDate: Timestamp; { date publication was created }
    Edition: Timestamp; { date of current edition }
    Formats: ResTypeList; { all formats available }
    fdType: OsType; { Pub file type }
    fdCreator: OsType; { Pub file creator }
  END;

```

CrDate: The CrDate field is the Creation date and time of the Publication.

Edition: The Edition field is the date and time of the latest edition (i.e. mod date).

Formats: The Formats field is a list of all ResType available for the latest edition.

fdType: The fdType field specifies which ResType in the Formats field that an application must understand in order to revise this Publication.

fdCreator: The fdCreator is the signature of the application that created or last revised the publication.

RevisePublication is used to ask the system to ask the appropriate application to revise the Publication to which theSect subscribes. The application is launched, and the Publication is opened, if necessary. This is equivalent to opening (double-clicking) the Publication in the Finder.

**FUNCTION RevisePublication(theSect: SubscriberHandle): OSErr;**

theSect: The theSect parameter is the handle to the SubscriptionRecord whose Publication is to be revised.

RevisePublication: The RevisePublication result is the returned error code. The possible return values are: NoErr, BadSectErr, BadFlavorErr, BadInfoErr, ???



### **3.3 Data Exchange Routines** *{Should these be streaming?}*

An application should call `GetPublicationInfo` to see what data formats are available and which edition is the current, before using `ReadPub`. `ReadPub` is used to read a single data type of a particular edition of a particular `Publication`. If the data format `whichType` does not exist `BadTypeErr` is returned. If the edition `whichEdition` is not the current edition, then `BadEditionErr` is returned. This probably means a new edition was Published while the read loop was in progress, in which case the application should start over and call `GetPublicationInfo`. When done reading the needed formats, the application should set the `Edition` field in the `SubscriptionRecord` to be the value used for `whichEdition`.

**FUNCTION** `ReadPub(theSect: SubscriberHandle; whichType: ResType; whichEdition: Timestamp; VAR ResH: Handle): OSErr;`

theSect: The `theSect` parameter is the handle to the `SubscriptionRecord` whose `Publication` data is to be read.

whichType: The `whichType` parameter specifies which `ResType` to read.

whichEdition: The `whichEdition` parameter specifies which edition to read.

ReadPub: The `ReadPub` result is the returned error code. The possible return values are: `NoErr`, `BadSectErr`, `BadTypeErr`, `BadEditionErr`, ???

This call is a prerequisite for revising a `Publication`. `RequestPubAccess` is used to gain exclusive write access to a `Publication`. Until `ReleasePubAccess` is called, all subsequent calls to `RequestPubAccess` for the same `Publication` by any user or application will return `PubInUseErr`. *{What if an app calls `RequestPubAccess` twice? this may be necessary for crash/error recovery} {future note: In the case of `PubInUseErr` on a network, can we return the Chooser name of the person using it? }*

**FUNCTION** `RequestPubAccess(thePubRef: PubRefNum): OSErr;`

thePubRef: The `thePubRef` parameter is the `PubRefNum` of the `Publication` to which exclusive access is being requested.

RequestPubAccess: The `RequestPubAccess` result is the returned error code. The possible return values are: `NoErr`, `PubInUseErr`, ???



ReleasePubAccess is used to release exclusive write access to a Publication. This call should coincide with the closing of the Publication window.

**FUNCTION ReleasePubAccess (thePubRef: PubRefNum): OSErr;**

thePubRef: The thePubRef parameter is the PubRefNum of the Publication of which exclusive access is being released.

ReleasePubAccess: The ReleasePubAccess result is the returned error code. The possible return values are: NoErr, BadPubErr, NoAccessErr, ???

BeginPubEdition is used to declare the start of a new edition of a Publication. BeginPubEdition fails if the application does not have exclusive write access to the Publication (see RequestPubAccess). Subsequent WritePub calls will write to the new edition. If BeginPubEdition is called again before an EndPubEdition, the new edition is "restarted", losing any data written since the previous BeginPubEdition.

**FUNCTION BeginPubEdition(thePubRef: PubRefNum): OSErr;**

thePubRef: The thePubRef parameter is the PubRefNum of the Publication of which a new edition is being created.

BeginPubEdition: The BeginPubEdition result is the returned error code. The possible return values are: NoErr, BadEditionErr, PubInUseErr, ???

EndPubEdition is used to declare the completion a new edition of a Publication. All data for the new edition should have already been written to the Publication. The call to EndPubEdition makes the edition accessible as the "latest edition". Any registered Subscriber will receive NewPubData events for the Publication. EndPubEdition also sets the edition in the Publication. The Finder file type and creator of the Publication are also set. The fdCreator should be the same that was returned by GetPublicationInfo or, if the formats are changed, it is your application's signature. The fdType is the minimal literacy ResType.

**FUNCTION EndPubEdition (thePubRef: PubRefNum; fdType, fdCreator: OSType): OSErr;**

thePubRef: The thePubRef parameter is the PubRefNum of the Publication of which a new edition is being created.

fdType: The fdType parameter is the new Finder file type of the Publication just updated.

fdCreator: The fdCreator parameter is the new Finder file creator of the Publication just updated.

EndPubEdition: The EndPubEdition result is the returned error code. The possible return values are: NoErr, BadPubRefErr, ???



`WritePub` is used to write a single data type to a particular Publication. If that data type already exists, it is overwritten. If exclusive access has not been granted through `RequestPubAccess`, `PubAccessErr` is returned.

**FUNCTION** `WritePub(thePubRef: PubRefNum; whichType: ResType; VAR ResH: Handle): OSErr;`

`thePubRef`: The `thePubRef` parameter is the `PubRefNum` of the Publication to which to write.

`whichType`: The `whichType` parameter specifies which `ResType` to write.

`ResH`: The `ResH` parameter is a handle to the block of memory to be written to the Publication.

`WritePub`: The `WritePub` result is the returned error code. The possible return values are: `NoErr`, `BadPubRefErr`, `PubAccessErr`, ???

57



### 3.4 Events

A `NewPubData<event>` is sent to an application if it has a registered Subscriber with mode set to `umAutomatic` and the Publication subscribed to has a different value for the `edition` field than the `SubscriptionRecord` does. The DC manager only compares the `edition` field of a `SubscriptionRecord` with its Publication's `edition` field at certain well know times. They are 1) when its Publication gets a new edition, 2) when the `SubscriptionRecord` is registered, and 3) when a volume is mounted causing its Publication to "come on-line".

`<event> NewPubData[ theSect: SubscriberHandle ]`

theSect: The `theSect` parameter is the handle to the `SubscriptionRecord` whose Subscriber needs to be updated from its Publication.

An `OpenPublication<event>` is sent to an application informing it to revise a Publication or open a document. It is used by the DC manager when a user issues the "Revise Publication" command, and when the user opens a Publication directly from the Finder. This event could also be used by MultiFinder when opening a document from the Finder for an already running application.

`<event> OpenPublication[ VRefNum: INTEGER; DirID: LongInt; Name: Str31; fType: OSType; access: ? ]`

VRefNum: The `VRefNum` parameter is the volume `RefNum` of the volume which has the file to open.

DirID: The `DirID` parameter is the `DirID` of the folder which contains the file to open This can be 0 if you are using working directories??

Name: The `Name` parameter is the file name of the file to be opened on the volume `VRefNum` in the folder `DirID`.

fType: The `fType` parameter is the file type of the file to open.

access: The `access` parameter is the mode with which the file should be opened. Example: read-only, read/write, read/deny write, etc.



{ The following events are optional, but not necessary:  
A quit application event. This would help Multifinder. It might be used to auto-quit an application that was auto-launched to revise a Publication. }

{ We still need routines or dialog resources for standard warning messages }

{ We still need to see what to do about SFGetfile and SFPutfile. Do we or developers create the DITL resource and code to handle the extra options? }

{ if SubscriptionRecords must be stored in the resource fork, we need some routines to help with this.

Ideas for resource based SubscriptionRecords...

NewSubscriber: creates a SR from the parameters

AddSubscriber: adds a SR to the current Resfile

- ChangedResource (is this too much of a good thing? size unchanging)

- WriteResource

- UpdateResfile

GetNewSubscriber: creates a SR by reading a resource

}

{ What do we do for Publication windows? new WDEF? special mini-icon in dragbar? special formatting of title? }



## 4.0 Developing a Diet Coke Application

Assuming you have read the "Diet Coke Human Interface ERS" document and the sections preceding this, you are familiar with the Diet Coke User Model, User Interface, Programmer Model, and ToolBox Interfaces. This section describes how they are all combined together in an application. The first part is organized in an event driven manner. That is, it describes what to do for each new kind of "event" an application might receive. The second part gives ideas on how to integrate Diet Coke into different document models.

### 4.1 Subscriber Creation and Deletion

#### Subscribe...

This creates a new Subscriber in the current document at the current selection. If anything was selected, it should be removed (into the undo buffer). The application next calls a variant of `selectFile` to ask the user which Publication to use. If the user cancels out of this dialog, the original selection is restored. If the user does not cancel, then call `NewSubscriber` with the selected Publication and initial `UpdateMode`. The `SubscriptionRecord` created by `NewSubscriber` should be added to the document structure. The creation of the new Subscriber will cause the DC manager to immediately send a `NewPubData` event, which in turn, causes your application to read the data out of the Publication for the new Subscriber. Note: this needs to be undo-able.

#### New Publication...

Do the same thing you do when creating a new document. The title bar of the window should state that it is an unsaved Publication. *{Will there be a different WDEF proc for Publication windows?}* Note: this is not undo-able, but it is cancel-able.



### Deleting a Subscriber

When the user makes a selection that includes a Subscriber and then deletes or replaces that selection (e.g. uses the Clear command, Paste command, or hits the delete key), the application needs to delete the Subscriber. To make this easily undo-able, don't immediately delete the Subscriber, instead `UnRegister` and move it (the `SubscriptionRecord`) with the rest of the data, to your "undo buffer". When the next action which overwrites the undo buffer occurs, call `DisposeHandle` to dispose of the memory. Note: this is undo-able.

### Cancel Subscription

This menu item should only be enabled if at least one Subscriber is in the current selection. The application should dispose of the `SubscriptionRecord` for each selected Subscriber and change its data structures so that the data that was in the Subscriber(s) is now editable and part of the "normal" data in the document.

### Subscription Info...

This menu item should only be enabled if exactly one Subscriber is in the current selection. The application should call `SubscriberInfoDialog` with the appropriate `SubscriptionRecord`. If the user does not cancel, the application should handle any application-specific attribute changes. The DC manager will take care of any `UpdateMode` change. (Does a change of `UpdateMode` constitute a document change? If so, apps need to know if changes were made in order to set its dirty bit. If app specific attributes depend on the `UpdateMode`, how does the app monitor `UpdateMode` changes during the dialog? what hooks are needed?) Note: this is cancel-able.

## 4.2 Handling New Editions

### Revise Publication

This menu item should only be enabled if exactly one Subscriber is in the current selection. The application should call `RevisePublication` with the selected `SubscriptionRecord`.



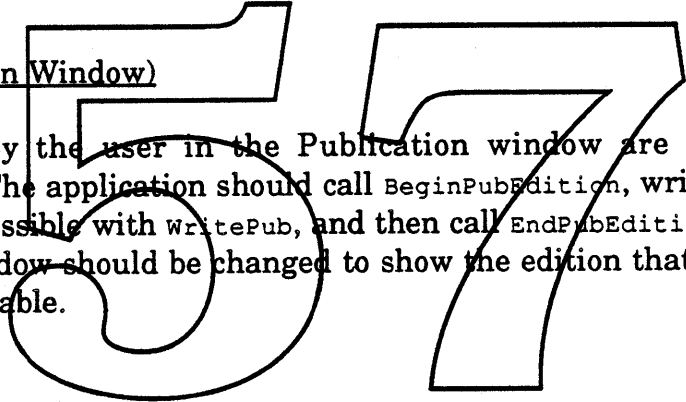


### OpenPublication <event>

This is the new toolbox event, not a user action. If the Publication specified is already open for revision, bring its window(s) to the front. If the specified Publication is not already open for revision, then begin the revision with `RequestPubAccess`. If it fails, warn the use with the appropriate alert (*standard?*) If it succeeds, open a Publication window. (*special Diet Coke window?*) In the window, display the data in a form editable by the user, as if it were a document. When the Publication window is active, all appropriate application functions should be available. The commands to **Subscribe...** and **Publication & Subscribe**, etc. should be disabled to prevent nested publications. Also, any application-specific restrictions on Publications should be enforced. Note: this is cancel-able (the user can close without saving).

### Save (on a Publication Window)

The changes made by the user in the Publication window are saved in the Publication on disk. The application should call `BeginPubEdition`, write the data in as many formats as possible with `WritePub`, and then call `EndPubEdition`. The title of the Publication window should be changed to show the edition that was written. Note: this is not undo-able.



### Close (on a Publication Window)

If changes have been made since the last Save, the application should ask the user if he would like to save the changes via a standard "Save? yes/no/cancel" dialog. When closing a Publication window, the application should call `ReleasePubAccess` to release its exclusive write access to the Publication.

## 4.3 Handling Subscriber Updates

### NewPubData <event>

This is the new toolbox event, not a user action. It contains the `SubscriptionRecord` handle of the Subscriber that needs to be updated. The DC manager will send this event to an application if it has a registered, automatically updating Subscriber which has become "out of date". In response, the application should call `GetPublicationInfo` to find the `ResTypes` available in the Publication and then call `ReadPub` to get the `ResTypes` of data it wants.



If the Publication is updated (by another application) while it is being read, one of the `ReadPub` calls will fail with "No such edition" error. The application should then start again with `GetPublicationInfo`. A robust application would keep count of how many times it had to start over and give-up/warn the user if the count gets unreasonable. (This only happens if the Publication continuously gets a new edition before the subscriber can read the previous one!)

`NewPubData` events can be received in the foreground or background, just like window update events. But, unlike window update events, an application will only get one `NewPubData` event per new edition of a Publication. If an application should ever ignore the event, it will not get another until it re-registers the Subscriber, the Publication gets another new edition, or the Publication goes off and then back on-line.

### Get Latest Edition

This menu item should only be enabled if at least one Subscriber is in the current selection. The application should check the edition date of the Subscriber vs the Publication (via `GetPublicationInfo`). If they are different, then treat this as if you just received a `NewPubData` event for the Subscriber. In either case, the appropriate user interface feedback of "highlighting" the Subscriber should be done. Note: this is not undo-able.

## 4.4 Documents on Disk

### Saving a Document

Do your normal save of the current document. Be sure to save the variable length `SubscriptionRecords` and the respective Subscriber edition data.

### Closing a Document

Again, close as normal. Be sure to "unregister" any `SubscriptionRecords` in the document. Using `DisposeSubscriber` will unregister and dispose of the memory allocated to the `SubscriptionRecord`.

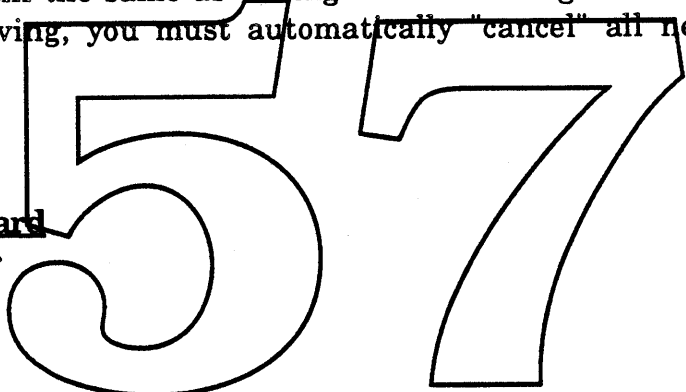


### Opening a Document

Opening a document should restore the document to its state before closing. This means, in addition to re-creating any application-specific data structures, that you need to re-create the `SubscriptionRecords`. To do this, allocate a handle that is the same size as the `SubscriptionRecord` on disk. Copy the `SubscriptionRecord` from disk into the memory block and then "register" it via `RegisterSubscriber`. *{This would be easier if `SubscriptionRecords` are stored as resources}* If any of the `Subscribers` have become "out of date", the DC manager will automatically send `NewPubData` events to the application.

### Reverting a Document

Revert to saved is still the same as closing without saving and then re-opening. To close without saving, you must automatically "cancel" all new editions in progress.



### 4.5 Using the Clipboard

#### Coercing the scrap

Applications are free to use whatever formats they choose for their private scrap. But when coercing their clipboard to the scrap, they need to use some standard formats and guidelines. One guideline is that applications which are not Diet Coke aware should be able to get a "static" copy of the data out of the scrap. This means the scrap needs to have a "static" copy of the data, as well as information describing where the `Subscribers` are in the scrap (if any), and the information about the `Subscribers` themselves. This strategy is similar to the way 'styl' was added to 'TEXT' - old apps just read the 'TEXT' part, new apps read both parts. *{ We should define some standards for this }*

### Cut Subscriber

When cutting a selection that includes `Subscribers`, the application must move the `SubscriptionRecords` into the clipboard and unregister them. There are two ways to do this. The first is to `UnRegister` each `Subscriber` and put the handles to them into the clipboard data structure. The second is to `BlockMove` the contents of each `SubscriptionRecord` into the clipboard data structure and then



DisposeSubscriber them. In either case you should never have a registered Subscriber in the clipboard. Note: this should be undo-able.

### Copy Subscriber

Copying a selection that includes Subscribers is very similar to cutting. The difference is that the data is copied, instead of moved, into the clipboard. Like Cut, there are two methods. One is to clone (HandToHand) the SubscriptionRecord and put the clone into the clipboard. The other is to BlockMove the SubscriptionRecord into the clipboard. Note: this should be undo-able.

### Paste Subscriber

If all the above rules were followed when cutting and copying, then pasting is simple. If you are using the handle in the clipboard method, just clone the SubscriptionRecord (HandToHand) and then RegisterSubscriber it. If you are using the SubscriptionRecord data in the clipboard, then create a handle of the correct size and BlockMove the data into it and RegisterSubscriber it. Note: this should be undo-able.

### Zeroing your private Scrap

Since Cut and Copy start by clearing your private scrap and both need to be undo-able, the clearing should really just move the contents into your "undo" buffer. When you eventually clear your undo buffer, you should only have to dispose of the handles to SubscriptionRecords – they should already be unregistered.



## 4.6 Derived Data Application Models

There are two ideas which are central to the Diet Coke user model: The first is that each Publication is "The Source" of some data. The second is that the user may go to The Source (i.e. the Publication), change the data, and then save the changes. This integrates well with Macintosh applications which do not introduce "derived data". Publications are then just like documents. A piece of text or a picture can be edited in a Publication and subscribed to in documents with no ambiguity or loss of generality. On the other hand, applications that do use and manipulate "derived data" need a little more thought.

First, the definition of "derived data": Data in a document is "derived" if the user cannot directly edit the data, but must change it indirectly from some other place. Some examples: A cell in a spreadsheet that contains a formula is derived data because the cell's value can only change as the result of changing the values in other cells. An instance of a header or footer in a word processor is derived data because in order to change its contents, you must "go to" a special window to do so. The result of a database query is derived data since you can only change the query result by changing the actual records in the database (and running the query again).

A slight problem occurs with derived data in Publications when the source of the derived data resides outside the Publication. This situation runs counter to the two central ideas of the Diet Coke model. First, since the source of the derived data resides outside the Publication, then the Publication is no longer "The Source". And second, the user can no longer go to the Publication and make changes directly; he must go outside the Publication to the source of the derived data. (Note that if the source of the derived data resides inside the Publication then neither of the two central ideas is compromised.)

The situation is not hopeless. If we review the Diet Coke user model we see that users expect that: 1) issuing a Save on a Publication will cause its Subscribers to get the new data, and 2) opening a Publication will allow them to make changes. It's possible for applications to adhere to the model to the point where the model is not corrupted in the user's mind, even when a Publication contains derived data whose source resides outside the Publication.

To meet the user's first expectation, stick to the rule that a Publication must be opened and saved for a new edition to be created. This reinforces the idea that the Publication is "The Source" (even though it is not) because the user still has control over when new editions are created. Do not implement a "back door" mechanism for silently creating new Publication editions based on derived data



sources changing in some other place.

In the spreadsheet example, where a Publication contains a formula that references a cell in another spreadsheet, the user must open the Publication in order to re-calculate the formula. This is already well-understood by spreadsheet users. But, a spreadsheet application that somehow allows values to be "pushed" into a spreadsheet Publication via a "back door" is not acceptable. A user opening a Publication would be unable to meaningfully edit the Publication formula or understand how the data gets into the Publication.

Note that the requirement to open and save a Publication in order to change it does not preclude the proverbial "stock server" application. For instance, the application could be designed so that a Publication looks like a table of stock names and prices. Once opened, the user could configure the Publication to be "auto-saved" for the remainder of the session. This would periodically save the Publication to allow Subscribers to get the latest stock prices. Additionally, the user could "Set Aside" the stock server application to hide its windows and let it run in the background (Set Aside is a forthcoming MultiFinder feature that allows you to make all of the windows invisible for any running application).

The second user expectation is that issuing the Revise Publication command will take him to "The Source" where he may change the data. If the Publication contains data which is derived from external sources, then the user does not get what he expects. However, if the Publication maintains user-visible references to the external sources, then at least the user will not be totally lost. In the spreadsheet example, the formula in the cell within the Publication should contain user-visible references to the other spreadsheets. In the database query example, the Publication should display the actual query. Thus, when the user "goes to" the Publication, he gets a pointer to the source of the data rather than the data itself. Within the context of the application, the user has an idea of how to affect a change to the Publication.

In summary, Publications should be as self-contained as possible. However, in the unfortunate case where a Publication contains externally derived data, the application should perpetuate the user model as much as possible. First, new editions should be created only as a result of saving the Publication - this reinforces the notion of the Publication as "The Source". Second, if a Publication has external data sources, the "link" between them should be stored only in the Publication. These "links" should be displayed as user-visible references. That way when the user goes to the Publication, he at least has an idea of how to change the Publication data. It should appear to the user that the Publication is the source, except that it has chosen to get its data from something other than user input.

57



# Glass Plus ERS

Ed Tecot x4-5418  
and  
Kevin MacDonell x4-0817

Blue Book Draft  
Friday, January 13, 1989

“Cleans Windows Plus A Whole Lot More”



57

# Introduction

This document is a first draft and should be taken with a grain of salt. All of the ideas herein are not only subject to change, they are expected to. The purpose of this draft is to outline our pre-mises and direction for public<sup>1</sup> review.

## Rationale

Macintosh CPUs are getting more and more powerful, and with Big Bang features, applications are getting richer and richer, and also, potentially more complex for the average user. Our user interface has not evolved much since its inception, and although such stability is a good sign of its quality, it has clearly reached some limits now, since many applications have started abusing it.

Large and multiple monitors are becoming more readily available and will soon become popular among many of our customers. These devices point out some of the flaws in our menu interface, most notably that a user must navigate a large area in order to make a menu selection.<sup>2</sup> It will behoove us to provide a solution, lest third parties develop their own, possibly inconsistent interfaces.

Adding features to Macintosh has had its consequences. Writing applications is now more difficult than ever before. The number of traps has mushroomed, the number of data structures has swelled to unprecedented levels. Glass Plus can hopefully reduce these numbers for future application developers. ~~Not everyone can wait for Pink.~~

Multitasking in the Blue environment will not be able to progress much further unless the windowing environment permits it. Currently, the MultiFinder cannot update while a menu is down, this *must* be fixed.

Also, we can take advantage of this opportunity to solve several similar problems with a uniform approach. A/UX has layer management needs which may also be solved by Glass Plus. In addition, several system areas can take advantage of a general layering approach.

Finally, new and interesting video hardware is starting to become a reality. Preparing the system for these things would be best done while similar code is being worked on.

## Goals

Glass Plus must live up to its rationale. Each of the five parts are listed below, with ideas to consider indented beneath.

- Give user control of menus.
  - Tear-off menus
  - Create and customize new menus from existing ones.
  - Multiple menu bars
- Provide useful features to application developers.
  - Floating windows
  - Promote self-consistent model of windows, layers, and menus.
  - Palette menus
  - Double buffering

---

1. This document is Apple Computer Confidential.

2. Assume the item does not have a key equivalent.

- Promote multitasking
  - Don't halt during menu selection.
  - Finer granularity.
- Simplify models for Apple system developers
  - Uniform Palette Manager
  - System Windows (Scripts, Handicapped Utilities)
  - A/UX Layers
- Support future video hardware
  - Hardware windows (Ground Zero)
  - Graphics accelerators (Monet)
  - Video overlay (Dagwood)

57

## User Controlled Menus

There are several problems with our current paradigm, most notably its inflexibility from the user's perspective. MacroMaker and QuicKeys attempt to address these problems, but suffer from some rigidity on their own part. It is only fair to note that these utilities are geared towards a different class of problems and only solve some of the menu issues by accident. Most of the problems deal with location and arrangement; the user really wants to customize his menus to suit a particular task, placing his tools near where they are likely to be used. It is important to note that Macintosh menus have one very important trait: Nothing is hidden from the user.<sup>3</sup> This must be preserved at all costs. I have three suggestions. None are any good. A significant human interface investigation in this area must begin immediately if Glass Plus is to succeed.

### Tear-off Menu

Additions to the Macintosh User Interface should only be made when they clearly benefit the user — thus making the Macintosh easier to use. It is clear that tear off menus do indeed fit this criteria. In general practice, a torn off menu is easier and more accurate to select than pulling down a menu in the menu bar. It is with this rationale that tear off menus are proposed as an enhancement to the Macintosh User Interface. [Tognazzini 1988] provides a basis for these ideas.

Upon deciding that tear off menus significantly enhance the Macintosh User Interface, the following plan is offered in support of tear off menus. The first issue to face is that all menus were created equal - no menu is more equal than another.<sup>4</sup>

*Menu Postulate 1:* All menus must be able to be torn off, and the method by which this is accomplished must be the same for all menus.

This means that there cannot be menus which are inherently made to be torn off, while others require some special operation to tear them off (i.e. HyperCard). We must preserve the drag-out-to-cancel method of canceling a menu selection, but we need to add a method of tearing off a menu. Several alternatives are being considered. In the first situation, tearing off can be accomplished by allowing the user to consciously change from "menu selection or cancel" mode to "tear off this menu" mode.

This mode will be selected by holding down the command-key while pulling down a menu. That is, when holding this key, menu selection is disabled and tearing off is the mode - releasing the command key while mouse is still down returns to selection mode. Dragging in a menu with the command key down pulls down the menu as usual, but none of the items in the menu become black as the pointer moves over them. It also creates a gray outline of the menu to appear and move with the pointer. Releasing the mouse with the command key held down creates the tear off menu.

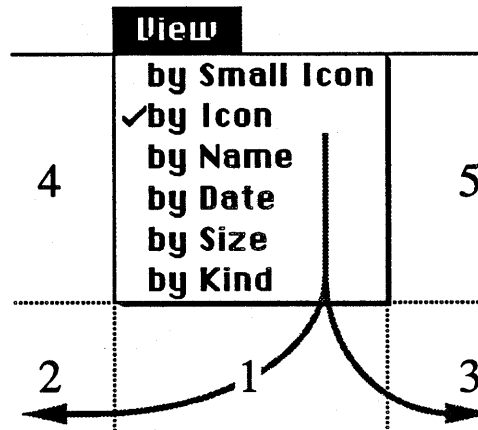
A more sophisticated method of tearing off menus might be accomplished via a tear off gesture rather than holding the Command key down. This gesture would be the only way to tear off a menu, other gestures in the menu would still allow the user to drag out and cancel. The following diagram illustrates the gesture:

---

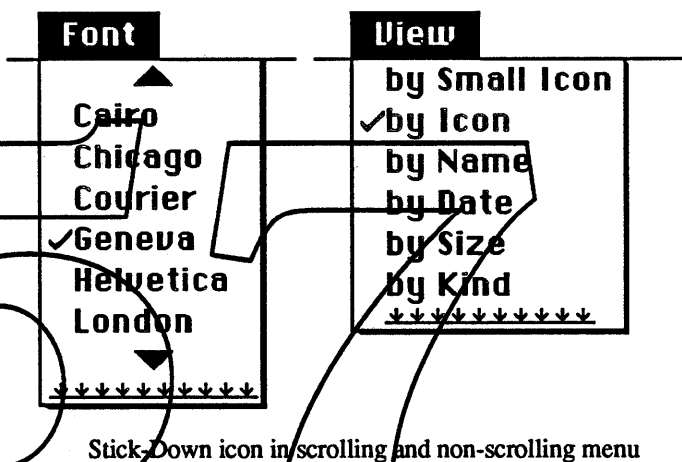
3. Hierarchical menus were just a bad dream; if you pinch yourself, they'll go away.

4. Apologies to Orwell.

By clicking on the menu item, dragging down through area 1 and releasing in areas 2 or 3, we have provided a metaphor for tearing off (eg. tearing off a paper towel from a roll). Releasing in areas 1, 4, or 5 after leaving the menu in area 1 does not create a tear off menu, it yields a canceling gesture. So does leaving the menu area through areas 4 and 5, regardless of where the mouse is released. Radius Inc. uses a gesture which requires that you drag out through areas 4 or 5 to tear off a menu. User testing will be the best way to determine a correct (or best) gesture for this. In any event, the ability to tear off any menu without any key sequences will be appreciated by power users and should simplify the operation for novices and handicapped users.

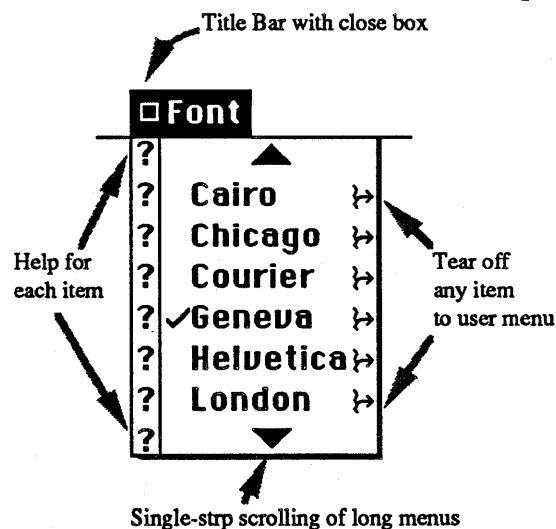


Another solution to the tear off menu issue is Stick-Down Menus. Stick-Down Menus are a *non-modal* method of allowing the user to pull down a menu and keep it down after the mouse is released. The user can now spend much more time considering the items in a menu and to make a selection without the stress of holding down the mouse button (especially significant for handicapped users). It also allows us to provide many more options within a menu than were previously possible.



To stick down a menu, a new item in the menu is added. This item is a small downward arrow with a bar at the bottom, and when selected, it flashes like a menu item and then menu does not retract. Once the menu is stuck down, it changes to reveal a number of user controls which perform various functions. The figure below describes some of these functions.

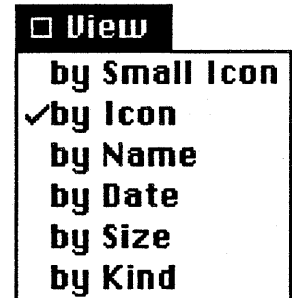
By clicking and dragging from the Title bar, the menu can be torn off. By clicking in the close box, the menu is retracted (un-stuck). Clicking on the question mark next to each item gets help for that item. Clicking on an item and releasing performs the normal function of that menu, but does not retract the menu. Clicking on an item and dragging out through the tear off icon will tear off that item and allow it to be placed in the user menu. In the case of a scrolling menu, each click on the up or down arrow advances the item list by one in that direction. When the last item in the menu is displayed for that direction, that arrow is replaced with the last item. Holding the mouse down on the arrow will auto-scroll.



## Behavior of Tear Off Menus

The behavior of tear off menus is a well defined hybrid of the functioning of windows and menus. These are defined below and the diagram is an example of a torn off menu:

- A torn off menu is contained in a small, special purpose window (windoid).
- Only one windoid should exist for a given menu at any one time.
- A windoid has a close box to allow the windoid to be disposed.
- The windoid has a title bar. The title in the title bar relates the tear off to its menu equivalent. The title bar provides an area to drag the windoid to a convenient position on the screen.



- Tear off menus should always "float" on top of all other application defined windows, in the same way that pull-down menus always pull down on top of other windows. If multiple windoids are displayed at the same time (2 or more menus torn off), they should have their own front-to-back ordering as regular windows do while still "floating" above all regular windows.
- Tearing off a menu for which a menu windoid already exists should dispose of the previous windoid and create a new one (helps to find lost windoids).
- Menu windoids should disappear when an application switch is performed in MultiFinder, just as that application's menu disappeared.
- Item status in the menu should always reflect status in windoid. Checking an item in the menu checks it in the windoid and vice versa. Likewise for graying an item, etc.
- Items deleted from a menu should be deleted from the menu windoid and the windoid should be resized accordingly.
- Tear off menus should be persistent (see discussion below).

## Persistence Of Menus

How persistent should a tear off menu be? An good example of this is demonstrated by the application SuperPaint, which combines many of the capabilities of MacDraw and MacPaint in one application. SuperPaint has a menu in the middle of the menu bar called "Draw" while the user is in drawing mode, and "Paint" while the user is in painting mode. This is quite useful in the context of the program. The nature of this program is to continually switch back and forth between these modes, thus switching the menu displayed. If the user was in Draw mode, the Draw menu could be torn off. Then, the user could switch to Paint mode, thus removing the Draw menu from the menu bar, and, by the rules given above, the torn off Draw menu windoid should disappear. The Paint menu would now appear in the menu bar.

If, however, the user switches back to Draw mode, should the previously torn off Draw menu windoid reappear (Persistence case 1)? If the user quits and the Draw menu is torn off, should it automatically reappear the next time the application is run (Persistence case 2)? If the user had torn off the Draw menu, but switched to Paint mode (thus making the Draw windoid disappear), and then quit, should the Draw menu windoid reappear when the application is run again and switched to Draw mode (Persistence case 3)?

Menu windoids should do what you expect them to do, and you expect them to mirror the actions of the menu it was created from. You expect the Draw menu to return when you change back to Draw mode, you expect the Draw menu to be there when you quit and then run the application again, and you expect that even though you were in Paint mode when you quit, that you still can get to Draw mode upon running the application again. You expect the tear off windoids to behave in this same way. Therefore, menu windoids should be consistent with all the cases of persistence outlined above.

Who should be responsible for maintaining this persistence? We believe that Glass Plus should automatically provide for Case 1 above (session persistence). Apple should then publish guidelines for developers to provide for Case 3 (separate invocation persistence), since this information will need to be written to a preferences-type file. This indicates the need for additional API to allow applications to determine which menus are torn off and what coordinates (global) are required to restore them later. A new resource format could be defined to tell the Menu Manager which menus are to be torn off at launch and InitMenus could automatically perform those operations. Other conveniences are also possible (see section on Application Developer Features).

### Custom Menus

As the complexity of applications available for the Macintosh increases, so do the number of menu selections available. While this is not an excuse for poor organization and design, it does point toward a need to let ~~the user have increasingly more control of the organization of his environment.~~ It may be that there is no clearly superior organization of menus and items which a developer can determine beforehand. In an effort to ~~add this problem, we can provide a method of creating user-defined menus.~~ These menus would be a user-defined collection the application menu items (a copy of the item, ~~not moving the application original~~) placed in a new user menu, usually created to be torn off. This presents a number of interface, context, and implementation issues to be resolved.

The ultimate approach is to allow the user to customize his menu environment in any way he desires. Lee Honigberg alluded to this with customizable tools for compound documents. In his model, new tool palettes could be created by copying the best of all the other palettes. The user would "pop the hood" off of any palette that he wished to modify. If you substitute item for tool and menu for palette, a similar approach might work for Macintosh. In order to preserve the original state, I might recommend that the original menus can only be added to.<sup>5</sup> In addition, perhaps some of this might be combined with MacroMaker, so that a user could put a macro in any menu that he desired. This would have the added benefit of getting rid of the macro menu that has caused so many problems in the past.<sup>6</sup>

Another direct manipulation option suggested by Bruce Tognazzini for customizing menus is even simpler than the one suggested by Honigberg. Holding the Command key while selecting a menu disables menu selection and enables tearing off that menu. Holding the Command-Shift keys would tear off a menu item when the user dragged out to the left or right of that item's rectangle.

---

5. Like in the MPW Shell.

6. Sorry, Donn.

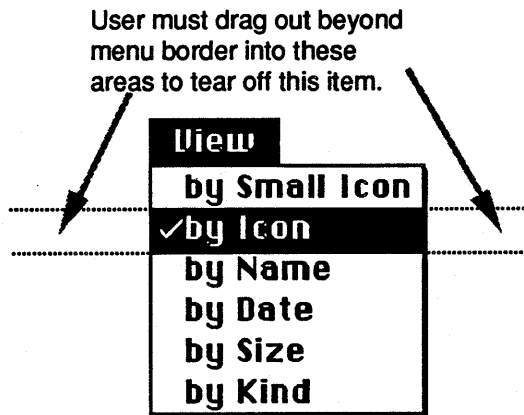


Fig. 1 Item tear off regions

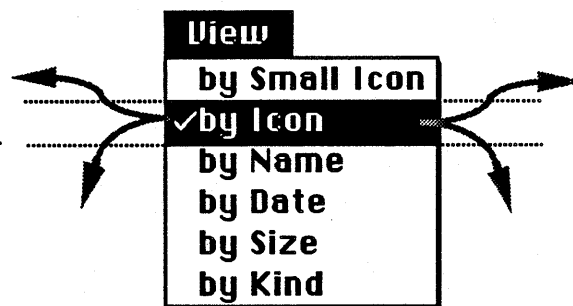


Fig. 2 Possible drag to tear off paths

Once the mouse is outside the menu, the menu retracts. This process has gotten the menu item out of its existing menu, but now that we have it, what should we do with it? The answer is we create a user menu in which to put the item if it has not yet been created. By moving out of the menu and tearing off the item, we create a new user menu (eg. ~~LS~~ menu) in the menu bar. By dragging the torn-off item over the user menu title (which would highlight to indicate proper mouse positioning) and releasing the mouse, that menu item is "dropped" into that menu. Releasing the mouse anywhere else will cause the menu item (really the gray outline) to "jump back" into the menu that it came from. That item will appear at the bottom of the menu. Sorting the menu items can be accomplished by tearing off items from the user menu and dropping them back into the same menu, thus moving that item to the bottom. If the user menu is already torn off, dragging to the title bar of the user window will invert the title bar, and allow the user to place the item in the menu via the window.

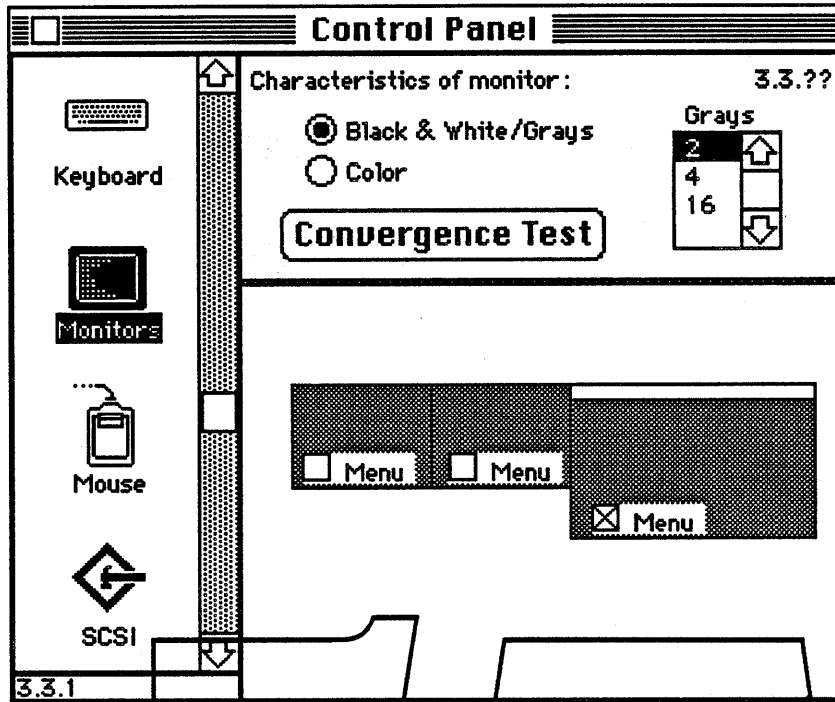
### Multiple Menu Bars

With the advent of multiple monitor systems, the Macintosh User Interface needs to address issues regarding users of these systems. *This discussion is ONLY relevant for multiple monitor environments.* The current scheme of menu selection requires the user to navigate through a potentially abstract path across monitor boundaries to find the monitor which has the menu bar, and then make a menu selection. This is cumbersome at best. Given various multiple monitor configurations, we need to provide ways to allow the user to place a copy of the current menu bar on any or all of his monitors. The familiar action of quickly pushing the pointer to the top of the screen to get at the menu bar is something that multiple monitor users should have at their disposal.

We should provide a method to add and remove menu bars from any or all monitors, thus making the path to the menu bar the familiar shove to the top of the screen, regardless of the monitor containing the cursor. At least two possible options exist. The first is similar to the current method of choosing the startup monitor in the Control Panel, the second is a direct manipulation approach. The choice of the direct manipulation approach requires some further investigation as to its feasibility.

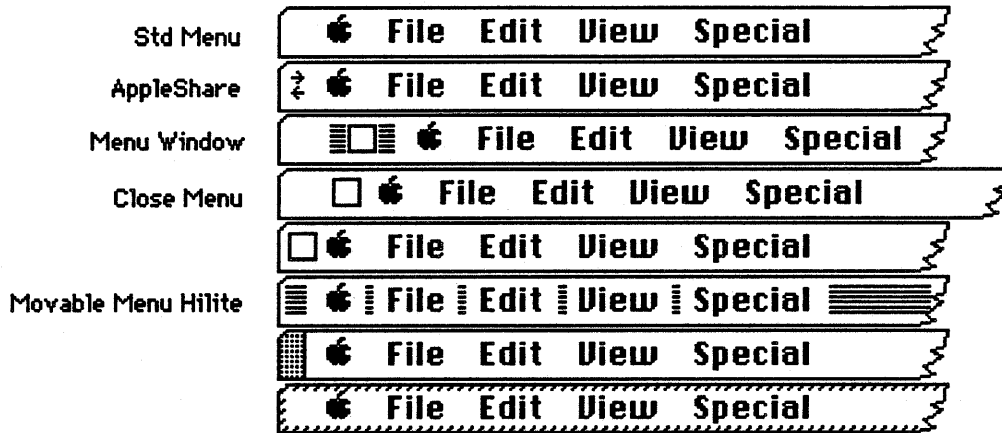
*The Control Panel Option:* This involves a modification to the Monitors 'cdev' to allow the user to specify which monitor(s) will have menu bars (check-box with each monitor). At least one monitor must have a menu bar. This would take effect immediately, if possible, otherwise it would take effect when the system is next restarted.





NOTE: This is an extremely simplistic example, see Jim Hull's new Monitors 'cdev' document for an elaborate description of what monitors will look like in the future.

*The Direct Manipulation Option:* If it is possible to move the main menu bar “on-the-fly” within the current constraints of the world, it would be preferable to move or duplicate the menu bar directly — rather than going to the Control Panel and rebooting. To do this however, we must provide a draggable region in an already crowded menu bar, or provide a key combination-click to allow menu bar dragging (thus disabling menu selection), and option-key combination-click-drag (ala MacPaint) to duplicate and drag. Upon releasing the menu bar on a monitor, it must “jump” to the top of the monitor. This may completely cover the title bar of a window and prevent it from being moved or closed without a window movement menu option (tile/stack windows). To remove a menu bar from a monitor, we must again steal space from the menu bar for a close-box, or do some more fancy keyboard work. All of these modifications to the look of the menu bar (with the exception of the first two) seem to distract the look-and-feel of the familiar menu bar.



[Vertelney 1988] suggests duplicating the menu bar if it is dragged to another monitor, and deleting a copy of the menu bar if it is dragged to another monitor that already has a menu bar. She also suggests allowing the user to adjust the location of titles within the menu bar.

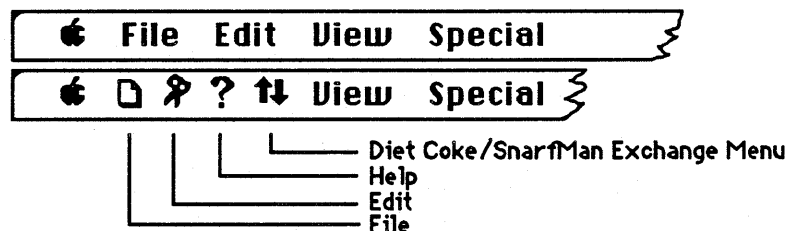
Another option open for investigation is to collapse the menu bar into a vertical fashion, more like a tear-off menu.

The Control Panel Monitors method of putting the menu bar on the startup monitor is something multiple monitor users are already familiar with, so the learning curve should be minimal. Moving menu bars via direct manipulation may put the user into a very precarious and unclear state. *For this reason direct manipulation should be considered, and then discarded.*

#### Other Menu Considerations

On very large screen monitors, Chicago 12 point menu bars get fairly small. Add the ability to change the font size to other larger sizes (Radius TPI uses 16 point). Applications which assumed the menu bar height was 20 pixels will have some problems, but everyone who did it right (i.e. used `mBarHeight`) will be O.K. Changing the font of the menu bar would not be allowed since it might disorient the next user (can you see San Francisco font in the menu bar? I would think the Mac was in serious need of repair!).

There is a need to add a help menu to the menu bar as a standard menu as the Apple, File, and Edit menus are now. This poses a serious constraint to the number and size of application menus available. The use of well-defined universally understood (i.e. the Apple) small icons instead of textual titles would provide two benefits. First, it would allow us to add the help menu into the same space that File and Edit now occupy — three menus for the price of two. Second, it would alleviate the need to translate these menu titles for international versions. Third, it would allow developers to use icons instead of text for some menu titles when appropriate.



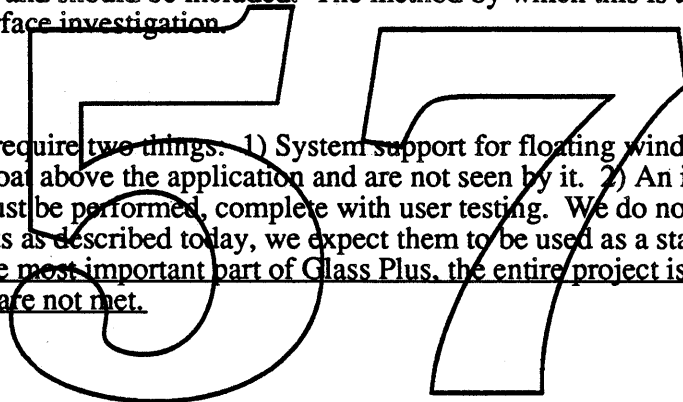
The current Finder menu and an example of a new menu with icons instead of text. Note how 4 new items fit in same space as 2 previously did.

Power keyboard users must continually take a hand off the keyboard to make a menu selection when command key equivalents are not provided for that item. However, not all menu items have command key equivalents, and when many items do, it becomes hard to remember them. It would be nice to be able to select *any* item in a menu via the keyboard. FullWrite Professional™ by Ashton-Tate Corporation has a unique method of allowing the selection of any menu item purely from the keyboard with a minimum of keystrokes. By typing Command-1, the first (File) menu is pulled down and held down and the first item in that menu is initially hilighted— but not selected. Command-period cancels any menu selection and retracts the menu. By typing the first few (yet only enough to be unique) letters of a menu item, that item is hilighted. Hitting Return or Enter will select that item and the menu retracts. This can be done very quickly by good keyboard users. As an added benefit, command key equivalents can be used, but if you forget them, you can always type the first few characters of the menu item name and hit return. A couple of drawbacks exist. You have to remember which menu number your item is in or you have to count the menus to determine which number to hit, you can only have 10 menus (assuming letters are used as command keys), and if the application already uses a command key equivalent using numbers instead of letters, it could be confusing or break the application.

We believe that allowing any menu item to be selected via the keyboard is a significant enhancement to the user interface and should be included. The method by which this is accomplished is again subject to user interface investigation.

#### Requirements

All of these components require two things. 1) System support for floating windows is needed; that is, windows which float above the application and are not seen by it. 2) An in-depth human interface investigation must be performed, complete with user testing. We do not suggest implementing these components as described today, we expect them to be used as a starting point for better ideas. As the single most important part of Glass Plus, the entire project is doomed to failure if these requirements are not met.



# Application Developer Features

Programming the Macintosh is too hard. Glass Plus could make it even more difficult unless we take specific steps to prevent this. Application developers want more features that *they* can use. As the self-appointed interface police, we want to make the best choices the path of least resistance. Developers should still be free to write their applications as they wish, but interface components which conform to Macintosh should be simple, even trivial, while significant diversions would require more effort. MacApp is a step in the right direction. But even MacApp must jump through hoops to achieve the proper result. These things should be in the Toolbox, not part of the application.

## Floating Windows

Application developers would like to provide windows which float above document windows, usually to provide tools which are accessible at all times. A more compelling reason is the fact that most menu manager changes will require it.

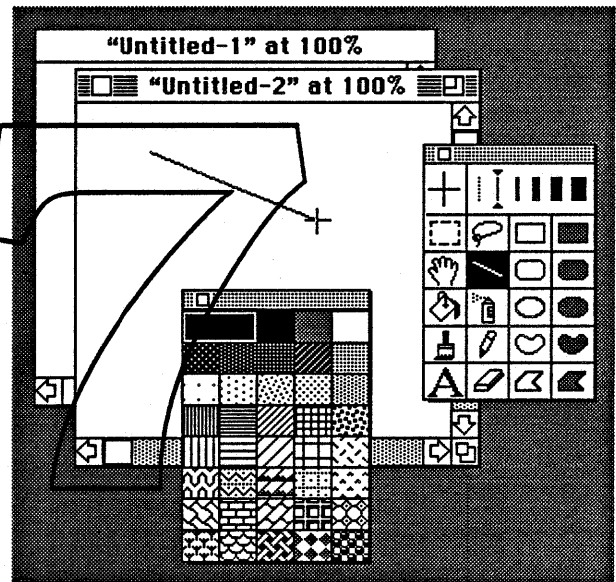
MacPaint™ demonstrates a good use of floating windows. Actually, these are tear-off menus, but there is no reason why they *must* be.

[Tognazzini 1988] proposes some guidelines for floating windows (referred to as windoids):

“Windoids have a close box, a visual indication of draggability, and an optional title. A visual indication of dragging can be a title bar-like region, a pointer change, or?”

“The last windoid clicked or opened is topmost. Other than that there is no implicit ordering of windoids.”

“Windoids hide when their application is switched out by MultiFinder, and reappear when their application is made frontmost.”



I think we should drop the whole notion of “windoids” and stick to floating windows. In particular, floating windows behave exactly like real windows except that they cannot be obscured by windows which do not float. It is important to note that unlike windoids, floating windows do not disappear when an application is switched unless the application wishes them to.

Glass Plus will implement floating windows by placing them in their own layer. Users are already familiar with layers; MultiFinder uses them liberally in order to group application objects together. An application which wanted to create a tool palette like MacPaint above would first create a new layer and then subsequently create the tool window in that layer.

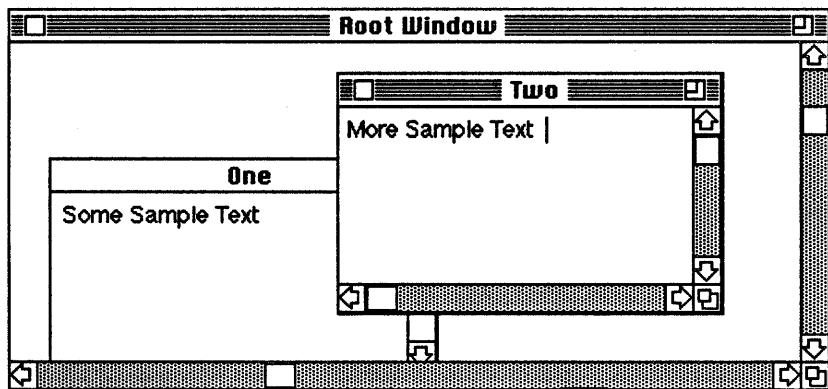
## Menus, Windows, Layers

Instead of treating menus, windows, and layers, as different objects, Glass Plus will treat them all in a similar fashion by introducing a new concept - hierarchical windows<sup>7</sup>. Hierarchical windows allow windows to “own” subordinate windows. Layers are hierarchical windows which have no frame and as such cannot be manipulated. Menus are windows which are created, re-

7. Windows cannot be hierarchical per se. Hierarchical windows are named such because they belong to a hierarchy, just like files under HFS do. A better name for these windows would be graciously accepted.

spond to mouse and key events, and destroyed automatically.

The basis of this is the *Hierarchical Window Manager*. The important notion here is that every window, except the root, which is owned by the window manager, has a parent. Every child window is subject to the whims of its parent. That is, each child is clipped to be drawn within its parent, and is moved when its parent is moved.



Here is the simplest example of a parent window and two children. Note that the children have a definite order and that they are clipped to their parent.

Layers are hierarchical windows whose "shape" depends upon its contents. A layer has no frame, and thus, cannot normally be moved or resized. Layers are used to group and divide other windows, as in MultiFinder, or to create floating windows as described above.

Hierarchical windows are distinguished from normal windows by their windowKind field.

