



A Technical History of Apple's Operating Systems

Amit Singh

A Supplementary Document for Chapter 1 from
the book *Mac OS X Internals: A Systems Approach*.

Copyright © Amit Singh. All Rights Reserved. Portions Copyright © Pearson Education.

Document Version 2.0.1 (July 28, 2006)

Latest version available at <http://osxbook.com/book/bonus/chapter1/>

This page intentionally left blank.

Dear Reader

This is a supplementary document for Chapter 1 from my book [Mac OS X Internals: A Systems Approach](#). A subset (about 30%) of this document appears in the book as the first chapter with the title **Origins of Mac OS X**.

This document is titled **A Technical History of Apple's Operating Systems**. Whereas the book's abridged version covers the history of Mac OS X, this document's coverage is considerably broader. It encompasses not only Mac OS X and its relevant ancestors, but the various operating systems that Apple has dabbled with since the company's inception, and many other systems that were direct or indirect sources of inspiration.

This was the first chapter I wrote for the book. It was particularly difficult to write in terms of the time and other resources required in researching the material. I often had to find and scavenge ancient documentation, software, and hardware (all actual screenshots shown in this document were captured first hand.) However, I couldn't include the chapter in its original form in the book. The book grew in size beyond everybody's expectations—1680 pages! Therefore, it was hard to justify the inclusion of this much history, even if it is interesting history.

I strongly believe that reading this document will help in understanding the evolution of Mac OS X, and, to some extent, of some aspects of modern-day operating systems. An important reality of technology, and of computing in particular, is that many things that we think of as “new ideas” are not quite new. As Alan Kay once said, “*Most ideas come from previous ideas.*” This is also true in the case of Mac OS X. As one digs the past of Mac OS X and its predecessors, one begins to understand how ancient things influenced not-so-ancient things, and the fact that we owe what we have today to many more people than are usually credited.

Please note again that this document is *not* the same as the book's first chapter. In particular, this document was not copyedited, composited, or proofread by the publisher. I prepared this PDF from my “raw” manuscript. Therefore, this document is not an example of the book's final typesetting or other production aspects.

I hope you enjoy reading this document and the book. Just as this document provides a super-detailed history of Apple's operating systems, the book itself is super-detailed on the internals of modern day Mac OS X. It is not at all a book about using Mac OS X—it is about the system's design and implementation. Therefore, I expect it to appeal to all operating system enthusiasts and students.

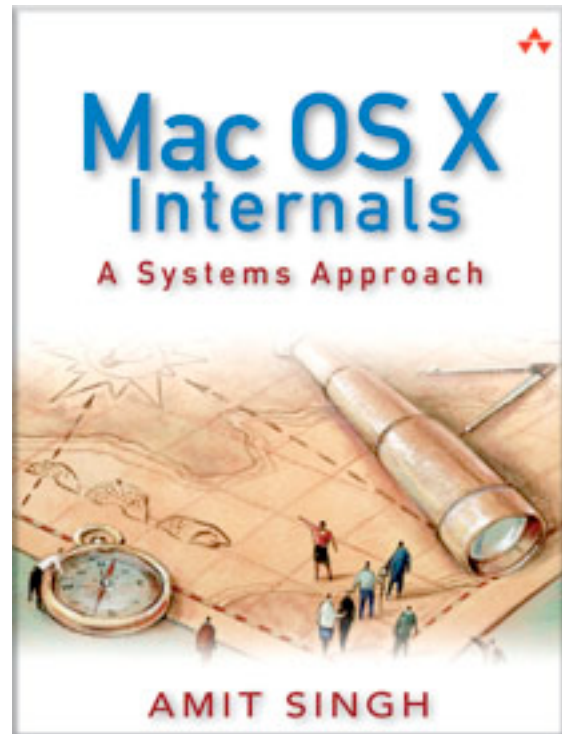
Amit Singh
Sunnyvale, California
<http://osxbook.com/contact/>

This page intentionally left blank.

About *Mac OS X Internals*

Mac OS X was released in March 2001, but many components, such as Mach and BSD, are considerably older. Understanding the design, implementation, and workings of Mac OS X requires examination of several technologies that differ in their age, origins, philosophies, and roles.

Mac OS X Internals: A Systems Approach ([Amazon page](#)) is the first book that dissects the internals of the system, presenting a detailed picture that grows incrementally as you read. For example, you will learn the roles of the **firmware**, the **bootloader**, the **Mach** and **BSD** kernel components (including the **process**, **virtual memory**, **IPC**, and **file system** layers), the object-oriented **I/O Kit** driver framework, **user libraries**, and other core pieces of software. You will learn how these pieces connect and work internally, where they originated, and how they evolved. The book also covers several key areas of the Intel-based Macintosh computers.



Over 1600 pages of the *what, how, and why* of Mac OS X!

A solid understanding of system internals is immensely useful in design, development, and debugging for programmers of various skill levels. System programmers can use the book as a reference and to construct a better picture of how the core system works. Application programmers can gain a deeper understanding of how their applications interact with the system. System administrators and power users can use the book to harness the power of the rich environment offered by Mac OS X. Finally, members of the Windows, Linux, BSD, and other Unix communities will find the book valuable in comparing and contrasting Mac OS X with their respective systems.

Please visit the book's web site (osxbook.com) for more information on the book, including a detailed [Table of Contents](#) and [links to reviews](#).

This page intentionally left blank.

A Technical History of Apple's Operating Systems

1.1. First Bytes into an Apple	6
1.1.1. Apple I.....	6
1.1.2. Apple II.....	10
1.1.2.1. Apple DOS.....	11
1.1.2.2. Apple Pascal.....	12
1.1.2.3. Apple CP/M.....	13
1.1.3. Apple III.....	14
1.1.3.1. Apple SOS.....	14
1.1.3.2. Apple ProDOS.....	16
1.2. Inspirations	17
1.2.1. Memex.....	18
1.2.2. Sketchpad.....	19
1.2.3. NLS: The oNLine System.....	21
1.2.3.1. The First Computer Mouse.....	21
1.2.3.2. A 5-Chord Key Set.....	22
1.2.3.3. Document Processing.....	22
1.2.3.4. Hypertext and Image Maps.....	22
1.2.3.5. Searching.....	23
1.2.3.6. Windows.....	23
1.2.3.7. Collaboration.....	23
1.2.3.8. Live, Interactive Collaboration.....	24
1.2.3.9. The Result.....	24
1.2.4. Smalltalk.....	25
1.2.5. Xerox Alto.....	30
1.2.5.1. Alto OS.....	33
1.2.5.2. Alto Executive.....	35
1.2.5.3. NetExec.....	36
1.2.5.4. Programming Facilities.....	37
1.2.5.5. Applications.....	38
1.2.5.6. Networking.....	38
1.2.5.7. Worms.....	40
1.2.6. Xerox STAR System.....	41
1.3. The Graphical Age at Apple	43
1.3.1. Lisa.....	44
1.3.1.1. Packaging.....	45
1.3.1.2. Processor and Memory.....	45
1.3.1.3. Display.....	46
1.3.1.4. Storage.....	46
1.3.1.5. Expansion.....	47

2 A Technical History of Apple's Operating Systems

1.3.1.6. Lisa OS.....	47
1.3.1.7. Lisa WorkShop.....	55
1.3.1.8. Lisa s Fate.....	56
1.3.2. The Macintosh.....	57
1.4. Many Systems for Many Apples.....	63
1.4.1. System Software Releases 2 - 6.....	63
1.4.2. What Color is Your System?.....	65
1.4.3. GS/OS.....	66
1.4.4. A/UX.....	68
1.5. Seeking Power.....	72
1.5.1. System 7.....	72
1.5.2. AIM for POWER.....	74
1.5.2.1. A RISCy Look Back.....	74
1.5.2.2. Apple Wants RISC.....	76
1.5.2.3. Apple Likes RISC: ARM.....	78
1.5.3. Mac OS for PowerPC.....	79
1.5.4. MAE.....	80
1.5.5. Apple Workgroup Server.....	84
1.5.6. NetWare for PowerPC.....	85
1.5.7. AIX for PowerPC.....	85
1.6. Quest for the Operating System.....	87
1.6.1. Star Trek.....	88
1.6.2. Raptor.....	89
1.6.3. NuKernel.....	89
1.6.4. TalOS.....	89
1.6.5. Copland.....	90
1.6.6. Gershwin.....	93
1.6.7. BeOS.....	93
1.6.8. Plan A.....	95
1.7. The NeXT Chapter.....	95
1.7.1. NEXTSTEP.....	96
1.7.2. OPENSTEP.....	101
1.8. The Mach Factor.....	102
1.8.1. Rochester's Intelligent Gateway.....	103
1.8.2. Accent.....	104
1.8.3. Mach.....	106
1.8.4. MkLinux.....	111
1.8.5. Musical Names.....	112
1.9. Strategies.....	113
1.9.1. Mac OS 8 and 9.....	115
1.9.2. Rhapsody.....	117
1.9.2.1. Blue Box.....	119