```
CONST HWindowKind = 0
```

Layers are distinguished by their definition function.

```
CONST LayerProc = 32
```

This is the structure of a hierarchical window. Since a hierarchical window descends from a regular window, the beginning of the record is the same. subWinList is a pointer to the children of the window. grayRgn is the portion of the window's content region that is visible - it is analogous to the visRgn in today's windows. The visRgn is the grayRgn with all of the children clipped out. This is so that drawing in a window does not draw on the children.

```
TYPE HWinPtr=WindowPtr;  
TYPE HWinPeek=^HWinRecord;  
TYPE HWinRecord =RECORD  
    window: WindowRecord;  
    subWinList: HWinPeek;  
    grayRgn: RgnHandle;  
END;
```

The interface to the Hierarchical Window Manager is necessarily simple. Only three calls are required, all other window manager routines work appropriately if passed a hierarchical window; searching for children if necessary.

```
FUNCTION NewHWindow(wStorage: Ptr; boundsRect: Rect; title:  
    Str255; visible: BOOLEAN; procID: INTEGER;  
    behind: WindowPtr; goAwayFlag: BOOLEAN;  
    refCon: LONGINT; subWinList: HWinPeek) :  
    HWinPtr;
```

Creates a new hierarchical window.

For example, to create a layer to be used by floating windows, the call would be:

```
FloatLayer := NewHWindow(NIL, emptyRect, NIL, TRUE, LayerProc, WindowPtr(-1), FALSE, 0, NIL);
```

```
PROCEDURE SetCurHWindow(theHWindow: HWinPtr);
```

```
FUNCTION GetCurHwindow() : HWinPtr;
```

These two calls set and return the *current hierarchical window*. Any subsequent call to any window manager routine which affects a window list will be applied to this hierarchical window. When an application is launched, this will be set to the appropriate layer. Hence, any application which does not use hierarchical windows will work the same as before.

For example, to create a floating window, you simply set the current hierarchical window to be the floating window layer before creating it, namely:

```
SetCurHWindow(FloatLayer);
```

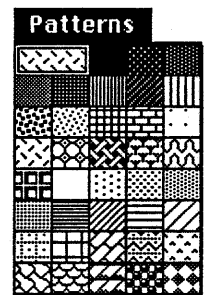
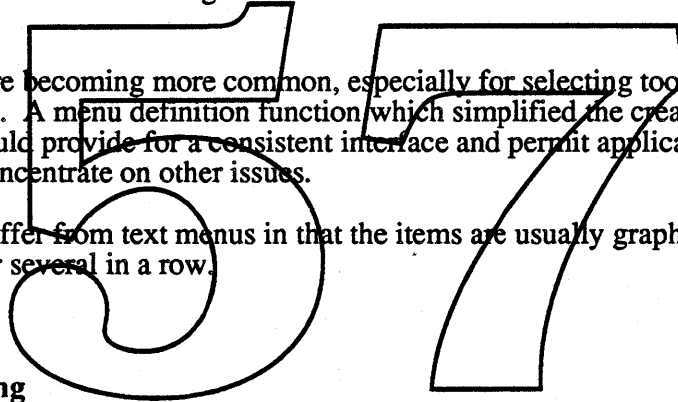
```
toolsWindow = GetNewWindow(toolsID, NIL, WindowPtr(-1));
```

Menus, instead of working around the window manager, will work with it. When a menu is displayed, a window will be created for it. The frame will be maintained by the window definition function, instead of being drawn by the menu manager. Instead of saving bits behind the menu and restoring them when the menu disappears, the menu manager will request this service from the window manager. This will put the window manager in charge of the tasks where it is best suited, and leave the menu manager to deal with the menus themselves.

### Palette Menus

Palette menus are becoming more common, especially for selecting tools, patterns, and colors. A menu definition function which simplified the creation of these menus would provide for a consistent interface and permit application developers to concentrate on other issues.

Palette menus differ from text menus in that the items are usually graphical, and often appear several in a row.



### Double Buffering

The Glass Plus menu manager requires the window manager to be able to save the bits behind a window and restore them when the window is destroyed or hidden. However, if a window is obscured by one of these windows, and someone draws into the obscured window, when the bits are restored, the data will not be correct. Instead, the window will be restored to the state it was when the front window was created.

A solution to this is double buffering. When a window is created, it can optionally request that the window manager buffer all drawing commands, either as bitmaps or pictures. When the window would normally require an update, such as when a part of the window is revealed, or it is resized, or moved to a different depth screen, the buffer will be used to update the area instead.

In addition to being double buffered itself, a window can double buffer on behalf of windows that it obscures. This way, menus could bestow the benefits of double buffering on other windows without requiring that all other windows be double buffered.

# Multitasking

MultiFinder is great, but some things are missing. Why does all background activity stop when a menu is down? Why does PrintMonitor slow everything down so much? One solution to these rhetorical questions is a preemptive scheduler. However, preemptive scheduling is not going to be enough. There are fundamental problems in the ToolBox that need to be corrected in order to obtain an elegant solution.

## Menu Selection

Currently, menus stop all background activity. MultiFinder can help by giving background tasks time during StillDown(), but then other tasks could draw on top of the menu. Here is where some of the previous ideas become a big win. By making menus use the window manager, menus can obscure other tasks and everything will still work correctly. Double buffering produces an even better solution in this environment.

## Finer Granularity

Currently, context switching only occurs at Get/WaitNextEvent. A preemptive scheduler increases the granularity within applications, however, the Toolbox is hardly reentrant. In particular, any part of the Toolbox that relies on the integrity of relocatable handles might not be reentrant. For example, the Font Manager might make a memory request which forces a compaction of the application heap. If it gets preempted, no windows can move, because that might require a recalculation of one of the visRgn's in the other task. In addition, if the Window Manager is calculating visRgn's, then no task can draw, because the visRgn's are in an indeterminate state.

One possibility is to classify the system as non-reentrant, and turn off preemption at every trap call. This seems a bit rude. Glass Plus needs to consider the implications of reentrancy and take some steps to allow a finer granularity of context switching.

## Requirements

Other areas of the Toolbox and OS need to be looked at with preemption and reentrancy issues in mind. Otherwise, such an effort within Glass Plus is simply wasted.

## Simpler Models

As hard as it is to write applications for Macintosh, it's even harder to write system code. Every change must consider the implications to thousands of applications across several hardware configurations. If we are to continue improving Macintosh, we must attempt to simplify what we have now. What Glass Plus makes easier for application developers will also become easier for us, but there are other areas to consider.

### Uniform Palette Manager

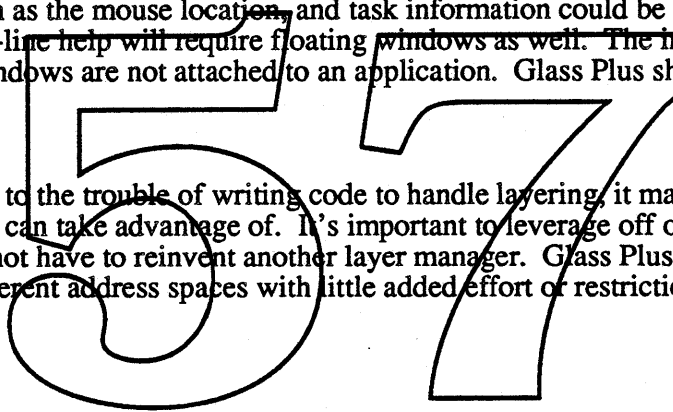
The Palette Manager attempts to do its job by getting control all over the place. What the Palette Manager really wants to do is act as a liaison between the application, the Color Manager, and the Window Manager. It should not have to have special hooks into MultiFinder in order to do its job. Glass Plus should be cognizant of these needs and provide for them where possible.

### System Windows

Not only do applications want floating windows, but other parts of the system want it too. I've discussed tear-off menus which are the impetus for this feature. However, the Script Manager can make use of this as well, providing for floating input method windows, such as Kanji. Even better utilities for our Handicapped users can use floating windows to enter text as well. Status information, such as the mouse location, and task information could be displayed for people who might use it. On-line help will require floating windows as well. The important thing is that these floating windows are not attached to an application. Glass Plus should be able to handle these cases.

### A/UX

Since I am going to the trouble of writing code to handle layering, it makes sense to do it in such a way that A/UX can take advantage of. It's important to leverage off of my efforts, so that the Unix team does not have to reinvent another layer manager. Glass Plus should be able to handle the notion of different address spaces with little added effort or restrictions.





## Future Hardware

The Macintosh of tomorrow will be very different from that of today. Glass Plus must not be dated. It is of no value of Apple if it becomes obsolete as soon as it is shipped. In particular, new video hardware is becoming available, and Glass Plus must address this.

### Hardware Windows

Cards such as Ground Zero allow windowing operations to be very fast and flexible. If Glass Plus can allocate the resources of these devices like the Palette Manager handles color lookup tables, it will be a big boon to its success. I see this problem as an extension of double buffering; it is the allocation that is the new problem.

### Graphics Accelerators

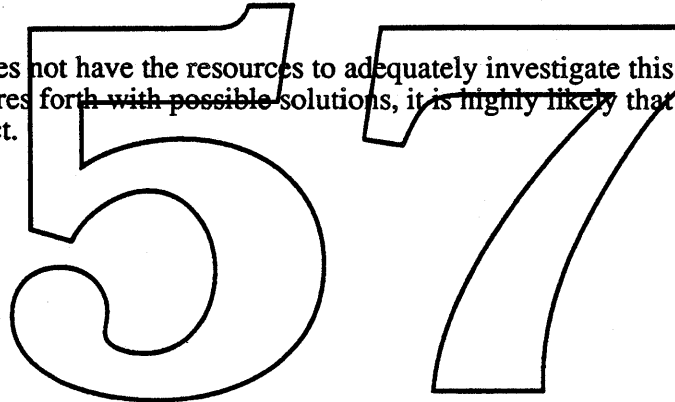
Graphics accelerators, such as Monet, will open up exciting new markets. I'm not sure if they impact Glass Plus, but it is important to find out.

### Video Overlay

Support for these cards present special problems. In addition to the resource allocation issues of Dagwood, video overlay devices need a fast path to the display.

### Requirements

The Glass Plus team does not have the resources to adequately investigate this area. Unless some other group ventures forth with possible solutions, it is highly likely that this will be dropped from the project.



## Bibliography

Tognazzini 1988

B. Tognazzini, A. Wagner; **Tear-Off Menus**, Human Interface Specification; January 20, 1988.

Vertelney 1988

L. Vertelney, K. Gomoll, B. Tognazzini; **Human Interface Design Recommendations for Multiple and Large Monitors**, Human Interface Specification; July 10, 1988

57

57

# SnarfMan

(a.k.a. Data Access Manager)

Thomas A. Ryan  
1/12/89

## Executive Summary/Product Abstract

Data Access: allowing a Macintosh user to retrieve and to store data on remote databases, is at the heart of any strategy that attempts to place Macintoshes in typical business environments.

SnarfMan provides an architecture for data access. It includes the role of application developers, data base tool builders, 3rd party database vendors, Network Innovations and, of course, system software's contribution. The SnarfMan project will implement:

- 1) a user interface to facilitate database access. The interface includes dialogs for session control, query execution and transaction control.
- 2) a simple interface for application developers. The interface is database and data language independent.
- 3) a database vendor interface that specifies a minimal capability set and allows vendors to distinguish their more salient features.

SnarfMan provides glue that enables every Macintosh application, even "non database" aware applications, to access database systems. It is a core technology that will distinguish Apple in the personal computer to mainframe access marketplace. SnarfMan utilizes an approach that allows applications, query languages, and physical databases to be gracefully decoupled and recombined in a novel ways. Furthermore, SnarfMan, in conjunction with Diet Coke (an inter application communication tool), will provide a dynamic database monitoring capability, in addition to the traditional downloaded snapshots.

## Introduction

On the one hand we have users running applications, on the other data base managers. The problem is to connect the two, with a Macintosh in the middle. SnarfMan solves this problem.

This document presents the SnarfMan architecture and the two main components that need to be implemented: SnarfBoy and SnarfTool. We begin by expanding on the problem areas that SnarfMan addresses, and then follow with a proposed design and a description of external interfaces. The last section contains a potpourri of issues such as dependencies, testing pitfalls and so on.

## Requirements / Problem Statement

The basic problem is to determine what data access software belongs in the Mac toolbox.. The underlying assumptions are that many applications will want to access remote data servers as well as local data files and that Apple has no unified framework or strategy to accomplish this.

It is a disservice to end users, tool builders and application developers when every brand of database has distinct sets human and programmatic interfaces for what are typically perceived as similar tasks. The challenge is to generate a solution that standardizes the user experience, has minimal impact on applications developers, yet still allows database vendors and data tool builders to distinguish themselves in the marketplace.

### Intended users

Three groups of users interact with SnarfMan each in different capacities. Firstly, humans will encounter SnarfMan when they wish to access data that is not native to the application. SnarfMan will present an end user with a list of canned queries, or the opportunity to formulate and ad-hoc query. Secondly, applications will communicate with SnarfMan via a small command set (API) that initiate SnarfMan data requests and receive result data. Lastly, database vendors will be expected to meet a minimal feature set and will provide database drivers that can be called by SnarfMan.

### Human Interface Requirements

Users should not have to learn to negotiate a different set of tools for every database that might exist. The requirement is for an intuitive database interface that is database manager independent. Ideally the location of the data, e.g. server, non-server, would also be transparent to the user. Data access should be ubiquitous: available from the finder as well as from within specific applications. It should not be necessary to run a separate data access application.

We want to encourage tool builders to develop innovative, novel, ad-hoc browsing capabilities tailored to specific business tasks (e.g. decision support). SnarfMan must not preclude their ability to do this.

### Application Program Interface Requirements

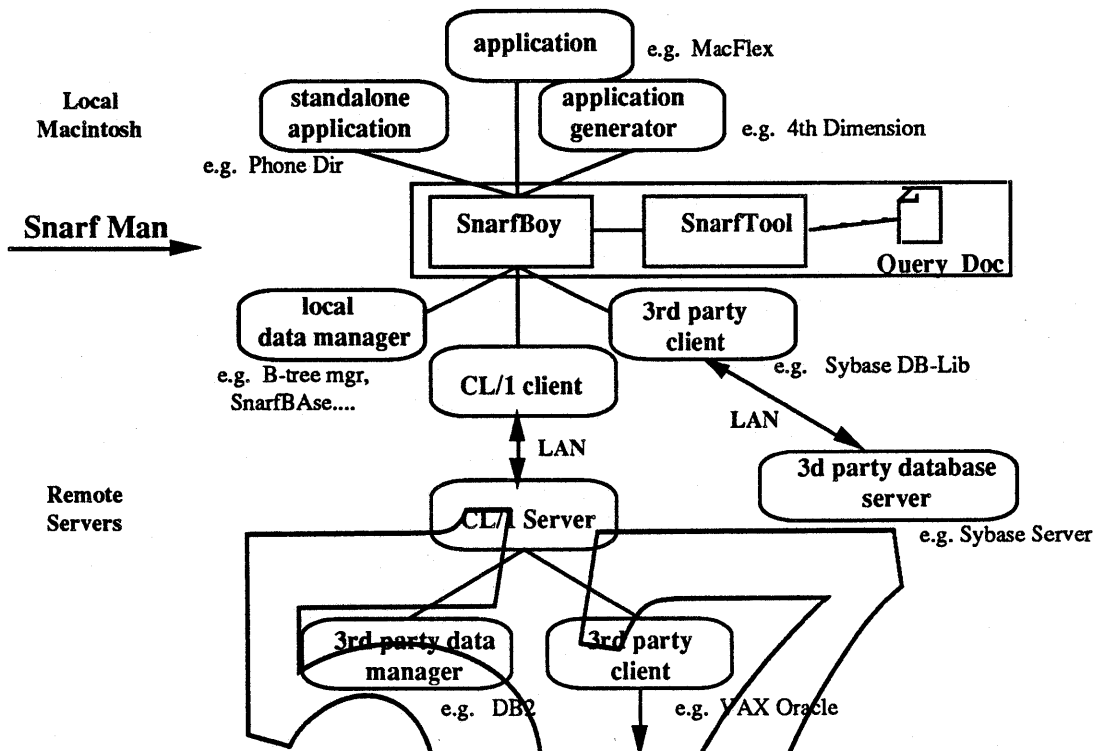
If we expect application developers to get excited about SnarfMan the application interface must be simple and attractive. Adding SnarfMan capabilities to an application must not be too traumatic. The interface must be database vendor independent since at application build time the suite of possible databases to access will typically be unknown. However, in those performance sensitive cases where the data paths are well defined, SnarfMan must provide the right length of rope.

### Database Vendor Interface Requirements

Database vendors must be able to "plug and play" in the Macintosh environment. We want to encourage them to build and support Macintosh clients to their database servers. Apple is not in position to impose a data language standard and it is unlikely that there will ever be an esperanto for data languages. Imposing the CL/1 API and language is an onerous burden to many database vendors, however, it is an attractive Apple solution in many other cases. Therefore the database specification must be standard enough to make SnarfMan feasible, must be compatible with CL/1, and must be flexible enough to allow database vendors to distinguish themselves from their competitors.

## Engineering Specification

The world fits together like this:



The key to the SnarfMan project is the SnarfMan monitor: SnarfBoy. SnarfBoy will field requests from applications, interact with the user, perform the required services, and return results to the application. Applications will interact with SnarfBoy through a programmatic interface. SnarfBoy will maintain the state of each open database connection and "in progress" data queries. Ideally SnarfBoy will be run asynchronously. At the bottom end SnarfBoy will call upon individual database drivers which are Mac resident database clients. These clients access specific database servers. Apple will provide a CL/1 client and it is expected that 3rd party database vendors will provide clients to access their own servers. Database drivers will be system documents and installed in the system folder. Note: it would be nice if they could all be collected into a folder within the system folder. The database drivers will be known as "DDEV"s.

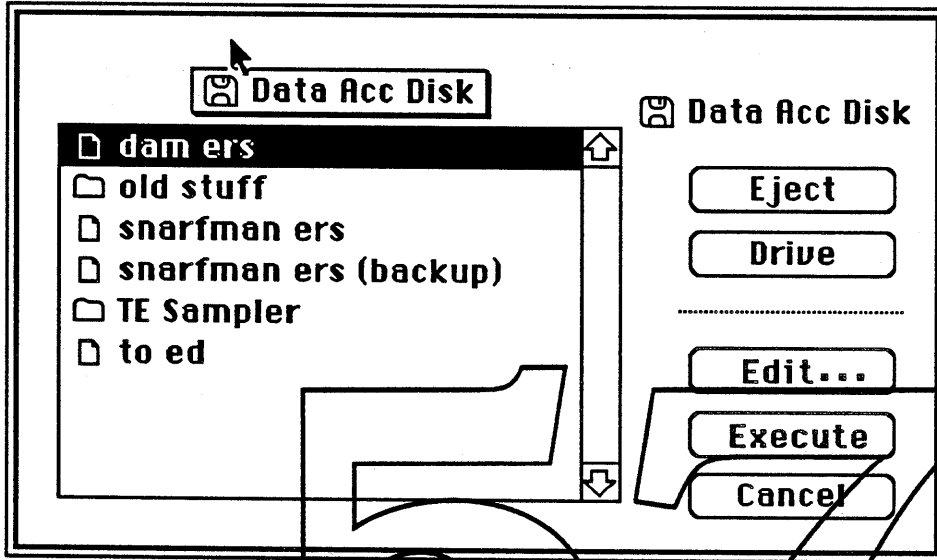
A typical request to SnarfBoy would be "Start\_Query". Assuming no other directions were given, SnarfBoy would present the user with a list of preformulated, "canned", queries, from which to choose. The list will be culled from Query Files that are external to the application and to SnarfBoy. Standard File will be used to find such query files. Data vendor specific commands are expected to appear in the query files. In the future SnarfBoy will be able to automatically translate between certain SQL dialects, but that is a longer term effort.

A standalone application, SnarfTool, will exist as the creator type of query files. SnarfTool will provide a thin layer above SnarfBoy and will store the results of queries into the clipboard or simple documents. It will serve as a sample application for developers to explore, as well as a test and debug tool. A hook to Diet Coke will allow query results to be stored as publications that SnarfBoy can automatically refresh.

Apple Confidential

## Human Interface

We expect applications, and the desktop (!?), to add a “data access ...” menu item under an existing menu header. Candidates include “File”, “Edit” and the proposed double arrow menu. There will also need to be a “Data Access ...” option available from Diet Coke’s publication definition sequence, or conversely, a “new Publication ...” command from SnarfTool. In either case, when SnarfBoy is activated from an application with the command “data access” the user might expect to see something like this:



Where presumably the files listed are query files, and others are grayed out or left out. If the user wishes to create a new query the short term solution is to open a simple text edit window and let them type the query into it, along with some other necessary parameters. Such a window would be available indirectly via “Edit...”. This is NOT a human friendly way to formulate queries. We will do better on Edit, stay tuned. A separate related project is investigating what a proper data query tool should look like.

Once a query has been selected SnarfBoy begins a query execution phase. If there are no current connections to the relevant database then the user may have to negotiate a login sequence. These sequences are vendor supplied and must be tailored to different database environments.

SnarfBoy will attempt to maintain the connection so that the user is not forced to repeat this trial for each query. Vendors are free to invent configuration files which eliminate user interaction but that approach is not currently recommended by SnarfMan.

The other access to SnarfBoy is via a “data control ...” command. This allows users to monitor and control the state of current database connections. User actions under “data control” include those to commit changes, roll back changes and close down sessions. “data control” could be a menu item at the same level as “data access...” or it could part of the “data access...” dialog. Our recommendation is the former. The data control dialog would look something like:

|                       | Open                                  | Commit                | RollBack                          | Close                 |
|-----------------------|---------------------------------------|-----------------------|-----------------------------------|-----------------------|
| <u>Parts Database</u> | <input checked="" type="radio"/>      | <input type="radio"/> | <input type="radio"/>             | <input type="radio"/> |
| <u>FooBar DB</u>      | <input type="radio"/>                 | <input type="radio"/> | <input checked="" type="radio"/>  | <input type="radio"/> |
|                       | <input type="button" value="Cancel"/> |                       | <input type="button" value="OK"/> |                       |

This dialog shows that two databases are open and that the user has decided to roll back all changes since the beginning of the transaction in database FooBar. When the dialog first opened all radio buttons were set to open. On "OK" that setting does not change anything with respect to the database. If Close is selected then the database will disappear from this panel next time around.

### Application Programmatic Interface

Minimally an application needs to learn about two new functions: "start\_query()" and "get\_results()" and one new event: "QUERY\_DONE". In the simplest case an application would call start\_query with basically no parameters. SnarfBoy would take over, interact with the user to find a query, and begin to execute the query. At this point the application would regain control and wait for a "QUERY\_DONE" event, signifying that data is ready. The application retrieves the data via calls to "get\_results()".

There are three flavors of Start Query. In the simplest form the name length and query fields are zero. In this case SnarfBoy interacts with the user to fill these fields. If the application knows which query file to execute, then that name can be passed directly. Likewise a tool such as a data browser may pass in a query specification which it formed under its own steam. SnarfBoy returns a unique query identifier to the application.

```

/* this data structure captures the basic information found in a query file */
typedef struct {
    int    brand_len;
    char   *brand;           /* What database brand are we */
    int    connection_len;
    char   *connection;     /* database name, location, login name, etc */
    int    query_len;
    char   *query;          /* The text of the query to execute */
    int    driver_len;
    char   *driver_specific; /* (not in the file) Space for driver doodling */
} sn_query;

/* for a whole query */
typedef struct {
    int    filename_len;
    char   *filename;       /* what file did we come from, if any */
    sn_query *q;            /* a query */
} sn_input;

/* activated queries have identifiers which select an internal SnarfBoy data structure */
typedef int  snarf_id;

```