1.9.2.2. <i>Yellow Box</i>	120
1.10. Towards Mac OS X.....	121
1.10.1. Mac OS X Server 1.x.....	123
1.10.2. Mac OS X Developer Previews.....	123
1.10.2.1. <i>DP1</i>	123
1.10.2.2. <i>DP2</i>	124
1.10.2.3. <i>DP3</i>	124
1.10.2.4. <i>DP4</i>	125
1.10.3. Mac OS X Public Beta.....	125
1.10.4. Mac OS X 10.x.....	127
1.10.4.1. <i>Mac OS X 10.0</i>	128
1.10.4.2. <i>Mac OS X 10.1</i>	129
1.10.4.3. <i>Mac OS X 10.2</i>	130
1.10.4.4. <i>Mac OS X 10.3</i>	131
1.10.4.5. <i>Mac OS X 10.4</i>	131
1.11. Others.....	133
1.11.1. Mac OS on the Pippin.....	133
1.11.2. Newton OS.....	137
1.11.2.1. <i>Newton OS</i>	137
1.11.2.2. <i>System Services</i>	138
1.11.2.3. <i>Application Components</i>	138
1.11.3. The iPod's Operating System.....	139

4 A Technical History of Apple's Operating Systems

C H A P T E R 1

A Technical History of Apple's Operating Systems

“Most ideas come from previous ideas.”—Alan Curtis Kay

The Mac OS X operating system represents a rather successful coming together of paradigms, ideologies, and technologies that have often resisted each other in the past. A good example is the cordial relationship that exists between the command-line and graphical interfaces in Mac OS X. The system is a result of the trials and tribulations of Apple, NeXT, as well as their user and developer communities. Mac OS X exemplifies how a capable system can result from the direct or indirect efforts of corporations, academic and research communities, the Open Source and Free Software movements, and of course, individuals.

Apple has been around since 1976, and many accounts of its history have been told. If the story of Apple as a company is fascinating, so is the *technical* history of Apple's operating systems. This chapter discusses operating systems that Apple has created in the past, those that it attempted to create, and some that influenced or inspired Apple. In this discussion, we will come across several technologies whose confluence eventually led to Mac OS X.

1.1. FIRST BYTES INTO AN APPLE

As 1975 came to an end, Steve Wozniak had finished his prototype of a home-brew computer built using inexpensive components. Hewlett-Packard, Wozniak's employer at that time, was not interested in his creation. Wozniak requested, and was soon granted, a release of the technology. On April first, 1976, Steve Jobs, Steve Wozniak, and an Atari engineer named Ronald Wayne founded Apple. The company's first product was Wozniak's computer: the Apple I.

1.1.1. Apple I

The Apple I was based on an 8-bit processor, the 6502, which was made by MOS Technology, Inc. of Norristown, Pennsylvania. The 6502 was designed by ex-Motorola engineers. It was similar to the more expensive 6800 from Motorola. Another alternative, the Intel 8080, was also more expensive than the 6502. The 6502 was chosen primarily because it was cheap—it could be had for \$25, whereas the 6800 and the 8080 were well over \$100 apiece.

MOS Technology advertised in 1975 that it would sell the 6502 at a discounted price of \$20 at the Wescon electronics show in San Francisco. The 6502's low price and popularity would eventually cause Intel and Motorola to lower the prices of their processors. Motorola had also sued MOS Technology over the similarity of the 650x processor line to the 6800. The 6501 was consequently withdrawn from the market.

The 6502 came in a 40-pin package. The Apple I used a clock oscillator with a frequency of 1.023 MHz. Moreover, in the Apple I, 4 out of every 65 clock-cycles were dedicated to memory refresh. Therefore, the effective cycle frequency was 0.960 MHz. Other key features of the processor included the following.

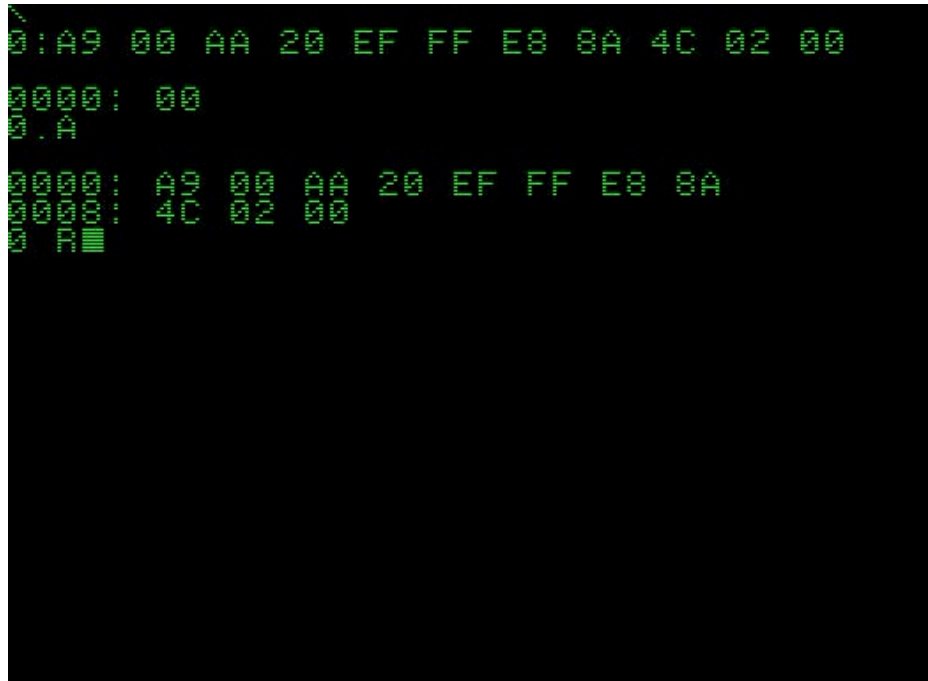
- An instruction set consisting of 56 documented instructions
- Interrupt capability with support for non-maskable interrupts
- A 16-bit address bus with an addressable memory range of up to 64KB
- Over a dozen addressing modes
- An 8-bit accumulator that was used for arithmetic and logical operations
- Two 8-bit index registers X and Y, which were typically used in indexed addressing modes
- A 16-bit program counter logically divided into “PCL” (low bits 0–7) and “PCH” (high bits 8–15) halves
- An 8-bit processor status register consisting of the following flags: Negative (N), Overflow (V), Break Command (B), Decimal mode (D), Interrupt Disable (I), Zero (Z), and Carry (C)
- An 8-bit stack pointer (the high byte of the logical stack pointer was hardwired to the value 1, limiting the stack to be 256 bytes in size, and to lie between addresses 0x100 and 0x1FF)

The Apple I used 16-pin dynamic RAM. It had sockets for up to 8KB of on-board memory. The total memory could be expanded to 64KB through a 44-pin edge connector. All refreshing of dynamic memory, including the off-board expansion memory, was performed automatically. The crystal oscillator was the source for the entire system timing.

The Apple I had a built-in video terminal. The output video signal was a composite signal consisting of sync and video information. It could be sent to any standard raster-scan-based video display monitor. In particular, the Apple I could be directly connected to a television with an RF modulator, yielding an automatic scrolling display with 24 lines per page, and a frame rate of 60.05 Hz. Each line had 40 characters, with the character matrix being 5×7 in size. Owing to the high cost of RAM, video terminals were designed with shift registers at that time. The Apple I’s display memory was comprised of seven 1K dynamic shift registers.

The Apple I also had a keyboard interface and a cassette board meant to work with regular cassette recorders. The “computer” was simply a motherboard (see

Figure 1–2 shows a test program entered at the monitor’s “backslash” prompt. The first line of hexadecimal numbers is the program itself. Its purpose is to print a continuous stream of ASCII characters on the display. Typing `0.A` prints a listing of the program, and typing `R` runs the program.



```
\
0:A9 00 AA 20 EF FF E8 8A 4C 02 00
0000: 00
0.A
0000: A9 00 AA 20 EF FF E8 8A
0000: 4C 02 00
0 R
```

FIGURE 1–2 The Apple I’s firmware-resident system monitor

Compared to the UNIX general-purpose time-sharing system, which was in its Sixth Edition then, the Apple I’s operating environment was decidedly puny. However, a contemporary computer system running UNIX would have cost many thousands of dollars. The Apple I was an attempt to make computing affordable for hobbyists, and as those behind it hoped, for the masses. Within the first nine months of the Apple I’s introduction, all but a few of the two hundred or so units manufactured had been sold.

1.1.2. Apple][

The Apple I had a life span of less than a year, but its successor would live much longer. Wozniak soon began work on the Apple][. Although based on the 6502 processor as well, the Apple][was introduced as an *integrated* computer: it came completely assembled in a beige plastic case. The contents of the original Apple][package included the following.

- An Apple][P.C. (“Printed Circuit”) Board complete with specified RAM memory
- A D.C. power connector with cable
- A 2” speaker with cable
- A Preliminary manual
- Two demonstration cassette tapes
- Two 16-pin headers plugged into locations A7² and J14 on the Apple][board

Latter models varied in their configurations as well as bundled accessories. For example, Apple][computers with 16KB or more memory came with game paddles and the STARTREK game tape.

The Apple][provided a machine-level monitor. Depending on the Apple][model, resetting the computer could either leave the user in the monitor, or could auto-start the ROM, dropping the user in BASIC. Calling a specific subroutine³ from BASIC allowed access to the monitor. The monitor provided commands for tasks such as the following.