```
/* loaded database drivers return one of these */  
typedef int    dd_id;
```

**sn\_query\_id**  
**Start\_Query(sn\_input s);**

There are three flavors of Start\_Query. In the simplest form the name length and query fields are zero. In this case SnarfBoy interacts with the user to fill these fields. If the application knows which query file to execute, then that name can be passed directly. Likewise a tool such as a data browser may pass in a query specification which it formed under its own steam. SnarfBoy returns a unique query identifier to the application.

**Describe\_Results(sn\_query\_id q)**

Tells the application what to expect on a subsequent Get\_Results. SnarfBoy will use the same protocols and formats as Network Innovation's CL/1 client. Refer to the CL/1 programmer's guide for format information.

**Get\_Results(sn\_query\_id q)**

Returns query results to the application. Refer to the CL/1 programmer's guide for format information.

**Commit(list of sn\_query\_ids or ddids)**

The application uses this call when a user wants to permanently post changes to a database, or a group of databases. In this latter case all but one database must support a two phase commit protocol. When this call returns either all databases will have committed or all databases will have rolled back. If necessary SnarfBoy will handle the two phase protocol for the application.

**Rollback(list of sn\_query\_ids or ddids)**

The application uses this call when a user wants to rollback changes to a database, or a group of databases. In this latter case all but one database must support a two phase commit protocol. When this call returns either all databases will have committed or all databases will have rolled back. If necessary SnarfBoy will handle the two phase protocol for the application.

**dd\_id**  
**Qid\_to\_DDids(sn\_query\_id q)**

Returns the database driver identifier, see below, that would be used to execute the query q.

**Publish\_Results(sn\_query\_id q)**

Like Get\_Results but informs SnarfBoy that these results should be turned into a Publication.

Applications will be also able to access database clients directly. Database vendors will be expected to provide a database driver conforming to the specification outlined below. One routine "link\_ddev()" will establish the Macintosh software linkage to the appropriate driver. More than one driver may be active at a time. Individual drivers will decide whether they can handle simultaneous active queries.

The link\_ddev() function is also available to those applications that want to bind more tightly to particular database clients. Such a binding bypasses the SnarfBoy monitor, but SnarfBoy will use the status command to maintain consistent state information.

Apple Confidential

**dd\_id**  
**link\_ddev(query \*q)**

This call will cause SnarfBoy to link the database driver code necessary to perform query q. If the driver is already loaded this command will return the dd\_id of that driver.

**unlink\_ddev(dd\_id d)**

Unloads the code associated with driver d.

**dd\_connect(dd\_id d, query \*q)**

SnarfBoy will call this routine once when work for the associated driver is about to begin. It is expected that this routine will establish a connection to the appropriate database. The driver is free to store driver specific information such as a session id or a transaction id in the driver\_spec buffer.

**dd\_start\_xaction(dd\_id d, query \*q)**

SnarfBoy will call start transaction when it needs to perform the query, q, in the associated database but dd\_status() indicates that a valid transaction is not open.

**dd\_send\_and\_execute\_query(dd\_id d, query \*q)**

SnarfBoy will call this routine to pass the query q to the appropriate driver.

**dd\_cancel\_query(dd\_id d, query \*q)**

This is an attempt to interrupt the query.

**dd\_describe\_results(dd\_id d, query \*q)**

Refer to the CL/1 programmer's guide for format information.

**dd\_get\_results(dd\_id d, query \*q, int blen, char \*buffer)**

Refer to the CL/1 programmer's guide for format information.

**dd\_prepare\_commit(dd\_id d, query \*q)**

SnarfBoy will initiate a two phase commit protocol with this call. SnarfBoy will not make this call unless the database brand supports two phase commit as indicated by a dd\_get\_status().

**dd\_commit(dd\_id d, query \*q)**

SnarfBoy uses this entry point to direct a database to commit the current transaction.

**dd\_rollback(dd\_id d, query \*q)**

SnarfBoy uses this call to direct a database manager to rollback the current transaction.

**dd\_disconnect(dd\_id d, query \*q)**

This command should cause the LAN connection to be dropped, all resources, buffers, etc to be freed and so on. SnarfBoy will not be using this driver for a while.

**dd\_get\_status(dd\_id d, query \*q)**

This routine should return a status word containing the following status bits.

|                     |                                             |
|---------------------|---------------------------------------------|
| DD_OPEN             | /* driver loaded, database open */          |
| DD_IN_TRANSACTION   | /* a transaction is active */               |
| DD_QUERY_ACTIVE     | /* sent a query, no response yet */         |
| DD_NO_DATA          | /* sent a query, but the answer is empty */ |
| DD_DATA_WAITING     | /* query results are in */                  |
| DD_PREPARE_PENDING  | /* started two phase commit */              |
| DD_PREPARED         | /* prepared to commit */                    |
| DD_COMMIT_PENDING   | /* started commit */                        |
| DD_ROLLBACK_PENDING | /* started rollback */                      |

57

## Dependencies, Open Issues, Testing etc..

Two main dependencies are a need to interface with Diet Coke, and the need for a decent CL/1 product. Among other things CL/1 in turn depends upon N&C's datacomm toolbox, in particular the LU6.2 driver and "Zoro Card" which allow IBM access. The Diet Coke interdependency is important from a "total product" standpoint, but not critical to SnarfMan's completion.

SnarfBoy is roughly analogous to the print monitor. However, there are some differences. The exact mechanics of what is actually linked with an application, what is a background task, and how the two communicate needs closer scrutiny. At the back end some details need to be worked out regarding the dynamic loading of database drivers, including the ability to protect SnarfBoy from potentially buggy third party drivers. Additionally none of the human interface proposals have been properly reviewed by someone from HIG (yet).

Testing in a networked environment is non-trivial. The combinatorial feature testing should not be an overwhelming difficulty but the code/component path length from user events to disk heads on a remote machine is very, very, very long. There are a lot of pieces that can break and we are likely to end up "testing" a lot of software that we don't own just because it is there. At a minimum we will require access to a CL/1 server which has access to IBM DB2, VAX/ORACLE and AU/X servers e.g. Informix. Furthermore we will want to test a Mac resident Sybase DB-LIB client as a 3rd party sanity check. The DB-LIB client may or may not be in the initial product offering but we must convince ourselves that a connection is feasible.

57

57

# Esperanto at a Glance

## The Alphabet of Esperanto

A a B b Ĉ ĉ D d  
 ah bo tao cho do  
 E e F f G g H h  
 eh fo go Joe ho  
 Ĥ ĥ I i J j K k  
 ĥho es jo ŝho ko  
 L l M m N n O o  
 lo mo no oh  
 P p R r S s Ŝ ŝ T t  
 po ro so ŝho toe  
 U u Ŭ ŭ V v Z z  
 oo uoe uo zo

28 Letters. There is no Q, W, X, or Y.

A, E, I, O, U have approximately the vowel sounds heard in *bar, bear, beer, bore, boor.*

C is not sounded like S or K, but like *ts* in *Tsar.*

J has the sound of *y* in *yes.*

The sounds of ĉ, ĝ, ĥ, ŝ, and ŭ are heard in *leech, liege, loch, leisure, leash,* and *leeyay.*



ESPERANTO is PHONETIC.

All letters sounded: one letter one sound.

ACCENT or STRESS falls on the last syllable but one.

No IRREGULARITIES. No EXCEPTIONS.

THE GRAMMAR is based upon SIXTEEN FUNDAMENTAL RULES, which have no exceptions.

THE PARTS OF SPEECH are formed from Root-Words by the addition of appropriate Letters.

o is the ending of the NOUN:

fakto  
 telefono  
 piano  
 gluo  
 fajro  
 tasko

ADJECTIVES end in

evidenta  
 longa  
 granda  
 frela  
 furioza  
 simpla

A

NOUNS and ADJECTIVES

form the PLURAL by adding

evidentaj faktoj longaj telefonoj grandaj pianoj

J *ej, ej* sound as in *my boy*

THE SIMPLE VERB HAS ONLY SIX INFLECTIONS.

INFINITIVE PRESENT PAST FUTURE CONDITIONAL IMPERATIVE

I AS IS OS US U

ESTI estas estis estos estus estu  
 LERNI lernas lernis lernos lernus lernu  
 HELPI helpas helpis helpos helpus helpu

N marks the ACCUSATIVE (*direct object*)

Mi (I) helpas lin (*him*)  
 Li (*he*) helpas min (*me*)  
 Ŝi lernas Esperanton

ADVERBS end in

energie  
 entuziasme  
 diligente

E

## Esperanto ERS

Kip Olson  
 January 13, 1989

“La voĵago estas la rekompenci”

## About Esperanto

Esperanto is a project to provide document interchange between Digital Equipment Corporation's DDIF compound document format and the Macintosh. A set of software converters will be created on the VAX to translate between DDIF and Macintosh formats. The VAX AFP server will be used to specify and transfer files, and a utility called Invoker will be created to initiate these conversions from the Macintosh.

The first converter to be produced will use MacWrite and PICT as the Macintosh data formats, because of their widespread acceptance by many other applications. When other more complex formats like Skia become available on the Macintosh they will also be used.

Esperanto is the first step in a more comprehensive strategy to integrate DDIF directly into the Macintosh environment. It should satisfy the majority of user requirements for simple data exchange and be easy to use. More long-range plans may include a cut and paste facility between VAX and Mac and a native toolkit to help developers read and write DDIF directly, if such a facility is actually required. The Apple Paris efforts to define an ODA converter may also provide solutions for supporting DDIF directly on the Macintosh.

## What is DDIF?

The Digital Document Interchange Format (DDIF) is the foundation of DEC's compound document architecture. Based initially on the Office Document Architecture (ODA) standard, it is a file format for storing revisable data such as text, graphics, page layout and scanned images, and can be extended to include other types of information as well. Keep in mind, however, that DDIF has deviated significantly from ODA in some areas, and can no longer claim to be fully compatible with it.

DDIF documents are completely revisable, which means the contents are broken down into logical structures which can be easily changed without disturbing the rest of the document. DDIF also defines substructures called generic contents which can be referenced repeatedly throughout the document. DDIF supports such document constructs as headers, footers, footnotes, chapters, tables of contents and multi-columns. Text can have style, use different character sets and fonts, and be flowed through rectangular galleys.

On the graphics side, DDIF defines lines, arcs, polygons, cubic bézier curves and paths. All shapes can have attributes such as line thickness, pattern and color. Text can be scaled, rotated and drawn along arbitrary paths. Scanned images can be represented in a number of formats with data compression.

Digital has provided a Compound Document Architecture (CDA) toolkit on the VAX to facilitate converting DDIF to other formats. This software library can manipulate DDIF files directly and presents DDIF data in a high-level, logical format which allows document converters to be easily created. The toolkit automatically resolves external references and generic contents, making it very dependent on the virtual memory environment provided on the VAX.

It should be noted that DDIF/CDA is not an architecture for applications, rather it is most useful for creating document converters. Digital already has converters to translate DDIF to PostScript, GKS and ClearText, and they are working on ODA and SGML. They have also created a format called TDIF to store table-based data like spreadsheets, and this is integrated into the CDA toolkit.

Digital is using DDIF as the standard encoding for documents exchanged through electronic mail, and is promoting the use of DDIF to store output from word processors and spreadsheets.

Therefore, in keeping with Apple's commitment to enhancing Macintosh/VAX connectivity, it will be important to be able to access data stored in this format.

### Using MacWrite as an Interchange Format

There are a number of reasons to choose MacWrite, the most important being that it is the lowest common denominator among word processor formats. Almost all word processors and page layout programs can read/write the MacWrite file format, making it the closest thing to an interchange standard the Macintosh world has at this time. Using MacWrite makes life easier for developers, since they do not have to support another document format. MacWrite also supports embedded PICT files, so text and graphics can be mixed in the document. Other, more complex formats could be used, like Microsoft Word, but they would not be as widely supported as MacWrite.

Putting the MacWrite converter on the VAX also provides some benefits. All of the software tools for manipulating DDIF documents are available under the VMS environment, so it makes sense to use them and leverage off of Digital's prodigious efforts in this area. The Macintosh user can then initiate conversions on the VAX, and then do something else with his Mac while the translation is going on.

Using the MacWrite format does cause problems, however, since it does not support many of the advanced document constructs that DDIF contains. Items like footnotes, chapters and multi-columns are not built in to MacWrite, so that information will be lost in the translation. Graphics are stored using PICT in MacWrite, and DDIF cannot be converted to PICT with 100 percent fidelity, so there will be some information loss there as well<sup>1</sup>. Converting in the other direction should be more successful, however, since most of the MacWrite features are a subset of DDIF.

Some of the third-party developers would not be able to absorb many of these advanced DDIF features in any case. Aldus in particular treats any external document type as if it contained no layout information at all. They just extract the basic text and picture elements and reformat the document their own way. Thus it would seem that the MacWrite format would be sufficient for this type of application.

---

1. David Cásseres has written a document explaining these problems in more detail.



The DDIF <-> MacWrite converter will provide this basic set of features:

*Sections*

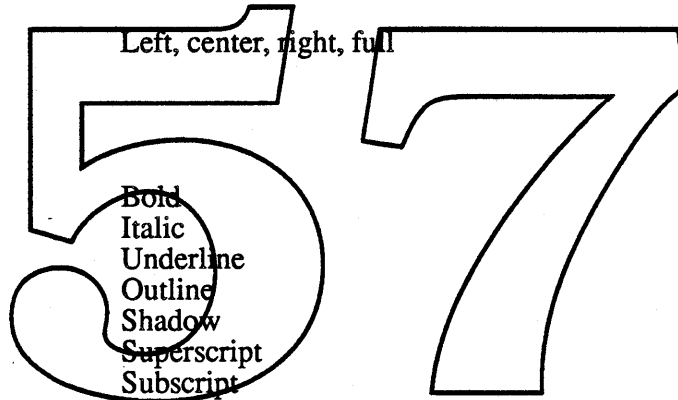
Header, Footer: Paragraph text, rulers, page number, date, time  
 Main Document: Paragraph text, rulers

*Paragraph Formatting*

Left Margin: 1 to 7 inches  
 Right Margin: 1 to 7 inches from left margin  
 Indent: First line only  
 Spacing: 1, 1.5, 2 lines  
 Tabs: 10 tab stops, left or decimal aligned  
 Justification: Left, center, right, full

*Text Attributes*

Styles:



Font Sizes: 9, 10, 12, 14, 18, 24  
 Fonts: Adobe Fonts (list to be determined)

*Graphics*

PICT format: Single graphic per paragraph

**The Converter Architecture**

The CDA toolkit architecture will be used to implement the DDIF to MacWrite and MacWrite to DDIF converters on the VAX.. The converters will be bundled with the CDA toolkit in VMS version 5.0 along with the other converters developed by Digital. They will be available on all systems that use DecWindows.

The VAX AFP File Server software being developed by Digital will be the link between the VAX and the Macintosh. The VAX file server defines an area of a VAX hard disk to be used for storing Macintosh files. Both native VAX files and Macintosh files can be stored in this area, and they will all appear on the Macintosh desktop when the VAX volume is mounted. A DDIF icon will be designed so that users can easily identify DDIF documents on the VAX.

The DDIF to MacWrite converter will create a MacWrite file on the VAX volume, complete with resource fork and Finder information, where it can then be dragged over to a Mac or simply opened up for editing. The MacWrite to DDIF converter will read the MacWrite document and output a DDIF file on the VAX. Note that both converters require that all input files be present on the VAX volume, and all output files must be created there as well. The user may be required to drag MacWrite files onto the VAX volume before initiating a conversion.

The conversion is initiated on the VAX with a Macintosh utility called the Invoker. This facility simply opens a session with the VAX over the network and issues a DCL command to begin the conversion. It then polls the VAX for completion status and checks for any errors. The converted file will appear on the VAX volume when the conversion is complete.

The Invoker will use the desktop services of NuFinder to present a clean and efficient interface to the user. An illustration of this interface is given in the following section. A stand-alone application may also be created to perform the same function, for use by systems without NuFinder. This function will also be available to developers as a Remote Procedure Call (RPC) so they can initiate conversions directly from their applications.

### The User Interface Model for Document Conversion

A typical user session might go as follows: Joe Apple has created the ultimate power-memo on his VAX, but realizes it needs to be jazzed-up a little to really make an impact. Since his VAX lacks any desktop publishing ability, he decides to bring the memo over to his Macintosh and spruce it up with some scanned images of his boss. First he goes to his Mac and mounts the VAX server volume on the desktop (Figure 1).

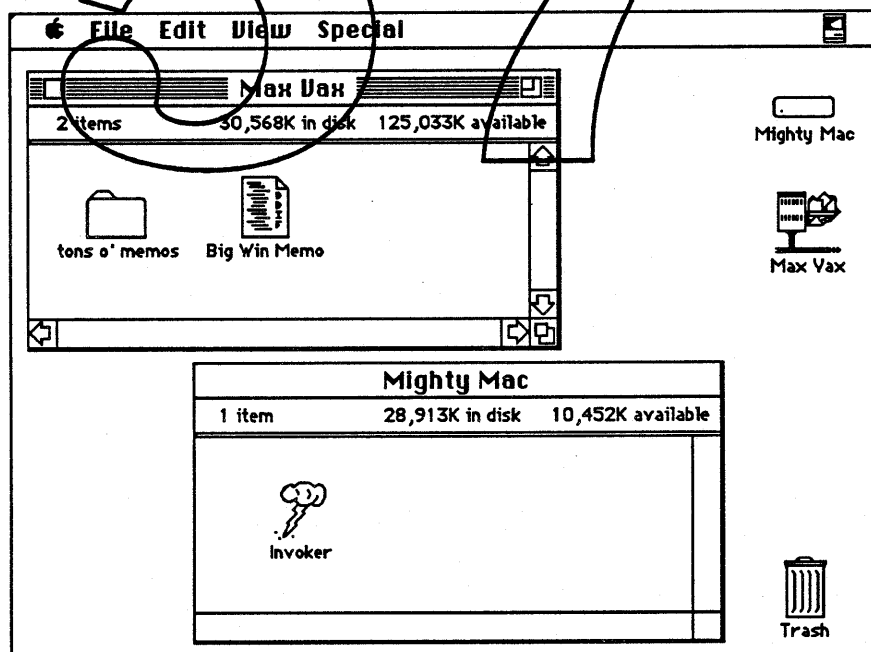


Figure 1. VAX file server is mounted

Finding the DDIF file containing the memo on the server, he drags it over to the Invoker icon and drops it in (Figure 2).

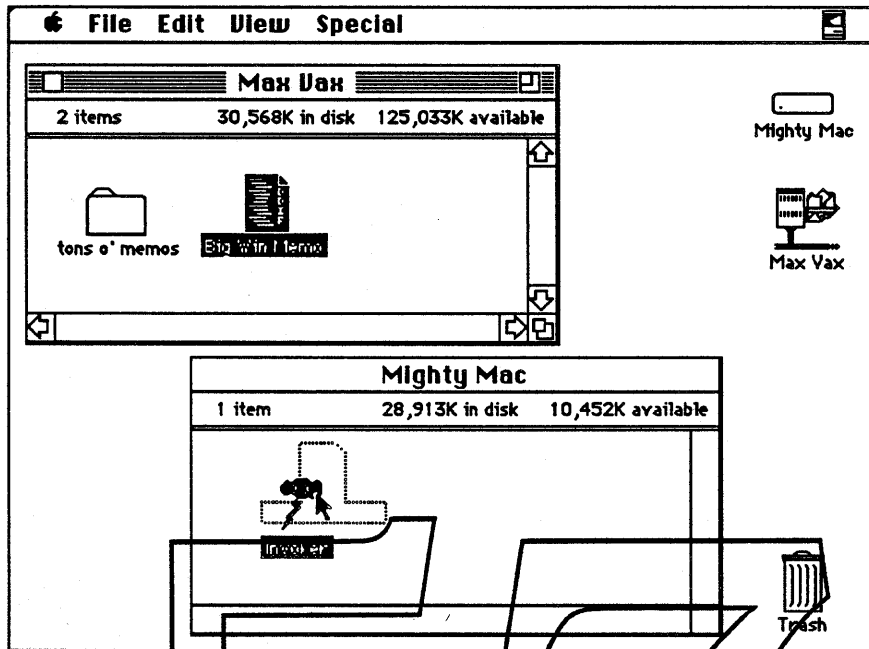


Figure 2. File conversion is initiated

The Invoker sends a command to the VAX to begin the conversion and puts up a dialog box naming the file to be converted and giving an estimate of the time it will take (Figure 3).

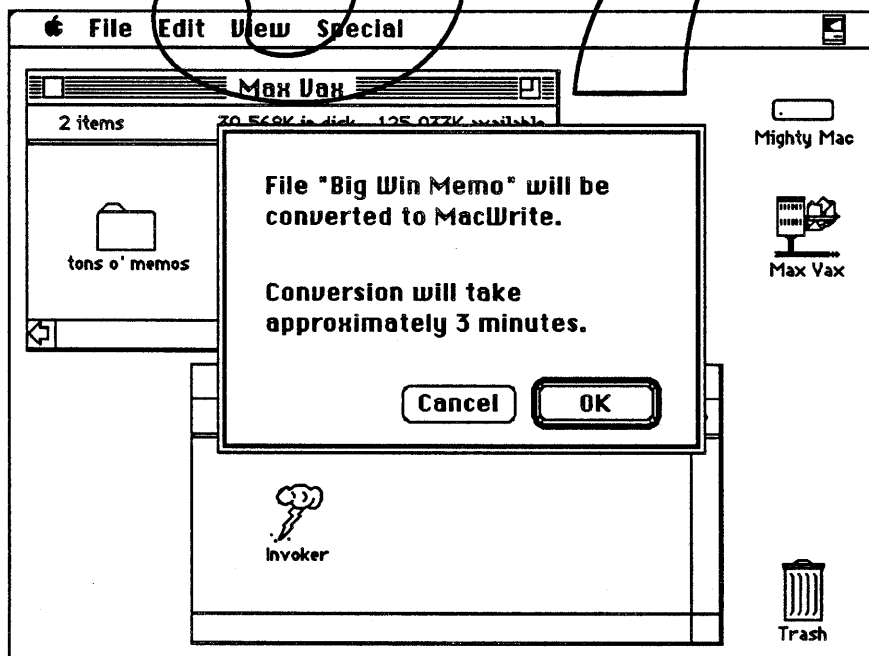


Figure 3. Conversion status is confirmed

During the conversion, Joe Apple is free to use his Mac to do other tasks, such as looking for the

best pictures of his boss. When the conversion is complete, a MacWrite file containing the contents of his memo appears on the server (Figure 4).

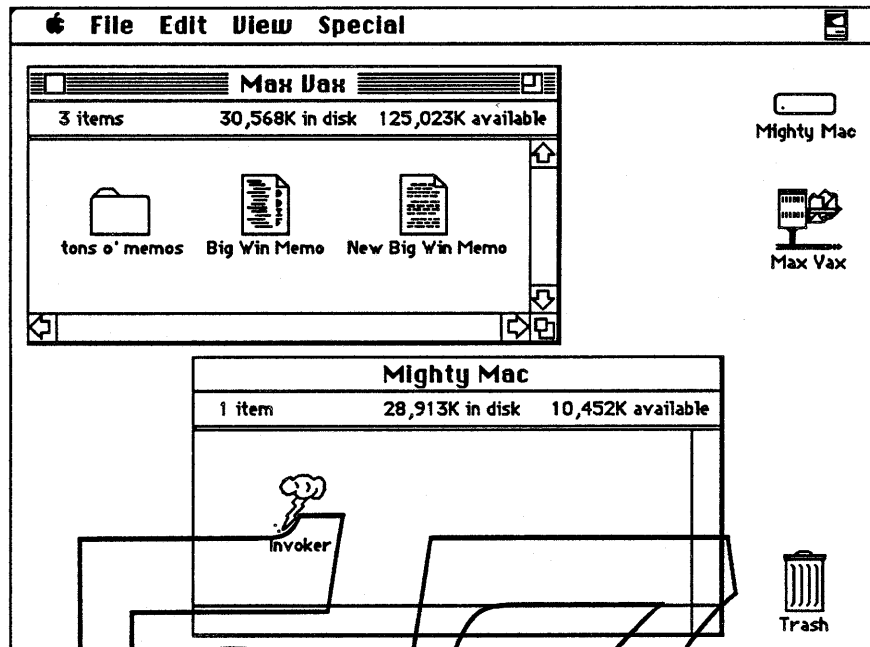


Figure 4. Conversion is complete

Joe Apple subsequently gets a large raise and promotion because of the effectiveness of his report.

### Project Requirements and Dependencies

#### 1.) Esperanto Requirements on the VAX:

- VMS 5.0
- DecWindows
- CDA Toolkit
- AFP File Server
- ADSP Remote Task Execution

The first three items are currently available from Digital. A third-party product called AlisaShare from AlisaSystems can be used to simulate the AFP File Server on the VAX. The ADSP Remote Task Execution ability is needed to initiate document conversions on the VAX. Digital has promised to do this but has not given any schedule at this time.

Digital has agreed to author the DDIF <-> PICT part of the converters, since they have done this type of thing before and know the DDIF graphic constructs very well. As of now they have not allocated any resources for this project nor have they given a schedule for its completion.

#### 2.) Esperanto Requirements on the Macintosh:

- NuFinder
- Invoker

NuFinder should be available by Big Bang, and the Invoker application can be used until then.

Apple has committed to providing the text portion of the DDIF <-> MacWrite converters. This will give us experience with CDA toolkit and allow us to control any other formats that might be appropriate for this type of conversion, like Skia.

57

The first three staves of the musical score are shown. They contain complex rhythmic patterns with various note values, rests, and dynamic markings such as *p* (piano) and *f* (forte). The notation includes slurs, accents, and other performance instructions.

# The Sound Manager and the Big Bang

Written By..... Michael Bartlow  
Music By..... Neil Cornia  
Misspellings By.. Mike and Neil  
Key Grip..... Phac Le Tuan

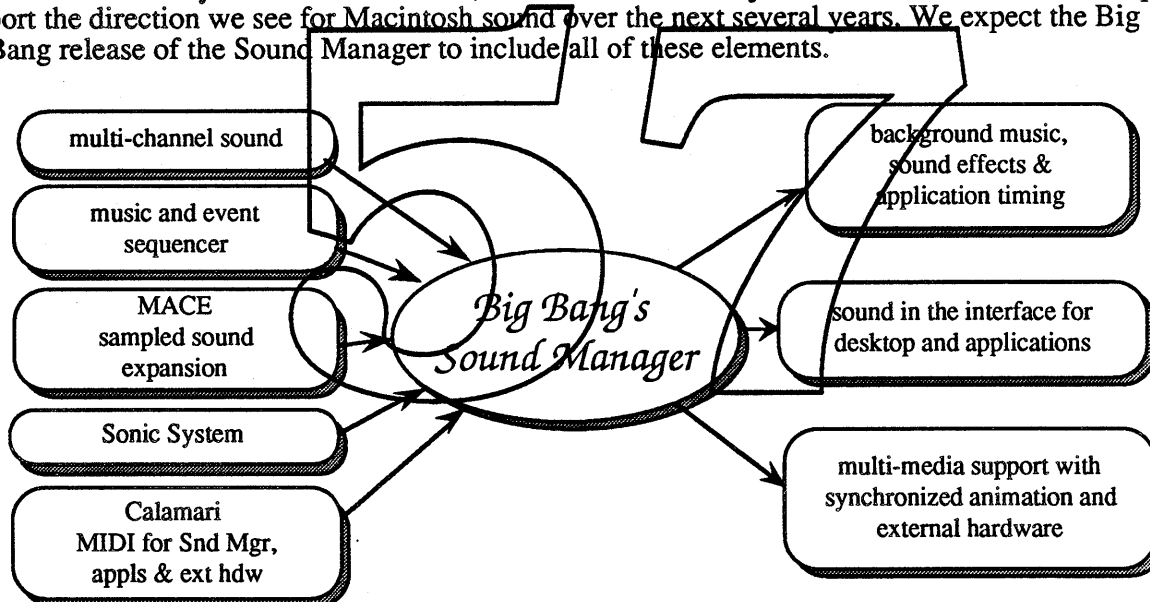
The bottom three staves of the musical score are shown. They continue the complex rhythmic patterns from the top staves. Dynamic markings include *Ad.* (Adagio), *ritard.* (ritardando), *crac.* (crescendo), and *accel.* (accelerando). The notation is dense with notes and rests.

## The Sound Manager and the Big Bang

It's quite a delight for us to plan the future of the Sound Manager. Both the interest in Macintosh sound and the software and hardware abilities to supply that sound are on the rise. The Sound Manager model is ready to be extended.

### 1. Sound Visions

The work we have before us lies mostly in the incremental and leading edge categories. There are problem areas left in the Sound Manager, most of which require extensions to the original design to make it as usable as our developers expect it to be, and as powerful as we want it to be. Beyond those extensions, we will add some major new functions in order to support the direction we see for Macintosh sound over the next several years. We expect the Big Bang release of the Sound Manager to include all of these elements.



The Sound Manager *must* be the arbiter of all sound events to present a unified structure to the developer and from there, to the end user.

This requirement is a far cry from where the Sound Manager was as of last fall. The documentation Apple provided was inaccurate and lacking even the correct overview essential to using the Sound Manager. Without documentation few developers would, or could, use our system sound functions. Our initial efforts provided that documentation, which was warmly received by the developer community.

Communication with the users of Mac sound has been a vital part of the design of our efforts. We are in daily communication with the inhouse groups of Multi-Media, HyperCard, Pink, and the Time Lords. We are very actively involved in the Apple sound community's efforts to specify the audio strategies and hardwares that will best assure our continuing leader-

ship of computer sound's development. We have gathered the many sound needs and ideas that were being requested from third party developers. Based on these needs and visions of Mac sound, we have designed our development goals for Big Bang and beyond.

## 2. Goals:

The basic model of the Sound Manager needs to be extended to work effectively with new markets, system software, and hardware. The first priority is to tackle the obvious deficiencies of the Sound Manager today, complete the features originally intended, prepare for the integration of new features as ATG and future CPUs make them eligible for system software support.

### 2.1 A broader range of Macintosh audio abilities need to be addressed.

The Sound Manager is designed to make the Application's sound code machine independent, using each model's capabilities to its fullest. In other words, it must be the QuickDraw for sound. Just as Color QuickDraw handles the translation of color to monochrome, so must the Sound Manager handle the range of sound processing power from 030 ASC (Apple Sound Chip) Macs to 68000 non-ASC Macs. The ASC, more powerful CPUs, and audio add-on boards can greatly extend the Mac's audio potential. Applications that exploit that potential must degrade gracefully on a Mac Plus or SE. To accomplish this the Sound Manager must be more knowledgeable about the environment in which it is running. Even greater enhancements in sound hardware are expected in future Mac machines.

### 2.2 Macintosh should be the multi-media platform of choice.

For low-end markets, such as education and low-cost business presentations, multi-media requires multiple channels of sound, including sound effects, music, or voice, all coming from a stand-alone Mac if possible. Synchronizing animation and other real-time processes (e.g. lighting, videotape) with sound is a must. For high-end multi-media, such as professional video or music editing, the Mac must be able to synchronize professional equipment with on-screen graphics, application timing, Mac-generated sound, voice and music.

### 2.3 The Sonic System.

We should encourage and guide the introduction of sound into the user interface. System Software can provide a 'path of little resistance' for developers and users by providing software tools that are sensible and robust. These tools will encourage developers to implement sound in a legal and consistent manner, and avoid user frustration from frequently broken software. Tastefully applied, sound can enhance the user interface by providing a predictable response to user actions. To support this **Big Bang** should include:

- our immediate fixes
- multiple channels of sound, safe and controllable by the application
- Visible Sounds: sound as a screen object, editable and manipulable by the user.



## 3. Incremental Changes:

The current system release does not provide many of the sound-related abilities developers have requested. Even the originally planned functions that are represented in the current set of abilities are not fully implemented.

### 3.1 Multi-channel Sound.

The current Sound Manager can play only one sampled sound at a time. If a SysBeep is called while the sampled sound synthesizer is playing a sound the menu bar flashes (if you're lucky). The Sound Manager should allow multiple sounds to play at the same time. On slower machines, the sound quality will degrade to allow enough processor time for other tasks. The key element introduced to allow such a feature is the Status Panel, a data structure where every current sound activity is registered.

### 3.2 Sound Expansion.

A sampled sound uses anywhere from 8,000 to 88,000 bytes of storage for every second of sound. The cost to developers in disc space and to Mac users in RAM is enormous as compared to the code used. Sound Expansion is being made available by MACE (Macintosh Audio Compression/Expansion) which is currently in testing. All future Sound Manager releases, including the Big Bang release, will include MACE expansion as part of the sampled sound synthesizer, providing automatic expansion of MACE-compressed sounds.

### 3.3 MIDI Patching.

Calamari is a software MIDI patcher that can send and receive MIDI data and pipeline this data between multiple MIDI-compatible applications running under MultiFinder. The Sound Manager will be able to use its MIDI-In and MIDI-Out synths to become a Calamari client. This will enable MIDI devices to play through the Sound Manager and enable the Sound Manager to drive MIDI devices. This will extend the capabilities of the Sound Manager to include external MIDI devices such as synthesizers and MIDI-Lighting controllers.

## 4. Proposed Additions:

The needs of sophisticated developers, and the complexities of multi-media software require significant enhancements to the existing Sound Manager. Today, the Sound Manager has limited real-time precision and no sequencing services. It provides no tools to help synchronize sound and graphics. It has an unusable MIDI interface and inadequate timing services.

The result of these combined deficiencies is that, in order to sequence music on a Mac, a developer must bypass system software, commandeer a hardware timer, and write his own code to handle real time events! The user or developer could sample an entire song at 22khz (22,000 samples/sec) and later play this back but, for minutes of music, this is memory intensive solution. Calamari (a MIDI patcher in software ), DJ, and Improvements to the Time Manager will address these crucial problems.

### 4.1 DJ - The Music and Event Sequencer.

If the Mac is to produce background music, asynchronous to the application/user interaction, if there is to be synchronization of graphics or events with that music structure, and if that timing structure is to be shared between applications, as multi-media appears to need to do, there must be a system software level music sequencer in the Mac.

By way of definition, a music sequencer times and synchronizes events. We gain a great deal by giving our sequencer the ability to handle two classes of events: sound events, which it would handle through the Sound Manager or MIDI, and non-sound events which would be cued to the application's code.

#### Examples of a Sequencer's use:

Such a sequencer would generate sound by playing back a stored note sequence using Sound Manager channels, add-on synthesis boards or external MIDI-connected synthesizers. As the sequence is read out, special 'notes' in the timing flow would generate callbacks to subroutines in applications, which could then cue an animation, start a video segment in a business presentation, or poll an external device. The sequence *could* have no audio or MIDI-output whatever, but simply provide sequencing for real-time events such as process control for a laboratory experiment or a factory.

Given only the needs of music sequencing and low-end multi-media, DJ would provide tools that could easily be extended by applications to cover a surprising variety of real-world sequencing, from animation to a broadcast schedule, from rudimentary inter-process communication to asynchronous control of slower external hardware, whose control protocols are understood only by the application and timed using the sequencing tool. In multi-media, the sequencer could generate a jungle ambience in stereo, using only samples of leaves rustling, a bird cry and a single stroke on a native drum, or cue the lighting and video climax at the end of a business presentation to Gorbachev.

### Why music in a computer?

Music is a non-verbal, universal language. No human mind needs it to be translated. No mind fails to notice it. No cultural meanings restrict or misinform some segment of the world's Mac users, a problem even our graphic interface has run into. With a sequencer that language becomes a system resource, able to take a single sampled sound and expand that to minutes or hours of comment.

Of course, we're not suggesting that the proper role of music in computers is to play a theme for the summing function in a spreadsheet, to play background music under a word processor to inspire the writer, or to replace the radio sitting next to the computer. What we are saying is that other computers that are capable of music find their musical abilities used. Our game support is among the lowest of any personal computer, and this sequencer provides a very powerful resource for getting around some of the problems that helped to minimize that market.

In the low-end multi-media markets, especially that of education, music is an almost necessary addition to system support, both for the savings in disc and RAM usage possible and for the entertainment value to educational users.

### 4.2 Visible Sounds - Making Sound as Affordable as Graphics.

The Macintosh is a multi-media machine to us, a collection of tools that allow the developer, and the user to affect the elements of his application or document with unprecedented ease. That control has been most effective in graphics and text to this point.

The tools for sound are among those considered by users and some developers to be too obscure and 'tricky' to use. Only a fraction of Mac users feel at home with **Font/DA Mover** or **ResEdit**, so many fonts, DAs, and sound resources are beyond their reach. Only in **HyperCard**, and, perhaps, **SoundMaster**, have the users been able to easily play with sound, and they have done so with considerable interest. Suddenly, for the first time, the popular bulletin board services have libraries of sound available!

System software has several efforts in process for Big Bang that make the user's control of these unreachables as available as the icons of the desktop, and as easy to transfer between applications as any other clipboard element. The leading examples of these are **Mover** coming in **NuFinder** and potentially **Diet-Coke**.

### 4.3 Sonic System - Making Sound a Standard Part of the User Interface.

Yes, friends, this is the area of the Sonic Finder, and of a whole lot more. The desktop has created a world of folders and windows, of directory hierarchies and of the relations of applications to documents. We think that the possibility of sound adding to some user's abilities in that world is well worth our investigation.

In Big Bang we hope to provide the system level hooks necessary to support the efforts of those of us that want to examine sound in the interface. The desktop audio-animation of the current Sonic Finder would be a sub-set. By putting sound hooks into the defprocs of interface elements such as scroll bars, close boxes, text editing and the like, we will allow sounds to be attached to those functions. Before any sort of release we intend to provide a method for the user to edit and control his sound interface, replacing the sounds as sets.

A position statement: Macintosh capitalizes on Apple's recognition that every mind is different. And despite that unacademic confusion, Apple bravely seeks to make tools for those minds. We provide a tool that fits not a laboratory or theoretically efficient average of what minds are, not some fantasy ideal mind we should all strive for, but the mind that grows in organic heads and bodies. Some think more verbally, some more graphically, some wield intuition like a screwdriver while others trust only in strictly printable logics. The world, and its inhabitants, are blends of every abstracting approach; our machines must not refuse to be a tool for any of the minds that may want to try us. If the sonic interface offends you, or if it doesn't fit your application's setting, turn it off.

Whereas music sequencing is a single, if complex, addition to system code, making sound a part of our interface is an effort to be implemented in many code areas throughout the system and finder. It's discussion and development are not the material for a few real-time specific programmers, but a many-year process of growth and exploration to be shared by system software designers, human interface experts, application developers, and, most of all, the users. As might be expected by anyone aware of Apple's treasured sense of integration, there is already work being done to provide the tools in system software that will feed this process.

Again, a discussion of this topic will be presented in the Future section of this paper. There we will consider our attraction to this possibility and some of the concerns many of us have.

## 5. Implementation

So, what is actually going on in the Blue System Software Toolbox Sound Manager group? How is it addressing our problems and providing for the future of Macintosh sound? We can best answer that by using the categories of sustaining, incremental, and leading edge (abstaining, detrimental, and bleeding edge) to organize the description of our current and future efforts.

### 5.1 Sustaining

#### Bug fixes

Since the inauspicious release of System 6.0, the Sound Manager has received some serious attention, review, and repair. Asynchronous calls to SndPlay are now asynchronous, the sysbeep behaves the same way on a Mac Plus as on a Mac II, and the wavetable synthesizer on the non-ASC machines works. Next on our repair list are: 'Anomalies' in the looping behavior of the noteCmd, some inefficiencies in the real-time code, proper stereo implementation, and changes to the sound code of HyperCard.

#### New machines

The '030-based MacIIx has been found to be wholly compatible with the Sound Manager. Sound problems have been found in a few audio and MIDI applications too clever to require the upward-compatibility protection of using system software.

On the other hand, there's Esprit. This is a unique situation in that it's the first 68000 machine to run an ASC chip, which is a problem because the ASC code had been written for the 68020 bus of the MacII. Also, the Harpo power manager has required new uses of the interrupts. Changes to cover both problems have been made and appear to be effective, but a few questionable behaviors have been found and will be looked at.

The other new machines to be introduced this year have been looked at for Sound Manager compatibility problems, and nothing there seems to be a problem. As they become available for testing and development each will be further tested by us, and by SQA's sound people.

#### Sampled Sound Synthesizer improvements

The sampled sound synthesizer is the most used of the Sound Manager's synths, because of the ease of generating a sampled sound wave, and because of the range of sounds samples can contain. This is the synth with the clearest future and the greatest need for development.

There are a couple of advertised features in the sampled synth that simply are not there, and they are widely noted deficiencies. These are stereo sound and the ability to handle a 44khz sample rate. As part of the incorporation of MACE into the sample synth, stereo is being included in all sampled synth playback, and the real-time needs of the sampled synth are being addressed, hopefully giving us 44khz playback of a stereo sampled sound without totally consuming the CPU.

In the Sound Manager Technote which was distributed in December, a new wave header format that indicates compression type, number of channels, sample size (we support only 8-bit samples now), and several other items from the AIFF (Audio Interchange File Format) was specified. The Big Bang Sound Manager will respond to this new resource format, expanding MACE-encoded sounds, truncating 16-bit samples, and handling sampling rate conversions appropriately.

Another feature we hope to include in the new sampled sound synthesizer is continuous sample play from disc. This has been requested by many developers, and, unfortunately, is being included in Farallon's non-system compatible sound package being released in January.

Continuous play from disc would use a buffer area which is smaller than the sound sample and use a double buffering scheme to update the buffer from disc, thereby minimizing RAM usage at the expense of some disc overhead. This is a trade-off of some importance in the 1-Meg world that is the majority of our market, and the most likely platform for low-end multimedia. The double-buffering scheme has already been used in test applications, even doing real-time MACE expansion while reading from disc. We are out of traps, so the current Sound Manager routines will have to be extended to indicate a call for auto-buffering.

## 5.2 Incremental

The following incremental changes do a little bug-fixing of problems, but beyond that they form the basis of the Sound Manager of the future. We will need to extend system sound abilities first to the '030 CPU power and add-on audio synthesizer boards already available, and then on to the even more powerful CPUs and MIDI & SCSI connected multi-media hardware we must expect in the future.

It's something of an open question as to just how much more can be pumped through a MacPlus without taking the entire CPU during sound generation, but it's an even more open, and relevant, question as to what collections of simultaneous sound might be expected of the Sound Manager at our high end in the next several years.

## Status Panel

As the first step towards multiple channels of sound, we will be adding an internal Status Panel to the Sound Manager. Basically, this will be an area to which the Sound Manager will make notes to itself whenever a channel of sound is set-up or disposed. The information to be included is not fully defined yet, but certainly it might include the name of the application requesting the channel, what type of synthesizer is in the channel, any possible summing information, priority or error response designations, etc.

There is currently no structure within the Sound Manager that describes what the Sound Manager is doing, and two problems arise from this.

First, there is no way for the Sound Manager to make a decision as to what new requests it can handle. In the current design only one synthesizer is allowed at a time, thus avoiding the problems of CPU or sound channel overloading, summing independent channels of sound, and the possibility of mixing any two types of sound for output.

## Sound Manager

---

The current Sound Manager makes no attempt to determine if a channel is currently in use before allocating a new one, and so has no reason, or method, to try to restore a channel after it has been interrupted. Problems occur whenever a sound channel is called for when another channel is already open. There are all too many examples of this situation, such as when a sysbeep interrupts an application with a channel open, or when, under MultiFinder, an application opens a sound channel as a prompt from background. If an application opens a sound channel, is then moved to MultiFinder background by an application that opens its own sound channel, and then returns to the foreground, there is no mechanism to inform it that its sound channel has been taken away! The first application continues to output sound commands, without being able to know that it isn't making a sound!

Another result of the absence of status monitoring, is that the statusCmd does little of what people had thought it did, and what it does do, it does ineffectively. Users of the Sound Manager had thought the statusCmd would tell them of a variety of things, from what channels were active, which are making sound now, what synths were initied, etc., etc. Without this sort of information, current applications don't know if they are making any sound, or whether or not their sound has been interrupted.

In fact, the statusCmd was intended only to report whether or not a synthesizer init call had been successful. Given the current tools, only some of these situations are preventable by the apps, (such as closing the channel at the suspend call from Multifinder, etc.), so those developers who have made the effort to sort out the Sound Manager, then defend their use of it against marauding sysbeeps and MultiFinder situations, soon come to realize that they just can't count on having a channel of sound if they use the Sound Manager! This is not a good thing.

The Status Panel, then, will serve two purposes:

First, to allow the Sound Manager a record of what its current situation is, so that it can decide if it can allocate another channel of sound when requested, so that it can restore a channel it used briefly for a sysbeep or a prompt from background, or, perhaps, to restore sound channels left open at a suspend event in Multifinder.

Second, the Status Panel should be used to support a new version of the statusCmd, for applications to use to determine whether or not they will get a channel they request, whether or not they are going to knock out another application's sound, whether or not there is a sysbeep going on, etc.

### Channel arbitration

Channel arbitration will be a protection of a sound channel once initied, using the Status Panel's record of who is doing what with which channel to respond to an incoming channel request (hopefully by restoring the previous user if a sysbeep or background prompt over-rides), and monitoring channel usage when an application is sent to background in MultiFinder while it has open channels.

## Sound Concurrency with Degradation

On machines with sufficient processing power, channel arbitration will be supplemented by the ability to play more than one channel at a time. If the application wants several channels playing simultaneously degradation will occur. If a sampled sound is playing and another sampled sound channel is requested by the application, the Sound Manager will check the Status Panel to determine if there is sufficient processor time to add the channel. If there is not enough time, the Sound Manager will try again with a degraded form of the application's request. If the application requested a stereo sampled synth with interpolation, it may have to settle for a mono sampled synth without interpolation.

This will provide graceful degradation of sound capabilities across the entire Macintosh line. As an added bonus this will provide graceful enhancement to applications as well. An application can request channels at full quality until it has exhausted the allowed percentage of processor time. A Mac Plus may allow only one or two channels, a MacII may allow up to four channels, and so on for future machines.

Hardware will impose some restrictions on concurrency. The ASC chip has different modes for sampled, ~~4~~ wave, and note synths, so concurrency will not allow mixing of different *types* of synths. There is a way around this problem, but the solution will require completely re-writing the wave and note synths for the ASC machines.

57

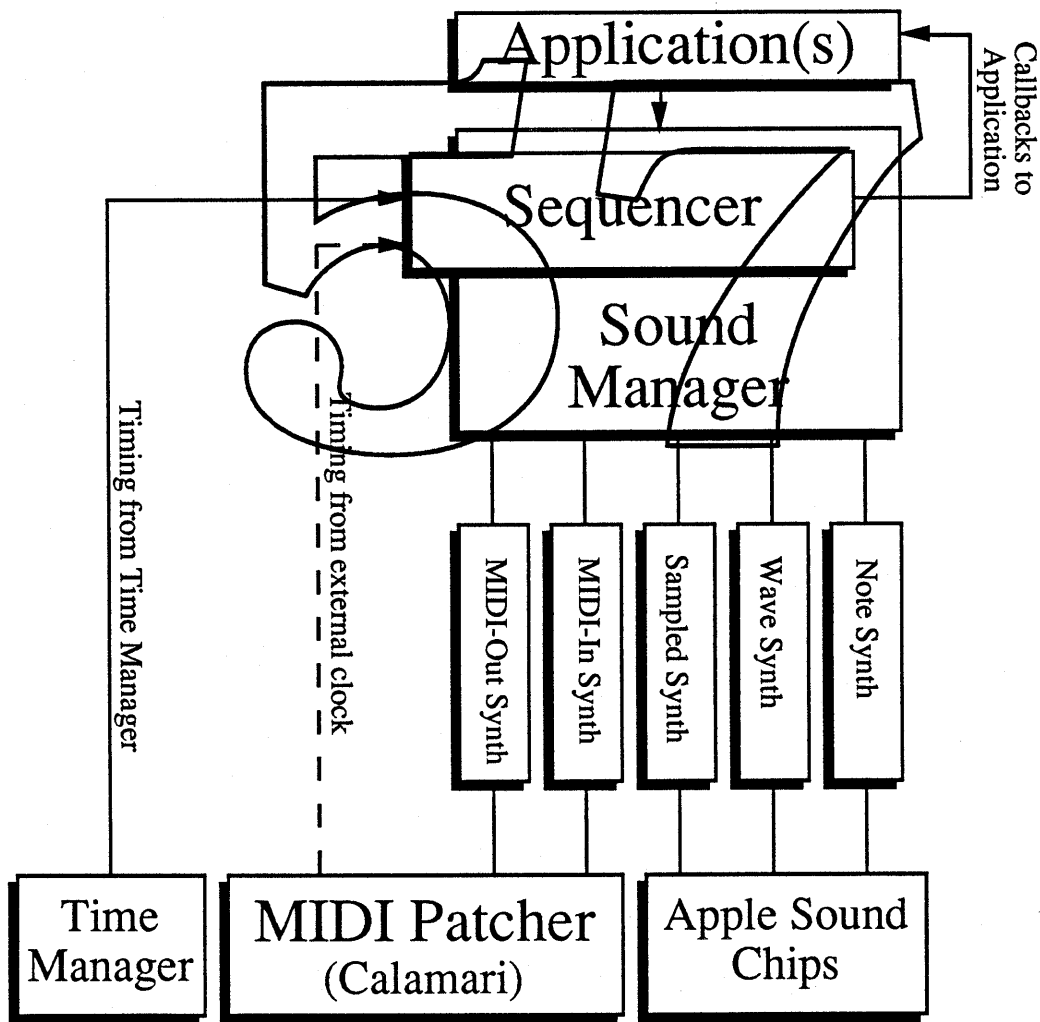


5.3 DJ

DJ is a music sequencer for the Sound Manager. A music sequencer is a device that takes a previously recorded stream of notes and other data, and outputs them to some other device that can interpret each note and, hopefully, play it back. By making the Sound Manager's MIDI-in and MIDI-out synthesizers into Calamari clients, DJ becomes a MIDI sequencer, able to drive a stage full of MIDI equipment, even from non-MIDI application code.

A callback mechanism will allow applications to embed 'reminders' in the sequencer's event flow, allowing DJ to become a *generic* event sequencer capable of providing timing, sequencing and synchronization for on-screen animation, sound, communication with external equipment, or the control of a year long experiment in a computer controlled lab. This provides an opportunity to establish standards for multi-media coordination of 'arbitrary' assortments of external media equipment and software.

And now, the details...



### Why add a Music Sequencer when applications could roll their own?

**DJ** is built on the system tools of the Time Manager, the Sound Manager, and Calamari. As you'll see in the following discussion, **DJ** fits almost classically into the arguments for any system software: that we provide a tool for developers that makes a function (real-time sequencing) more easily and reliably available and less costly in development time to the developer.

Currently, many developers would like to include music in their applications but cannot because of time constraints or lack of knowledge. Here we also have the great benefit of providing a platform for an important new standard, one which may provide a critical method of communication between the families of applications that we hope to encourage in the multi-media market.

**It may be useful to some readers to have a brief review of the abilities offered by the system tools DJ utilizes.**

#### The Time Manager

Real-time services are the goal of the Time Manager, offering applications and system users the ability to request interrupts at specific intervals from the Time Manager, either on a one-shot or periodic basis. The Time Manager is sensitive to the error in real-time response any traditional micro-processor system almost guarantees, and provides information to the user to help him respond to any error in the timing. The Time Manager currently has not enough precision to support sound and MIDI requirements, and we have operated independently of it in the past.

It is something of an open question as to whether or not the Time Manager will be improved in Big Bang to the point that it can provide the timing services required for Calamari and the Sound Manager. Calamari and the Sound Manager currently work just fine sharing the sound community's hardware timer, but any application that wants to use Timer 1 and is NOT a Calamari or DJ client, will be a source of contention. If the Time Manager gets itself into better shape, we will both move over to it.

#### Calamari & MIDI

Calamari is a MIDI patcher providing the first system MIDI services stable enough to be usable. Having cured a glaring deficiency in our system's abilities, it went on to produce a new, significantly useful and elegant level of integration of personal computers with MIDI, a level no other system offers. For those who have only a vague idea "what the big deal is about MIDI", we'll start first with a description of MIDI.

### Why MIDI is a requirement for Macintosh system software.

MIDI was developed as a communication protocol for music and is now the unchallenged standard used to connect music synthesizers and support equipment. Very little of the music developed or performed today is done without the use of such systems. The MIDI standards define the serial message structures that are sent to and from MIDI devices. For example, all MIDI synths respond to the same message for "start a note in channel one at A-flat, with a volume of 67"; an individual synth might not be able to use all the information included, but all MIDI synths are able to read the message.

The impact MIDI has had on its industry is an example of the quality and pervasiveness all standards-makers have always hoped for. MIDI has become the digital glue that connects music and entertainment equipment from music development through live performance.

Entire stage set-ups are driven routinely across the same daisy-chained MIDI cable. Those stages can interconnect controlling computers and any number of synthesizers, the sound processing devices, from the guitarist's wah-wah pedals, to echo units, to the mixing board that mixes all the audio for the performance, and, beyond sound, even the lighting board that controls all stage lighting and effects.

Just as importantly, MIDI has become the dominant approach to music composition and recording. Its editing capacity, and the software developed to realize that potential have made it essential in education and all areas of professional music, from one-man lounge acts to the frame-synchronized scoring of movies and video production. All are synchronized through the MIDI protocols of communication, wiring and timing management.

### Calamari - the Macintosh internal MIDI Patch Bay

Before **Calamari**, that is in today's system releases, MIDI applications crashed into each other and the Sound Manager. As noted above, the Time Manager we currently offer is not precise enough for MIDI protocols, so MIDI applications had to take over the timer used by the Sound Manager to get adequate hardware support in Mac for a world-wide standard. Operating outside of system protection, all MIDI applications could be clobbered by so routine an event as a sysbeep (in most situations), or many situations under Multifinder. With luck one MIDI application maximum could be run. Despite the ugliness of this, some of the finest, and certainly the most popular and professional of all MIDI applications have been written on the Mac, and the Mac is still the computer of choice for most musicians, professional and amateur.

Now, with **Calamari**, the MIDI networks and protocols are extended into the Mac. Using software simulations of MIDI cables, the MIDI user now finds that he can connect multiple MIDI applications running under MultiFinder with external MIDI-connected devices, including audio synthesizer add-on boards already on the market. Beyond the normal services of MIDI, **Calamari** offers full professional-quality timing conversions and services, bringing MIDI to SMPTE conversions, and a desk accessory-easy control of inter-application and external device communication that may be a graphic model for the general, non-musical inter-process and application communications yet to come in system software.

**Calamari** will be introduced at the January NAMM (National Association of Music Manufacturers) show in Anaheim the same weekend as MacWorld. As things now stand, it will be licensed to developers and distributed as an INIT. We hope that by Big Bang it will be included on our Utility disc; Sound Manager power is extended by it to include the entire MIDI world, even if no **Calamari**-aware application is running at the time.

If an application wishes to use **Calamari** (that is, a MIDI application that needs more services than **DJ** alone will supply) then it must act as a **Calamari** client. The demands are minimal and quite obvious: the application must sign on to **Calamari**, telling it what sort of time-base it's talking into and wants to be informed in, assigning time ports, data in ports, and data out ports as it requires. This system of ports is almost exactly that of MIDI, the differences are only improvements, and quite within any MIDI developer's expectations. A **Calamari** message format wraps around the normal MIDI messages, but again, the difference is minimal and adds obvious and desirable functionality from the developer's perspective.

An example of a typical Mac/**Calamari**/MIDI use: the score for a 30-second commercial is the goal. The available equipment is a DX7 synthesizer, a Roland drum machine, and a MIDI-driven mixing board. The digital equipment is a Mac with MIDI interface box running **Calamari**, a music sequencing application and applications to edit and control the external equipment. Let's put the musician towards the end of his development process: he's written the music and is now fine-tuning the tone of the finished product by listening to it being played by the sequencer application through his three pieces of music equipment. Somehow the cymbals aren't right.

Without interrupting the sequencer's control of his music setup, he mutifinders over to the editor applications and tweaks individual values on the cymbals being generated by the drum machine. All the tempos remain as he programmed, all the instrument switching and blending continues as his listener would hear it, and all the while, he's playing with a single control value out of the possibly hundreds involved, using the graphic interface the drum machine's editor found most appropriate.

Now the tone of the cymbal is there but it's still annoying. The sequencer continues reciting the current piece, as another MultiFinder click brings the mixing board's editor to the foreground. A little playing around and the problem is found: the cymbals' fade-in was too quick, and the curve the editor was generating was based on the wrong sort of algorithm. A few radio buttons poked, a scroll bar or two moved, either by on-screen mousing or through the control wheels of the DX7 keyboard, and just the right blend is found, reviewed and saved.

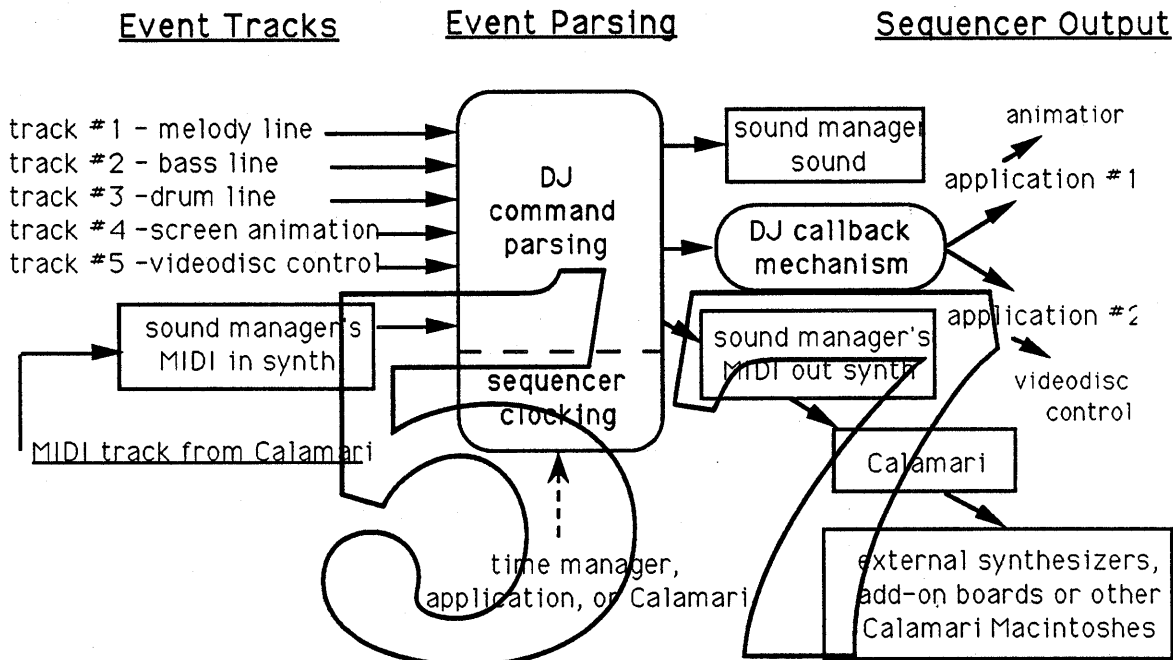
To any musician, this is power, the power for your music and your musical talent to "be their best."

Without **Calamari** one application at a time would have been used, an edit, an application switch to the sequencer, a review of the change, and application switch, and another blind shot at an edit. This is the best most users have today. It will be archaic after the next NAMM show.

**The Description of DJ.**

What do we hope DJ to add to the MIDI/ Mac world of Calamari?

- Make the development and control of music through the Sound Manager as easy as calling an individual sound
- Make the MIDI world available to non-MIDI applications running under MultiFinder
- Synchronize music or any other event flow with anything an application can control.



**DJ, the Sequencer,  
doing a typical multi-media application**

**Two complications in verbally describing DJ:**

Two areas complicate this description, so we'll point them out right now.

First, DJ will run either with or without Calamari. If the user wants to use MIDI, or SMPTE timing, as a part of DJ's input or output, then Calamari will be required; it costs too much in code space and development time to duplicate the timing services and message control abilities Calamari offers. If Calamari is not present then DJ will generate its own clock, either internally or as supplied by an application. That clock service will drop some of the features MIDI includes, almost certainly tempo will be a constant, almost certainly the variety of timing bases will be reduced, etc.

The second complication to this description is the variety of uses applications could put this to. We plan only three formats for sequencer input: MIDI, note (that is, beat) -based tables, and a table format for millisecond timed events. There are three types of output: Sound

Manager commands, MIDI and callbacks to applications running under MultiFinder. To list the possible uses for these relatively few input and output modes, touching only briefly on the combinations of those modes through synchronization, independent operation, etc., boggles the mind, and would drag the attention far too far away from the consideration of the development of DJ.

### A Reassurance on Development:

Those are the complications of this description, now an assurance: DJ adds only three functional areas to the current Sound Manager, so it might just be develop-able for Big Bang! They are command format parsing/translation, the clock generator/converter to DJ's output, and the callback mechanism to alert applications of events timed at the interrupt level. They aren't trivial, but they also aren't as diverse as the applications this may serve.

### DJ's Development

Some of the crucial elements of DJ are a part of the Sound Manager's future whether there is a DJ or not.

The current Sound Manager has two MIDI synths, MIDI-in & MIDI-out, that just, plain don't work. They will be reborn in Big Bang as Calamari clients, interfacing the Sound Manager to the MIDI world.

When the MIDI-in synth receives commands from a Calamari attached MIDI source, an application or a real-world synthesizer, it will convert those MIDI calls to Sound Manager commands and send them to a Sound Manager software synthesizer for playback through the Mac speaker. Some arrangement of this provides a developer wishing to include music in his application, but not willing to master the Sound Manager, to use a standard MIDI set-up to develop his music, using the Mac as his output device during development just as it will be in the final application. If he wants to hire some musical help, he'll only have to find any MIDI-capable musician, a bit more common than the music/computer expert required to write for the Mac today.

In the same way, conventional Sound Manager code, if sent to the MIDI-out synth of the Sound Manager, will be converted to MIDI code and sent to Calamari for distribution to whatever may be listening in the external MIDI world. This is the method that will be used in the future to drive most audio add-on boards, such as the DigiDesign DSP-based board already marketed.

If nothing more of DJ makes Big Bang than these, several of the advantages we want will be realized. As a very worst-case fallback, the MIDI synths alone make a pretty good boost to the Sound Manager.

### DJ's additional requirements

New stuffs required by DJ, as noted above, are input parsing and translation, clock generation, and the callback mechanism. I don't really see where the first two topics need to be gone into here; they're somewhat predictable to a programmer and pointlessly obscure to a non-programmer.

Some general comments on the input formats might be of interest to those who expect to be able to use it in their work. If MIDI or SMPTE based control is to be a part of the system, then **Calamari** will be controlling the clocking and the in/output of those devices. Even then a non-MIDI, non-SMPTE application or purpose might be included as a track by using one of the **DJ**-supplied formats.

When the first application initiates **DJ**, it will declare itself the master application and define the master track's time-base, tempo, MIDI/non-MIDI, error reporting, how many callback numbers it requires. Other apps that sign on, or perhaps other services requested by the master application, will specify slave tracks, and these too will have a list of parameters, such as whether its timing is synchronized to the master track, what its timebase is (beats or milliseconds), error correction, how many callback numbers it requires. This information will be used by **DJ** to set-up its callback mechanism, to select the form of parsing it will apply to the channel, what timebase, etc.

We plan two formats for event flow tables which would be unique to **DJ** (that is, not the MIDI-in synth's parsing of MIDI). The first would be a beat based note file format, which may end up being MIDI, with a few extensions to conventional MIDI, perhaps.

The second, a millisecond based format, would have two command line forms:

- 1) "event, time till next event"
- 2) "loop to x, after this duration."

'Event' would be an indicator of a callback to the application. 'x' would specify how many commands back to jump to provide looping of an event series. Some conditional test command might be included, but what that conditional would test is not clear; probably it would be up to the application to monitor the state of things and clear or rewrite the commands of a track when it wanted to change the selected event series.

### The DJ Callback Mechanism

Comments now on the callback mechanism are strongly invited, because, of course, the expression of that mechanism will be crucial to this thing's usefulness to animators and multi-media designers. This is the description of our current view of how things will go; we are asking for input here from the ATG Multi-Media group and the Time Lords, in particular, and anyone who reads this, in general.

The problem to be solved here is, simply, **DJ**'s going to be parsing at interrupt time; it's timing will be interrupt driven, whether it's clocked by the Time Manager, **Calamari**, or internally. How then to call a routine in applications residing in the domain of non-interrupt time? Posting special events would be great, if we could send to a particular application's event queue, and if we got a revision of all the languages that generate the event queue independent of the application being written in them. It looks like the solution is to have the applications allocate a **DJ** queue and then to poll it when they run their event queue.

When **DJ**'s command parsers encounter a call for an external callback, the sequencer would write to the appropriate application's queue:

- 1) the callback's identifying number,
- 2) a long word defined by the application (either a pointer or data value),
- 3) the current time (to allow the application to check for timing error, so that it may "gracefully degrade" its own situation).

This queue, which the application would have allocated initially, would then be checked by the application in its event loop and be acted upon as the application saw fit.

This allows the application to synchronize any sort of event to the sequencer's time flow.

Examples of these events (I really can't resist this any longer) might be a frame update in an animation sequence, a cue to an external CD for a music start, the dialing sequence for a PBX for an automated data transfer, a call to another Mac through MIDI connections to start the lead singer's stage rotating (and cue the strobe light sequencer application), a call to another application to ask it to read a shared area of memory for an update to an IPC connection, a lab animal's feeding schedule and the equipment that monitors him, a factory's security system, the energy management logic of a skyscraper's heating and cooling control system, a network of church bells, fireworks and cannons for the finale of The Great Gates of Kiev, or, perhaps, well, what have I left out?

DJ's callback mechanism, supported with standards of use can generate multi-media inter-application abilities, and even event loop communication between application. Given that it occurs within a sequencer, either of the functions can easily be called in a timed structure by any application as a slave channel.

### A note on a possible multi-media approach.

It's somewhat difficult to imagine what sorts of applications might wish to share a synchronization. What sense does it make for a presentation to be made of a master track developed on one application running perhaps the music and onscreen graphics' timing, while another is running a second document fitted to the first, if the development and synchronization of the two apps is left as an added task to the author of the presentation. Like our MIDI composer, that author is the someone we are supposedly trying to shield from any of this sort of document/application intra-munging complexity. He must be allowed to be thinking of his presentation, not of "doing computers." We would like to suggest that a master editor/slave application model be fostered.

If the multi-media author can interactively develop his presentation (just as our MIDI user did above), by sitting at a master editor application that will handle the callback assignments and insertions in his DJ master track for the slave applications and their equipment, and generate through them the documents they are to run when those callbacks are made, then we've saved our author's attention from the distractions inherent in our united nation of protocols. Then, when 'presentation foobar', as our Mr. Oren tells us, is called up, the author can react and add to the system's output in real-time, as he wants to, and can ignore the mess of inter-and intra-communication, -synchronizations, and 'I-don't-want-to-know-either's involved in running arbitrary equipments using even more arbitrary protocols.

We suggest then that master editor applications be capable of being told which slave applications or external equipments will be involved, and that that master app then provide an interface to design a core track, stored and edited as the master document. When that master document is played back, the author can then manipulate attached equipments through slave apps that know of particular equipment's protocols. They will be sent the author's input through the master, generating a slave document synchronized by the master's protocols with the original master document. A presentation developed by this master/slave model, then is a set of documents, all of which can be called into synchronized action by calling up just the master document.



That was just a note to the multi-media thinkers; may it begin our discussions on the design of DJ's callback mechanism and the design of its eventual uses.

To quote Ms.Tracy Ullman: "Enough already, go home!"

57

## 5.4 Visible Sounds

The Mac has until now provided the user with sound tools best considered miserably weak and thoroughly unusable.

Third parties jumped in early to provide the profusion of tools each of them felt important; since they had so little from us to support or guide them, an impressive jungle of incompatible efforts resulted. A recent in-house survey of sound tools available to Mac users quickly generated a page full of shareware icons alone; only some of them were recognized by Mac's sound community in-house. In the past, sound has been treated like some sort of font, everybody uses it but nobody but an expert has ever tried to use the tools to do anything special with them. This is not the Mac ideal.

Even the Sound Manager itself was quite beyond its target group; the developers had only one chapter from Inside Mac V, to use as their guide. That chapter was wholly contradictory, where clear it was incorrect, and the overview it provided was not of any Sound Manager its authors had ever heard of.

To encourage the user to think of sound as a sense Mac is a tool for, many developments have occurred. To make that tool available to the average user is much of the goal that drives the many efforts, including ours, in Visible Sounds.

Two Big Bang efforts from outside the sound community are going to be very effective in changing the availability of sound to the user. They are **Diet-Coke** and the **Mover**.

### Diet-Coke

With Diet-Coke, applications can produce documents which can include sections permanently imported from other applications, in the form of Subscriptions to Publications.

### Sonic Manipulation with Diet-Coke

One advantage of Diet-Coke documents is that each type of information included can be manipulated by the application that best fits it. Sound, as much as any media, requires very specific applications to process it, especially for the non-professional audio user. Using Diet-Coke, sound may be fitted to a document, then trimmed using an audio specific application. Whether that sound be a sound effect, a sampled sound or a sequence, the user's ability to manipulate that sound will be directly extended by his available applications.

Imagine then, an application that wants to provide voice notation. Either it can include the code for handling a digitizer, along with the Sound Manager abilities needed to play any sound, or it can subscribe to a digitizer application's document through Diet-Coke. There is more to an application hooking a digitizer to the system than just reading a serial port. Any sampled sound can benefit from filtering, enveloping, effects processing like mild echoes, and certainly sound compression.

The place to find these audio abilities will be found in a current digitizer application. Its latest update will have been done by a staff concerned with sound, and well-informed as to the Mac's latest sound developments. There's just no reason to expect every document that wants to include voice notation to put out revisions every time a new audio compression scheme is added to system software. **Diet Coke** offers the services of professional specialty applications to any other type of document that may want to use them.

In order for **Diet Coke** to provide such audio service, three system level abilities are required.

- 1) There must be a standard interchange format for sound. And there is, the 'snd' resource. Such a resource must be defined for music sequences generated by **DJ**.
- 2) There must be a simple way for the subscriber application to play the sound so that it doesn't find itself supporting abilities the user may not include. Again, that ability is included in the current Sound Manager in **SndPlay**, a command that requires only three lines of code needed to play any sound resource. And, again, such a call needs to be provided for **DJ** sequences.
- 3) There must be a graphic representation of the sound in the subscriber document. This is a **Diet Coke** need; the presence of a sound resource in a file doesn't say where it is to be played. This may be a matter for human interface to consider, but it is certainly the sort of question we try to help out on.

Our role in the Diet-Coke development for now is advisory, and enthusiastic. It serves our system level purposes so well there is little reason to get in the way.

For Big Bang, we expect only to make the connection of sound resources under Diet-Coke to be well-informed and compatible with the future of the sound system as we see it. In the following year we expect to participate more actively in developing the potentials of inter-application communication as it applies to Mac sound.

### Mover

From the Finder group comes **Mover**, something of a desktop graphic replacement for Resedit, Font/DA Mover and Sound Movers all rolled into one.

Here in Bluebook the place to know about Mover is, after all, the Mover section of the Bluebook. Now, here some notes on how sounds live under its services.

For Big Bang, **Mover** will provide graphic presentation of fonts, DAs, sounds, and some network services. A double click on the system file icon opens a Mover window in which sit all these resources as icons. Any of these can then be moved into a Mover file from which any of them can be opened. An open DA just functions as a DA would, had it been called through the Apple menu. An open font might show a Workshop with samples of the font and perhaps an editing facility.

When a sound is selected and clicked, it plays; when a set of sounds are selected and clicked, they all play sequentially; no more SoundPlay style shareware apps, no more need for a sound-interested user to go out and buy a professional quality sound editor just to hear the sounds his disc holds.

When a sound is opened there will eventually be Workshop provided, and that's where we start to get involved. In such a Workshop, a technically naive user of sound would edit many of the very perceptible characteristics of sound the Mac can control, and the user wants to control. Among these certainly would be the pitch of the sound, if it is to repeat itself a number of times

or indefinitely, various filtering or enveloping effects might be included to turn a sample of a cat's meow into a space gun blast curling off into the cosmos.

Fun stuff, until now available only to us pros. More 'non-gratuitous', serious minded things would be available to the common user there as well. In particular sounds might be compressible to save space, or de-compressed to be given to a friend whose system doesn't have the appropriate system update.

Another area we hope to apply Mover to is the Sonic System. Perhaps the System file, or dedicated dummy icon just for this purpose, could be opened to a Workshop for the Sonic System. That Workshop would include a way to select the sounds, or sets of sounds, to be attached through the Sonic System table to the graphic interface elements of windows, buttons etc. A select would set the area of sonic definition, either desktop or system level and the icon placement would somehow attach sounds to graphics.

One proposal is that the Sonic System's Workshop would be a proto-desktop with windows and controls and the rest of the interface there, perhaps with sonic attachment points attached to each. These might be socket-like things to which one could drag a sound icon from a Mover window or the open system file, then attach it with an informative "snick". Once a scroll bar had been connected to a sound this way, that scroll bar could be operated in the proto-world, and for that matter in any available real-world window on the desktop at the moment, the sound left as a lasting attachment, or moved back to its own Workshop for editing.

It's an open question as to whether or not the Mover, audio or Sonic System Workshops, or the resources to develop any of these will be ready for development for Big Bang. We're very interested in these developments for their unique, very new-feeling presentation of sound control, for drawing interest and attention both to the new world of sounds in the Mac, and as a great demonstration of the new approach to resources Mover offers.

### 5.5 Sonic System

An example on the Sonic System's possible future, perhaps a part of our children's future:

A student sits down to a long research paper's last session. His **MacGalactica XVII** boots up when he calls to it. As the screen loads his desktop, the sound of a mailbox creaks open and a bird flies off to the left: he knows the modem-mailbox contains new messages, and that none of them are a bill. On the desktop the new messages glow with a pulse. In the right speaker a shark tail begins to stroke the water at a nasty little tempo: that one's the charge card bill down there in his unpaid box, right where it's been for over two weeks now. The shark slaps the water surface with annoyance, just a little more often than it did the day before. "No matter."

As he goes to get coffee his desktop sings through the Progress Tune. Moving through the files on the desktop, oldest to newest, the Tune sounds the document sound, with volume proportioned to age, quieter for the old, building louder to the newest, with pitch varying with the file size, deepest tones for the largest files. If he had been watching the monitor he would have seen each icon flash as its tone was played.

When the first progress tunes came out in shareware, they were haughtily dismissed by the pundits of the computer media, with the usual round of "gratuitous", "inefficient", and "non-essential"-s that had met quite nearly every development in Macintosh history. Now, two years later, the Progress Tune's popularity had made it system software.

It had been found that users didn't make that many document changes in a day, so the tune evolved slowly, and to each user's ears the daily differences were noticed, and sub-verbally, playfully, refreshed his memory of what he had been doing. As the user changed files the pattern of the tune changed, and it is, and will always be, the recognition of patterns that ears and minds are built for. The information couldn't be hard-core: it only cartooned the work flow, and left the user with a fresh ending on the tune as a reward for his latest work.

If one asked our student what the Progress Tune was for him, he'd answer that it was just a little private memory game he liked to play. He didn't try to remember the tune from yesterday or which file changes caused what change; he just noticed the changes.

He taps his paper's icon. The pitch is a little lower than yesterday's, "at least that'll sound good to our good Miss Brownley." Two cycling wind swirls, one full of musical scale movement, one of random, decidedly computer-esque tones, swelling in the speakers as they appear on the desktop, coming out from his document to the charting and music documents published in his paper.

A tap on the music swirl unravels it, and a window appears to tell him that the score to that missing Stephen Foster reference was located by the Swiss net-

work. His paper's editor has linked it into the audio appendix of the paper and then ran it through his latest series of algorithms. A tap and he hears the music. He makes a voice notation at the appendix and another in his guide notes.

Back to the desktop where a tap for the charting program calls up the latest of his speculative transforms. "Somehow there's a calculus for musical clarity, and I'm close." The 3-axis analysis run the night before and inter-appl'ed into his paper already, plays the tune it analyzed as it animates the graphic series it generated. "Promising, better than before. Still not the music calculus I need, and today is the day I needed it."

Opening his work stack, his paper, and his CD controller gives him, in order, the sounds at the blackjack table he scored on last winter, the opening music sequence of his paper, and the name of the CD in his player, read in the voice of his ex-girl-friend, "I've really gotta change that thing."

The paper's the order of the day, "What to do, what to do? Overview, that's who," a tap on the blimp button gives him a quick "blimp" that hisses off into the sky as the paper drops away into the screen to Overview Distance. His paper's sections gel into the summary diagrams he's stared at far too many times already. He selects them all and taps once; each lights and sounds its size by pitching that hyena whoop his brother sent him. Just like yesterday, that last soprano "whEEP" reminds him that the section marked "Conclusions of our Investigation" is still just about empty. "In we go, then." The air hiss builds in volume and drops in pitch as his mental blimp drops him into his empty conclusions area.

"Didn't I make any notes on this?" The search dialogue is filled in with a satisfying cymbal bash of the return key and Search goes off humming through the guide notes. Search finishes with a "ding" that's more of a "doing" and he's looking at an empty list window. "Great, no notes. Then the time has come; I've got to come up with some conclusions."

Three hours later, the "network message received" tune breaks through the hail of keystrokes and document links, and he sits back with a smile, "I'll bet it's the call from the ski club." He closes the paper's editor app, pausing to hear the expected sequence of saves and back-ups to the right discs, and clicks the modem response app. The face the directory shows him is Miss Brownley's. The priority tone he hears is not a casual one...

Allright, settle down, you knew there'd be a student version of the Knowledge Navigator someday.

Again, we of the Blue Toolbox Sound Manager Development Group would like to state our position that Sonic System is not our idea of a replacement for the graphic desktop, nor do we think it a future requirement for any user. We know that a lot of people want to work with sound in the interface, including us, and in Big Bang, hope to provide the tools for that initial exploration.

### What we hope to provide in Big Bang:

We're certain that IF the world wants a Sonic System ability, it will be continually evolving, just like every other part of human interface. In the Big Bang release, we hope to provide only a basic tool set that can be used by developers and Mac System Software, and to encourage Human interface Groups to explore the earliest notions of a sonic interface.

In particular, we intend to look into putting hooks into the defprocs of the Control Manager, Window Manager, Menu Manager, the Finder, Dialogue Manager, Text Edit & Standard File. These hooks would call the 'new' sound trap we're sharing with Calamari, to call a body of code to be called the Sonic System. That code would refer to a table with the current sound resource assignments, dig out the appropriate one, massage its pitch if required and play it.

Given this set-up, the desktop will have a set of sounds the user can specify, and applications will have a set of sounds. Application developers will be able to specify the sounds their applications would make. By changing that sounds as the user moves through the application, developers would have the ability to use context sensitive sonic reinforcement.

We expect to see applications using one set of sounds for the document windows and editing, another for help functions in the applications, another for windows providing filing or printing functions, etc. ~~The limit of memory and disc space will be a mighty tight lid on these uses, so hopefully that will protect us from the problem of 'too many sounds to be meaningful.'~~ As the users experience the sonic interface and react to it, we expect a learning curve to appear to guide future development.

Some comments on the Sonic System idea seem necessary, as it is an unprecedented extension to something so critical as the Macintosh interface. Some notes then to answer the question:

### Why should we consider adding sound to our very precious interface?

Macintosh is the computer of the individual, the machine that seeks to fit most usefully the person using it. To say that that person understands and controls his world through constant integration of his senses is a bit obvious, a bit too commonly noted, and the most important fact we have to consider in any interface discussion. Our minds perceive by a dynamic combination of all the senses, and sound is a critical part of that synthesis. If the user can bring more of his internal tools to the interface, if his view of his computer-world can be made to be more like his view of the rest of his life, he will feel more at home, more confident, and be more productive.

When our graphic interface was introduced, many a computer pro comfortably noted that it was a pointless toy, a waste of processor power and user time, and no one would like it after the uniqueness wore off. "A command line is all the interface anyone will ever use." They had forgotten that the comfort of the user, his feeling of control and understanding, was more important to him, and to productivity, than some engineering abstraction. They had forgotten that everyone's mind works a little differently from anyone else's, and that use of sight is a greater commonality among users than a background in UNIX or CP/M.

That same group is now sagely observing that sound in a computer interface is a toy and unimportant to the user. "A beep is all the computer sound anyone will ever use." Again, let the users decide; it's their machine. If they don't like sound in the interface, we'll certainly allow them a path to switch back to that beep.

Macintosh product differentiation almost calls for this exploration. We are the computer that knows that the best interface is a tool fitted to the mind. Sound is an essential given in human perception. By incorporating it into our interface first, before the "Industry Leaders" copy us, again, we will be defining the new language of computer audio, we will be setting the path for other systems' use of sound, and we will affirm and strengthen our image and role as the computer developer that produces the best tool for the mind.

Personal computers have brought up new needs that we could address with sound:

**Notification of background tasks** - such as printing or messaging in network situations, a language of notices from networks or MultiFinder background tasks inform the user without disrupting his foreground activity, even when his foreground activity isn't in the computer.

**Navigation** - Confusion reigns as the graphic interface allows the user to roam about in information hyper-spaces and applications that offer many modes with which to explore and manipulate that information. As he moves through a massive stack or application, the user's uncertainty breeds discomfort, then distrust. He needs new cues for the new scales of information availability.

The Sonic System approach offers unobtrusive guidance here. Perhaps the areas of an application that provide the user with information on what this application is able to do, such as help areas, or info areas used to describe the fields of data stored, have a set of sounds for the scroll bars-in-progress, or page flips. Given that those sounds are different from the same functions when being used to manipulate or search the document the user is using the application to manipulate, then the current application mode is reinforced with each user action.

Everything he does responds to support his understanding of where he is in the application, and where he isn't. It sounds superfluous in discussing it, but to any of us, when in the position of working with a document using an application that shows nothing of its functional structure, that offers a meta-toolbox of invisible abilities linked invisibly in complex ways, the sort of subconscious, almost subliminal response that sets of sounds echoing the applications sets of abilities, becomes as necessary a cue as the click of a dashboard switch, or the sound of our housekey sliding into the lock the way we know it has to in order to work.

**Differentiation of graphic interface elements** - The difference between a document and an application is obvious only to an experienced user. We power users forget how confusing this 'interface for the rest of us' is to most of the people who use it. And, perhaps, we also forget just how reassuring each expected response from the interface can be to someone who doesn't "do computers for a living."

Part of becoming a power user is learning an application's use of icons on the desktop to indicate which file is the application and which the documents. The difference is not one that any amount of experience with Mac apps can teach; each developer will come to his own, quite unpredictable conclusion as to what 'arbitrary' abstract icon will represent what. If the Sonic System's Finder sounds a different sound for the document as opposed to the application, that initial learning comes easier, in fact, it comes immediately, and then, with each use of either icon, responds as the user expects, and, though the user may not "need" the reinforcement, though he may never notice it again, with each use his expectations of the icons and their application, his feeling of comfortable control of the application is reinforced.

Again, discussion abstracts the feeling, and engineering minds might want to dismiss it as un-necessary or wasteful; the user won't be so inclined to dismiss his comfort, even though he can't point to what it is that has increased it.



The few audio capabilities available today have already given rise to wholly unpredicted uses and interest in sound as an interface element.

"**Scientific Auralization**" is currently being used on many data-reporting projects, from super-computers, predicting upper atmospheric transitions, to urinalysis. The idea is to express a data analysis' complex output in some audio format and let the considerable pattern-matching abilities of the ear alert the user to subtle but crucial anomalies. I mention this not as a goal worthy of specific system code, but as an indication of the unexpected uses of sound, which were recognized only after the tools to try them were produced. Note also the bandwidth and sophistication of this unused sense.

**HyperCard** gave users new control of sound, and they used it more than anyone expected, not just as a controlled event but as audio-animation for buttons and user-actions. True, more than a few body functions were sampled and played through many a stack, but the popularity of sounds in HyperCard, and now as used through SoundMaster is already established, and longer-lasting than the pundits had predicted. Few users leave any of the sounds on all the time, but they go out of their way to get and share the ones they like.

### Epilogue

Leading the development of sound in personal computers is Dangerous Stuff! The risk we take in being the first in this exploration is an enormous one: if the first attempts are poorly tested, if the sounds and uses they are put to are not carefully selected, then a permanently silly image is a possibility for both sound in all applications and in the Mac in general. The connotations of specific sounds are clear to any listener, and vary widely between listeners. Heavy testing of any sound used in an application must be a prerequisite for any commercial effort! Certainly, a careful watch on the developers' use of sound, and, perhaps, some organized group discussions on the topic may be in order.

Many users will never want sound, many will always want it, and the rest will 'take-it-or-leave-it.', at least that's what they'll report if asked. The same range of opinions was and is honestly reported about the desktop interface!

Second, this is Big Impact Stuff! Our minds are built to seek the new sensation.

Video games dramatized and commercialized a fact personal computers may have forgotten: it is the new ability that draws attention and comment. A new avenue of communication fascinates the user, and demands that he "grok" it.

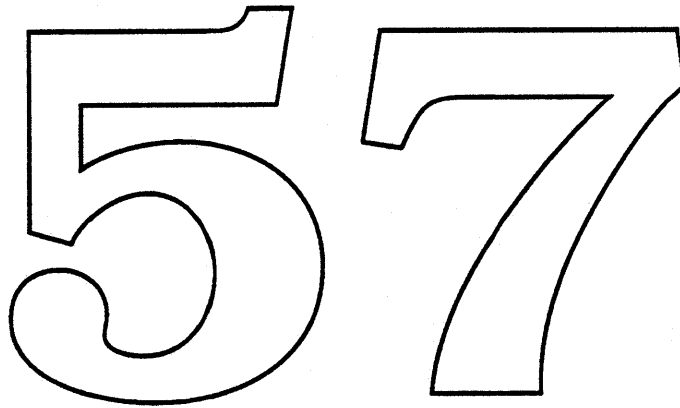
Even more than most senses, the reaction to sound is inherently non-verbal, emotional, personal, individual, involving, and imaginative, and it is uniquely so in each and every user that encounters it. Sound has never been used interactively in serious computer interface. This is an enormous potential for us to explore and develop.

So, how do we investigate sound in the interface?

The Sonic Finder is to be released soon, with enhancements allowing the user to easily stop the sounds, or to replace them as sets. It is felt by everyone involved, as far as I have been able to tell, that it is essential to the Sonic Finder that the user be allowed some easy control over it, and that no two users will be found that like the same set of sounds. Where information is expressed by audio variation, such as when an object's size is expressed by varying the pitch of

the sound, those variations will be applied to whatever set of sound is selected in the same way. We hope to determine a set of useful language basics this way, and will then apply them consistently.

After the Sonic Finder testing we plan to incorporate sound-calling hooks into the xDEFs of the system to produce the Sonic System. The idea then would be that any application could use these hooks to make the interface elements within the application respond with sound. At this point, sound becomes available as a tool for navigation, warning, or entertainment as the developer sees fit.



## 6. Future

This area might better be titled “not for Big Bang;” so much of the material we’ve described is preparation for future Mac sound hardwares and markets, it’s not clear what part of it is the future and what is not. Even in considering inclusion in Big Bang as the criterion, there’s some question. For example, Text-to-Speech improvements, although well underway, cannot be considered a guaranteed part of Big Bang, because we don’t have control over it at this point.

### 6.1 Text-to-Speech

**The situation:** “Macintalk sucks” so badly that even if we told it say that simple phrase, we probably couldn’t understand it. Our inability to improve it lies in long-frozen legal issues, but our ability to replace it is vibrant, current and being actively pursued.

Currently a group in ATG has an investigation ongoing with a third party developer. This investigation is in too early of a stage to insure text-to-speech by Big Bang, or for that matter, to be able to report that MacinTalk’s successor has been found.

The situation today then is as follows. Voice samples are used by their approach and those voice(s) have been selected, sampled, and are now being processed by the developer. In a month (or three) there will be code returned to us that would represent the voice quality this new process would provide. At that point the interest group, which includes Blue Sound Manager, will convene and evaluate this approach. If it is of interest the developer will then be funded to complete work on a version for system inclusion. Throughout this period the interest group will meet to discuss issues of system integration, quality, call structure etc.

With luck, and adequate development money, we’ll have a female and a male voice available, but that is not expected until the end of the year, so it will not be a Big Bang feature, I’m afraid.

Of course, we have preliminary ideas of how to hook text-to-speech ability into the system. One possibility is to add sound manager synths for the various Mac models. The details of the calls that should be supplied are, as suggested above, open to comment or suggestion.

Nothing compresses speech like text-to-speech, no other audio approach offers such interactive potential. On the other hand, nothing else sounds so much like a talking computer; so where a Hollywood-defined computer-speech effect, or an interactive, memory-cheap verbal intelligibility are required, text-to-speech is a must. It’s never going to be like talking to another human, that doesn’t mean it is useless.

### 6.2 Miscellaneous

**We actually do know something of Sound Manager’s future:** It will be the system service that integrates developments in sound from ATG and third party developers with the abilities of the machine they run on. It will be the Sound Manager that provides hooks for new hardwares and processor power. That’s what we’ve prepared for this year, with the new ability to handle multiple channels, to get a handle on gracefully degrading performance to fit to available CPU power, with better Sound Manager self-reporting and management with the new Status Panel, with the important improvements to the sampled sound synth, MACE and Sound Manager support for Calamari.

**Sound is only now becoming available to the Mac developers:** As noted above, our efforts for Big Bang are our response to the needs and interests we found unaddressed by the current Sound Manager. If they are successful then we should see a year of new uses of sound in the Mac of unprecedented quantity, and unpredictable quality.

Calamari is introduced in January as a licensed product and should be released with Big Bang as a system extender. The world of MIDI and Mac should literally burst open, the Mac-rags should be delighted to report on such a graphic, easy-to-describe, and evidently valuable development as Calamari, and the Mac should end the year as the unchallenged owner of the computer/music market.

After Big Bang, we expect to see some changes for the better in non-MIDI sound and the Mac. We're hoping to see music behind games, many explorations into multi-media emitting and education based on DJ, much more use of prompts from MultiFinder background using the multi-channel abilities of the Big Bang sound manger, and MACE sound compression used on most sampled sound applications.

If the Sonic System makes it into Big Bang, some of the effort and interest that produced third party bandaids for our missing audio abilities, will be able to be applied to using sound in the interface. It's not really a matter of a total invasion of all the interfaces Mac will ever have again; more realistically, the Sonic System will be applications trying to use sound to clear up some of the confusion and learning curve problems that now separate the Power Users from the new-comers.

Given a Sound Manager with these abilities, as new audio needs and functions emerge from ATG, Multi-Media groups, HyperCard, and the developers, the Sound Manager offers them each the connection to the system for their solutions, a place ready to integrate their output into the audio abilities already present. As new machines go to stronger and faster processors, we will add new versions of our old synths that optimize those new performance horizons.

When a DSP appears on our motherboard, and the processing of sound becomes so vastly less time-limited, the improvements we put into the Sound Manager of Big Bang will be fully used: channels of CD-quality sound will be assigned and mixed as easily and as productively as in a sound studio today, speech generated over background music will be a standard for presentations from single Macs, sound effect and music manipulations that rival all but the most powerful of today's professional quality synthesizers will be available with minor loading on the main processor, PBX functions will be available on the desktop for personal use, and there will be available to every developer's and user's imagination the audio power to support 'any' sound use in the interface of presentations, education, games, and applications.

**A future of responding to the future is the expectation then:** No one is ready to predict what multi-media might become over the next two years. It may be a substitute for professional editing studios, it will more likely be a style of access to gigabyte storage technologies that are just around the corner.

There is no one that can say which limits in Mac sound will be the most troublesome to developers, or how Apple might decide to distribute sound hardware over the next years' machines.

We've tried to anticipate all of these answers and positioned our development efforts for Big Bang accordingly. We'll see how we did, then do it again.

57



**MACINTOSH  
AUDIO COMPRESSION  
(V 1.0)**

**External Reference Specifications**

**57**

Hugh Svendsen  
Ron Dumont

January 10, 1989

**\*\*\* Apple Confidential \*\*\***

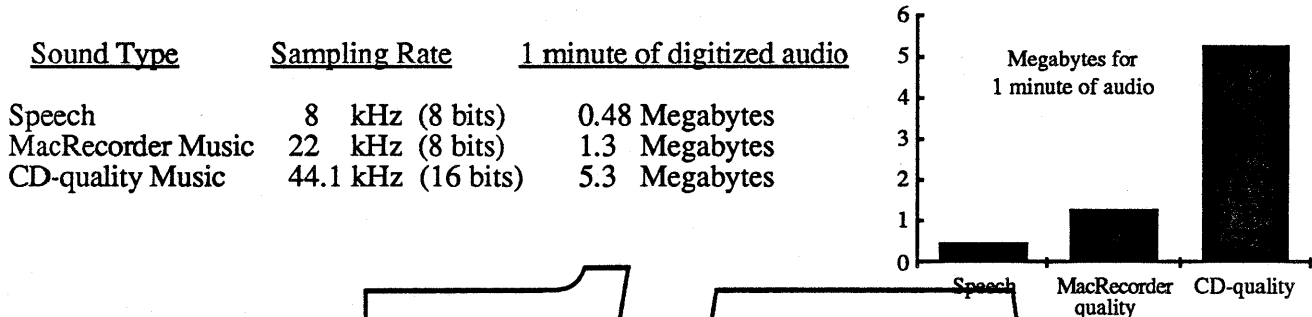
## Table of Contents

|         |                                                         |    |
|---------|---------------------------------------------------------|----|
| Chapter |                                                         |    |
| 1.0     | Statement of Purpose . . . . .                          | 1  |
| 2.0     | Data Sheet . . . . .                                    | 2  |
| 3.0     | Theory of Operation . . . . .                           | 6  |
| 3.1     | 3:1 Compression / Expansion . . . . .                   | 6  |
| 3.2     | 6:1 Compression / Expansion . . . . .                   | 9  |
| 4.0     | Sound Manager Integration . . . . .                     | 13 |
| 4.1     | Overview . . . . .                                      | 13 |
| 4.2     | Data Structures . . . . .                               | 14 |
| 4.3     | Command Integration . . . . .                           | 17 |
| 4.4     | Constants . . . . .                                     | 19 |
| 4.5     | Data Types . . . . .                                    | 20 |
| 5.0     | ACE and Related Sound Commands . . . . .                | 22 |
| 5.1     | ConvertCmd . . . . .                                    | 22 |
| 5.2     | SizeCmd . . . . .                                       | 24 |
| 5.3     | BufferCmd . . . . .                                     | 26 |
| 5.4     | ContinueCmd . . . . .                                   | 27 |
| 5.5     | VersionCmd . . . . .                                    | 28 |
| 6.0     | Recommended Practices . . . . .                         | 29 |
| 6.1     | Installation . . . . .                                  | 29 |
| 6.2     | Recording / Playback . . . . .                          | 29 |
| 6.3     | Examples of Usage . . . . .                             | 29 |
| 7.0     | Testing Considerations . . . . .                        | 30 |
| 7.1     | Ideas for Testing the ACE Snth's . . . . .              | 30 |
| 7.2     | Ideas for Testing the Expansion-Playback Mode . . . . . | 31 |
| 8.0     | Design Decisions . . . . .                              | 32 |

## 1.0 STATEMENT OF PURPOSE

### Background

Audio Compression decreases the amount of memory space required to store digitized audio information. This is important because digitized audio information require large amounts of memory for relatively short amounts of time. For example, one minute of CD-quality music requires 5.3 megabytes. One minute of music digitized on a Macintosh via the MacRecorder™ from Farallon requires 1.3 megabytes. One minute of telephone-quality speech requires slightly less than 0.5 megabytes.



Thus, many Macintosh multimedia applications, including stackware, are shipping with a high percentage (up to 80% in some cases) of the physical media space being occupied by audio information.

### Benefits

- \* Increase the content and richness of multimedia applications
- \* Ease distribution issues for VARs by reducing the number of disks required
- \* Spur new applications (e.g. Talking Dictionary, Language Learning, Transactional Translation, ...)

### Purpose

The purpose of the Audio Compression product is to:

- \* Increase the multimedia capabilities of Apple's CPUs
- \* Help VARs differentiate their Apple-based applications over the (PC-based) competition
- \* Ultimately sell more Apple CPUs and raise barriers to competition

### Specific Project Goals

- \* Develop and deliver a software-only Audio Compression / Expansion (ACE) Toolkit for the Mac Plus, Mac SE, Mac II and compatible follow-on CPUs (such as Esprit and Mongoose)
- \* Provide high quality audio compression that handles both speech and music
- \* Provide real-time playback of compressed audio that is compatible with existing applications using playback of digitized audio
- \* Complement and enhance existing third party audio digitization products, such as MacRecorder

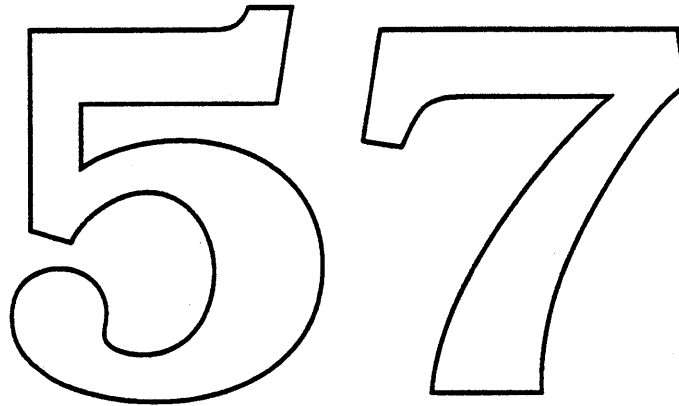


## 2.0 DATA SHEET

**ATTENTION: The following issue has not yet been resolved:**

**\* Real-time 6:1 compression**

This is designed to be used by providers of digitizer products only and not by general software developers. It is unresolved at this time, how we should distribute and support the digitizer-specific software. Until this is resolved, we have explicitly included the real time 6:1 compression algorithm in the toolkit which would be widely distributed.



## Implementation

The Audio Compression / Expansion (ACE) Toolkit is implemented as a software-only solution that uses only the built-in processor in the target CPUs along with the existing built-in sound hardware.

### CPUs Supported

- \* Mac Plus
- \* Mac SE
- \* Mac II
- \* Future Macintosh systems

### Compression / Expansion

- \* 3:1 - Very high quality compression for both speech and music
- \* 6:1 - High compression, good quality for both speech and music

### Memory Requirements (in bytes)

| <u>Module</u>                                | <u>Code Resource ID</u> | <u>Disk Space</u> | <u>RAM Space</u> |
|----------------------------------------------|-------------------------|-------------------|------------------|
| 3:1 Buffered Compression / Expansion         | snth 11                 | 7758              | 7756             |
| 6:1 Buffered Compression / Expansion         | snth 13                 | 9128              | 8818             |
| Real-time playback on ASC-based systems (1)  | snth 4101               | 10,584            | 10,284           |
| Real-time playback on Sony-based systems (2) | snth 4101               | 11,107            | 10,800           |

Notes:

- (1) ASC = "Apple Sound Chip" currently used in Mac II, Portable and later Macintosh models  
 (2) Sony = "Sony Sound Chip" currently used in the Mac Plus and original Mac SE

### Processor Bandwidth Required for Buffered Expansion

| <u>Expansion Ratio</u> | <u>Mac Plus</u> | <u>Mac II</u> |
|------------------------|-----------------|---------------|
| 3:1                    | 39%             | 24%           |
| 6:1                    | 42%             | 25%           |

## Major Functions

**Compression:** Takes a buffer of digitized audio and compresses it by 3:1 or 6:1. Compression may be done in place (use the same buffer) or the output may be directed to another buffer.

**Expansion Playback:** Takes a buffer of compressed audio information, expands it and plays it through the sound hardware. This process runs in real-time across all supported Macintosh machines.

**Buffered Expansion:** Takes a buffer of compressed audio information and expands it into another memory buffer. The expanded data may then be played out as normal digitized audio. The amount of real time required to perform *Buffered Expansion* is less than that required for *Expansion Playback* since it does not incur the extra overhead of the actual playback process.

## Compression / Expansion Functionality

**3:1 compression / expansion** is well-suited for music and other high fidelity sounds. It preserves the full bandwidth of the sampled signal. 3:1 compression is not real-time on the Mac Plus and Mac SE. The 3:1 expansion runs in real-time across all Macintosh machines.

**6:1 compression / expansion** allows a relatively high compression ratio with good quality results. It would be used for speech, mid-quality music or in cases where the highest compression ratio must be used to maximize the amount of shippable audio information. It reduces the frequency bandwidth of the original signal by a factor of 2 to allow for higher compression. 6:1 expansion is real-time on all Macintosh machines.

**Real-time 6:1 compression** provides real-time compression across all Macintosh machines; the other 6:1 technique produces higher quality audio but does not compress in real-time on the Mac Plus and Mac SE. This technique is mainly designed for compression of speech sounds for applications such as voice-annotated documents or dictation. It produces compressed sound data compatible with the higher quality 6:1 technique so that only a single 6:1 expansion routine is required.

| Computer                  | 3:1 Compression | Real-Time 6:1 Compression | 6:1 Compression | 3:1 and 6:1 Expansion | Stereo Expansion / Playback | Mono Continuous Playback | Sample Rate Conversion |
|---------------------------|-----------------|---------------------------|-----------------|-----------------------|-----------------------------|--------------------------|------------------------|
| Mac Plus                  | Non-Real-Time   | Real-Time                 | Non-Real-Time   | Real-Time             | No (1)                      | Yes (2)                  | No                     |
| Mac SE                    | Non-Real-Time   | Real-Time                 | Non-Real-Time   | Real-Time             | No (1)                      | Yes (2)                  | No                     |
| Portable                  | Non-Real-Time   | Real-Time                 | Non-Real-Time   | Real-Time             | No (1)                      | Yes                      | Yes (3)                |
| Mac II and 030 successors | Real-Time       | Real-Time                 | Real-Time       | Real-Time             | Real-Time                   | Yes                      | Yes (3)                |

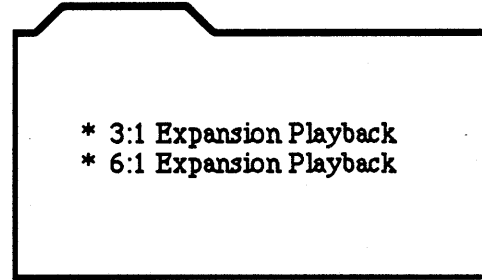
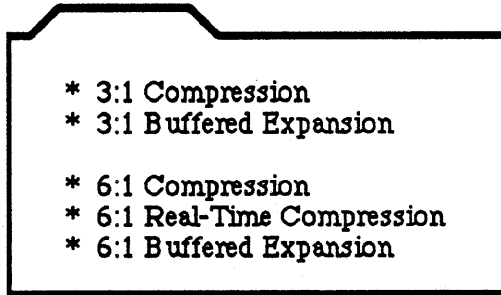
Notes:

- 1) If a stereo snd is received on these machines, only the right channel is played.
- 2) Multiple buffers may be played continuously on these machines only when the buffers expand into a multiple of (3 \* 370) bytes.
- 3) The drop sample tuning technique is used to perform sample rate conversion on these machines.

## Sample Rates

A sample size of 8 bits linear is presumed. The 3:1 and 6:1 compression / expansion methods are independent of sample rate. The best audio quality is achieved when the input sampling rate matches the output rate of the sound hardware built into the CPU (i.e. 22.254 kHz). Due to processor bandwidth constraints, the Mac Plus and the Mac SE cannot perform sample rate conversion during real-time expansion playback. Sounds not recorded at 22.254 kHz are played back at a different pitch on the Mac Plus and the Mac SE. Thus it is highly recommended that all sounds be recorded at 22.254 kHz to achieve consistency of playback across all platforms.

## Distribution



The Expansion Playback modules will eventually be part of the Macintosh system disc and are thus contained in a separate folder. The Compression and Buffered Expansion modules need to be included with the applications which make use of them. Until the Expansion Playback modules are included in the Macintosh system disc, developers will also need to include these modules in their applications.

## Compatibility

### Software Compatibility

*Sound Manager Compatibility:* ACE will be compatible with the Sound Manager. Existing applications which make use of the Sound Manager to play digitized audio will transparently be able to play compressed audio, without modifications to the application. ACE will be implemented inside the Sound Manager and will have its own commands as well as transparent-mode operation.

*File Standards:* A Compressed Audio File Format (CAFF) will be created, based on the existing Audio Interchange File Format (AIFF). There will be no modifications to the existing AIFF, and the CAFF will very closely resemble AIFF but with an emphasis on *storage* rather than interchange compatibility.

### Hardware Compatibility

*Apple Sound Hardware:* ACE will run on all existing and all planned sound hardware for Macintosh systems. This includes the Mac Plus, Mac SE, Mac II, future CPUs and any new versions of Apple sound hardware.

*Apple IIGS:* A version of ACE is already available as part of the system toolkit for the IIGS. It supports compression rates of 2:1 and 8:3. The compressed data is not compatible with the Macintosh ACE toolkit since we are taking advantage of the higher processing capabilities of the Macintosh to provide higher quality / compression ratios. If it becomes a priority, the IIGS ACE algorithms may be ported to the Macintosh or a new algorithm, compatible across both systems, may be developed. At this point, IIGS compression compatibility is not an issue since the prominent usage is for developers to provide increased audio information when shipping applications. No network-based file sharing applications for compressed audio, such as voice annotated documents, are available or under development. It is easy for developers who use both systems to take the original digitized audio information and run it through the appropriate compression toolkit.

*Third Party Digitizers:* ACE will be available to third party digitizer providers such as Farallon (MacRecorder) and Articulate Systems (Voice Navigator), Digidesign (Sound Accelerator), Authorware (SoundWave) and Magnum Software (TeleFlex) for inclusion with their products. This will allow them to enhance their product line and maintain audio data compatibility with other applications such as HyperCard.

3.0 THEORY OF OPERATION

3.1 3:1 Compression / Expansion

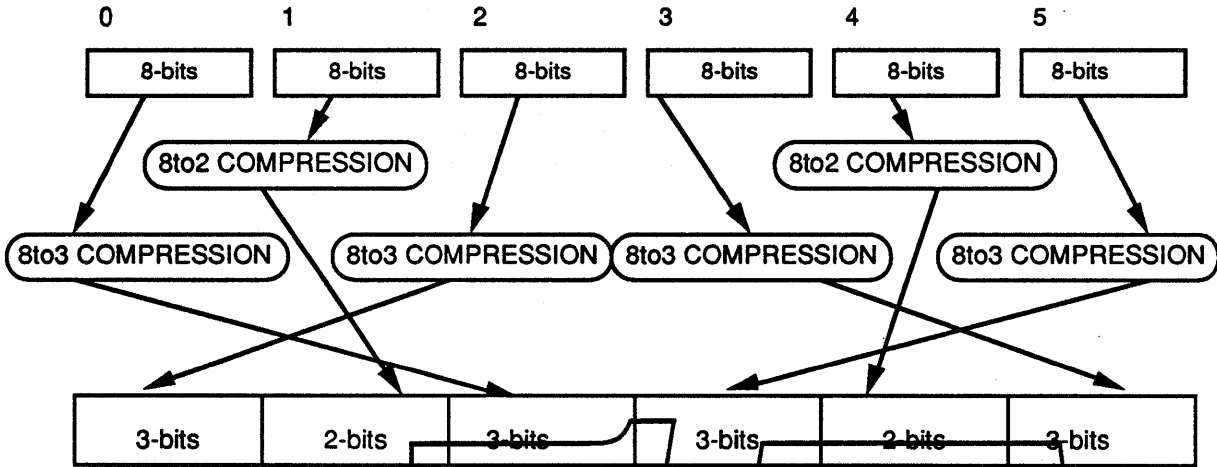


Figure 3.1.0: Bit Packing Scheme for 3to1 Compression

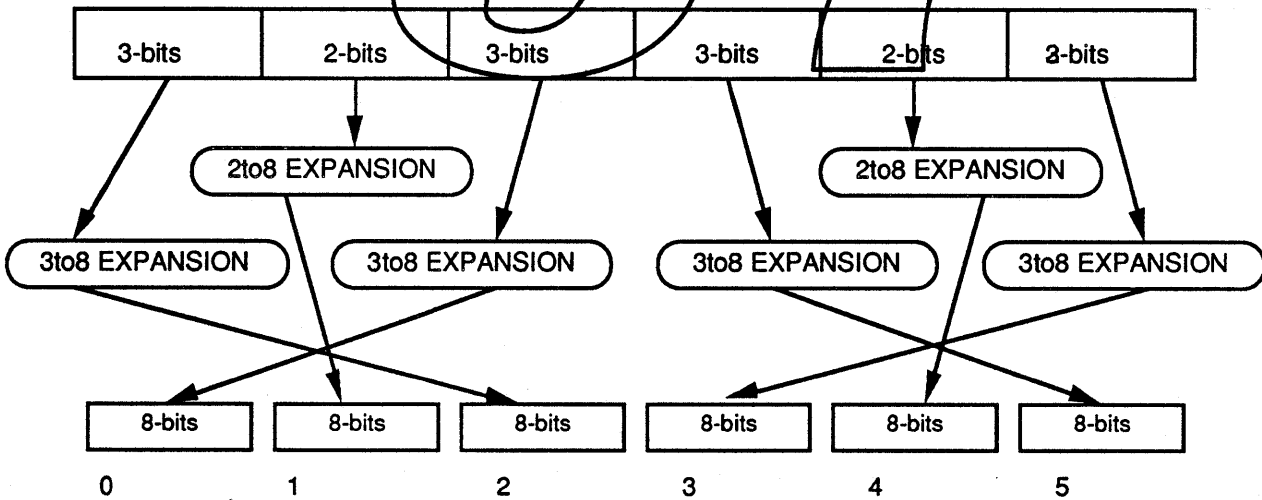
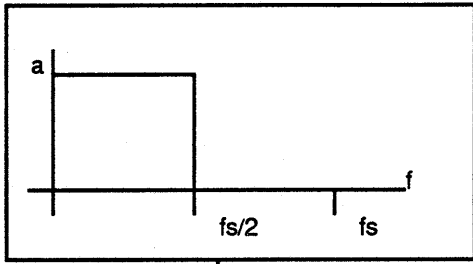
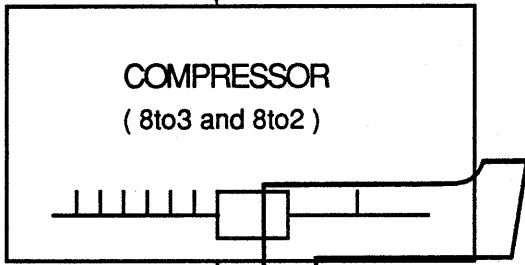


Figure 3.1.1: Bit Packing Scheme for 1to3 Expansion



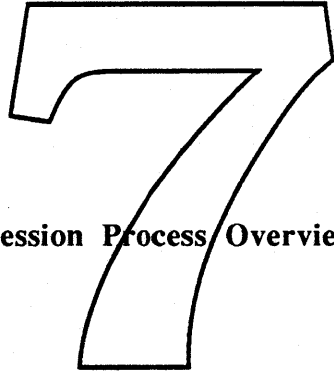
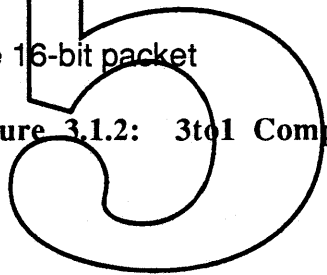
Frequency Range of Input Signal. For a sample rate of 22kHz, the highest frequency in the Input Signal is 11kHz.

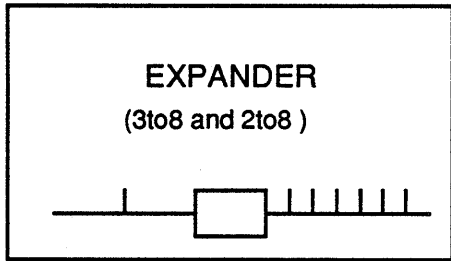


The 3to1 Compressor takes as input six 8-bit samples and produces one 16-bit packet.

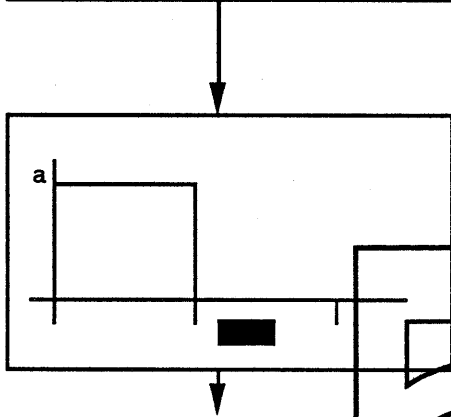
OUTPUT: one 16-bit packet

Figure 3.1.2: 3to1 Compression Process Overview





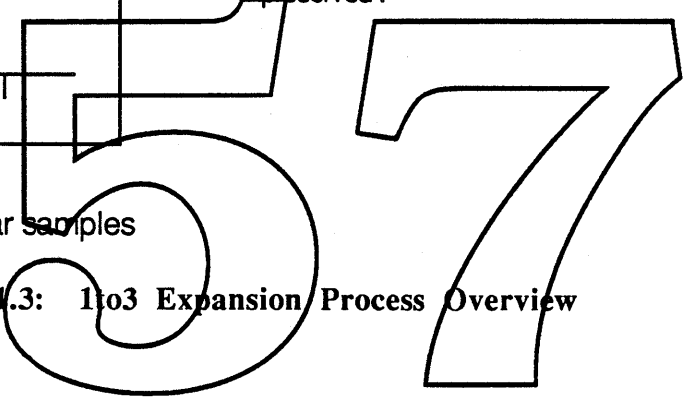
The 1to3 Expander takes in one 16-bit packet and produces six 8-bit linear samples.



The Frequency Range of the Reconstructed Signal. Note that the full bandwidth is preserved.

OUTPUT: six 8-bit linear samples

Figure 3.1.3: 1to3 Expansion Process Overview



57



3.2 6:1 Compression / Expansion

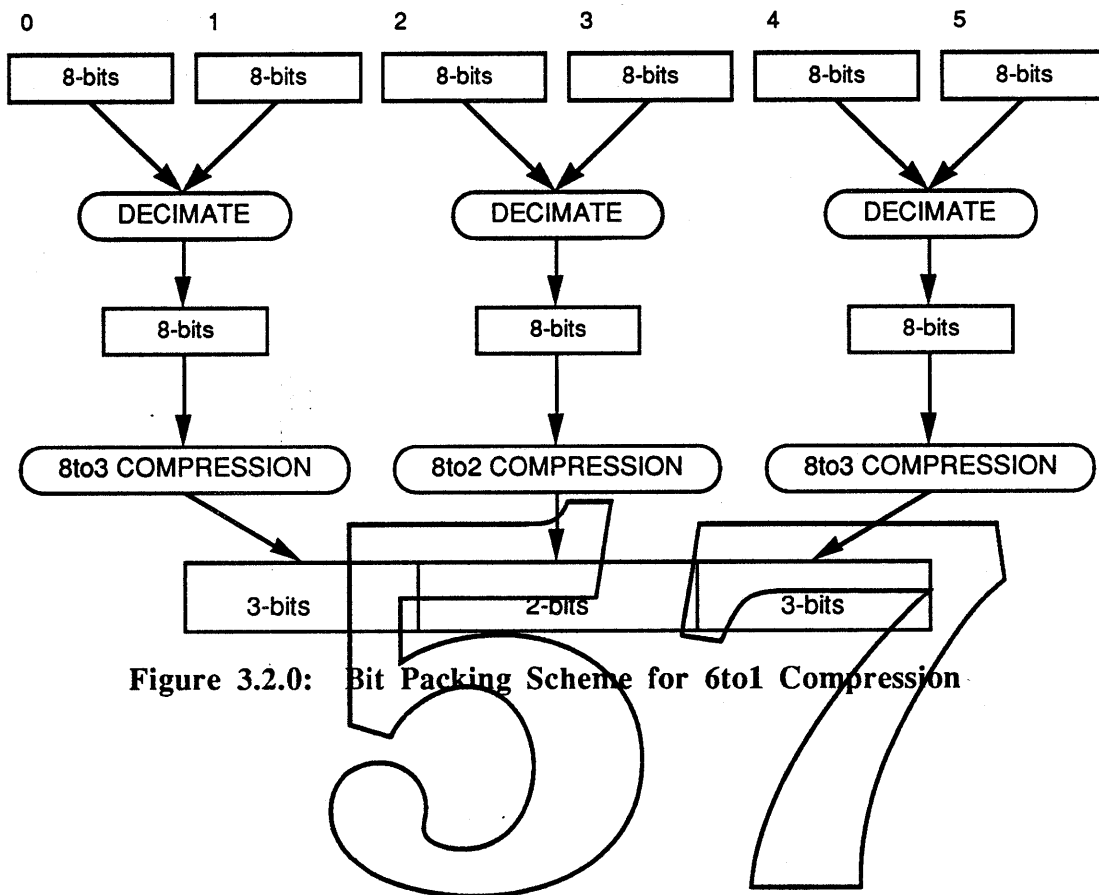


Figure 3.2.0: Bit Packing Scheme for 6:1 Compression

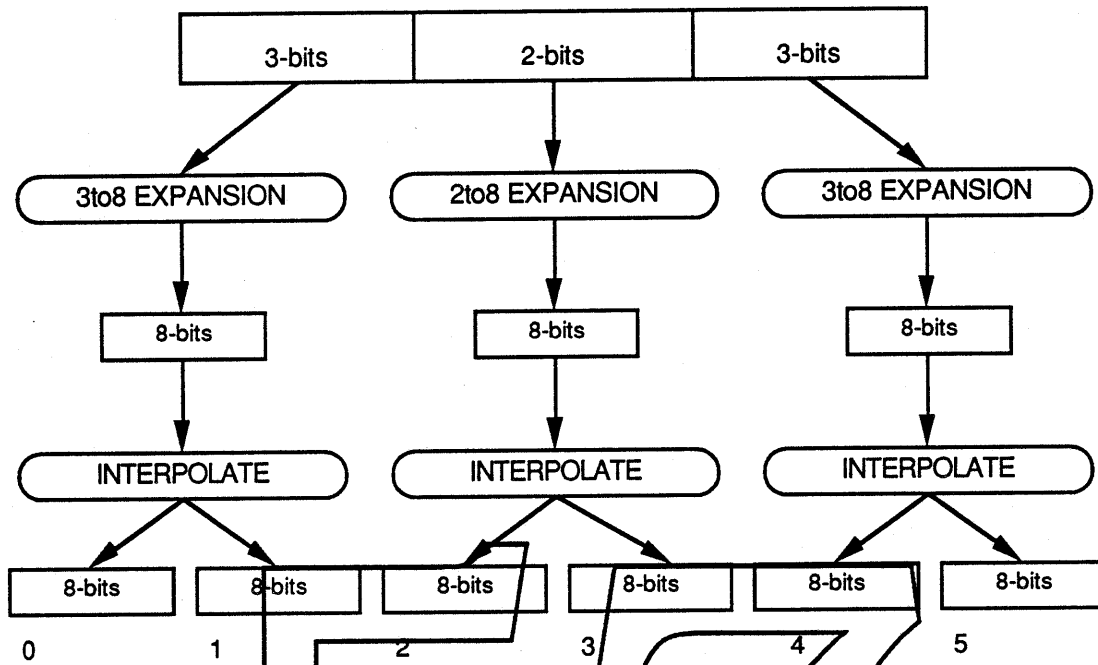


Figure 3.2.1: Bit Packing Scheme for 1to6 Expansion

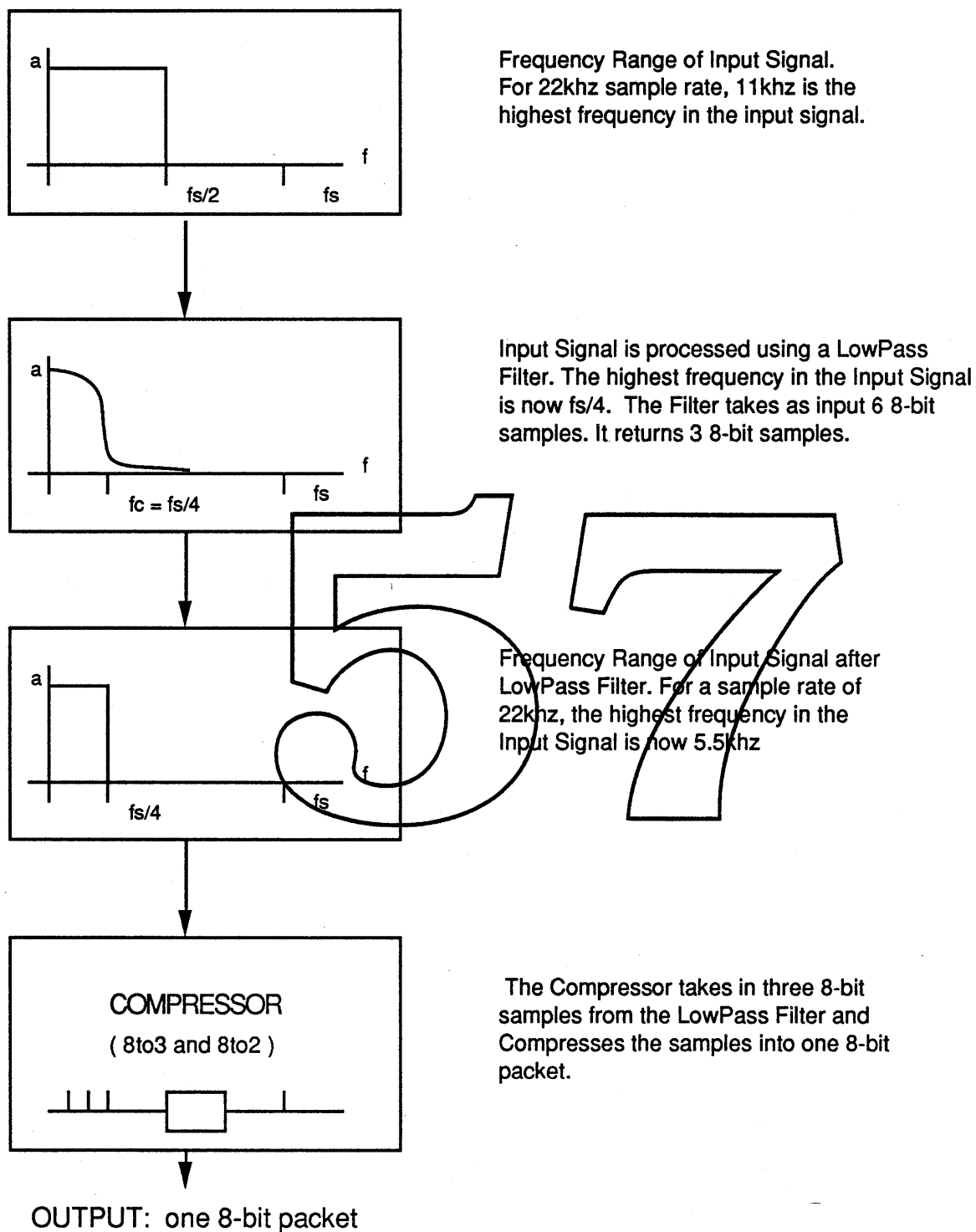
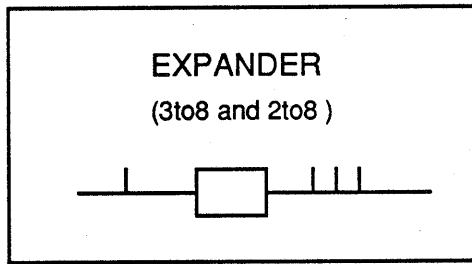
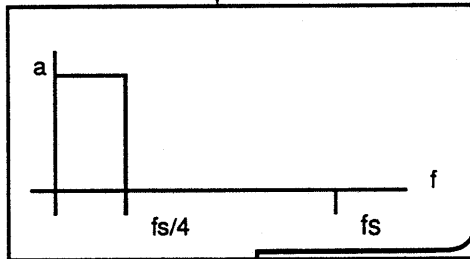


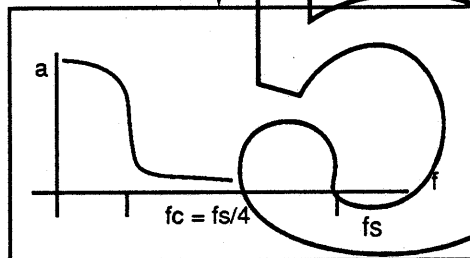
Figure 3.2.2: 6to1 Compression Process Overview



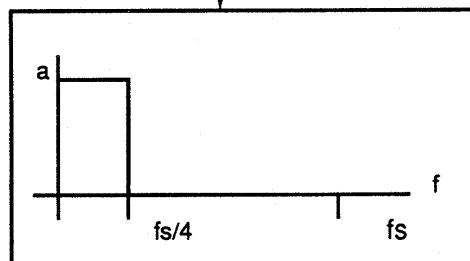
The Expander takes in one 8bit packet and expands it into three 8-bit samples.



The Frequency Range of the Reconstructed Signal. The bit rate of this signal is 1/2 the bit rate of the original Input Signal.



To return the Reconstructed Signal to the bit rate of the original Input Signal, run it through a LowPass Filter and perform Interpolation. The Filter takes three 8-bit samples as input and returns six 8-bit samples. The cutoff frequency of the filter is  $fs/4$ .



The Frequency Range of the Reconstructed Signal (output). Note that the highest frequency in the Reconstructed Signal is 1/4 the sample rate of the original Input Signal.

OUTPUT: six 8-bit linear samples

Figure 3.2.3: 1to6 Expansion Process Overview

## 4.0 SOUND MANAGER INTEGRATION

### 4.1 Overview

There are 3 main functions provided for Audio Compression / Expansion:

#### 1. COMPRESSION

Takes a memory buffer of digitized audio and compresses it into another or the same buffer. Consecutive calls to the *Compression* routine may be made for audio segments which do not easily fit into a single buffer. Version 1.0 supports two compression ratios: 3:1 and 6:1. The 3:1 algorithm provides extremely high quality audio at the full digitizing bandwidth. The 6:1 algorithm provides a higher compression ratio for those applications requiring more audio information in a limited space (e.g. floppy disk).

#### 2. EXPANSION PLAYBACK

Takes a memory buffer containing compressed audio data, expands it and plays it out through the internal sound hardware in real time. The output is heard over the CPU speaker or through an external speaker via the audio jack on the rear of the CPU box. *Expansion Playback* runs in real time on the Mac Plus, Mac SE, Mac II and future Macintosh systems.

#### 3. BUFFERED EXPANSION

Takes a memory buffer of compressed audio data and expands it to normal audio data in another buffer. The expanded audio data may then be played through the Sound Manager with minimal processor overhead during playback. This command is useful for applications which require screen updates or user interaction during playback, such as animation applications. The audio quality is slightly higher with this method since we can take more time to expand the data than in the *Expansion Playback* routine. The *Buffered Expansion* routine handles both the 3:1 and the 6:1 compression ratios.

The COMPRESSION and BUFFERED EXPANSION routines will be integrated into the Sound Manager as Snths, called the *ACE Snth's*. As such it will not be necessary to allocate a sound channel to use one of these snth's. Access to the ACE Snth's will be made exclusively through the SndControl function call. This differs slightly from traditional snths in that the *ACE Snth* cannot be linked to a sound channel and as a result should not be called with SndNewChannel and SndDisposeChannel. Since the *ACE Snth*'s can not be placed at the end of a soundChannel queue, it cannot be called using the SndDoImmediate or SndDoCommand function calls. The only way to access the ACEsnth's is through SndControl.

The EXPANSION PLAYBACK routine will be integrated into the present Sampled Sound Synthesizer. The sound commands presently used to invoke Sampled Sound Synthesizer will still be used, but the data structures passed by the commands are different for the EXPANSION PLAYBACK routine.

## 4.2 Data Structures

The ACE Snth adds four new data structures into the Sound Manager:

- \* ConversionBlock
- \* CmpSoundHeader
- \* StateBlock
- \* LeftOverBlock

The existing SoundHeader structure is also changed.

The **ConversionBlock** is a 6-word data structure used presently only by the ConvertCmd. The inputPtr and outputPtr fields are both pointers to CmpSoundHeaders. The destination field lets the Sound Manager know where in the output CmpSoundHeader to put the samples it generates. The unused field is reserved for future use by Apple. The ConversionBlock Structure Definition appears in Figure 4.1 below.