- Examining, changing, moving, and verifying memory
- Performing cassette I/O

² The Apple][’s keyboard connector was a 16-pin IC socket at location A7 on the main circuit board, whereas a 16-pin game I/O connector was located at J14.

³ Typing CALL -151 at the BASIC prompt provided access to the monitor.

- Configuring the video mode
- Assembling, disassembling, running, and debugging machine code

Upon its release, the Apple II was the first personal computer to display color graphics. Apple provided special software to make use of the Apple II's capabilities, including its graphical prowess. For example, the *High-Resolution Operating Subroutines* package provided graphical subroutines usable from both BASIC and machine language. Using these subroutines, the programmer could initialize high-resolution mode, clear the screen, plot a point, draw a line, and draw or animate predefined shapes.

Various Apple II machines were made during the long life-span of the Apple II line: the Apple II+, IIe, IIC, IIe Enhanced, IIe Platinum, the 16-bit IIGS, and the IIC+ that was introduced in 1988 as the last Apple II model. Many of these models had multiple revisions themselves. As we will discuss next, several operating systems were created for the Apple II family.

1.1.2.1. *Apple DOS*

Shortly after the release of the Apple II in 1977, it was realized that a better storage solution than the existing one (which was based on cassette tape) was imperative for the computer. Wozniak created a brilliant design for a floppy disk drive—the Disk II—and thus there was need for a disk operating system (DOS). Apple's first version of a DOS was released as Apple DOS 3.1 in July 1978.

Apple DOS is unrelated to Microsoft's popular MS-DOS. During a time when it was a luxury to have disk drives, and for an operating system to support them, many such "disk operating systems" had the term DOS in their names.

The first release of Apple DOS had the version 3.1 (as opposed to, say, 1.0) because one of the implementers, Paul Laughton, incremented a revision counter of the format *x.y* every time the source code was recompiled. The counter started with

x being 0 and y being 1. Every time y reached 9, x was incremented by 1. Apple DOS was beta tested as version 3.0. Figure 1–3 shows a screenshot from Apple DOS 3.3.

```

]CATALOG
DISK VOLUME 254
*A 0003 HELLO
*I 0003 APPLESOFT
*B 0006 LOADERS.OBJ0
*B 0042 FPBASIC
*B 0042 INTBASIC
*A 0003 MASTER
*B 0003 MASTER.CREATE
*I 0003 COPY
*B 0003 COPY.OBJ0
*A 0003 COPY.A
*B 0003 CHAIN
*A 0014 RENUMBER
*A 0003 FILEM
*B 0003 FID
*A 0003 CONVERT13
*B 0003 MUFFIN
*A 0003 START13
*B 0003 BOOT13

```

FIGURE 1–3 Apple DOS 3.3

1.1.2.2. Apple Pascal

The *p-System* was a Pascal language and development system that was very popular in the 1970s and the early 1980s. It was created at the University of California, San Diego (UCSD). It was a portable operating system, essentially a stack-based virtual

machine running *p-Code*,⁴ with UCSD Pascal being the most popular programming language for it.

In 1978, most of Apple's software development was around the BASIC and assembly languages. Two Apple employees, Bill Atkinson and Jef Raskin, were instrumental in introducing the Pascal language at Apple. They also licensed the p-System from UCSD.

UCSD students Mark Allen and Richard Gleaves developed a p-Code interpreter for the 6502 in the summer of 1978. This interpreter became the basis for the Apple][Pascal⁵ that was released in 1979. Apple][Pascal included a compiler, an assembler, a filer, a modal editor, and various utility programs. The system was command-line-driven. Pascal code would compile into p-Code, which would then be interpreted by the 6502-native interpreter. It supported program modules called *units*, which could be segmented, and therefore, were typically memory-resident only when needed. Apple][Pascal's memory usage was limited to 64KB.

Apple Pascal lived as a product for five years, with the Pascal support in the Macintosh Programmer's Workshop (MPW) eventually replacing it.

1.1.2.3. Apple CP/M

Microsoft introduced a coprocessor card, a circuit board named *SoftCard*, in 1980. It was originally called the *Microsoft Z-80 SoftCard*, but Microsoft had to rename it to avoid a lawsuit from Zilog, the makers of the Z-80 processor. The SoftCard plugged into a slot and added a Z-80 processor, essentially converting an Apple][into two computers. An Apple][with a SoftCard could run Z-80 programs based on

⁴ p-Code was akin to what is now commonly known as bytecode.

⁵ Apple's Pascal system for the Apple][was derived from a specific implementation of the p-Code architecture: UCSD Pascal II.1.

the popular CP/M operating system,⁶ which had a rich software library of programs such as dBase and WordStar. Programming languages such as Microsoft ANSI Standard COBOL and FORTRAN could also be used on the Apple II.

Besides SoftCard, there existed other coprocessor cards for the Z-80 and for other processors such as the Motorola 6809. For example, the *Stellation Mill* 6809 card allowed the OS-9 real-time operating system⁷ to run on compatible Apple machines.

1.1.3. Apple III

The Apple III was introduced in 1980 as a computer for business users. Apple originally planned to ship the Apple III with a suite of office programs: a spreadsheet, a word processor, a database program, and a business-graphics program. However, the Apple III shipped late, and initially ran only the VisiCalc spreadsheet program. It had a new operating system called *SOS*.

1.1.3.1. Apple SOS

SOS⁸ officially stood for “Sophisticated Operating System,” although it was apparently an acronym for “Sara’s Operating System,” named after an engineer’s daughter.

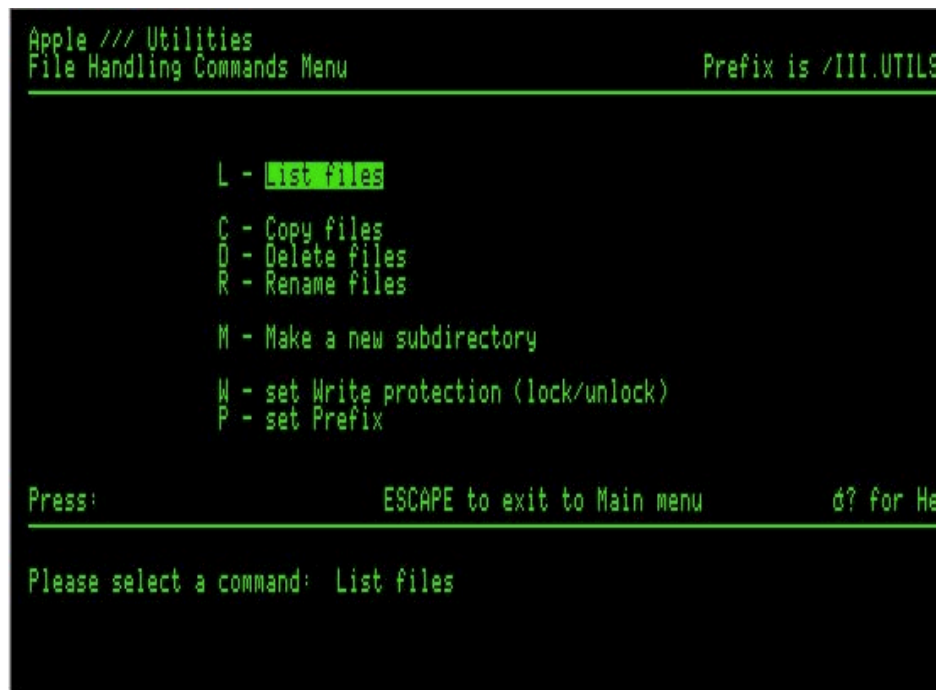
Every SOS program loaded the operating system into memory. A SOS application disk consisted of a kernel (`SOS.Kernel`); an interpreter (`SOS.Interp`), which could be the application itself, or something that the application used; and a set of drivers (`SOS.Driver`). Unlike the Apple II, which used ROM-based device drivers

⁶ CP/M was created by Gary Kildall, who founded a company called *Intergalactic Digital Research* to market the system.

⁷ Unrelated to Mac OS 9.

⁸ Pronounced “sauce” (which would make it “Apple Sauce.”)

(such as those residing on I/O card ROMs), SOS device drivers were RAM-based, and could be “installed,” which was a first for a commercial operating system at that time. Figure 1–4 shows a screenshot from SOS.



```
Apple /// Utilities
File Handling Commands Menu                               Prefix is /III.UTILS

L - List files
C - Copy files
D - Delete files
R - Rename files

M - Make a new subdirectory

W - set Write protection (lock/unlock)
P - set Prefix

Press:                               ESCAPE to exit to Main menu       d? for Help

Please select a command: List files
```

FIGURE 1–4 Apple SOS

The Apple /// also had a Pascal system that was derived from Apple II Pascal, but had language extensions, access to up to 256KB of memory, a floating-point implementation called Standard Apple Numerical Environment (SANE), and the ability to access SOS from Pascal programs using the `SOSIO` unit.

Steve Wozniak once called SOS “the finest operating system” for a micro-computer. He lamented, however, that SOS was closed-source. In general, Apple’s attitude with the Apple /// was that of starkly heightened secrecy as compared to earlier systems.

SOS evolved into Apple ProDOS.

1.1.3.2. Apple ProDOS

Apple ProDOS was first released as version 1.0 in October 1983. Based on SOS, it was Apple's replacement for Apple DOS 3.3. ProDOS provided better facilities for programming in BASIC, assembly language, and machine language; better interrupt handling; faster disk I/O with direct block access; and so on. Figure 1-5 shows a screenshot from ProDOS.