```
typedef struct
{
short
short
CmpSoundHeader
CmpSoundHeader
}
ConversionBlock, *ConversionBlockPtr;
```

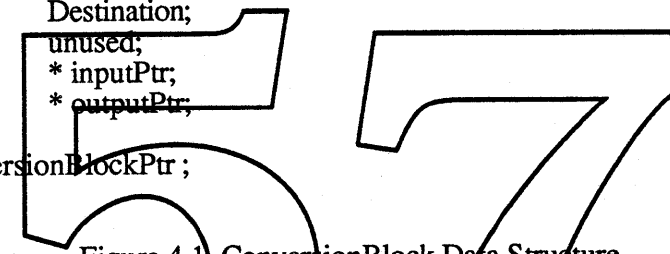


Figure 4.1 ConversionBlock Data Structure

The **CmpSoundHeader** is a data structure for use only by the convertCmd, sizeCmd, BufferCmd, and ContinueCmd SndCommands. It is the in memory representation of the CAFF ( Compressed Audio File Format ) file storage standard. The CmpSoundHeader is a logical extension of the old SoundHeader. As such, the first seven fields of the CmpSoundHeader, with the exception the encode field, are identical to those of the old SoundHeader. The encode field of the CmpSoundHeader is used to determine the meaning of the 21 words that follow it. The CmpSoundHeader appears below in Figure 4.2.