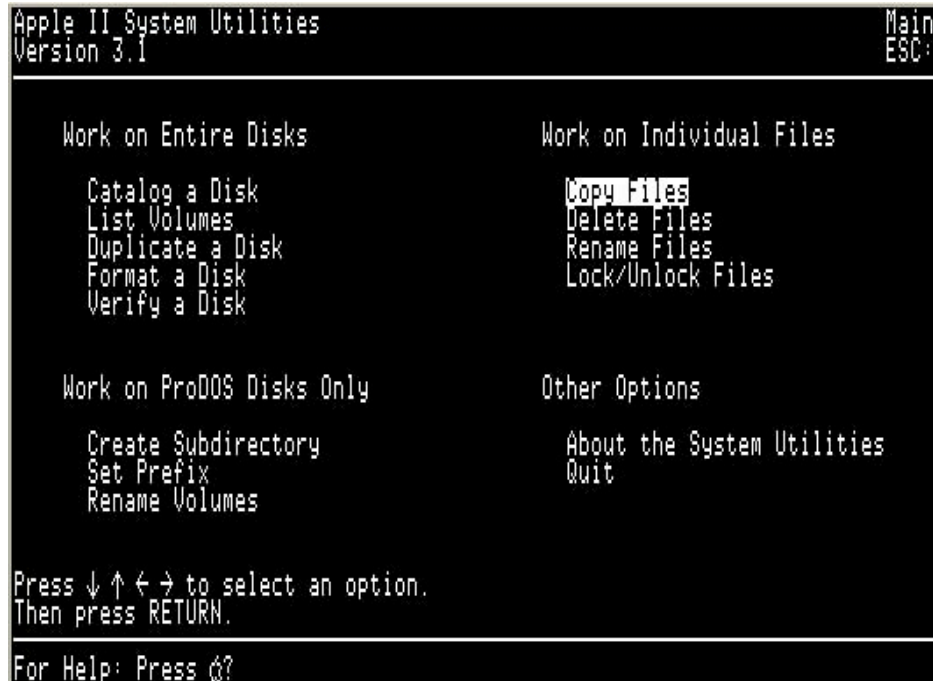


FIGURE 1-5 Apple ProDOS

ProDOS also had a relatively sophisticated hierarchical file system with features such as the following.

- Multiple logical volumes on one physical volume
- Support for up to 20 different file types, of which 10 could be user-defined
- Up to 8 open files at any given time
- An arbitrary number of files in a subdirectory, although the volume directory was limited to a maximum of 51 files

When the 16-bit Apple II was released, ProDOS was at version 1.1.1. It forked into ProDOS 8 and ProDOS 16 for 8-bit and 16-bit processors, respectively.

1.2. INSPIRATIONS

1984 is well known in the Apple world as the year the Macintosh was introduced. In 1983, Apple had released the Lisa computer, which represented a fundamental step forward in mainstream personal computing. Lisa and Macintosh were greatly inspired—directly or indirectly—by the work done at Xerox Palo Alto Research Center (PARC). Sources of such inspirations included the Smalltalk environment, the Xerox STAR system, and the Xerox Alto. Most of the ideas pioneered in these systems remain relevant today—in Mac OS X and in other modern systems.

To fully understand the lineage of the Macintosh, we must take a few detours and go further back in time, first to 1945, then to 1963, and finally to 1968—before the advent of UNIX, long before Apple or Microsoft were even founded, and in fact, decades⁹ before the first version of Microsoft Windows or the Macintosh were released.

⁹ The first version of Microsoft Windows was released on June 28, 1985. Microsoft had made the first Windows announcement in November 1983. Microsoft's selling point was that Windows provided a new software development and runtime environment that used bitmap displays and mice, thus freeing the user from the "MS-DOS method of typing commands at the C prompt (C:)."

1.2.1. Memex

Vannevar Bush published an article titled *As We May Think* in the July 1945 edition of *The Atlantic Monthly*.¹⁰ Bush was then the Director of the Office of Scientific Research and Development in the United States. His article described, among several incredible visionary insights and observations, a hypothetical device Bush had conceived many years earlier: the *memex*.

The memex was a mechanized private file and library—a supplement to human memory. It would store a vast amount of information and allow for rapid searching of its contents. Bush envisioned that a user could store books, letters, records, and any other arbitrary information on the memex, whose storage capacity would be large enough. The information could be textual or graphic. Other features of the memex included the following.