```
typedef struct
{
char *samplePtr;
unsigned long length;
Fixed sampleRate;
unsigned long loopStart;
unsigned long loopEnd;
unsigned char encode;
unsigned char baseNote;
unsigned short numChannels;
unsigned short sampleSize;
extended AIFFSampleRate;
Ptr MarkerChunk;
Ptr FutureUse1;
Ptr FutureUse2;
Ptr StateVars;
```

```

Ptr          LeftOverSamples;
unsigned short compressionID;
unsigned short packetSize;
unsigned short snthID;
unsigned short FutureUse3;
char         sampleArea[0];
}
CmpSoundHeader, *CmpSoundHeaderPtr;

```

Figure 4.2 CmpSoundHeader Data Structure

The **samplePtr** indicates the location of the compressed sound frames. If samplePtr points to nil then the frames will be located in the sampleArea of the CmpSoundHeader. Otherwise the samplePtr points to a buffer that contains the frames.

The **length** field indicates the number of frames contained in the CmpSoundHeader.

The **sampleRate** field indicates the sampleRate at which the frames were sampled before compression, as expressed in Apple's Fixed point representation.

The **loopStart** and **loopEnd** fields should be used to indicate the loop points of the sound before compression.

The **encode** field of the CmpSoundHeader is used to specify the data structure being used. Presently three *SoundHeaders* are supported by the Sound Manager. They are the SoundHeader, the ExtSoundHeader and the CmpSoundHeader. If the encode field is set to zero the data structure is the original SoundHeader. If the encode value is 1, the data structure is an ExtSoundHeader. An encode field of 2 indicates the data structure is a CmpSoundHeader.

| <u>encode</u> | <u>constant</u> | <u>data structure</u> |
|---------------|-----------------|-----------------------|
| 0             | stdSH           | SoundHeader           |
| 1             | extSH           | ExtSoundHeader        |
| 2             | cmpSH           | CmpSoundHeader        |

Figure 4.3 Summary of present encode values

The **numChannels** field is used to indicate how many packets are in a frame. If the sound is a mono sound the number of channels is one. If the sound is a stereo sound the number of channels is 2.

The **sampleSize** field indicates the size of the original non-compressed sample. Presently the Sound Manager only works with 8-bit samples.

The **AIFFSampleRate** field indicates the sampleRate at which the frames were sampled before compression, as expressed in IEEE floating point representation.

The **MarkerChunk** is used to specify synchronization information.

**StateVars** is a pointer to a **StateVarBlock**. The **StateVarBlock** is used to store the state variables for a given Algorithm across consecutive calls.

**LeftOverSamples** is a pointer to a **LeftOverBlock**. This block can be used to store samples that would have been truncated across Algorithm invocations.

The **compressionID** is used to identify the compression Algorithm used on the samples in the **CmpSoundHeader**. If the **compressionID** is set to 0, it indicates that the samples are not compressed.

| compressionID  | constant | Comment                           |
|----------------|----------|-----------------------------------|
| notCompression | 0        | normal samples, no compression    |
| twoToOne       | 1        | Implemented only on the IIGs      |
| eightToThree   | 2        | Implemented only on the IIGs      |
| threeToOne     | 3        | Implemented only on the Macintosh |
| sixToOne       | 4        | Implemented only on the Macintosh |

Figure 4.4 Summary of present compressionID's

The **packetSize** field indicates the size of the smallest element that a given expansion Algorithm can work with. This number is specified in bits. The **packetSize** for the 3to1 Algorithm is 16 bits, the **packetSize** for the 6to1 Algorithm is 8 bits.

The **snthID** field indicates the Resource ID number of the Sound Manager snth that was used to compress the packets contained in the **CmpSoundHeader**. A 3:1 snd would have a **snthID** of 11, and a 6:1 snd would have a **snthID** of 13.

When the **samplePtr** field points to nil the sample frames follow the data structure in the **sampleArea**.

The **StateBlock** is to be used for storing the state variables of the various Algorithms across consecutive calls. This is necessary because the code for the snth's exist as code resources, and the resource is only locked down for the duration of the call. Being able to save the state of a given Algorithm is useful for other reasons also, it allows the programmer to be able to process any number of channels simultaneously by restoring and saving the state of each channel on each call. This, by the way, is how the real-time stereo playback is achieved on the Apple Sound Chip machines.

```
typedef struct{
short                stateVar[StateBlockSize] ;
} StateBlock, *StateBlockPtr;
```

Figure 4.5 StateBlock Data Structure

The **LeftOverBlock** is used for saving samples or packets that would have been lost due to truncation on a given compression call. For example, take the case where a programmer sends 7 samples to the 6to1 Compressor for several calls. If a **LeftOverBlock** were not used then the 7th sample would be lost on each call. The results would not be pleasant. If a **LeftOverBlock** were used however, the 6to1 snth would save the seventh sample from the first call and concatenate it onto the buffer from the next call. On the third call, 2 samples truncated from the second call would be concatenated onto the buffer for the third call and so on. This may have limited use for the present 6to1 and 3to1 algorithms where the Compression ratio's are fixed



and it is easy for the programmer to know exactly how many samples to send each time to avoid truncation. In the future however, Apple may decide to release a variable packetSize Compression scheme and it will be quite impossible for the user to know how many samples to send on a given call to avoid truncation.

```
typedef struct
{
    unsigned long          count;
    char                  sampleArea[LeftOverBlockSize] ;
}
LeftOverBlock, *LeftOverBlockPtr;
```

Figure 4.6 LeftOverBlock Data Structure

As implied above, the old SoundHeader data structure will have to be changed slightly. The upper byte of the basenote field will now be used to indicate the meaning of the Data Structure. If the encode field is equal to zero, then the structure is interpreted as a new SoundHeader. If the encode field is non-zero, the structure is not a standard SoundHeader. Refer to Figure 4.3 above for possible encode values.

```
typedef struct {
    char          *samplePtr ;
    unsigned long length ;
    fixed         sampleRate ;
    unsigned long loopStart ;
    unsigned long loopEnd ;
    unsigned char encode ;
    unsigned char baseNote ;
    char          sampleArea[0] ;
} SoundHeader , *SoundHeaderPtr;
```

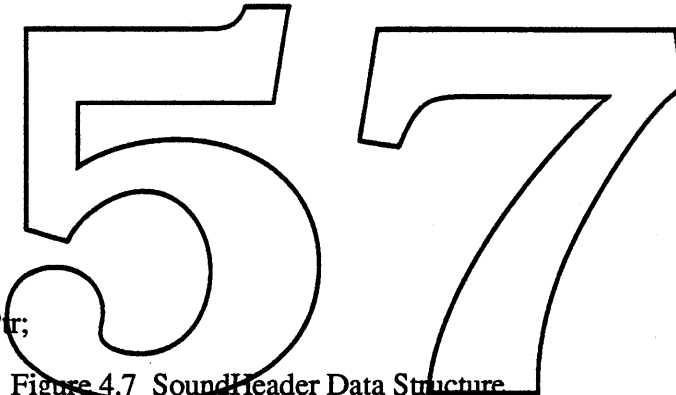


Figure 4.7 SoundHeader Data Structure

### 4.3 Command Integration

Two new commands have been defined for use with the ACE snth's. Those commands are the convertCmd and the sizeCmd.

Since the EXPANSION PLAYBACK routine will be coupled with the Sampled Sound Synthesizer, some of the commands presently accepted by the Sampled Sound Synthesizer will be altered slightly. Those commands are the BufferCmd and ContinueCmd.

A table reference of each command is given in section 5.0.

#### 4.3.1 ConvertCmd SndCommand

The first of the two new Sound Commands is the convertCmd. It is called by an application to either compress or expand a group of audio samples. The formal definition of the convertCmd is listed below:

##### ConvertCmd SndCommand

cmd = convertCmd  
 param1 = compressionID value (used only to convert from normalized to compressed samples)  
 param2 = pointer to ConversionBlock

Param1 is used to indicate the compression algorithm that will be used to compress input samples. Param1 is only used when the compressionID field of the input CmpSoundHeader indicates that the input samples are normalized samples (i.e. convert from normalized to compressed). When expanding compressed samples, param1 is ignored and the convertCmd uses the compressionID value contained in the input CmpSoundHeader.

Param2 is a pointer to a ConversionBlock, which in turn points to two CmpSoundHeaders. The Destination field of the ConversionBlock is used to tell the Sound Manager where in the output CmpSoundHeaders to write the samples it will generate. If Destination = insideCmpSH, the samples are put in the sampleArea of the output CmpSoundHeaders. If Destination = outsideCmpSH, the Sound Manager checks to see where the samplePtr of the output CmpSoundHeaders points. If samplePtr is nil, the Sound Manager allocates the necessary memory. Otherwise the Sound Manager writes the samples into the buffer pointed to by samplePtr.

If the StateVar pointer in the input CmpSoundHeader points to nil, the requested Algorithm will be initialized at the beginning of the call, otherwise the Algorithm will attempt to read its state out of the StateBlock pointed to by StateVar. If the StateVar pointer in the output CmpSoundHeader does not point to nil the Algorithm will attempt to write its state into the StateBlock pointed to by StateVar. Both the input StateVar pointer and the output StateVar pointer may point to the same StateBlock.

If the LeftOverSamples field in the input CmpSoundHeader does not point to nil, the samples or packets contained in the LeftOverBlock will be catenated onto the present call. If LeftOverSamples does point to nil, no samples or packets will be catenated. If the LeftOverSamples field in the output CmpSampleHeader does not point to nil, any samples that did not get processed due to truncation will be saved into that LeftOverBlock.

Warning! For programmers who do not want to use the LeftOverBlock, always send a number of samples that is a multiple of the Compression schemes packetSize.

#### 4.3.2 SizeCmd SndCommand

The other new sound command is the sizeCmd. The SizeCmd is for use by those programmers who want to do their own memory allocation before using the convertCmd described above. The sizeCmd will indicate the amount of memory required for expansion of a specified buffer of compressed data. The formal definition for the sizeCmd is listed below:

##### SizeCmd SndCommand

cmd = sizeCmd  
 param1 = compressionID value (used only to convert from normalized to compressed samples)  
 param2 = pointer to CmpSoundHeader

Param1 is used to indicate the compression algorithm that will be used to compress input samples in the succeeding convertCmd call. Param1 is only used when the encode field of the input CmpSoundHeader indicates that the input samples are linear non-compressed samples (i.e. preparing to convert from normalized to compressed data). When preparing to expand compressed samples, param1 is ignored and the sizeCmd uses the compressionID value contained in the input CmpSoundHeader.

Param2 points to the `CmpSoundHeader` that will be used as the input `CmpSoundHeader` in the succeeding `convertCmd`. When called, the `sizeCmd` looks at the length and `compressionID` fields of the `CmpSoundHeader`. If the `compressionID` field indicates that the samples it holds are compressed, the `sizeCmd` then uses the `compressionID` field to calculate how many bytes the samples will expand into. If the `compressionID` field indicates that the samples held by the `CmpSoundHeader` are linear non-compressed, the `sizeCmd` looks at the `compressionID` held in its `param1` and calculates how many bytes the samples will compress into. The value the `sizeCmd` calculates is stored in `param2` on return from the call.

Note that the `SizeCmd` does not account for any samples or packets that may be stored in a `LeftOverBlock`. It is intended for the user that wants to make only one call to a given algorithm. For the user that wishes to make several calls, it is advisable to allocate an output buffer large enough to hold an Algorithm's maximum output (ie worst case). For the `sixToOne` and `threeToOne` algorithm, these numbers are easy to calculate since both algorithms have fixed output ratios. For future compression Algorithms Apple will release the necessary information for the user to calculate what the largest output ratios are.

The way existing sound commands work with the Sampled Sound Synthesizer will have to be altered slightly as a result of its integration with the EXPANSION PLAYBACK routine. This affects the following sound commands: `BufferCmd`, and `ContinueCmd`. Presently `param2` of these instructions always points to a `SoundHeader` data structure. After the EXPANSION PLAYBACK routine has been integrated, however, `param2` of these instructions will also be able to point to `CmpSoundHeaders`. As mentioned before, the Sampled Sound Synthesizer knows which it is looking at by inspecting the value of the `encode` field.

#### 4.4 Constants

The following are new constants being added to the Sound Manager to support Audio Compression / Expansion:

##### encode field constants

|                      |                    |   |                                     |
|----------------------|--------------------|---|-------------------------------------|
| <code>#define</code> | <code>stdSH</code> | 0 | standard <code>SoundHeader</code>   |
| <code>#define</code> | <code>extSH</code> | 1 | extended <code>SoundHeader</code>   |
| <code>#define</code> | <code>cmpSH</code> | 2 | compressed <code>SoundHeader</code> |

##### compressionID field constants

|                      |                            |   |                                                        |
|----------------------|----------------------------|---|--------------------------------------------------------|
| <code>#define</code> | <code>notCompressed</code> | 0 | <code>CmpSoundHeader</code> has non-compressed samples |
| <code>#define</code> | <code>twoToOne</code>      | 1 | available only on IIgs                                 |
| <code>#define</code> | <code>eightToThree</code>  | 2 | available only on IIgs                                 |
| <code>#define</code> | <code>threeToOne</code>    | 3 | available only on Macintosh                            |
| <code>#define</code> | <code>sixToOne</code>      | 4 | available only on Macintosh                            |

##### new snth Resource ID numbers

|                      |                    |    |                                                |
|----------------------|--------------------|----|------------------------------------------------|
| <code>#define</code> | <code>ACE_3</code> | 11 | code resource ID for 3to1 compressor/ expander |
| <code>#define</code> | <code>ACE_6</code> | 13 | code resource ID for 6to1 compressor/expander  |

##### Destination constants

|                      |                           |   |                                                   |
|----------------------|---------------------------|---|---------------------------------------------------|
| <code>#define</code> | <code>outsideCmpSH</code> | 0 | put samples in outside buffer                     |
| <code>#define</code> | <code>insideCmpSH</code>  | 1 | put samples or packets in <code>sampleArea</code> |

##### numChannel constants

|                      |                            |   |              |
|----------------------|----------------------------|---|--------------|
| <code>#define</code> | <code>monoChannel</code>   | 1 | one channel  |
| <code>#define</code> | <code>stereoChannel</code> | 2 | two channels |

ACE Error Codes

|         |              |   |                                           |
|---------|--------------|---|-------------------------------------------|
| #define | ACESuccess   | 0 | success                                   |
| #define | ACEMemFull   | 1 | memory could not be allocated for output  |
| #define | ACENilBlock  | 2 | nil block pointer                         |
| #define | ACEBadComp   | 3 | invalid compressionID                     |
| #define | ACEBadEncode | 4 | invalid encode field                      |
| #define | ACEBadDest   | 5 | invalid Destination field ConversionBlock |
| #define | ACEBadCmd    | 6 | invalid SndCommand number                 |

packetSize constants

|         |                      |    |                               |
|---------|----------------------|----|-------------------------------|
| #define | sixToOnePacketSize   | 8  | packetSize for 6to1 Algorithm |
| #define | threeToOnePacketSize | 16 | packetSize for 3to1 Algorithm |

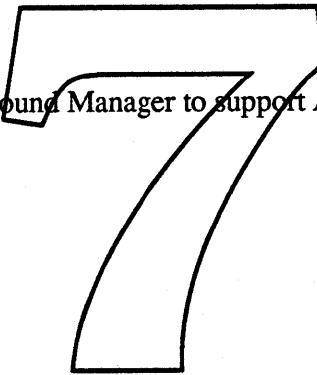
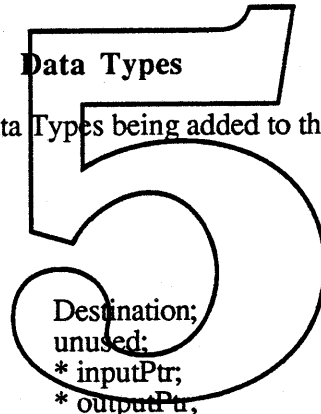
data structure sizes

|         |                   |    |                                   |
|---------|-------------------|----|-----------------------------------|
| #define | StateBlockSize    | 64 | StateBlock data structure size    |
| #define | LeftOverBlockSize | 32 | LeftOverBlock data structure size |

**4.5 Summary of new Data Types**

The following are new Data Types being added to the Sound Manager to support Audio Compression / Expansion:

```
typedef struct
{
short
short
CmpSoundHeader
CmpSoundHeader
}
ConversionBlock, *ConversionBlockPtr ;
```



```
typedef struct
{
char
unsigned long
Fixed
unsigned long
unsigned long
unsigned char
unsigned char
unsigned short
unsigned short
extended
Ptr
Ptr
Ptr
Ptr
Ptr
}
*samplePtr;
length;
sampleRate;
loopStart;
loopEnd;
encode;
baseNote;
numChannels;
sampleSize;
AIFFSampleRate;
MarkerChunk;
FutureUse1;
FutureUse2;
StateVars;
LeftOverSamples;
```

```

unsigned short    compressionID;
unsigned short    packetSize;
unsigned short    snthID;
unsigned short    FutureUse3;
char              sampleArea[0];
}
CmpSoundHeader, *CmpSoundHeaderPtr;

```

```

typedef struct{
short            stateVar[StateBlockSize] ;
} StateBlock, *StateBlockPtr;

```

```

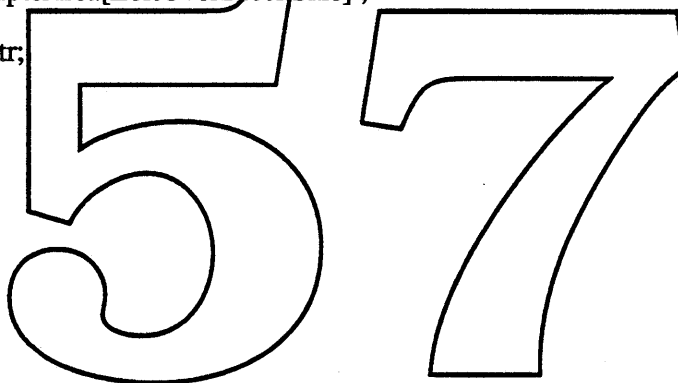
typedef struct
{
unsigned long    count;
char            sampleArea[LeftOverBlockSize] ;
}
LeftOverBlock, *LeftOverBlockPtr;

```

```

typedef struct {
char            *samplePtr ;
unsigned long   length ;
fixed          sampleRate ;
unsigned long   loopStart ;
unsigned long   loopEnd ;
unsigned char   encode ;
unsigned char   baseNote ;
char           sampleArea[0] ;
} SoundHeader , *SoundHeaderPtr;

```



## 5.0 SOUND COMMANDS

### 5.1 ConvertCmd

#### Overview

The `convertCmd` is used for gaining access to the COMPRESSION and the BUFFERED EXPANSION routines (i.e. the *ACE snth's*) through the Sound Manager. When the `convertCmd` is used, the *ACE snth's* receive a `CmpSoundHeader` of audio samples. The audio data will then be either compressed or expanded, depending on the parameters passed.

#### Parameters

`cmd` = `convertCmd`  
`param1` = compressionID value (used only to convert from normalized to compressed samples)  
`param2` = pointer to `ConversionBlock`

#### Calling Procedure

The `convertCmd` is invoked by passing the *ACE snth's* a `convertCmd SndCommand` using the `SndControl` function call. The `id` field passed in the `SndControl` call should be the resource ID number of the code resource that contains the desired Compression Algorithm. The `SndCommand` passed should be of type `convertCmd`. The appropriate values for the `SndCommand` are listed below in the *Parameters* section. The `ConversionBlock` pointed to by `param2` in turn points to two `CmpSoundHeader`s. The input `CmpSoundHeader` contains the input samples, the output `CmpSoundHeader` is filled out by the *ACE snth* according to the conversion it performs.

If the `StateVar` pointer in the input `CmpSoundHeader` points to `nil`, the requested Algorithm will be initialized at the beginning of the call, otherwise the Algorithm will attempt to read a saved state out of the `StateBlock` pointed to by `StateVar`. If the `StateVar` pointer in the output `CmpSoundHeader` does not point to `nil` the Algorithm will attempt to write its state into the `StateBlock` pointed to by `StateVar`. Both the input `StateVar` pointer and the output `StateVar` pointer may point to the same `StateBlock`.

If the `LeftOverSamples` field in the input `CmpSoundHeader` does not point to `nil`, the samples or packets contained in the `LeftOverBlock` will be catenated onto the present call. If `LeftOverSamples` does point to `nil`, no samples or packets will be catenated. If the `LeftOverSamples` field in the output `CmpSampleHeader` does not point to `nil`, any samples that did not get processed due to truncation will be saved into that `LeftOverBlock`.

Warning! For programmers who do not want to use the `LeftOverBlock`, always send a number of samples that is a multiple of the Compression scheme's `packetSize`.

#### Returns

The `CmpSoundHeader`, pointed to by the `outputPtr` field of the `ConversionBlock`, contains the converted audio samples. The appropriate fields of the `CmpSoundHeader` are filled to describe the converted samples. The *ACE Snth* returns error codes in the `cmd` field of the `SndCommand` data structure it receives. There are presently seven defined error codes returned by the `convertCmd`. Those error codes are:

```
#define ACESuccess 0 success
#define ACEMemFull 1 memory could not be allocated for output
```

```

#define ACENilBlock      2    nil block pointer
#define ACEBadComp      3    invalid compressionID
#define ACEBadEncode    4    invalid encode field
#define ACEBadDest      5    invalid Destination field ConversionBlock
#define ACEBadCmd       6    invalid SndCommand number

```

### Data Structures Used

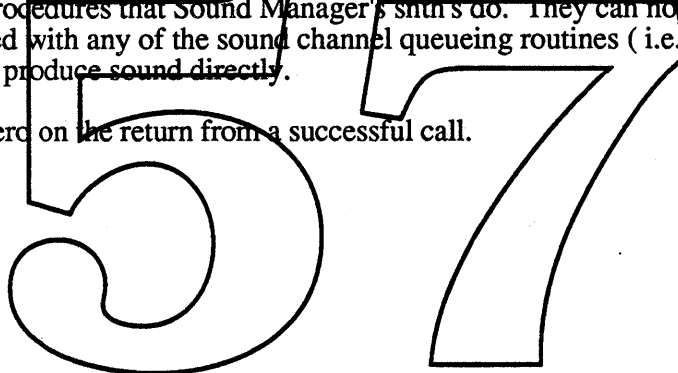
SndCommand (1), CmpSoundHeader (2), ConversionBlock (1), StateBlock ( 1 or 2 both optional ), LeftOverBlock ( 1 or 2 both optional )

### Warnings

The convertCmd is for use only with the *ACE snth's*. It is not for use with the Note Synthesizer, WaveTable Synthesizer, Sampling Synthesizer, MIDI Input Synthesizer, or MIDI Output Synthesizer.

The Compression snth's are not snth's in the traditional sense. They are really code resources that happen to conform to some of the calling procedures that Sound Manager's snth's do. They can not be linked to a sound channel and thus can not be called with any of the sound channel queueing routines ( i.e. SndDoCommand, SndDoImmediate ) . They do not produce sound directly.

Note that the cmd field is set to zero on the return from a successful call.



## 5.2 SizeCmd

### Overview

The sizeCmd is for those programmers who want to do their own Memory Management. When an ACE *snth* receives a SizeCmd, it looks at the type of samples and number of the samples in its CmpSoundHeader. It then calculate how much memory a converted version of the input will occupy. The sizeCmd will work for either compression or expansion of samples.

### Calling Procedure

The sizeCmd is invoked by passing the ACE *snth's* a sizeCmd SndCommand using the SndControl function call. The id field passed in the SndControl call should be the ID number of the ACE *snth* that is to be invoked. Presently there are two Compression Algorithms. The Resource ID numbers for the ACE Algorithms is listed in Section 4.4. The SndCommand passed should be of type SizeCmd. The appropriate values for the SndCommand are listed below. The CmpSoundHeader pointed to by param2 of the sizeCmd should be identical to the CmpSoundHeader used for the input of a successive convertCmd.

### Parameters

cmd = sizeCmd  
 param1 = compressionID value (used only to convert from normalized to compressed samples)  
 param2 = pointer to CmpSoundHeader

### Returns

On return, Param2 points to a long word that indicates how much memory is needed for a successive convertCmd call. The ACE *Snth* returns error codes in the cmd field of the SndCommand data structure it receives. When using the sizeCmd it is possible to receive the following error codes.

|         |              |   |                                          |
|---------|--------------|---|------------------------------------------|
| #define | ACESuccess   | 0 | success                                  |
| #define | ACEMemFull   | 1 | memory could not be allocated for output |
| #define | ACENilBlock  | 2 | nil block pointer                        |
| #define | ACEBadComp   | 3 | invalid compressionID                    |
| #define | ACEBadEncode | 4 | invalid encode field                     |
| #define | ACEBadCmd    | 6 | invalid SndCommand number                |

### Data Structures Used

SndCommand (1), CmpSoundHeader (1)

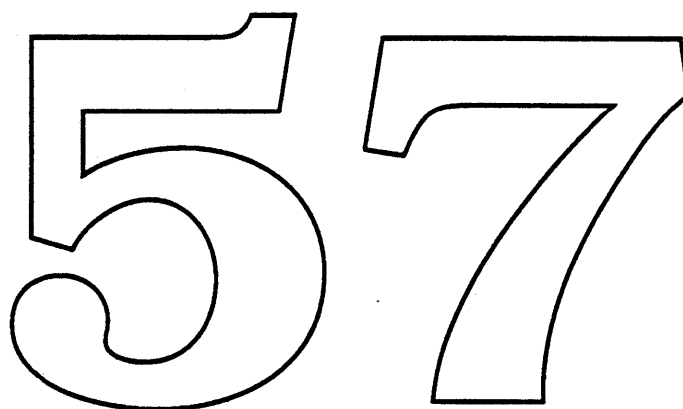
### Warning

The sizeCmd is for use only with the ACE *snth's*. It is not for use with the Note Synthesizer, WaveTable Synthesizer, Sampling Synthesizer, MIDI Input Synthesizer, or MIDI Output Synthesizer. Note that the cmd field is set to zero on the return from a successful call.

Note that the SizeCmd does not account for any samples or packets that may be stored in a LeftOverBlock.



It is intended for the user that wants to make only one call to a given algorithm. For the user that wishes to make several calls, it is advisable to allocate an output buffer large enough to hold an Algorithm's maximum output (ie worst case). For the sixToOne and threeToOne algorithm, these numbers are easy to calculate since both algorithms have fixed output ratios. For future compression Algorithms Apple will release the necessary information for the user to calculate what the largest output ratios are.

The image shows the numbers '57' in a large, bold, outlined font. The '5' has a thick top bar and a rounded bottom, while the '7' has a thick top bar and a curved bottom. The numbers are centered on the page.

### 5.3 BufferCmd

#### Overview

The BufferCmd is an existing Sound Manager SndCommand. Presently it is for use only with the Sampling Synthesizer. With the integration of EXPANSION PLAYBACK routines into the Sampling Synthesizer, the BufferCmd will also be used to pass compressed samples.

#### Calling Procedure

The BufferCmd is sent to the Sampling Synthesizer using the SndDoImmediate and SndDoCommand function calls. The SndDoCommand inserts the BufferCmd at the end of the specified SndChannel queue. The SndDoImmediate call bypasses the queue and passes the command directly to the synthesizer for immediate processing. To expand and playback a buffer of compressed samples, pass the Sampling Synthesizer a BufferCmd with the parameters filled out as listed below. For EXPANSION PLAYBACK, param2 should point to a CmpSoundHeader instead of a SoundHeader. The SoundHeader can still be used for passing buffers of 8-bit linear non-compressed mono-channel samples.

#### Parameters

cmd = BufferCmd  
 param1 = nil  
 param2 = either: pointer to CmpSoundHeader (for compressed or normalized samples)  
 or: pointer to SoundHeader (for normalized samples)

#### Data Structures Used

SndCommand (1)  
 CmpSoundHeader (1 if compressed or normalized samples are passed)  
 SoundHeader (1 if 8-bit linear non-compressed mono-channel samples are passed)

#### Warning

The BufferCmd is for use only with the Sampling Synthesizer. The playback of multiple BufferCmd's when passing compressed samples is not guaranteed to be continuous on the Macintosh Plus and the Macintosh SE.

## 5.4 ContinueCmd

### Overview

The ContinueCmd is an existing Sound Manager SndCommand. Presently it is for use only with the Sampling Synthesizer. With the integration of EXPANSION PLAYBACK routine into the Sampling Synthesizer, the ContinueCmd will also be used to pass compressed samples.

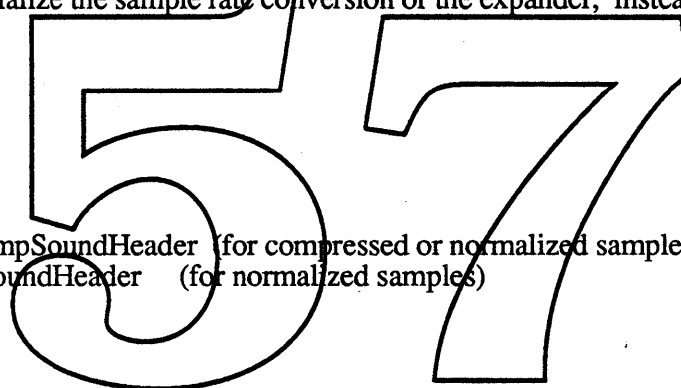
### Calling Procedure

The ContinueCmd is sent to the Sampling Synthesizer using the SndDoImmediate and SndDoCommand function calls. The SndDoCommand inserts the ContinueCmd at the end of the specified SndChannel queue. The SndDoImmediate call bypasses the queue and passes the command directly to the synthesizer for immediate processing. The calling procedure for the ContinueCmd is identical to that of the BufferCmd.

The ContinueCmd is used for producing continuous Playback. To produce a continuous sound, send the first segment of the sound with a BufferCmd. The BufferCmd will initial both the sample rate conversion algorithm and the requested expander. Then send all following segments of the sound with ContinueCmds. The ContinueCmd does not reinitialize the sample rate conversion or the expander, instead it uses the present state of each.

### Parameters

cmd = ContinueCmd  
 param1 = nil  
 param2 = either: pointer to CmpSoundHeader (for compressed or normalized samples)  
 or: pointer to SoundHeader (for normalized samples)



### Data Structures Used

SndCommand (1)  
 CmpSoundHeader (1 if compressed or normalized samples are passed)  
 SoundHeader (1 if 8-bit linear non-compressed mono-channel samples are passed)

### Warning

The ContinueCmd is for use only with the Sampling Synthesizer. The playback of multiple buffers when passing compressed samples is not guaranteed to be continuous on the Macintosh Plus and the Macintosh SE. It is only continuous when the compressed packets that are sent expand into a multiple of ( 3 \* 370 ). The playback of multiple buffers of stereo sound is not continuous on the Mac Plus, Harpo or SE under any conditions. Note that when a stereo sound is received on a Mac Plus, SE or Harpo, only the right channel is played. Also note that on the MacII when a stereo sound is played, only the right channel can be heard through the external speaker. To get both channels a pair of headphones or external speakers must be used. On the Fafnir both channels of a stereo sound can be heard through the external speaker.

## 5.5 VersionCmd

### Overview

The VersionCmd is an existing Sound Manager SndCommand. It is a SndCommand that is accepted by all Sound Manager snth's. The VersionCmd is used by both the user and the System to determine the Version of the snth's.

### Calling Procedure

The VersionCmd is sent to all snths with the SndControl trap only. The two new Ace snth's are no exception.

### Parameters

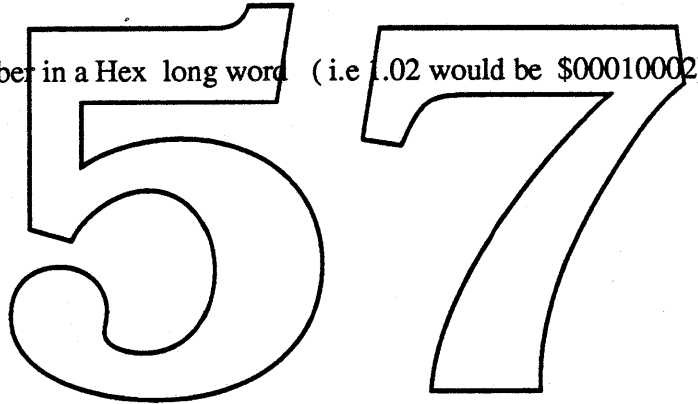
cmd = VersionCmd  
param1 = nil  
param2 = nil

### Returned

param2 = Version number in a Hex long word (i.e. 1.02 would be \$00010002)

### Data Structures Used

SndCommand (1)



## 6.0 RECOMMENDED PRACTICES

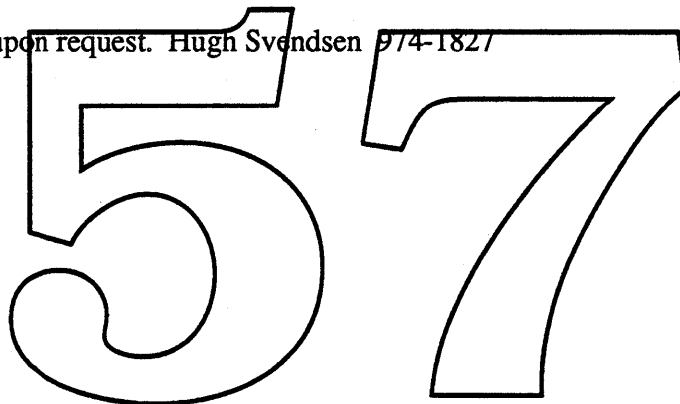
- Do not pass a compressed sounds using the SoundCmd. This mode of operation is not supported on any machine !!!
- 22kHz is the only supported sample rate on the Mac Plus and SE. Therefore to maintain compatibility across all machines it is strongly recommended that all compressed snds be created at 22khz!!!

### 6.1 Installation

### 6.2 Recording / Playback

### 6.3 Examples of Usage

Code examples available upon request. Hugh Svendsen 974-1827



## 7.0 TESTING CONSIDERATIONS

### 7.1 Ideas for Testing the ACE Snth

The following list outlines some features/uses for the ACE snth's. They are being listed to give those testing the snth's some ideas about what it can be used for, and as a result, maybe some insight into how to best test it.

Because the state of the ADPCM algorithms can be saved across consecutive calls, it will be possible to convert multiple channels of sound simultaneously. If the state of the ADPCM routines are not being saved and restored correctly by the respective snth's each time, it will be immediately apparent in the output waveforms.

The convertCmd performs expansion if the CmpSoundHeader indicates the samples are compressed, and compression if the samples it receives are non-compressed. Therefore it should be possible to send an input CmpSoundHeader through two consecutive ConvertCmd's, and receive as the output a CmpSoundHeader functionally equivalent to the original.

To make the use of the ACE Snth's as painless as possible, the Snth's return error codes when called incorrectly. The error codes are returned in the `err` field of the SndCommand. The possible error codes returned are:

ACESuccess = success  
 ACEMemFull = memory could not be allocated for output  
 ACENilBlock = nil block pointer  
 ACEBadComp = invalid compressionID in either param1 of SndCommand or compressionID of CmpSoundHeader  
 ACEBadEncode = invalid encode field parameter in CmpSoundHeader  
 ACEBadDest = invalid Destination field ConversionBlock  
 ACEBadCmd = invalid command number (Snth only accepts ConvertCmds and SizeCmds)

There are three possible places an ACE Snth can output samples. Samples can be written into a buffer allocated and passed by the caller. They can be written into a buffer that the Snth itself allocates, or they can be written into the `sampleArea` of the output CmpSoundHeader. These modes of operation could be tested in the following manner:

- Pass an ACE Snth a ConversionBlock with the Destination field set to `ord(outsideCmpSoundHeader)`. The `samplePtr` field of the input CmpSoundHeader should be pointing to nil. The ACE Snth will allocate the space for the output samples and write the samples into that space.
- Pass an ACE Snth a ConversionBlock with the Destination field set to `ord(outsideCmpSoundHeader)`. The `samplePtr` field of the input CmpSoundHeader should be pointing to a buffer large enough to hold the samples to be generated. The ACE Snth will write the output samples into that space.
- Pass the ACE Snth a ConversionBlock with the Destination field set to `ord(inCmpSoundHeader)`. The output samples will then be written into that space. Note that the user has to allocate the space at the end of the CmpSoundHeader.

There are two ways samples can be passed to an ACE Snth. The first is in a buffer passed by the user, the second is in the `sampleArea` of the input CmpSoundHeader. These modes of operation could be tested in the following manner:

- Pass an ACE Snth an input CmpSoundHeader with the samplePtr field pointing to an input buffer. The ACE Snth will look in this buffer to find the input samples.
- Pass an ACE Snth an input CmpSoundHeader with the samplePtr field set to nil. The ACE Snth will then look in the sampleArea of the CmpSoundHeader to find the input samples.

The baseNote field of the output CmpSoundHeader should be identical to the baseNote field of the input CmpSoundHeader.

The compressionID field of the output CmpSoundHeader is filled out according to the operation performed. If passing a convertCmd causes compression to take place, the compressionID field of the output CmpSoundHeader indicates the compressionID of the algorithm used. Otherwise the compressionID is set to zero.

The number of channels specified in the input CmpSoundHeader Should always be one. The ACE snth's do not accept stereo snd's. If the user wishes to convert a buffer of stereo samples/packets, he should first separate the samples/packets into to separate contiguous buffers for processing.

If a convertCmd causes compression to take place the packetSize field will be altered. If threeToOne compression takes place the output CmpSoundHeader packetSize field will be set to threeToOnePacketSize. If sixToOne compression takes place the output CmpSoundHeader packetSize field will be set to sixToOnePacketSize. If expansion takes place the packetSize field of the CmpSoundHeader will actually indicate the sampleSize of the expanded data. Currently 8-bit linear is the only non-compressed sample format, therefore the output sampleSize would be 8 bits.

## 7.2 Ideas for Testing the Expansion-Playback Mode of the Sampled Sound Synthesizer

The following list outlines some features/uses for the Expansion-Playback mode of the Sampled Sound Synthesizer. They are being listed to give those testing the Synthesizer some ideas about what it can be used for, and as a result, maybe some insight into how to best test it.

In the past when sending the Sampled Sound Synthesizer a BufferCmd, only SoundHeader data Structures were used to pass the samples. The new version of the Sampled Sound Synthesizer should be able to accept non-compressed samples in a SoundHeader, and both compressed and non-compressed samples in a CmpSoundHeader.

It should be possible to send a BufferCmd followed by multiple ContinueCmds for continuous Expansion-Playback of mono compressed samples on all Macintosh CPU's. On all Apple Sound Chip machines except the harpo, it should be possible to produce continuous expansion and playback of stereo compressed sounds.

- A more rigorous test for multiple buffer playback could include using a double buffering scheme to send alternating buffers for extra-long continuous playback.

## 8.0 DESIGN DECISIONS

I'm sure that if any one really takes the time to read and understand this ERS, they will ultimately have some questions as to why some things were implemented the way they were. Therefore I have taken the liberty to answer some questions that I think someone would have after reading this document.

*Why is there no sample rate conversion on the Mac Plus and SE?*

Not enough real-time. Real-time expansion-playback on a Mac Plus takes close to 95% of the processor's bandwidth. There simply was not enough real time for sample rate conversion. Sample rate conversion might be possible on a SE but it would mean yet another snth and even more disk space.

*Why don't you use interpolation instead of drop sample-tuning on the MacII and up?*

Because the Mac Plus can not do sample rate conversion we are already mandating that user compress only at 22khz. Therefore it seemed to make some sense not to include a feature that we were telling everyone not to use.

*Why is the buffered 6to1 and 3to1 in two different code resources?*

So only the algorithm that is being accessed is read into memory and not both algorithms. This is a good practice to follow to handle possible future compression algorithms.

*Does the Compression Algorithm accept 16-bit samples?*

No, presently the Algorithm accepts only 8-bit linear samples. However there is nothing in the Algorithm that prevents this as a future option.

*Do the Buffered Compression/Expansion Algorithms accept stereo samples?*

No, to process stereo interleaved packets the packets should first be compacted into two separate contiguous buffers and sent separately for processing. Direct conversion of interleaved packets/samples might be a good future extension to the buffered compression/expansion snth's however.

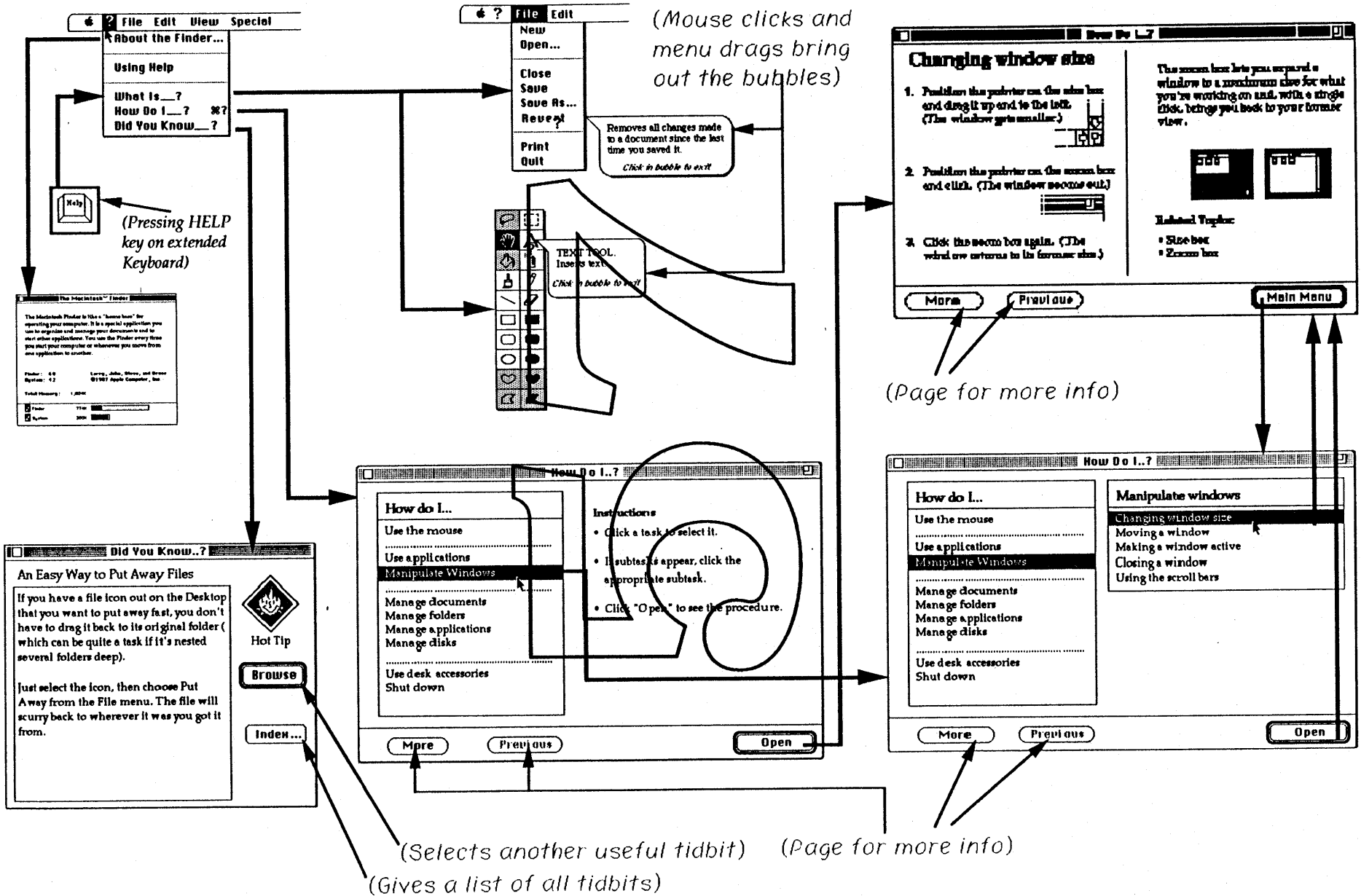



57

# Help Manager User Interface Summary

1/13/89

(Information Menu)



The Human  Interface Group

# Help Manager Human Interface

ERS  
57

James Sulzen

January 13, 1989

Version 0.5

Please return comments and/or markups to me by January 20 at:

x4-3142, or

M/S 27-AO, or

AppleLink: Sulzen

APPLE COMPUTER

COMPANY COFIDENTIAL

# Table of Contents

|                                               |          |
|-----------------------------------------------|----------|
| <b>1.0 Introduction</b> .....                 | <b>3</b> |
| <b>2.0 Overview</b> .....                     | <b>3</b> |
| 2.1 Types of Help .....                       | 3        |
| 2.2 Information Menu .....                    | 5        |
| 2.3 Information Menu Commands.....            | 7        |
| 2.4 Help Authoring Tool.....                  | 8        |
| 2.5 Help Database Localization.....           | 9        |
| <b>3.0 Bubble Windows</b> .....               | <b>9</b> |
| 3.1 Bubble Mode.....                          | 9        |
| 3.2 Entering and Exiting Bubble Mode.....     | 10       |
| 3.2.1 Entry Mechanisms.....                   | 11       |
| 3.2.1.1 "What Is" Menu Command.....           | 11       |
| 3.2.1.2 HELP Key.....                         | 11       |
| 3.2.1.3 Alternative HELP Key.....             | 12       |
| 3.2.1.4 Button or other Screen Graphic.....   | 12       |
| 3.2.2 Exit Mechanisms.....                    | 13       |
| 3.2.2.1 "What Is" Menu Command Toggle.....    | 13       |
| 3.2.2.2 HELP Key.....                         | 13       |
| 3.2.2.3 Clicking in the Bubble.....           | 13       |
| 3.2.2.4 Close Box in Bubble.....              | 14       |
| 3.3 Bubble Window Display.....                | 14       |
| 3.4 Help Information Display Sizing.....      | 15       |
| 3.5 Menu "What Is" Help.....                  | 16       |
| 3.5.1 Menu Titles.....                        | 17       |
| 3.5.2 Alternative Menu and Item States.....   | 18       |
| 3.5.3 Miscellaneous Menu Issues.....          | 19       |
| 3.5.3.1 Duplicate Menu Item Strings.....      | 19       |
| 3.5.3.2 Hierarchical and Scrolling Menus..... | 20       |
| 3.6 Standard Bubble Messages.....             | 20       |

|     |                                           |    |
|-----|-------------------------------------------|----|
| 3.7 | Bubble Window Suggestions .....           | 21 |
| 4.0 | "How Do I" Help .....                     | 21 |
| 4.1 | Functional Description .....              | 21 |
| 4.2 | Help Fonts .....                          | 24 |
| 4.3 | How Do I Suggestions .....                | 24 |
| 5.0 | Did You Know Help .....                   | 25 |
| 5.1 | Suggestions .....                         | 26 |
| 6.0 | Additional Help System Capabilities ..... | 26 |

## List of Figures

|           |                                                 |    |
|-----------|-------------------------------------------------|----|
| Figure 1  | - Sample Bubble Window for Menu Help .....      | 5  |
| Figure 2  | - Sample Bubble Window for a Tool Palette ..... | 5  |
| Figure 3  | - "How Do I" Help Dialog .....                  | 6  |
| Figure 4  | - Did You Know Dialog Box .....                 | 6  |
| Figure 5  | - Standardized Information Menu .....           | 7  |
| Figure 6  | - Pop-out Help Menu .....                       | 8  |
| Figure 7  | - Example Finder "About..." Window .....        | 9  |
| Figure 8  | - Alternative HELP Key CDEV .....               | 13 |
| Figure 9  | - Bubble Mode Explanation .....                 | 14 |
| Figure 10 | - Bubble Window with Close Box .....            | 15 |
| Figure 11 | - Bubbles for Non-standard Menus .....          | 18 |
| Figure 12 | - Sample Menu Title Bubble Window .....         | 19 |
| Figure 13 | - Sample Help Message for a Disabled Item ..... | 19 |
| Figure 14 | - "How Do I" Topics and Sub-Topics .....        | 23 |
| Figure 15 | - "How Do I" help message .....                 | 24 |
| Figure 16 | - Unzoomed One Page HDI Help Display .....      | 24 |

## 1.0 Introduction

This document describes the Macintosh help user interface. It describes the types of help and ways users will interact with the help system.

Though not strictly part of the user interface, HIG guidelines have been included where relevant. In particular, developer guidelines are provided describing how to present the help system to users. However, guidelines for help content and organization are not provided - they are forthcoming in a document from Interactive Education.

The *Los Angeles* font is used to note comments, ascerbic asides, and concerns which need further addressing.

This will be the first help system ever released as part of Apple's system software. It is the standard to be promoted to third party developers and one which we hope they will subsequently promulgate in their applications. Given the unique and even experimental nature of the help interface, any and all input is solicited and would be greatly appreciated.

## 2.0 Overview

### 2.1 Types of Help

The Help Manager will directly support three kinds of help as described below.

- a. "What Is" (Bubble Help). This displays "bubble" windows next to screen objects. The contents of the bubbles briefly explain what the screen object is. (See Figures 1 and 2).
- b. "How Do I". This displays a floating windoid from which the user can select topics and obtain help. The topic lists are constructed such as to aid the user with answering questions about how to perform a function, given the user knows what is needed to be done (see Figure 3).
- c. "Did You Know". This help displays a window that shows messages which describe little-known or particularly useful program features (see Figure 4).

Other Types of Help. There are several other varieties of help that research has indicated would also be useful. These other varieties are not being implemented due to either lack of time or the need for further research. However, it is expected that the Macintosh help system will evolve to incorporate these and other capabilities in the future. The help capabilities described above do address the most pressing needs of users and can be implemented within the Blue time-frame. See section six of this document for information about additional capabilities.

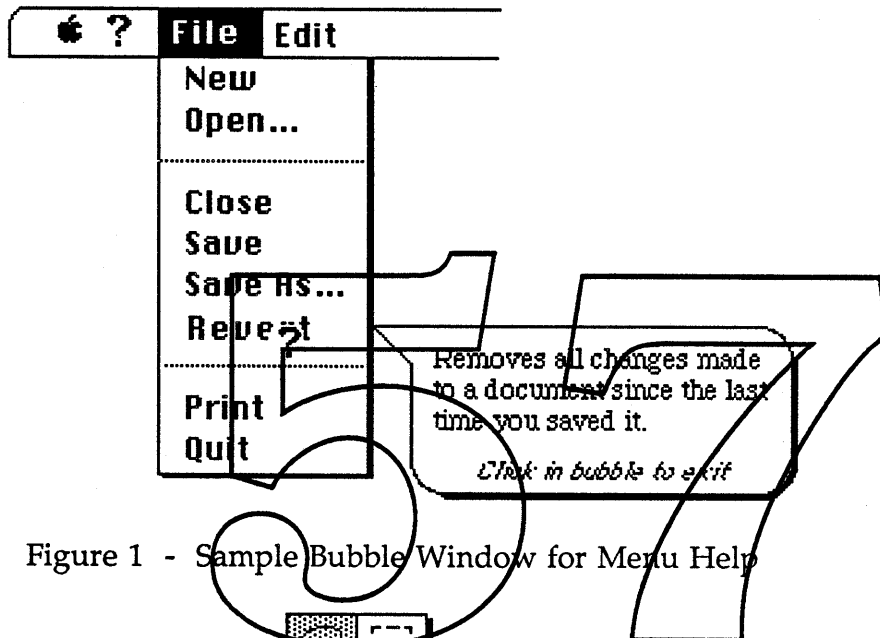


Figure 1 - Sample Bubble Window for Menu Help

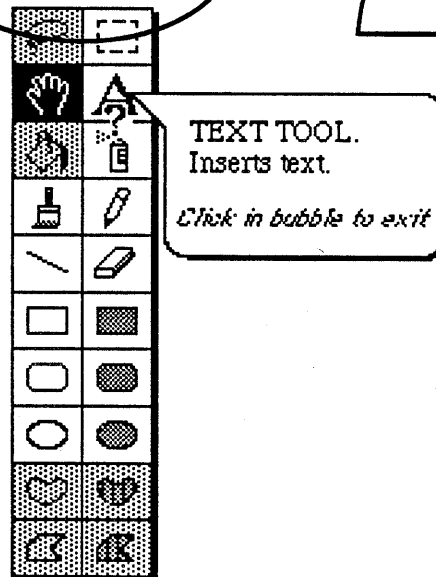


Figure 2 - Sample Bubble Window for a Tool Palette

[The More and Previous buttons should be interchanged.]

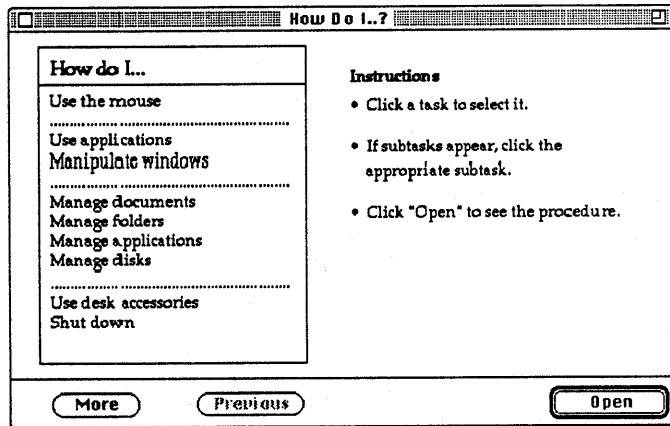


Figure 3 - "How Do I" Help Dialog

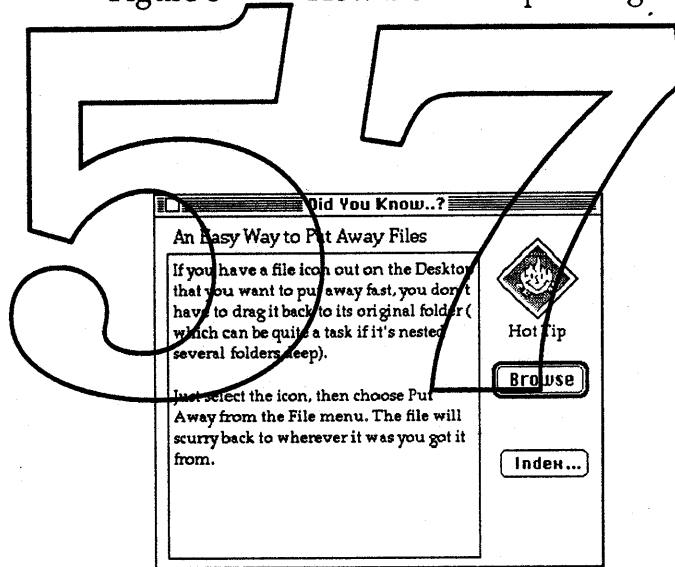


Figure 4 - Did You Know Dialog Box

## 2.2 Information Menu

Each application supporting help should have an Information menu which lists the commands shown in Figure 5. The Information menu is the start of a unified strategy for providing user on-line information needs. The intent is to make help and information access as much a part of the whole user interface as items like the File menu and cut/copy/paste metaphor. Consequently, information commands are being given their own menu to make them much more visible to users.



Information Menu Placement. The exact placement of the menu in the menubar is not yet decided. This is arbitrary as far as the Help Manager is concerned and need not be decided for some time. There are currently four possibilities: First in the menubar, second, following Edit, and last. The ultimate decision will depend such factors as:

1. Frequency of use of its and similarly infrequently used commands,
2. Potential user confusion,
3. Esthetics of appearance,
4. Traditional menubar organization (whatever that means under the present circumstances...), and
5. Potential user reaction.

**i** Menu Title Localization. The menu title can vary depending on the country for which a machine is localized. On this side of the Atlantic, the question mark is considered the sign for information. However, in Europe it is a stylized lower-case "i" (see figure). [God knows what it is anywhere else...] Since there is no universal symbol for information, and a single and simple symbol is desired to conserve menubar real estate, the menu title must be an localizable graphic symbol.

Menu Guidelines. Note that the "About.." command has been moved from the Apple menu to the Information menu and COMMAND-? is the recommended command key for the "How Do I \_\_\_?" command. Any other of the application's help or information related commands can also be placed in this menu after the standard help commands. A separator line should be used before such commands.

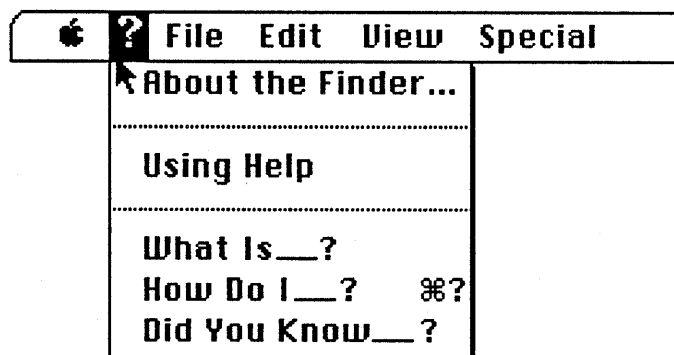


Figure 5 - Standardized Information Menu

The organization and content of the Information menu is a guideline. It is not programmatically required. As in all other user interface guidelines, the organization of the Information menu should be followed if at all feasible. However, applications with unique or stringent requirements might provide different commands and access to the help system. For example, the relevant help commands could be placed together under a different menu (such as the Edit menu), or hierarchically under a "Help" menu item.

Non-menu Help Access. There are situations where the menus are not accessible (i.e. during modal dialogs). Making help available then requires using a screen graphic or control that allows the user to obtain help.

A button might well do for turning on Bubble Mode if that is all that is needed (see below about Bubble Mode). More sophisticated requirements which need access to several help forms can use a pop-out menu to allow the user access to the various help systems (see figure below). If a menu is used, it should have a similar organization to the Information menu so users will feel familiar with it.

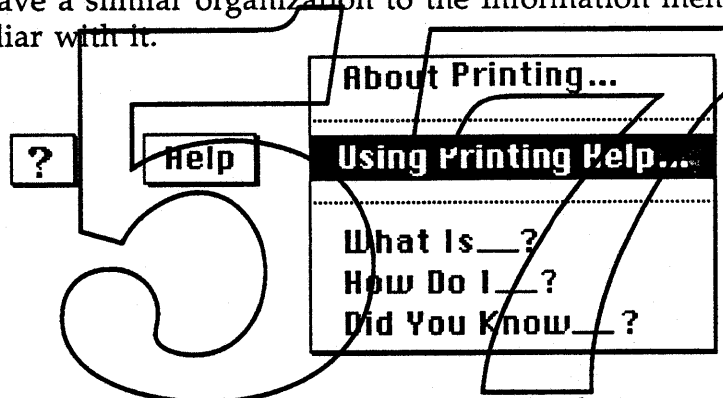


Figure 6 - Pop-out Help Menu

### 2.3 Information Menu Commands

Using Help Command. Selecting the Using Help command from the Information menu displays a dialog which describes the various kinds of help which are available as well as how to use them. A standardized dialog is provided for all applications which can be used or not as an application developer sees fit.

About... Command. The "About..." command under the Information menu is intended to succinctly provide "Goal" help. Goal help tells a user the basic purpose and primary uses of the application. It answers the user's basic question "What can I do with this?".

Traditionally, the "About..." box has contained copyright, version, and miscellaneous status information, as well as author credits. It is still reasonable for the window to contain some all or none of this information, but the "goal" information should be emphasized in its placement and appearance. (Note that the window should also be non-modal so that users can keep it on the desktop if they desire.) The figure below is an example of the above recommendations.

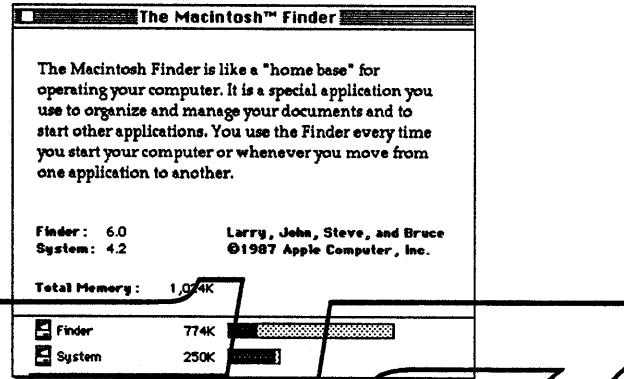


Figure 7 - Example Finder "About..." Window

Other Help Commands The Information menu's "What Is\_\_?" command places the application into Bubble Mode as described in section three of this document. The "How Do I \_\_?" and "Did You Know\_\_?" commands cause their respective windows to be displayed front-most. In addition, selecting "Did You Know\_\_?" causes a new piece of help information to be displayed in the Did You Know window.

## 2.4 Help Authoring Tool

The authoring tool allows an application's help information to be created and added to the application. It also allows an application's help information to be translated to foreign languages. While an Authoring Tool is not strictly required to use the Help Manager, providing one is considered critical to the success of the help system. The Authoring Tool itself, is to be provided by Interactive Education and will be defined by them in a future document.

*[The help authoring tool definition needs to be addressed]*

## 2.5 Help Database Localization

The help system operates in all international script systems. Only the help information itself needs to be translated to provide a foreign language version of an application's help system.

Note that several formats will be supported for actual help data. Among these are PICT data to allow pictures and text to be freely mixed. It is strongly recommended that no text information be included in bit maps in graphics data. Such text would be inordinately hard to access and so to translate.

## 3.0 Bubble Windows

Bubble Windows are the primary feature of "What Is" help. They display help information directly adjacent to the feature being described. They have limitations, especially as regards program features which have no currently displayed screen representation. However, their straightforward simplicity and appearance are very appealing to users.

### 3.1 Bubble Mode

To use "What Is" help a user must put the application into Bubble Mode. This is a very major program modality that affects the application's entire user interface behavior. In Bubble Mode the application's normal user interface behavior is suspended and instead mouse clicks cause a bubble window to appear. The bubble "points" to the graphical object associated with the cursor position and contains information describing the object.

MultiFinder. Bubble Mode only affects the current application. MultiFinder functioning and other applications are not affected if MultiFinder is running (i.e. clicking on another application's window will bring that application forward). If single Finder is running then Bubble Mode will apply to executing DA's (and DA's can turn on Bubble Mode and affect the running application and other DA's).

Holding down the option key turns on bubble mode.

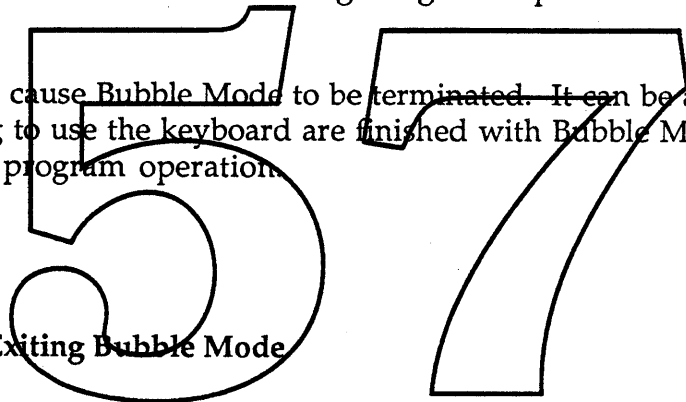
Cursor Shape. While in Bubble Mode the cursor is changed to the shape of of its menu title. *[What is suitable for non-roman scripts and other*

languages or if a translator opts for a word title for the menu rather than a symbol?]

**Basic Bubble Behavior.** A click anywhere outside the menubar while in Bubble Mode causes a bubble window to appear for each mouse click. However, when dragging through menus, a bubble appears when the cursor enters a menu item and disappears when it leaves. This means bubble behavior is actually slightly different between menus and the desktop. However, since this difference mimics the mouse behavior associated with each area, users should have no confusion and find it quite natural. The difference in behavior does lead to some different considerations, especially in the case of menus. The issues relating to menus are discussed in their own section below.

Clicking anywhere on the screen causes a bubble to be displayed, even if only to say that there is no help associated with the area that was clicked. This prevents a first time "What Is" user from getting no response to a mouse click.

Keystrokes should cause Bubble Mode to be terminated. It can be assumed that users wanting to use the keyboard are finished with Bubble Mode and are resuming normal program operation.



### 3.2 Entering and Exiting Bubble Mode

The accepted means for entering and exiting Bubble Mode has not been generally agreed upon as of this writing. This is because there are a number of alternatives and no one of them is adequately satisfactory. Therefore, several entry/exit mechanisms will be discussed. User testing will be performed in the very near future to determine which entry/exit mechanisms are best.

Note that programming the entry/exit mechanisms are largely the responsibility of the application. The specific mechanisms chosen will be standard across all applications, but can be selected in due course without particularly affecting Help Manager development.

### 3.2.1 Entry Mechanisms

This section describes alternatives for entering Bubble Mode. The alternatives below are each expected to be implemented, but need to be tested for utility.

#### 3.2.1.1 "What Is" Menu Command

This command exists under the "Help" menu. Selecting it initiates Bubble Mode. While in Bubble Mode the menu title is inverted (signifying the command is still being performed). This is likely to be the standard way all applications allow users to enter Bubble Mode, but needs to be tested for simplicity and clarity of access.

#### 3.2.1.2 HELP Key



The Apple Extended Keyboard has a HELP key on it. While this key is depressed Bubble Mode will be enabled. Releasing the HELP key turns Bubble Mode off (i.e., "spring-loaded" action). Entering Bubble Mode via the HELP key requires only that the cursor be changed to the standard Bubble Mode shape; no other Bubble Mode indications need be done (such as inverting the Information menu title). The HELP key is supported on all machines with an attached Extended Keyboard.

Redundant Bubble Mode. The HELP key is not available on all machines. Consequently, an alternative entry mechanism is necessary which IS universally available (i.e., the "What Is\_?" menu item). With two mechanisms it is possible for the user to try to redundantly operate (i.e. press the HELP key after having entered Bubble Mode via "What Is\_?"). In this case, exiting Bubble Mode while the HELP key is down (by some means other than releasing the key) leaves the application in Bubble Mode. Releasing the HELP key then turns off Bubble Mode.

Mouse Accessibility to Help. There should always be a mouse-accessible means of gaining Bubble Mode in situations where an application supports bubble help. Users needing assistance can be counted upon to scan the screen and menus looking for help. Relying only on the HELP key shuts out users who do not know it exists (the very population most likely to need the help). The HELP key is really only intended as a short-cut (albeit a universal one) for getting quick help.

HELP Key During Dragging. Note that the existence of the HELP key means that a user can put the application into Bubble Mode at any time: Even during a drag event (such as when a menu is pulled down). Menus seem like

the only case of much significance. Pressing HELP while dragging an icon, extending a selection, or scrolling, does not seem to call for bubbles immediately popping out. The consequence of pressing HELP while a menu is pulled down is discussed in the menu bubble help section of this document.

### 3.2.1.3 Alternative HELP Key

A single key-combination (such as CMD-SHIFT-~) can be set by a user from the Control Panel to act as the HELP key (see figure below). While pressed, it functions identically to the HELP key as described above. A suitable default key is initially set for every system. (CMD-SHIFT-~ is suggested: It has the advantage of limited current usage among applications and is fairly closely grouped for one-handed access). Alternative HELP keys must be supported for all keyboard models.

Allowing users to custom select the HELP key solves the problem of conflict with existing applications, but also means the HELP key can change from one person's machine to another. The desirability of this feature needs to be established by user testing.

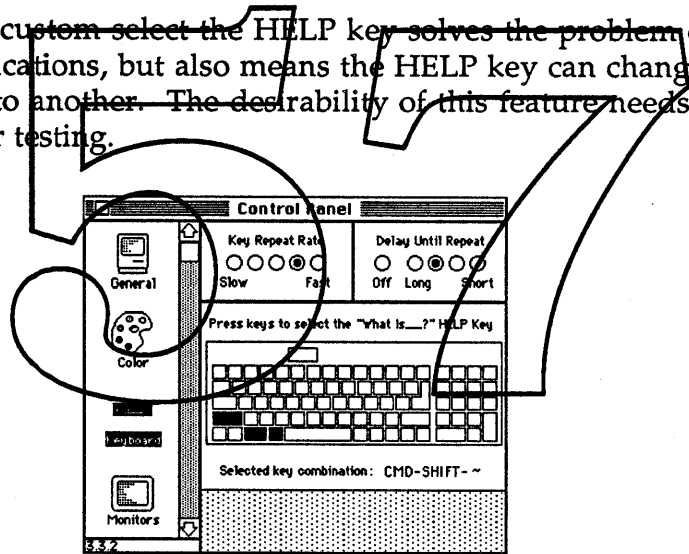


Figure 8 - Alternative HELP Key CDEV

### 3.2.1.4 Button or other Screen Graphic

Other mechanisms are certainly possible for entering Bubble Mode. An example might be buttons in dialog boxes, pop-out menus as previously mentioned, or a "Help" tool in a tool palette which causes bubbles to be displayed. While applications can customize their own way of allowing access to help, they should only do so when the standard ways of doing so are insufficiently effective.

### 3.2.2 Exit Mechanisms

The following describes alternative ways a user exits Bubble Mode. As with the entry mechanisms these are going to be tested for their user interface utility.

#### 3.2.2.1 "What Is" Menu Command Toggle

The "What Is" menu item is checked on entry to Bubble Mode. Selecting the "What Is" command again turns Bubble Mode off. It is the only menu command that has any effect in Bubble Mode. This mechanism has the drawback that it may not be self-evident, especially with relatively naive users. However, it is readily implemented and can be universally available [except in HyperCard...].

#### 3.2.2.2 HELP Key

As noted above, releasing the HELP key (or alternative HELP key), causes the application to leave Bubble Mode.

#### 3.2.2.3 Clicking in the Bubble

An italicized string is displayed at the bottom of every bubble that says "Click in bubble to exit". Clicking in the bubble terminates Bubble Mode.

*[What's italicized Chinese and Arabic look like...?]*

Alternative wordings for the italicized message are:

- Click here to quit
- Click here to close
- Click in bubble to exit

Exiting Bubble Mode this way requires that a bubble be visible on the screen at all times. This means either that there must always be a bubble on the screen for the user to click. The figure below shows that bubble.

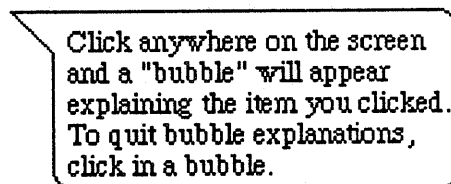


Figure 9 - Bubble Mode Explanation



The precise wording of all the messages must still be worked out. All suggestions are gratefully appreciated.

### 3.2.2.4 Close Box in Bubble

It has been suggested to let the user click a close box in the bubble to turn off Bubble Mode. The figure below shows such a bubble. This still requires that a bubble always be available to click in; as a feature, it needs to be tested for user comprehension, but has the advantage that it does not particularly change the bubble's esthetics.

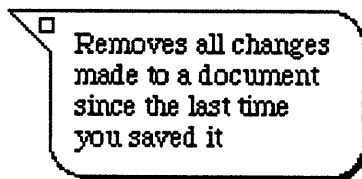


Figure 10 - Bubble Window with Close Box

### 3.3 Bubble Window Display

Bubble windows have several aspects as to how they are drawn:

- a. **Frame.** The frame is a rounded rectangle with a triangular section at one corner. The triangular section can appear at any of the window's four corners as described below. The window has a drop shadow.  
*[The bubble appearance is certainly open to comment and suggestion, especially as regards the drop shadow relative to the pointer being in each of the four corners.]*
- b. **Positioning.** The window's location is biased away from the object being pointed at and towards the screen's center. The biasing means that if the object is towards the right of the screen the bubble window will appear to the left of the object, if towards the upper screen the bubble appears below the object, etc. The bubble tip (the triangular section) is placed at the window corner nearest the object so that it "points" at the object.
- c. **Sizing.** A bubble window's size is dependent upon the amount and type of information to be displayed within it.

Help Information and Hotspot. The above features can be entirely calculated from two items of data: The help information and a hotspot. The help

information is the message to be displayed for a given bubble window, while the hotspot is the screen point where the bubble's tip is to be placed.

Help Data Types. Help information can be of several types: Text, styled text, or QuickDraw pictures. The window's size is calculated from the help information as described below.

Hotspot Location. The hotspot will typically be at the center of an object or just off to one side. When the spot is off to one side, the offset direction must follow the biasing-towards-screen-center rules described above else the bubble window can wind up overlaying the very item being described. (An example of this is menu bubble windows where the bubble is biased towards the side of a menu item - see below.)

The Bubble Layer. Bubble windows take precedence over the display of anything else on the screen. This means that a bubble is always the top-most item being displayed and therefore always fully visible (it is even above the "plane" in which menus appear). Only one bubble appears on the screen at a time so that displaying a new bubble window causes any previous one to disappear.

### 3.4 Help Information Display Sizing

Dynamic Bubble Sizing. Bubble window size is calculated from the size of the help information: String length information for text, and bounding rectangle for QuickDraw pictures. Dynamically calculating a bubble's text display means a help developer need only provide the help text. The calculation also simplifies translating the help information into a foreign language.

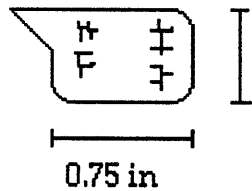
Size Calculation. The window size can be taken directly from the bounding rectangle for pictures. Text data is slightly more involved. The string length is used to calculate the pixel area needed to draw the text. This area is used to calculate a rectangle with a height-to-width ratio of two-to-three (two-to-three is roughly a golden ratio). The Text Edit manager is used to draw the text. This produces a rectangular box of text from which the window's size can be calculated. Given the sizing information and hotspot location the window's positioning and frame can also be calculated.

Italicized Quit String. The italicized string is added to the bottom of window to tell users how to exit from Bubble Mode. This must be done for both text and picture help information. *[Where's the string go for international scripts and pictures?]*

Screen Boundaries. A bubble window containing text data should be forced to fit entirely on the screen even if this means ignoring the above sizing

calculation. If there is insufficient screen space to display all the help information, the bubble should be clipped by the viewable screen boundaries with text being truncated from the end of the string. *[Is text truncated from the end for backwards scripts (i.e. Hebrew and Chinese)?]* This may mean there are aberrant cases that display only some fraction of the help information.

Likewise, if there is insufficient room to display picture data it is simply drawn to its normal dimensions and the bubble is drawn partially off screen.



Minimum Bubble Size. A bubble should never be narrower than 0.75 inch or less than 0.5 inch high (say 1.5 cm by 1 cm). This is both for esthetic purposes as well as for setting boundary limits.

### 3.5 Menu "What Is" Help

This section discusses the "What Is" user interface directly related to menus.

Menu Bubble Behavior. When in Bubble Mode, a bubble window is displayed next to each menu command as the mouse is dragged through a menu. The bubble is displayed only while the cursor is in the command. Commands with no associated help information have a bubble displaying "No help information available". Releasing the mouse causes the menu and any displayed bubble to be removed and the bubble in Figure 9 to appear.

Menu Item Flashing. Menu items are not inverted while the mouse is being dragged through them. There were a number of circumstances during user testing when users expected a command to be "live" because the menu seemed to otherwise behave normally. Avoiding the standard inversion behavior gives users a visual cue that the menu is in fact not live *(it's memorex)*.

HELP Key Revisited. The HELP key may be pressed or released at any point during menu operation. When it is pressed while a menu is pulled down Bubble Mode behavior should immediately ensue (i.e. bubbles appear and menu inversion is turned off). Releasing the HELP key (and leaving Bubble Mode) should immediately re-enable normal menu behavior.

Bubble Positioning. The exact positioning of the bubble depends upon the type of menu being accessed. Generally, this should be to the side of the menu so the item is not obscured. Palette menus (or other rectangularly

arrayed menus) may need to overlay onto the menu to make the bubble point to the item (see the figure below).

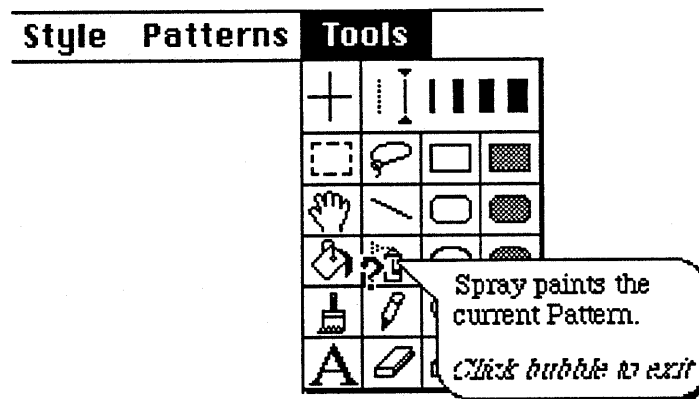


Figure 11 - Bubbles for Non-standard Menus

Text Menu Bubble Placement. For the system standard text menus, as each bubble is displayed it is positioned either to the immediate right or left of the menu command (see Figure 1). Which side the bubble appears on depends upon which side of the screen the menu itself appears: Left side menus have bubbles appear to their right, and right side menus have bubbles on their left. (This is in accordance with the bias-towards-center bubble positioning technique.) Menus which straddle the middle of the screen have bubbles positioned to their left, on the presumption that the left will almost always have sufficient room to accommodate a bubble (bubble size is not known until it is displayed). The right side is more likely to be squeezed when an unusual menu is being displayed.

### 3.5.1 Menu Titles

When the cursor is in the menu title, a bubble is displayed which gives an over-all description of the menu's purpose. The hotspot for this bubble is positioned immediately below and to the middle of the menu title. Its appearance is shown by the figure below.

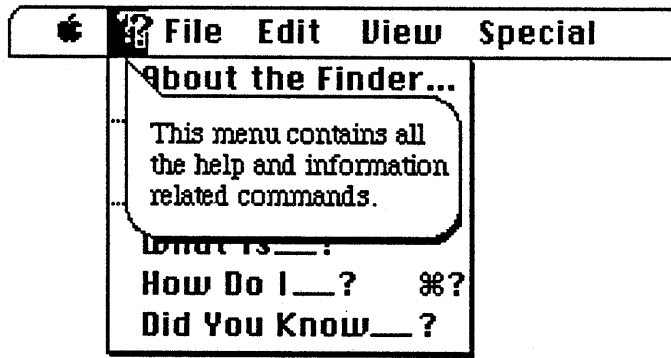


Figure 12 - Sample Menu Title Bubble Window

### 3.5.2 Alternative Menu and Item States

When a menu or menu item changes state, such as being disabled or having its text checked, the corresponding bubble information should change where it makes sense to do so. There clearly are cases where the help information can remain the same, but there are other cases where users would find it very helpful to have a different description. It should be up to an application's help author as to how the current state affects the help message.

The help information for all menus and items should be able to depend upon an item's enabled/disabled state. Help information for standard text menus can also depend on whether or not the item is currently checked.

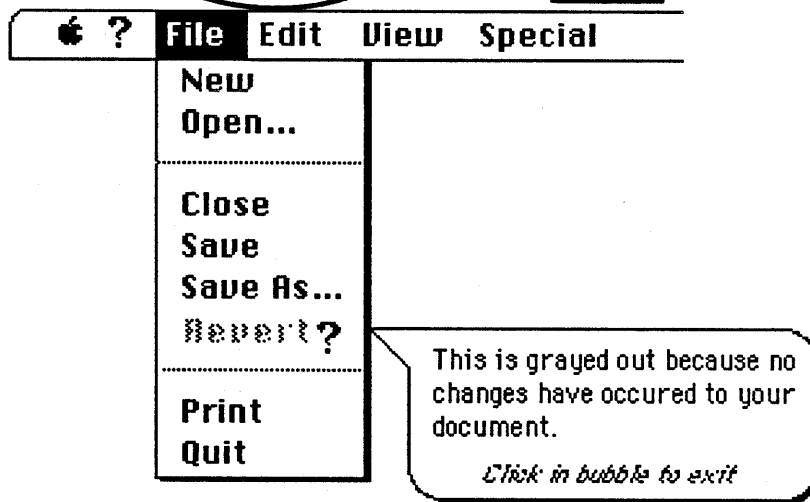


Figure 13 - Sample Help Message for a Disabled Item

Bubble Message Look-up Priorities. For standard text menus, the following priorities will determine which help information to display when several are available:

1. If an item is disabled, display a disabled-state bubble message,
2. If an item is enabled and has a check character, display a check character associated message,
3. Else display the standard bubble help message.

Likewise, when an entire menu becomes disabled, the disable-related messages should be displayed.

Alternative Message Guidelines. As a guideline, the alternative messages should give a user more information than just that "the item is disabled" or "You have selected (i.e. checked) this item" (sic). The additional messages should provide at least an indication as to why the state is altered or as to how it can be changed back. This is especially true as to disabled menu items: Users often times are mystified as to why a given menu item is disabled and would appreciate some illumination.

### 3.5.3 Miscellaneous Menu Issues

#### 3.5.3.1 Duplicate Menu Item Strings

All standard text menu items with help information must have a unique name. This is because the item string is used as the look-up key for accessing the item's help information. This is a fairly simple system and reasonably robust. However, this look-up system does not work when an application has duplicate menu strings.

Duplicate menu strings are disambiguated by first doing a look-up on the menu string itself. If this fails to find anything, the menu title will be prepended to the item string to perform a look-up (i.e. "File:Open"). *[A colon separator would be very nice; do Chinese and the like believe in colons?]*

Note that another technique to obtain uniqueness, albeit a more obscure one, is to append blanks to one or more of the item strings. This has the advantage of simplicity, but would abhor too many readers to be seriously proposed. *[Besides, do all scripts have blank characters?]*

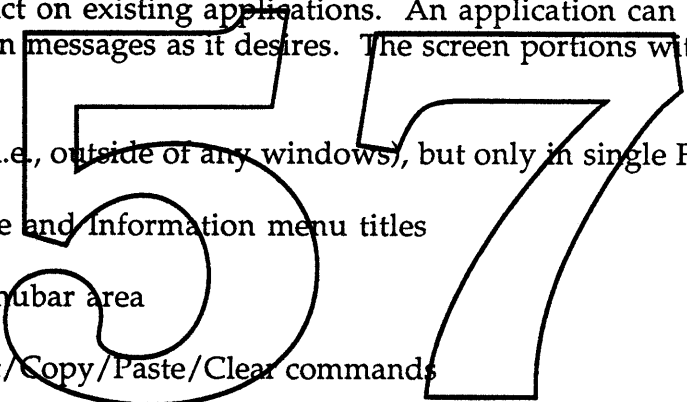
### 3.5.3.2 Hierarchical and Scrolling Menus

Menu items having associated sub-menus must function normally even when in Bubble Mode - the sub-menu must pop-up rather than a bubble appearing for the menu item. Consequently items with sub-menus have no bubble information displayed for them and the sub-menu's parent item inverts normally.

Scrolling menus must also behave normally - i.e. scrolling when the bottom or top is reached.

### 3.6 Standard Bubble Messages

Certain parts of the screen have standard bubble messages associated with them. This allows them to be moved, changed, or new features to be added with minimal impact on existing applications. An application can suppress or substitute its own messages as it desires. The screen portions with standard messages are:

- 
- Desktop (i.e., outside of any windows), but only in single Finder.
  - The Apple and Information menu titles
  - Blank menubar area
  - Undo/Cut/Copy/Paste/Clear commands
  - The standard help commands in the Information menu
  - Standard window features below:
    - Close box
    - Zoom box
    - Drag area
    - Growbox in a document window
    - Scrollbars located to the extreme right or bottom of a document window. The scrollbar parts will be differentiated.

Note that the growbox and scroll bars cannot be counted upon to be reliably interpreted for every application. In the somewhat rare cases when scrollbars at the extreme window edges do not scroll the document, the application can substitute its own messages or suppress the Help Manager's entirely. The same argument applies to the growbox.

### 3.7 Bubble Window Suggestions

This section lists comments and suggestions that might be incorporated within a bubble window implementation at some future point.

*Dialog Manager support.*

*Close box (persistent bubbles).*

*User editable help information.*

*Hypertext links and active graphics elements.*

## 4.0 "How Do I" Help

### 4.1 Functional Description

Selecting the "How Do I...?" command from the Information menu causes the How Do I (HDI) window to be displayed. In its initial state this window appears as shown by Figure 3. This window has two display rectangles: Topic and sub-topic. The topic area is initially displayed with a list of topics from which the user can choose. The sub-topic area is covered by instructions as shown in Figure 3.

HDI Miscellanea. The window has a number of special behaviors. For one, the window floats above all of an application's windows. This allows a user to readily refer to the help information while trying to use it. Also, the window is sized to display only the topic area if there are no sub-topics. This allows the HDI facility to also be used for a Glossary capability *[which is not being implemented initially...]* (see "On-line Help Summary Report" by Kathleen Gomoll for a definition of Glossary). Because of its floating windoid behavior, the window's appearance is different from standard document windows (dotted rather than lined drag area, and rounded corners).

Sub-topics and Getting Help. After clicking on a topic, the sub-topic area will appear (if there are associated sub-topics) and the window will appear as in the figure below. Clicking a sub-topic will cause the window to display the selected help information (see Figure 15). (During testing, users indicated they had a strong expectation that just a single click, rather than a double click, would display the help information.) The Open button is enabled only



when help information can be presented. It exists because it is possible to have help associated with a topic whether or not it has associated sub-topics.

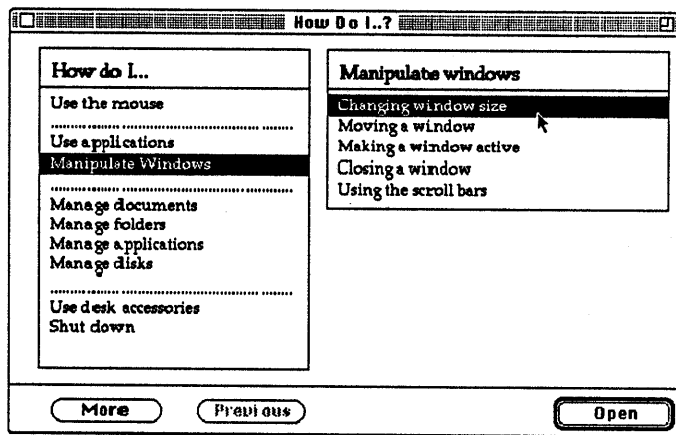


Figure 14 - "How Do I" Topics and Sub-Topics

*[What are the implications for displaying the topic/sub-topic lists in vertically oriented script systems?]*

**Paging.** The More and Previous buttons in the figure above function to allow a user to page the Topic area. *[Alternative paging mechanisms are being investigated - such as a general purpose paging "control".]* The sub-topic area cannot be paged or scrolled. This places a limit on the number of sub-topics that can be associated with a topic. Paging is used in the topic area rather than scrolling to allow the help author better control over how the topics are presented. It has been general observation confirmed by research that such lists are more comprehensible when carefully laid out and that paging gives the best layout control.

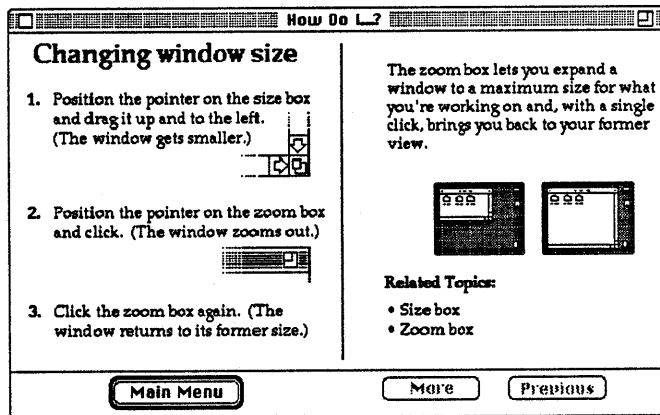


Figure 15 - "How Do I" help message

Help Information Display. The help information in the figure above can be paged forward and backward via the More and Previous buttons. The window can also be zoomed to show one or two pages of help information (see figure below). Paging causes the entire screen to be replaced by preceding or following help information (i.e., two "pages" at a time). If only one page of help information exists the window will be displayed in the one page size. Closing and reopening the window (without quitting the application) does not change the window display (whether displaying help information or the topics list).

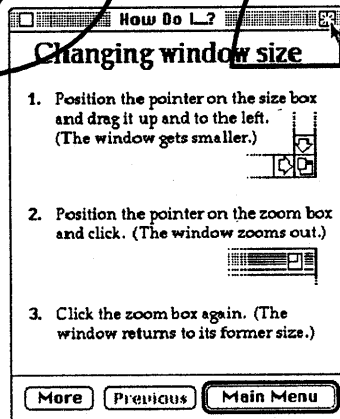


Figure 16 - Unzoomed One Page HDI Help Display

Help Information Overflow. Picture information which is too large to fit on a page is truncated at the right and bottom. Text data is simply flowed onto successive pages (where a page is half of a full screen). *{How are*

*"international" pictures with backwards/upside down scripts truncated?]*

**Multiple Topic Lists and Direct Help Display.** The HDI facility has a number of generalizations to broaden its utility. First off, it is possible for an application to have more than one topic list. This allows the application to have contextually dependent help systems; the one to be used can be selected from context. It is also possible for the application to have the HDI window open to a specific information display without forcing the user to go through the topic/sub-topic stage. This allows an applications to support a more intelligent help facility when it is able to anticipate a user's need.

An example usage of the above generalizations might be a "Why" button in an error message dialog. Clicking the "Why" button would display help information which gave possible explanations for the error. It would also say to click the Main Menu button for more information. Clicking Main Menu would give a topic list which listed the general explanation (just read by the user) and also listed more detailed explanations for each of the possible causes of the error. Since the window floats it can still appear over modal dialog and alert windows. *[Is this going to cause Multifinder conrptions?]*

#### 4.2 Help Fonts

Considerable emphasis is being placed upon well laid-out help information, even to the extent of forcing help authors to work within fixed page boundaries. This is all well and good except when the information must be displayed in a font different from the one in which it was laid-out. Consequently, help authors must have reasonably strong assurance that the fonts they use will indeed be available at the time their help information is displayed.

*[Another plea for some nice fonts in ROM...]*

#### 4.3 How Do I Suggestions

This section lists comments and suggestions that might be incorporated within a HDI implementation at some future point.

*Thesaurus and key-word search capability*

*User editable help information*

*Active graphics elements supporting Hypertext (and HyperCard-like) links to other help displays*

*Multi-media displays and/or access such as sounds and CD-ROM*

*Animated example help*

*Allowing users to gray-out or hide topics, sub-topics, and specific pages of help information.*

*Customized bookmark notations*

## **5.0 Did You Know Help**

Did You Know (DYK) help allows a user access to little-known program features that "power users" often know about. Study has indicated that the DYK facility can be an efficient way to increase a user's facility with an application.

Selecting the "Did You Know\_?" command from the Information menu causes the DYK window to be displayed as shown in Figure 4. This window displays a useful or obscure piece of information that users often do not know about. Pressing the Browse button causes another item to be displayed. DYK sequences through its list of facts, starting over again once all facts have been displayed.

The Index... button displays a list of all DYK entries (this is just the HDI main topic window). Selecting an item from the list displays it and starts the sequencing from there.

The window is non-modal so users can send it backwards and bring it forwards. Closing and re-opening the window does not change the displayed item.

As with the HDI system, there can be more than one list of facts so that an application which has reason can display different lists in different contexts.

*[Is it useful to allow applications to open the DYK window to a specific fact?]*

## 5.1 Suggestions

*Bookmarks*

*Allowing users to selectively suppress entries*

*Novice/expert settings*

*Being able to start the list from the beginning*

*Animated Examples*

## 6.0 Additional Help System Capabilities

This section describes a number of capabilities which research indicates could be very useful, but which time makes unlikely will appear in an initial release of the Help Manager.


*Bookmarks - letting users note where information was found.*

*Glossary - "What Is" information for items and concepts which don't have an immediate screen graphic.*

*Notes - Attaching customized notes to Bookmarks*

*Thesaurus search capability in HDI*

*Printing the Help Manual*

The Human  Interface Group

# Help Manager Programmer's Toolbox Interface

ERS  
57

James Sulzen

March 22, 1989

Version 0.5

Please return comments and/or markups to me at:

408-974-3142, or

Apple Computer, M/S 27-AO, or

AppleLink: Sulzen

APPLE COMPUTER

COMPANY COFIDENTIAL

# Table of Contents

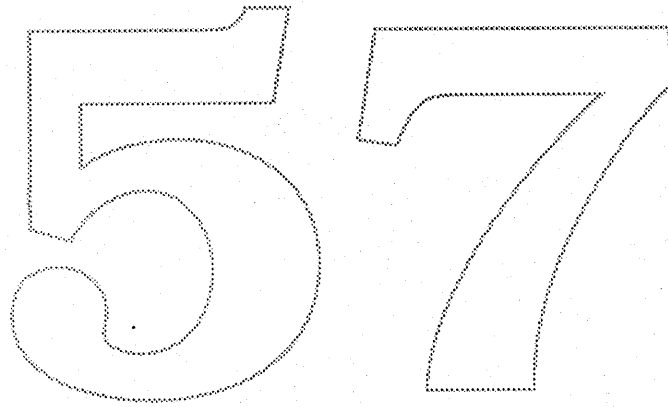
|                                                      |    |
|------------------------------------------------------|----|
| 1.0 Introduction.....                                | 1  |
| 2.0 Programmer's Model.....                          | 1  |
| 3.0 Help Displays.....                               | 3  |
| 4.0 Help Data Base Organization.....                 | 5  |
| 4.1 General.....                                     | 5  |
| 4.2 Help Keys.....                                   | 5  |
| 4.3 Help File Chain.....                             | 5  |
| 4.4 Help File Organization.....                      | 5  |
| 4.4.1 HelpRec Organization.....                      | 5  |
| 4.4.2 Record Types.....                              | 7  |
| 4.4.2.1 End User Help Information Records.....       | 7  |
| 4.4.2.2 Aggregate Records.....                       | 8  |
| 4.4.3 HelpRecords in Resource Files.....             | 8  |
| 4.5 Help File Resources.....                         | 9  |
| 4.5.1 Help DRVRs.....                                | 9  |
| 4.5.2 finf Default Text Font and Style Resource..... | 10 |
| 4.6 Dialog Help Items.....                           | 10 |
| 5.0 Data Structures.....                             | 11 |
| 5.1 HelpDisplayRecord.....                           | 11 |
| 6.0 Defining A Help Manager DRVR.....                | 12 |
| 6.1 General.....                                     | 12 |
| 6.2 DRVR Messages.....                               | 13 |
| 6.3 DRVR Chaining.....                               | 15 |

|       |                                                                       |    |
|-------|-----------------------------------------------------------------------|----|
| 6.4   | Standard System DRVR's .....                                          | 15 |
| 6.4.1 | Standard System HowDoI DRVR.....                                      | 16 |
| 6.4.2 | Standard System DidYouKnow DRVR.....                                  | 16 |
| 6.4.3 | Standard System Bubble DRVR.....                                      | 16 |
| 7.0   | Using the Help Manager .....                                          | 17 |
| 7.1   | Application Support of User Help Access .....                         | 17 |
| 7.2   | Initialization .....                                                  | 17 |
| 7.3   | Help Data Base Support.....                                           | 18 |
| 7.4   | Bubble Mode Support .....                                             | 18 |
| 7.5   | How Do I Support.....                                                 | 19 |
| 7.6   | Did You Know Support.....                                             | 19 |
| 7.7   | Other Help Manager Procedures .....                                   | 19 |
| 8.0   | Help Manager Procedures.....                                          | 19 |
| 8.1   | Help Manager Initialization.....                                      | 20 |
|       | HMInit - Initialize the Help Manager .....                            | 20 |
| 8.2   | Bubble Window Support .....                                           | 21 |
|       | HMSetBubbleMode - Enable/Disable Bubble Mode.....                     | 21 |
|       | HMGetBubbleMode - Return the current Bubble<br>Mode setting.....      | 21 |
|       | HMGetNewBubble - Create a new bubble<br>HelpDisplay.....              | 22 |
|       | HMNewBubble - Create a new bubble without<br>using a help file.....   | 22 |
| 8.3   | Help Displays .....                                                   | 23 |
|       | HMGetNewDisplay - Create a new HelpDisplay.....                       | 23 |
|       | HMNewDisplay - Create a HelpDisplay without<br>using a help file..... | 24 |



|                                                                            |    |
|----------------------------------------------------------------------------|----|
| HMDisposeDisplay - Dispose a HelpDisplay.....                              | 24 |
| HMGetCurrentDisplay - Return current<br>HelpDisplay.....                   | 24 |
| HMSetCurrentDisplay - Set the current<br>HelpDisplay.....                  | 24 |
| HMSetSelection - Set the Current HelpDisplay's<br>Display Context.....     | 24 |
| HMGetSelection - Get the Current HelpDisplay's<br>Display Context.....     | 25 |
| 8.4 Information Routines.....                                              | 25 |
| HMDrawIcon - Draw the international help icon.....                         | 25 |
| HMParamText - Help Manager's ParamText<br>procedure.....                   | 26 |
| HMGetHelpRec - Get a HelpRec from the current<br>HelpDisplay.....          | 26 |
| 8.5 Help Data Base Access.....                                             | 26 |
| HMReadRecord- Read a HelpRec.....                                          | 26 |
| HMStuffNewRec - Create a new HelpRec in<br>memory.....                     | 27 |
| HMExtractRec - Obtain information from a<br>HelpRec in memory.....         | 27 |
| HMDisposeRec - Dispose a HelpRec which is in<br>memory.....                | 27 |
| HMGetRecEntry - Extract a HelpEntry field from a<br>HelpRec in memory..... | 27 |
| HMSetRecEntry - Insert a HelpEntry into a<br>HelpRec in memory.....        | 28 |
| 8.6 Help Files.....                                                        | 28 |
| HMOpenFile - Add a file to the current help chain.....                     | 28 |

|                                                                 |    |
|-----------------------------------------------------------------|----|
| HMCloseFile - Remove a file from the current<br>help chain..... | 28 |
| Appendix - HelpRec Formats.....                                 | 29 |
| A.1 Aggregate HelpRecs.....                                     | 29 |
| A.2 Picture HelpRecs.....                                       | 30 |
| A.3 grph HelpRec.....                                           | 30 |
| A.4 TEXT HelpRecs.....                                          | 31 |
| A.5 styl HelpRec.....                                           | 31 |



## 1.0 Introduction

This document defines the Programmer's Interface to the Macintosh Help Manager. It assumes the reader is familiar with the "Help Manager Human Interface ERS".

This document describes the programmer model (the way a programmer conceptualizes programming the Help Manager), the help data base organization and specific Toolbox Interface supported by the Help Manager. It describes the Help Manager in sufficient detail to allow a Macintosh programmer to implement an application's help facilities using the Help Manager.

Parenthetical comments in the *Los Angeles* font are interspersed in the text. These comments note unresolved issues and areas which may not be implemented or are likely to change from the description.

## 2.0 Programmer's Model

The basic model a programmer has of the Help Manager is as a set of routines and desktop drivers which pull help information from files (resource or data forks) on a user's disk and displays the information in an attractive and consistent way across all applications. The Help Manager is responsible for retrieval and display of the help information, and for handling user events to allow browsing of the help. The application is responsible for defining the Help Database (by opening the relevant help files), and for telling the Help Manager when and what kind of help to display.

The Help Database consists of the set of open help files. A help file consists of variable sized records called **HelpRecs**. Most HelpRecs contain end user help information (or more simply "Help Information"). However, HelpRecs are also used to contain indexing information (called Aggregate records). These records tell the Help Manager how to aggregate help items so a user can browse the data base. Additional help files may be opened and therefore added to the Database by the System or other software (i.e. DA's, packages, etc.).

A help file is created by a **Help Author** using the **Help Authoring Tool**. The Help Author is typically a technical writer who is responsible for creating an application's on-line help information. The writer will use the Authoring Tool to enter and organize information in an application's help files. The Tool is responsible for building help files which the Help Manager is able to access.

The Authoring Tool is the only software component that creates and makes major changes to help files. It alone contains most of the logic for writing and modifying help files. The Help Manager can read help files and can append simple text information (for user annotations).

An application can display one or more help-related windows which are known as **Help Displays**. A Display can handle events from the user, receive messages from the Help Manager, or receive messages from the application through the Help Manager. It responds to events and messages by accessing the help Database and updating the displayed information as appropriate.

HelpDisplays are implemented via a cooperative arrangement between the Help Manager and a Macintosh **Help Manager Driver** (or **HM DRVR**). The Macintosh Device Manager Driver mechanism is used to implement Displays because it provides a well-defined means for creating software entities which operate semi-independently of the application, but which can still process events and operate a window of their own.

Note that the HM DRVR mechanism used by the Help Manager is not exactly the same as a Desk Accessory. The HM DRVR name does not appear under the Apple Menu and the DRVR has a number of I/O calls it supports which DA's do not have. However, it does take advantage of a number of aspects of the Desk Manager's support for DA's. The details of HM DRVR's are more fully discussed in the section on how to write an HM DRVR.

The Help Manager supports two basic classes of help: **Bubble help** ("WhatIs") and **explanatory help** ("HowDoI" and "DidYouKnow"). A different type of HelpDisplay is used to implement each of these classes. An application may define its own classes as well.

The Help Manager provides a certain amount of support specific to Bubble help. It supports a **Bubble Mode** which can be enabled or disabled. When enabled and when a user clicks in a window or on the desktop, an application which supports Bubble Mode causes bubble windows to be appropriately displayed. During Bubble Mode, the System causes bubbles to appear for menus (via support from the Menu Manager). Additionally, the Dialog Manager can display bubbles for dialog windows as noted in the Dialog Manager section below. While Bubble Mode is enabled, the application can perform no functioning whatsoever except to display bubbles, allow menus to be pulled down and/or clicked (i.e., clicking in tear-off menus), and to exit Bubble Mode.

HowDoI/DidYouKnow HelpDisplays are created by a single call to the Help Manager. Once created, these Displays run as Macintosh Drivers and receive window and mouse events via the standard Desktop Manager event passing

mechanisms. The user may interact with them by clicking in their windows to browse the help Database. The HelpDisplays are disposed by clicking in their Closebox.

### 3.0 Help Displays

HelpDisplays are the basic display mechanism of the Help Manager. They encapsulate both window appearance and behavior.

Each HelpDisplay has one window associated with it. All events associated with that window (mouse downs, updates, activates, etc.) are handled by the HelpDisplay's HM DRVR code (see below for more information about HM DRVR's).

A HelpDisplay's window is divided into a number of logical areas as noted by the diagram below:

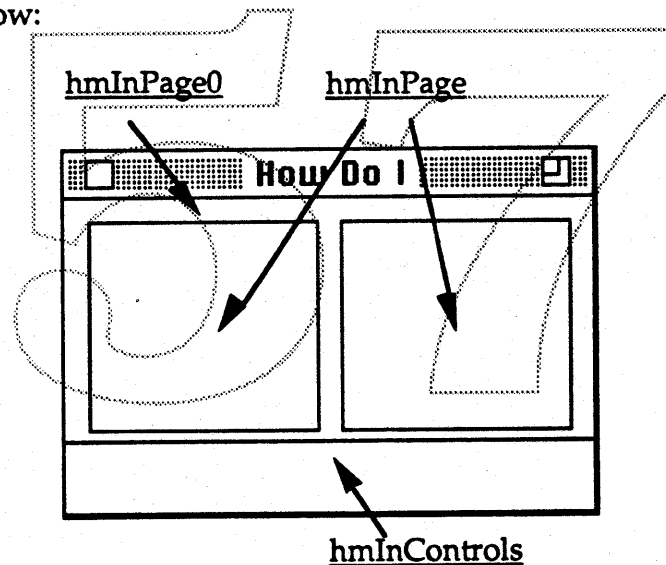


Figure 1 - Logical parts of a HelpDisplay

The meaning of each of the areas is as follows:

**hmInControls** Designates an area on the screen where window controls are generally placed. This may be a value from -1 to negative infinity. The different values allow different sets of controls to be displayed in the hmInControls area. The standard system HM DRVR's only respond to -1.

**hmInPage**

This area consists of the upper part of the HelpDisplay window's content region as noted in the diagram above. The hmInPage area is actually divided into one or more displayable panels (the above diagram has two display panels). There are an arbitrary number of "logical pages" associated with the panels. These pages are displayed in the panels in sequence by user manipulation of the controls in the hmInControls area. The pages are numbered starting with one and increasing in sequence.

**hmInPage0**

Designates the content area not in any of the other areas. This value can also designate the entire content region for simple HelpDisplays (such as for bubbles).

The exact size of the each area can be obtained by a call to a HelpDisplay's HM DRVR. An HM DRVR may support all or only a sub-set of each area as suitable for its own needs. The standard system HowDoI/DidYouKnow HM DRVR supports two display panels in the hmInPage area and one value for hmInControls (the diagram above illustrates this HelpDisplay organization). The standard bubble HM DRVR supports only one display panel and no hmInControls area. An HM DRVR should only respond to HelpDisplay area values which it knows about and ignore any others.

Since the hmInPage values range from one on up, they can be referred to as hmInPage1 for page 1, hmInPage2 for page 2, etc. Each page is associated with only one of the display panels. This association is determined by the HelpDisplay's HM DRVR. For the two panel HelpDisplay shown above, the left panel is associated with odd pages and the right panel is associated with even pages. The association means that knowing a page number means also knowing where that page is displayed.

The Help Manager supports three "levels" of Help Records for each HelpDisplay. The first level is associated with hmInPage1, the second level with hmInPage2, and the third level with hmInPage3 and all larger hmInPage values. More levels are possible, but the HM DRVR itself has to provide the support for the additional levels.

The contents of a level's HelpRec determines what is displayed in the associated page. For the system standard HowDoI/DidYouKnow HelpDisplays, hmInPage1 and hmInPage2 pages are used to display table-of-

contents information; the level three HelpRec contains end-user help information.

Much as there must be a current GrafPort before calling QuickDraw routines, there must be a current HelpDisplay for the Help Manager. It is the application's responsibility to insure the correct HelpDisplay has been setup (by calling HMSetDisplay) before Help Manager routines are invoked.

## 4.0 Help Data Base Organization

### 4.1 General

Each HelpRec has a HelpKey associated with it. An application passes a HelpKey to the Help Manager to get the associated HelpRec retrieved.

HelpKeys can be interpreted as calls upon the Resource Manager as explained below in the section on HelpRecords in Resource Files. This allows help information to be stored in resource forks such as the System and application files.

### 4.2 Help Keys

HelpKeys consist of up to 31 ASCII characters. The string itself is a Pascal String type with the first byte being a length count.

### 4.3 Help File Chain

When a file is first opened it is automatically added to the front of the Help Chain. Satisfying a search request for a particular HelpRec involves sequencing through the files on the Help Chain from most recently opened to least recently opened.

### 4.4 Help File Organization

#### 4.4.1 HelpRec Organization

HelpRecs have a specific format as defined by the diagram below.

*[Need to change field names to "hrXXXX"]*

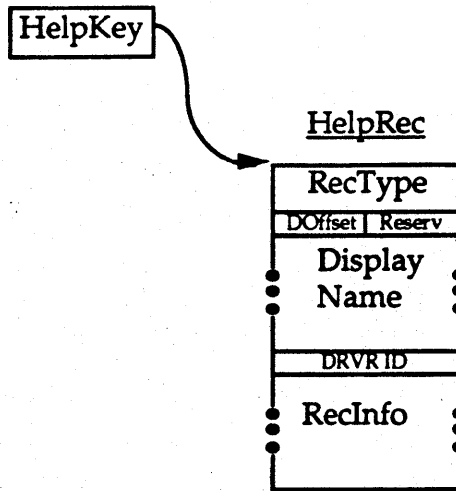


Figure 2 - HelpRec Organization

The fields of a HelpRec are defined as follows:

- RecType** A four ASCII byte value that defines the type of the HelpRec.
- DOffset** This is a one byte field which defines the byte offset from the beginning of the HelpRec to the RecInfo field. The DOffset field allows for a variable field area in the HelpRec. Values of less than six are reserved.
- Reserv** Reserved one byte field and must be zero.
- DisplayName** This is an optional field. The DisplayName is a Pascal string which is used to display this help item's name to the user. If it is not present, the HelpRec's key is used as the DisplayName. (A zero-length DisplayName string means this field is unused.) The DisplayName field is padded with nulls if necessary so that the following field starts on a word boundary.
- DRVRID** This is a word-length optional field. The DisplayName field must be present if this field exists. If present, it is a HelpDisplay ID to a resource of type "DRVR". This value is identical to the hdisplayID parameter in the HMGetNewDisplay call. A value of -32768 means this field is unused.



**RecInfo** This field is variable-length and contains the record's help information.

The above record organization is intended to support a very flexible and extendible data structure. It is designed to allow help data bases to be readily extended in the future, but still function with earlier versions of the Help Manager.

#### 4.4.2 Record Types

Every HelpRec has a RecType value which defines the record's data type, and to some extent defines the record's format. The RecType information is used by the Help Manager to determine how to use the record's RecInfo. A RecType is a four character ASCII value (RecType values are typed as ResType's).

Help Information records contain data that is displayed directly to a user. Aggregate records contain keys to other HelpRecs. These keys typically point to either other Aggregate records or Help Information. The Help Manager utilizes several types of Help Information and Aggregate record types. Applications can define their own types as they see fit, so long as the new types do not conflict with current Help Manager usage.

The RecTypes recognized by the Help Manager are defined in the following section.

##### 4.4.2.1 End User Help Information Records

Records which contain end-user help information can be one of several types:

|              |                                                                                                                                                 |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>TEXT</b>  | The data is raw text data and will be formatted by Text Edit for display.                                                                       |
| <b>STR</b>   | The data is a single pascal string.                                                                                                             |
| <b>styl</b>  | Same as TEXT, but the data is TE's styled text data.                                                                                            |
| <b>PICT</b>  | QuickDraw picture data.                                                                                                                         |
| <b>grph</b>  | Mixed picture, TEXT, or styl data. This data type allows text separation from graphic information which simplifies localizing a help data base. |
| <b>other</b> | Other types of information are possible. When the Help Manager encounters a non-standard data type it invokes the                               |

HelpRec's DRVR function to display the help data. If the HelpRec has no drvID, then the HelpDisplay DRVR is called. System standard DRVR's ignore unknown RecTypes.

Figures defining the above four record types are given in the appendix at the end of this document.

The redundancy in data types (i.e. TEXT and styl, PICT and GRPH) is to support different levels of sophistication among application developers. The TEXT type is supported partially for developers porting from other development platforms. They may have much help information in TEXT format, but not have the resources to carefully format it into a nice PICT layout. The TEXT format is also useful for simple bubble help messages stored in a resource fork. The PICT type is supported because it is an extremely common type of data on the Mac and is directly supported by QuickDraw. The grph type is expected to only be created by the Authoring Tool.

#### 4.4.2.2 Aggregate Records

Aggregate records are used by the Help Manager to support multiple HowDoI and DidYouKnow HelpDisplays from the same Help Database.

- |             |                                                                                                                                                                                                                                                                                                               |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>INDX</b> | Contains a list of HelpKeys. These HelpKeys are used to construct lists of help information that a user can access. Selecting an item in the list causes the corresponding HelpKey to be used to load and display the item.                                                                                   |
| <b>LIST</b> | Similar to an INDX HelpRec, but it is interpreted as a list of HelpRecs. This list is treated exactly the same as if the HelpRecs it points to were all "packaged up" in one large record. A LIST allows a help author to have different ways to organize the same help information within the same Database. |

The above record types are also defined in detail in the appendix.

#### 4.4.3 HelpRecords in Resource Files

When the Help Manager is unable to locate a record by searching down the Help Chain, it will interpret the HelpKey as a Resource Manager request. It does this by using the first four characters of the HelpKey as a resource type and the succeeding characters as either a resource ID or resource name. If the fifth through last characters in the HelpKey constitute a valid ASACII integer,

they are converted to a sixteen bit integer to use as a resource ID. Otherwise, the fifth through last characters are used as a resource name. Then either GetResource or GetNamedResource are called as appropriate to retrieve a HelpRecord.

The search follows the current Help Chain, NOT the resource chain.

When a help resource is found in this way, DetachResource is called to disassociate the help item from the resource file. The resource is also re-organized by formatting it into a HelpRec. The reformatting that occurs is described by the diagram below.

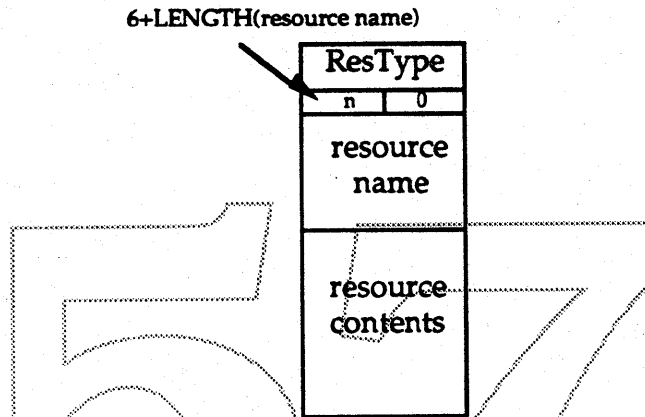


Figure 3 - Resource help item converted to a HelpRec

The HelpRec's RecType is set to the resource's ResType, the DisplayName is set to the resource name (if there is one) and the RecInfo is set to the resource contents.

## 4.5 Help File Resources

Each help file may have a number of resources that are used by the Help Manager. This section describes those resources.

When a help file is first opened, its resource fork is also opened. This allows Help Manager resources to be retrieved from the help file. These resources are used as described in the following sections.

### 4.5.1 Help DRVRs

Each HelpDisplay has an HM DRVR associated with it at the time the Display is created. Additionally, each help file can also have an HM DRVR with the

same resource ID as given by the HelpDisplay's DRVRIID. Besides these two DRVRIID's, the application and System resource files can have DRVRIID's with the same ID. These four DRVRIID's are chained together and collectively can define the HelpDisplay's behavior and appearance. This capability is described in detail in the section on DRVRIID's below.

#### 4.5.2 finf Default Text Font and Style Resource

A finf,0 resource defines the default size, styling and font to be used for TEXT data derived from a file.

### 4.6 Dialog Help Items

When Bubble Mode is enabled, the Dialog Manager will display bubble windows for items in the dialog window. When the Dialog Manager receives a mouse down event for one of its windows, it generates a HelpKey of the following form:

"TEXT" + <HELP\_STATIC\_TEXT> + <item\_number>

where the plus sign denotes string concatenation.

The <item\_number> is the Dialog Manager's item number converted to positive ASCII integral value before the concatenation. The <HELP\_STATIC\_TEXT> is the content string of the last static text item in the item list. The item must be disabled, its rectangle must not be visible, and the item string must begin with the letters "HELP". The above formulation allows simple TEXT strings to be associated with each of a particular dialog's items. *[The word "HELP" must be internationalized.]*

## 5.0 Data Structures

### 5.1 HelpDisplayRecord

Every HelpDisplay is defined by a HelpDisplayRecord which is defined below.

```
TYPE HelpDisplayRecord = RECORD
    hdRecSize:      INTEGER;      (# of bytes in this record)
    hdNextRec:     HelpDisplayHandle; {link to next rec}
    hdHotspot1:    Point;          {primary hotspot for bubbles}
    hdHotspot2:    Point;          {secondary hotspot}
    hdRefcon:      LONGINT;        {appl defined field}
    hdWindow:      WindowPtr;     {Display's window}
    hdNPages:      INTEGER;        (# of displayable pages)
    hdControlPage: INTEGER;        {cur inControlPage value}
    hdDrvRRefnum:  INTEGER;        {refnum slot in Unit Table}
    hdDrvRID:      INTEGER;        {the original drvRID}
    hdDrvChain:    Array[0..3] of HDDrvRRecord; {Drv chain info}
    hdHelpLevels:  Array of HDLevel[0..0];
END;

HDDrvRRecord = RECORD
    hdDrvRHandle:  Handle;          {handle to DRVR code}
    hdDCE:         DCtlHandle;     {DRVR's DCE}
    hdDrvRFlags:   INTEGER;        {0:DrvR init'd}
END;

{Interface file needs to be changed to HDLevel from HDPAGE}

HDLevel = RECORD
    hdHelpRec:    HelpRecHandle; {HelpRec for this page}
    hdKey:        HelpKey;       {HelpKey for this HelpRec}
    hdPage:       INTEGER;       {Page this rec is set to}
    hdItem:       INTEGER;       {selected item w/in the helprec}
END;

HelpDisplayPtr = ^HelpDisplayRecord;

HelpDisplayHandle = ^HelpDisplayPtr;
```

## 6.0 Defining A Help Manager DRVR

### 6.1 General

Help Manager Drivers (HM DRVR's, or more simply, just DRVR's) are desk accessories with extended functionality, but which do not appear in the desk accessory menu. The desk accessory architecture is used because it is an existing mechanism which allows a body of code to process events independently of the running application. DA's (or more properly speaking, device drivers) also have other capabilities which can be exploited by the Help Manager, such as menus and being able to run in parallel with the application.

Macintosh device drivers accept a control code in the csCode field of the paramblk passed to a driver's control routine. This control code tells the driver what control function to perform. Additionally, one or more parameters for the control call are passed in the paramblk's csParam field. For the purposes of simplifying the discussion below, the various values of csCodes will be referred to as "messages" and the corresponding values of the csParam field will be referred to as parameters for the call.

The general operation of a HM DRVR is as follows. The application tells the Help Manager to create a HelpDisplay. The Help Manager opens the required HM DRVR which causes the Device Manager to allocate a Unit Table slot and a Device Control Entry, and to call the HM DRVR's Open routine. The Open routine allocates the DRVR's global data storage and returns.

The Help Manager then passes the hmCreateWindow message to the DRVR. The DRVR should create the HelpDisplay window and set the WindowPtr into the hdWindow field of the HelpDisplayRecord. The DRVR should also define the number of displayable pages by calling HMSetDisplaySize.

The Help Manager will then call the DRVR with the hmDisplayPage message for each displayable page. The DRVR is passed the page number to display, and if relevant, the item within the page which is selected.

After this the user may interact with the HelpDisplay by clicking in it. As the user does so, the Desk Manager passes events directly to the DRVR. The DRVR should respond to these events as appropriate.

A DRVR may decline to service a particular control call and can pass it on to the next DRVR in the chain by calling HMPassCall. This will cause the Help

Manager to select the next DRVR in the chain and call it with the appropriate parameters.

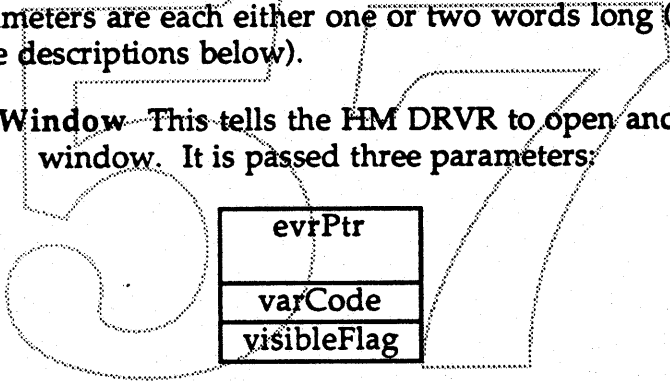
Note that there are a number of small integral parameters that are passed to a DRVR by the Help Manager (such as the HelpDisplay area values). In general, a DRVR must only respond to such values it knows about and ignore all other values.

## 6.2 DRVR Messages

HM DRVR's can receive a number of messages. The Desk Manager can send any of its messages that it would normally send to Desk Accessories (see the Desk Manager Chapter of IM). The Help Manager can also send a number of messages as detailed below.

The descriptions below define the messages the Help Manager can send to a DRVR. The parameters are each either one or two words long (thin boxes or thick boxes in the descriptions below).

**hmCreateWindow** This tells the HM DRVR to open and initialize a window. It is passed three parameters:



|             |
|-------------|
| evrPtr      |
| varCode     |
| visibleFlag |

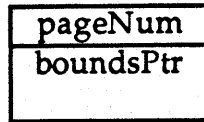
where evrPtr is an address of an EventRecord associated with the creation of the HelpDisplay, varCode is the low four bits of the drvID parameter from the HMGetNewDisplay call, and visibleFlag is TRUE if the window is to be initially visible.

**hmSizeWindow** This tells the DRVR the number of displayable pages the HelpDisplay should have. The DRVR should change the window size appropriately. However, it should not change anything displayed in any of the pages. The Help Manager will do this by messaging hmDisplayPage.

|          |
|----------|
| numPages |
|----------|

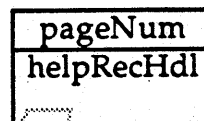
where numPages is the number of display pages.

**hmGetBounds** This message tells the DRVR to return the bounding rectangle for the given page.



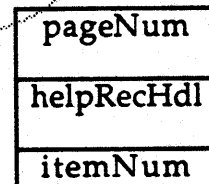
where pageNum is the page number to get the bounds of, and boundsPtr is the address of a rectangle where the bounds are to be stored.

**hmDisplayPage** This message tells the DRVR to actually draw a page. Its parameters are as follows:



where pageNum is the page number to be drawn, and helpRecHdl is a handle to a HelpRec which is to be used to draw the page.

**hmSelectItem** This tells the DRVR to select a particular item within the given page. The effect of this call is exactly the same as if the user had just selected the given item. The csParam field is passed as follows:



where pageNum is the affected display page number, and helpRecHdl is the HelpRec for the page, and itemNum identifies the item within the page to be selected (if non-zero).

Before calling the DRVR with a message the Help Manager sets up a number of things. It sets the Help Manager's current Display to the DRVR's HelpDisplay and sets the current grafport to the HelpDisplay's window. For drawing, it also makes some adjustments to the HelpDisplay's GrafPort. In particular it does a SetOrigin to the affected part of the Display so that the



area's topLeft point is the window's origin. Also, the clipRgn is set to encompass only the affected Display page. (Note that in the case of hmInPage0 this includes the entire window content region.) When the HM DRV is called with a Desk Manager message, it must do any necessary set-up itself.

### 6.3 DRV Chaining

A HelpDisplay can have a number of DRV's associated with it as noted above. In particular when a HelpDisplay is created its drvID value is used to retrieve possibly up to four DRV's. These four DRV's are put on a logical "chain" and can be called in sequence to execute a single message.

The first DRV is derived from the initial HelpRec's DRV ID field (if there was one). The second DRV comes from the HelpRec's file. The third DRV comes from the application file's resource fork and the fourth from the System resource file. The resource id for the last three DRV's is derived from the hdisplayID parameter in the HMGetNewDisplay call.

When a DRV is called it may decline to service the call by calling HMPassControlCall. This will cause the Help Manager to call the next DRV in the chain. A DRV may change its parameters in anyway it chooses, and the changes are passed on to the next DRV in the chain.

The DRV chain allows an DRV to perform only the functions it knows about. It can pass "standard" behavior messages to other DRV's and so does not need to re-implement functions that are adequately handled by existing system or application DRV's.

The first two DRV's in the chain allow a developer to change their application's help behavior without changing the application. This allows new help system capabilities to be added to applications in a fairly seamless way.

### 6.4 Standard System DRV's

There are five standard drvID's available to all applications:

- 0 The standard HowDoI HelpDisplay which floats.
- 1 The standard DidYouKnow HelpDisplay which floats.

- 8 A non-floating standard HowDoI HelpDisplay.
- 9 A non-floating standard DidYouKnow HelpDisplay.
16. A standard bubble HelpDisplay.

The HowDoI and DidYouKnow DRVR's implement their respective standard behavior and appearance as defined in the "Help Manager Human Interface ERS".

#### **6.4.1 Standard System HowDoI DRVR**

If the DRVR is given a page 1 HelpRec of RecType INDX it will construct a paging list of the record's help items. If the INDX record points off to other INDX blocks they will be used to construct other paging lists in Page 2. End user help information is displayed in Pages 3 and 4.

A page 1 HelpRec of RecType LIST will create a single scrolling list containing the display names of items pointed to by the LIST record.

A page 1 HelpRec of end user help information will just cause that record to be displayed in the normal way.

#### **6.4.2 Standard System DidYouKnow DRVR**

The initial HelpRec used to define the HelpDisplay should be of type INDX or LIST. If it is an INDX record then the record's HelpKeys must all point to end user help information. A LIST type HelpRec's keys should all point off to either LIST or help information HelpRecs.

#### **6.4.3 Standard System Bubble DRVR**

This implements the standard bubble appearance and behavior. The DRVR must be passed only Help Information HelpRecs.

## 7.0 Using the Help Manager

This section discusses how an application uses the Help Manager to support the Information Menu.

### 7.1 Application Support of User Help Access

An application which has the Help Manager service the Information Menu only needs call the following Help Manager routines to fully support on-line help:

- HMInit - Initialize the Help Manager.
- HMOpenFile - Open a help file.
- HMGetBubbleMode - Get the current Bubble Mode state.
- HMGetNewBubble - Display a Bubble window using a HelpKey.

If the application needs to handle the Information Menu itself, then it may need to call the additional following routines:

- HMSetBubbleMode - Set the current Bubble Mode state.
- HMGetNewDisplay - Create a HelpDisplay using a HelpKey.
- HMCloseFile - Close a help file.

Three other procedures are required for applications which need to dynamically create bubble messages (such as the Finder):

- HMStuffNewRec - Create a new HelpRec.
- HMNewBubble - Create a bubble window from a HelpRec.
- HMDisposeRec - Dispose of a HelpRec.

The use of all of the above routines is described below.

### 7.2 Initialization

An application must first call HMInit to use any facility of the Help Manager. It can pass the menu ID of the Information Menu to HMInit and the Help

Manager will then service all items on the menu (see the description of HMIInit for details). This allows applications which support the standard Information Menu items to off-load responsibility for the menu to the Help Manager. This is in keeping with the least-path-of-resistance philosophy of Toolbox Manager design. Applications which pass the Information Menu ID to HMIInit need only do two other things to fully support on-line help: Open the appropriate help file(s), and provide Bubble window support for application windows (see the descriptions below).

### **7.3 Help Data Base Support**

An application developer must support on-line help by creating the necessary Help files. Help information can be stored in separate help files or in the application and/or System file resource forks. Only standard system information should be stored in the System file. Help information in either of the resource forks is always available while the application is running. However, the application must open other help files by calling HMOpenFile on them before a user can get the help information. HMCloseFile can be called when the information is no longer needed.

### **7.4 Bubble Mode Support**

The application can enable or disable Bubble Mode by calling HMSetBubbleMode. It can find out the current Mode by calling HMGetBubbleMode. If an application is handling the Information Menu itself (rather than letting the Help Manager handle it), it should enable Bubble Mode when the user selects the "What Is\_?" menu item. The Help Manager will disable Bubble Mode when the user clicks in a bubble, so the application need do nothing further to disable Bubble Mode.

An application which supports Bubble Mode should have the bubble help information stored in the Help Database for all menu commands and all screen graphics which a user can click upon. The bubble information for each menu item must be stored in a HelpRec whose HelpKey is the same as the menu item string. The system standard MDEF will use the item string as a HelpKey to obtain the required help information. The HelpKeys for help information associated with screen graphic items can be anything the application developer finds convenient. Users need never see this information themselves.

When Bubble Mode is enabled, it is up to the application to display Bubbles when the user clicks in the content area of application windows. The application must do its standard mouse hit-testing to determine what screen item was clicked. If the user clicked a screen graphic which has help information, the application should call HMGetNewBubble. It must pass

HMGetNewBubble a hotspot (which will generally be the center of the item clicked upon), and a HelpKey which should be a reference to a HelpRec with the appropriate help information for the screen graphic.

Alternatively, the application may want to dynamically create the help string to be displayed in the bubble. In this case, the application needs to call HMStuffNewRec to place the information into a HelpRec. It then calls HMNewBubble to display the new bubble, and can immediately call HMDisposeRec to dispose of the HelpRec created by the HMStuffNewRec call.

## 7.5 How Do I Support

When the user selects the "How Do I\_\_?" item from the Information Menu, the application should call HMGetNewDisplay to create a floating HowDoI display. (This menu support is unnecessary if HMInit was passed a menu ID.) It should pass a HelpKey to a HelpRec which has a rectype of INDX. The HelpKeys in the INDX record must all be to other HelpRecs, which which can themselves be INDX reotypes. (These secondary INDX reotypes will be used to generate the sub-topics scrolling list.) The HelpDisplay will handle all necessary user events, including disposing of itself.

## 7.6 Did You Know Support

Support for DidYouKnow is the same as for HowDoI, except a DidYouKnow type of HelpDisplay is created in response to the user selecting the "Did You Know\_\_?" menu command.

## 7.7 Other Help Manager Procedures

There are many other Help Manager procedures, but these are intended strictly for use by HM DRVR's and by the Help Manager itself. An application which just uses the system standard help types (WhatIs, HowDoI and DidYouKnow), need not concern itself with them.

## 8.0 Help Manager Procedures

Procedure calls which are enclosed in a box in the following section are to be implemented. Procedure calls which are not enclosed in a box have no commitment for implementation.

## 8.1 Help Manager Initialization

### **HMInit - Initialize the Help Manager**

**PROCEDURE HMInit(stuff: LONGINT; menuID: INTEGER; aboutProc:  
ProcPtr);**

This procedure must be called before any other Help Manager routines. It initializes internal Help Manager data structures. The stuff parameter should always be zero (it is intended for future compatibility). The menuID parameter is the menu ID of the Information menu. The Help Manager will service user selections off of this menu by appropriately calling Help Manager routines. If the menuID parameter is zero, then the application itself must respond to all user selections from the Information Menu. The aboutProc parameter is the address of a parameterless procedure which will be called in response to the "About..." menu item (when menuID is non-zero).

The Help Manager performs Information Menu handling as described below:

- About...                      Calls the aboutProc procedure.
- What Is\_\_?                    Toggles the current state of Bubble Mode.
- How Do I\_\_?                   Creates a floating HowDoI HelpDisplay using a HelpKey of "How Do I\_\_?".
- Did You Know\_\_?              Creates a non-floating DidYouKnow HelpDisplay using a HelpKey of "Did You Know\_\_?".
- other menu items              Creates a non-floating HowDoI HelpDisplay using the menu item string as the Display's HelpKey.

Note that an application cannot enter Bubble Mode if it does not call HMInit, even when the HELP key is pressed.

## **8.2 Bubble Window Support**

The procedures in this section support bubble window creation and display.

### **HMSetBubbleMode - Enable/Disable Bubble Mode**

**PROCEDURE HMSetBubbleMode(mode:Boolean);**

This procedure enables Bubble Mode if the mode parameter is TRUE and disables it otherwise. When Bubble Mode is enabled the application should appropriately display bubble windows in response to user clicks. The standard menu MDEF procedure will display bubble help as the user drags through menus. **NO NORMAL APPLICATION FUNCTIONS SHOULD BE EFFECTIVE DURING BUBBLE MODE** except to disable bubble mode.

While Bubble Mode is enabled, the Help Manager ensures that there is always a bubble window displayed which will inform the user as to how to quit bubble mode. When Bubble Mode is turned off, bubble windows are disposed which were displayed as a result of Bubble Mode being enabled.

An application cannot enter Bubble Mode if it never called HMInit. Also, it should disable Bubble Mode when it receives a MultiFinder SUSPEND event (but NOT re-enter Bubble Mode on the succeeding RESUME event).

### **HMGetBubbleMode - Return the current Bubble Mode setting**

**FUNCTION HMGetBubbleMode: Boolean;**

This function returns TRUE if Bubble Mode is enabled, and FALSE otherwise.

## HMGetNewBubble - Create a new bubble HelpDisplay

**FUNCTION HMGetNewBubble(key: HelpKey; hotspot1, hotspot2: Point; evr: EventRecord): OSErr;**

This function reads a HelpRec and uses its contents to display a bubble window. It creates a HelpDisplay that defines the bubble and makes the Display the current one.

The hotspot parameters define the bubble tip location in global coordinates. The hotspot1 parameter is the preferred location. When hotspot2 is different from hotspot1, normal bubble placement and orientation are modified. Normally, the bubble tip will be placed at hotspot1 and the bubble will be biased away from hotspot2. However, if there is not enough room to fit the bubble between hotspot1 and the screen edge, hotspot2 will be used and the bubble's direction will be biased away from hotspot1. Most calls to HMBubGetNew will just pass the same value for hotspot1 and hotspot2.

The use of the two hotspots is necessary when the bubble must appear to the side of the item the bubble points at, rather than directly over it (such as for menu items). In this case hotspot1 and hotspot2 define the corner points of a bounding box which the bubble is to be placed entirely outside of.

The evr parameter should be the EventRecord that is associated with the bubble. If there is none, evr can be NIL. Making the evr parameter odd signifies that the HelpKey parameter is a text string which is to be used as the actual help information. This allows applications to trivially construct and display bubble messages.

## HMNewBubble - Create a new bubble without using a help file

**FUNCTION HMNewBubble(hrec: HelpRecHandle; hotspot1, hotspot2: Point; evr: EventRecord) : OSErr;**

This function is the same as HMGetNewBubble, but accepts a HelpRecHandle in place of a HelpKey. The contents of the hrec parameter can be created dynamically.



## 8.3 Help Displays

### HMGetNewDisplay - Create a new HelpDisplay

**FUNCTION HMGetNewDisplay(key: HelpKey; hdisplayID: INTEGER; visible: Boolean; evr: EventRecord): OSErr;**

This function creates a HelpDisplay and makes it the Help Manager's current display.

The key parameter is used to load a HelpRec which defines the HelpDisplay's contents. This HelpRec is set as the level zero HelpRec. If a HelpDisplay already exists with the same HelpKey as the key parameter, the existing Display will be made frontmost.

The hdisplayID parameter is analogous to a the window manager's procID parameter in a NewWindow call: A resource ID is computed from it. The resource ID is to a resource of type DRVR which will control the appearance and behavior of the HelpDisplay. The resource ID calculation is given below:

$$16 * \text{DRVR\_resourc\_ID} + \text{variation\_code}$$

Note that the above expression has a form identical to the NewWindow's window procID (the variation code is the low four bits of the proc ID).

If the visible parameter is TRUE then the HelpDisplay will be initially visible. Otherwise it must be made visible by doing a ShowWindow call on its window. This allows an DRVR or application to set-up a window in anyway it desires.

The evr parameter is the event record associated with the event that caused the HelpDisplay to be created. It is ignored if it is NIL.

The DRVR uses the RecType of the HelpRec to decide what to display in the HelpDisplay as defined in the Help Database section of this document.

*[How can a developer use an DRVR built into his own app without having to at least create a dummy DRVR resource?]*

### **HMNewDisplay - Create a HelpDisplay without using a help file**

**FUNCTION HMNewDisplay(hrec: HelpRecHandle; hdisplayID: INTEGER;  
visible: Boolean; evr: EventRecord): OSErr;**

This function performs exactly the same function as HMGetNewDisplay, but accepts a HelpRecHandle in place of a HelpKey. This allows applications to dynamically construct a HelpDisplay.

### **HMDisposeDisplay - Dispose a HelpDisplay**

**PROCEDURE HMDisposeDisplay(hd: HelpDisplayHandle);**

Disposes of a HelpDisplay and all of its associated data structures. Any HelpRecs associated with the Display will also be disposed.

### **HMGetCurrentDisplay - Return current HelpDisplay**

**FUNCTION HMGetCurrentDisplay : HelpDisplayHandle;**

Retrieves the current HelpDisplay.

### **HMSetCurrentDisplay - Set the current HelpDisplay**

**FUNCTION HMSetCurrentDisplay(hd: HelpDisplayHandle);**

Sets the current HelpDisplay.

### **HMSetSelection - Set the Current HelpDisplay's Display Context**

**Function HMSetSelection(pageNum: INTEGER; hrec: HelpRecHandle): OSErr;**

Sets the current HelpDisplay's displayed information as if the user had selected the given help information themselves. The pageNum parameter is a

hmInPage value. The hrec parameter specifies a HelpRec to use. A null value removes the current hrec. To completely set a HowDoI HelpDisplay's display context to end user help information, it is necessary to first call HMSetSelection to set page one's HelpRec, then call it to set page two's HelpRec, and then set page three and four. Setting the HelpRec's in any other order than lowest page to highest page will prevent the selected HelpRec's from actually being drawn into the HelpDisplay.

### **HMGetSelection - Get the Current HelpDisplay's Display Context**

**Function HMGetSelection(pageNum: INTEGER): INTEGER;**

This procedure returns the item number in the given page which the user last selected. If there is no such item then it returns negative one. To find the complete sequence of HelpKeys that identify the path the user took to display a given piece of help information, it is necessary to call HMGetSelection for page one, and then page two. HMGetSelection returns the item numbers that the user selected from each scrolling list. HMGetHelpRec could then be called to get the HelpRec for each page. The page's return value from the HMGetSelection call could then be used (along with the page's HelpRec obtained from HMGetHelpRec) to call HMGetRecEntry to retrieve the HelpRec's data for the given item number (this would usually be the HelpKey associated with the item).

## **8.4 Information Routines**

### **HMDrawIcon - Draw the international help icon**

**PROCEDURE HMDrawIcon(h,w: INTEGER);**

This procedure draws the international help icon into the current grafport and at the current pen position. The icon is positioned to the right and below the pen. The hw parameter specifies the height and width to which the icon is to be scaled. A (0,0) h & w values will scale to the icon's default size (suitable for display in the menubar).

## HMPParamText - Help Manager's ParamText procedure

PROCEDURE HMPParamText(string0, string1, string2, string3: Str255);

Just like the Dialog Manager, the Help Manager substitutes the above four string values for the constant strings "^0", "^1", "^2", "^3" in help text it displays. The strings are only substituted into TEXT and styl type of help information. For styl records a TEInsert is performed to insert the text over the ^X string; this way the inserted characters will adopt the styling of the surrounding text.

## HMGetHelpRec - Get a HelpRec from the current HelpDisplay

FUNCTION HMGetHelpRec(level: INTEGER): HelpRecHandle;

This function returns the HelpRec associated with a given page of the current HelpDisplay.

## 8.5 Help Data Base Access

The procedures in this section define the Help Manager's database capability.

*[They also probably define what the hdbp resource has to look like...]*

## HMReadRecord- Read a HelpRec

FUNCTION HMReadRecord(key: HelpKey; VAR helprec: HelpRecHandle;  
offset, nbytes: LONGINT): OSErr;

Retrieves a full or partial HelpRec from the Help Database. The key parameter specifies the HelpKey to use. The helprec parameter is returned as a handle to the specified data. The offset parameter is a byte offset from the beginning to the HelpRec. nbytes is a byte count of the number of bytes of data to read from the record. If nbytes is zero the entire HelpRec will be read into memory.

### **HMStuffNewRec - Create a new HelpRec in memory**

**FUNCTION HMStuffNewRec(VAR helprec: HelpRecHandle; recType:  
ResType; varData: Ptr; nbytes: INTEGER): OSErr;**

This function is a convenience function to create a simple HelpRec with no hdisplayID or DisplayName values. It simply creates a HelpRec of type recType and stuffs the varData into the HelpRec's RecInfo field.

### **HMExtractRec - Obtain information from a HelpRec in memory**

**PROCEDURE HMExtractRec(helprec: HelpRecHandle; VAR recType:  
ResType; VAR hdisplayID: INTEGER; VAR dispName: Str31;  
VAR varData: Ptr; VAR nbytes: INTEGER);**

### **HMDisposeRec - Dispose a HelpRec which is in memory**

**PROCEDURE HMDisposeRec(helprec: HelpRecHandle);**

This releases a HelpRec which was previously obtained by a call to HMRecordRead or HMRecNew. This procedure must be called to release HelpRecs.

### **HMGetRecEntry - Extract a HelpEntry field from a HelpRec in memory**

**PROCEDURE HMGetRecEntry(helprec: HelpRecHandle; itemNum:  
INTEGER; VAR data: Ptr; VAR nbytes: LONGINT);**

This procedure extracts data from the RecInfo field of a HelpRec. It is used primarily for extracting a HelpKey from an INDX or LIST record. It will never move memory.

### **HMSetRecEntry - Insert a HelpEntry into a HelpRec in memory**

**FUNCTION HMSetRecEntry(helprec: HelpRecHandle; itemNum: INTEGER;  
data: Ptr; nbytes: LONGINT): OSerr;**

This is the opposite of HMGetRecEntry and puts an item into the RecInfo of a HelpRec. This routine may move memory.

## **8.6 Help Files**

### **HMOpenFile - Add a file to the current help chain**

**FUNCTION HMOpenFile(vRefNum: INTEGER; dirID: LONGINT; fileName:  
Str255; VAR refNum: INTEGER): OSerr;**

This function adds a file to the head of the current Help File Chain. If fileName is nil then the refNum parameter is taken as a refnum to an already open file.

### **HMCloseFile - Remove a file from the current help chain**

**FUNCTION HMCloseFile(refNum: INTEGER): OSerr;**

Closing a file does NOT remove it from any help chains. This procedure must be called to cause a file to be removed from the current Help Chain (allChains is FALSE), or all Chains (allChains is TRUE). This routine does not close the file - it only removes it from a Chain for the sake of searching for HelpRecs.

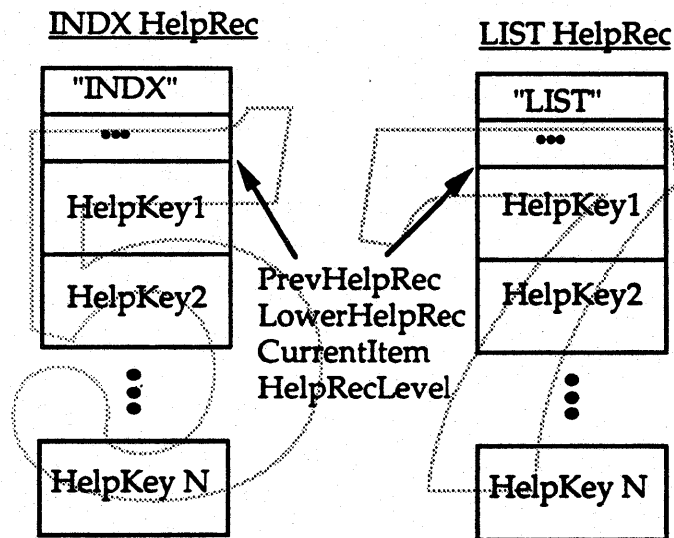
If the refNum parameter is -1 it means to remove the current help file. If it is 0 it means to remove the System file refnum, if it exists in the current Chain.

# Appendix - HelpRec Formats

This appendix lists the HelpRec's that the Help Manager supports. [GRP HelpRec's may not be supported.]

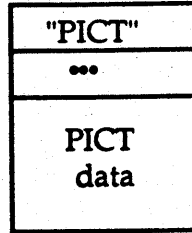
Detailed descriptions of each one are TBD. The fields labeled as "..." in the following diagrams stand for the standard HelpRec header fields DOffset, Reserv, DisplayName and DRVRIID, which have their already defined meanings.

## A.1 Aggregate HelpRecs



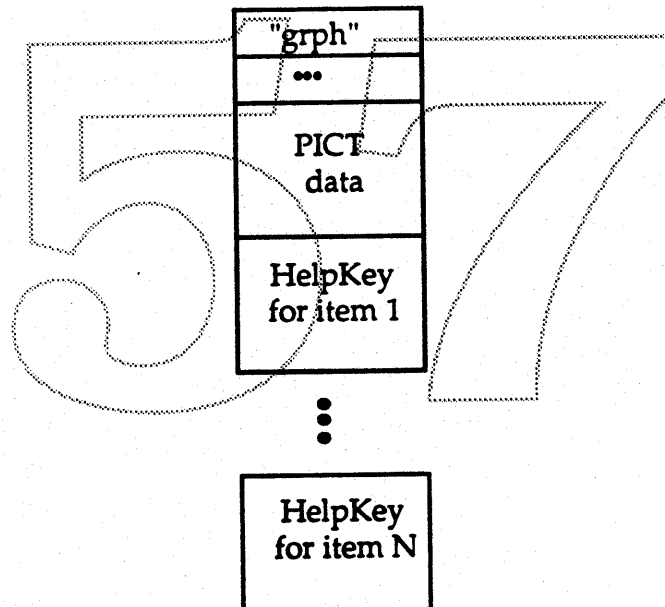
The above HelpRecs are the Aggregate record types and are defined in the Help Database Organization section of this document.

## A.2 Picture HelpRecs



The above record defines a PICT type of HelpRec. The PICT data field is a QuickDraw picture.

## A.3 grph HelpRec



The above diagram defines the grph HelpRec structure. The text items are HelpKeys to HelpRecs which will be used to display text information. The PICT data is a QuickDraw picture with imbedded picture comments. Each picture comment defines a point where the next HelpKey item is to be displayed. Note that these HelpKeys cannot be to a PICT type of HelpRec since QuickDraw does not allowed nested drawing of pictures.



# The Blue Interface Group

Scott Jenson  
x1576, MS 27-AJ, AppleLink: JENSON

Wednesday, March 15, 1989

TO: Lots of Folks  
CC: Even more...  
RE: New Chooser Design

## Summary:

This document describes the new chooser interface for Big Bang. It briefly notes the reasons motivating the new design, describes the current interface prototype, and then discusses some compatibility issues. One purpose of this document is to flush out any pending compatibility issues with this new design. Please send comments if I've overlooked something.

## Motivations:

### Better integration with the Desktop.

With NuFinder, more "chosen devices" will now exist on the desktop. AppleShare is an example that currently places devices on the desktop but Ginsu will also do it. As other device types might need to do this as well, we need to have a common mechanism.

### Better integration to a particular task.

As the Chooser is currently a DA, it's difficult for an application to ask the user to choose a device. They are effectively forced to say "Go choose a printer" when it would be better to not only bring it up for them but also filter the presentation to relevant device types.

### Limit impact on existing and soon to be released RDEVs.

Probably the most critical in terms of releasing in time for Big Bang: we can't change the Dhooser in a way that forces a significant rewrite of several device types.

## Approaches:

### Window Browser

The Window Browser as demonstrated by Jack Palevich and Annette Wagner lets the user browse the network from a Finder-level window. This integrates browsing much better into the Finder as any device can be brought to the desktop by dragging. It also expands the usefulness of NuFinder Views as they can apply to the network as well.

The biggest disadvantage of this model is that it's really a network browser, and doesn't

handle direct connect devices or network configuration(ala Network CDEV) well. After some work trying to tie the two together it appears that an additional "Connections Tool" would be more appropriate. Another problem with the window browser is that it forces users to the Finder for all choosing. This may seem like a feature and not a bug but many tasks, such as printing shouldn't force the user to close a dialog, wade through several layers of windows, and then choose a printer. A final but potentially minor point is that many RDEVs would need to be rewritten to properly fit into the direct manipulation environment of the Finder, properly respond to dragging, double clicking, "Get Info...", etc. Please don't quote me out of context; these things aren't wrong, of course, it's just the engineering cost and user pain of rev'ing several device drivers could be high.

### Enhance Chooser

This idea is to slightly modify the existing Chooser with a uniform interface mechanism to "save" favorite devices and also provide a modal calling structure like SFGGetFile. This approach is attractive primarily for simplicity and safety. Not only would it require minimal engineering changes to the Chooser but RDEVs could run without change. Equally important is the ability to let apps now bring up a chooser dialog like Standard File exactly when the user needs it. Finally, it's also possible to cleanly integrate the Networking CDEV so users don't need bop back and forth between the chooser and control panel (see details below).

The biggest disadvantage is that this design is too safe, leaving our network access technology effectively stagnant for yet another year. It's also not very well integrated into the desktop; there is a place to get things and a place to view what you've found. It feels cumbersome and could potentially be confusing to users.

### **Current Prototype:**

The current decision is to go with the chooser enhancement. It is not only more practical considering the constraints of Big Bang but it also solves the difficult problem of bringing the chooser directly to the user when needed. This isn't necessarily the best approach. Given more time, the Browser/Connections tool combination could be much more effective, easier to use, and provide a better model for the user. Development work should continue on these projects for a potential future release.

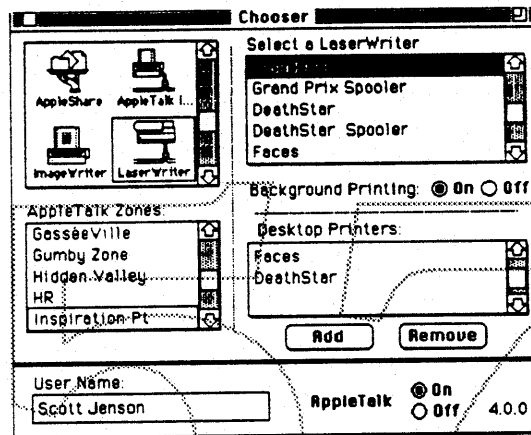
The basic Chooser will still be a DA and look similar to the existing one. The immediate differences are:

- 1) The user name has moved.
- 2) The Network CDEV is now replaced with a popup of Network drivers if they exist. If the system only has one driver then no popup appears.
- 3) The lower right corner now has an area to add common devices.

Printing behaves the same way with this new chooser but by clicking on the "Add" button, the selected printer will now appear on the desktop. Unlike volumes, these

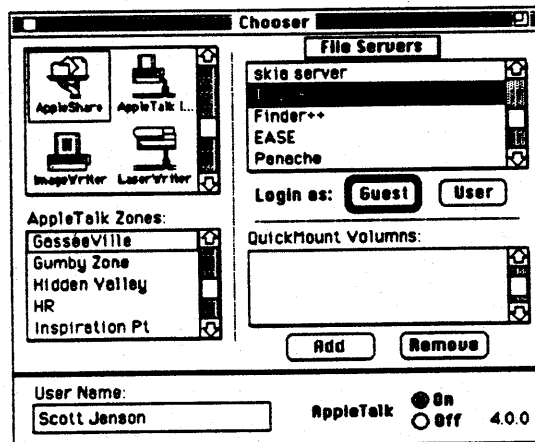
printers can be moved any where in the file system, or users can create SOFAs to them if they wish. Dragging documents to these printers will be the equivalent of "Print One", all print settings will now be saved with a document so no dialog will appear. The term "Desktop Printer" might be misleading and may change.

In addition, at PrintJob time, all Desktop Printers will appear in a popup menu letting users quickly switch between common printers without requiring a trip to the chooser. This is a non-trivial addition; once users find their favorite printers, they could conceivably **never** need to use the chooser again.



Chooser for selecting a LaserWriter

The AppleShare interface changes slightly to incorporate this new common device area (see figure below). File servers list now behaves like StdFile, double clicking on a server defaults to Guest access (This can change...) so browsing servers can be a much easier, quicker task. Logging on as User would ask only for a password. "Add"ing a volume to the common area could then prompt for the necessary stuff. Don't get too excited about the "QuickMount Volumes:" label, it's just a place holder for now. It's not clear yet if AppleShare wants to go the QuickMount route in preparation for Black Book or the boot mount route we have today.



Chooser for selecting a File Server

### Details:

Users should be able to select zone and then device in browsing the network, they currently must start with the device first.

It would be very convenient to have "Alpha-Select" on all fields, i.e. TAB goes from field to field, and typing selects a particular item. This is very important for consistency with NuFinder and StdFile. A particular example would be the File Servers list for AppleShare; it should use the ⌘ - up arrow and return as StdFile does. Non-active lists would remember their selections with an outline selection. While only having one fully inverted selection at a time should significantly reduce the "Christmas tree" effect that confuses many users in the current Chooser, this could be potentially confusing.

A rather sticky issue is flow of control between Chooser and RDEVs. For consistency and easy of implementation, it would be nice if the Chooser could handle all List Manager access. This would keep all user interaction between the various lists consistent. This is a tricky issue of backward compatibility and ease of implementation and is still being looked at.

This Chooser should replace the existing one for both Big Bang AND 6.0.X. A side effect of this is the Network CDEV now "goes away" from the user's point of view but further development of the internals will continue in N&C. This is a coordination issue that needs to be resolved between system software and N&C.

### **Compatibility:**

This design *should* be completely compatible with all RDEVs. AppleShare and Ginsu are changing as they want a common interface to bring something to the desktop. With the old chooser, there is a rect available for any "interface specific" RDEV needs. This area is now larger but should be a minimal compatibility risk.

One purpose of this document is to flush out any pending compatibility issues with this new design. Please send comments if I've overlooked something.