- It would be incorporated into an ordinary looking desk, thus fitting in the user's normal work environment. Its user interface would consist of a keyboard, buttons, levers, and translucent projection screens.
- Its primary storage medium would be microfilm.
- New material would be primarily available for purchase on microfilm. A variety of content, ranging from books to daily newspapers, could be trivially inserted into the memex.
- The memex would have photocopying abilities so that users could conveniently add their own content to the machine's data store.
- A lever would allow a user to flip forward and backward through pages of information, with the browsing speed dependent on how far the lever was moved. A shortcut would warp to the first page of the index.
- The user would be able to arbitrarily annotate preexisting information in the memex.
- The memex would not only index its content normally, but would allow the user to build "trails" of information, connecting one piece to another to build associations between related bodies of knowledge. Thus, Bush

¹⁰ "As We May Think," by Vannevar Bush (*Atlantic Monthly* 176:1, July 1945, pp. 101–108).

described the core idea behind the web: the hyperlink—*almost half a century before the first web browser was created.*

Bush made numerous uncanny predictions. He suggested that some day an entire encyclopedia would be available on a storage medium the size of a matchbox. He envisioned that people from all walks of life—chemists, historians, lawyers, patent attorneys, physicians, and so on—would use the memex to perform hitherto impossible or inordinately difficult tasks.

The modern-day Internet, the development and proliferation of high-density storage media, and the critical dependence of computer users on searching, are resounding testimonies to Bush’s foresight.

1.2.2. Sketchpad

In January 1963, Ivan Edward Sutherland, who was a graduate student at the Massachusetts Institute of Technology, submitted his Ph. D. thesis titled *Sketchpad, A Man-Machine Graphical Communication System*. Sutherland’s thesis advisor was Claude Elwood Shannon, the world-renowned mathematical engineer who is known as “the father of information theory.”

Sutherland had begun work on a drawing system for the TX-2 computer in the fall of 1961. The Sketchpad system was primarily designed and developed within the next year and a half.

TX-2

The TX-2 computer was built in 1956. It had a word length of 36 bits, but it supported breaking a 36-bit word into independent sub-words, even allowing sub-word lengths to differ. It had a large amount of memory for its time: a vacuum-tube-driven core of 64K words, and a faster, transistor-driven core of 4K words. It had a paper-tape reader and could also use magnetic tape as auxiliary storage.

The primary input device of Sketchpad was the *light pen*: a hand held photodiode device with the shape and dimensions of a pen. The light pen was connected to the computer by a coaxial cable, through which it communicated to the computer when its field-of-view encompassed a spot on the TX-2's display. The pen's barrel could be rotated to vary the distance between the photodiode and the lens at the pen's tip, thus adjusting the field-of-view's size.

As the user drew directly on the screen using the light pen, Sketchpad interpreted the drawing. Using straight-line segments and circle arcs as basic drawable shapes, the user could draw more complex shapes. The system treated on-screen shapes as *objects*¹¹ that could be operated upon. For example, transforms such as rotation, scaling, and translation could be applied to existing drawings. Once drawn, shapes could be saved, and reused as primitive units.¹² The light pen was used in conjunction with push-button controls. For example, the *draw* control created a new line segment or arc, with the drawn shape's end-point remaining attached to the pen. Other examples of controls included *circle center*, *move*, *delete*, and *instance*.

Sketchpad was demonstrated by creating electrical, mathematical, mechanical, scientific, and even artistic drawings. Moreover, arbitrary mathematical conditions, which could be the result of complex computations, could be applied to drawings. Sketchpad would automatically satisfy the conditions and alter the drawings to pictorially display the results. It had various other mathematical abilities that made it a powerful graphical input program—for example, it allowed sub-pictures within pictures, with no intrinsic limit on the nesting level.

Sketchpad drawings, or rather, their topologies, were stored in special files using data structures optimized for fast editing of the drawings.

Sutherland received the Turing award in 1988 for his pioneering contribu-

¹¹ Sketchpad could be seen as an object-oriented graphics editor, and a precursor to object-oriented programming.

¹² Thus, Sketchpad provided the first example of the Prototype design pattern.

tions to the field of computer graphics. His work on Sketchpad would prove inspirational in the development of Smalltalk, which in turn would be an inspiration for the advent of the graphical user interface at Apple.

1.2.3. NLS: The oNLine System

On December 9, 1968, an array of astounding technologies was demonstrated at the Convention Center in San Francisco during the Fall Joint Computer Conference (FJCC). Douglas Engelbart and his team of seventeen colleagues—working in the Augmentation Research Center at the Stanford Research Institute (SRI) in Menlo Park, California—presented NLS (oNLine System). NLS was an online¹³ system they had been working on since 1962. The “astounding” adjective is justified by the quantity and quality of innovation demonstrated on that single day.

Engelbart said¹⁴ at the beginning of his presentation, “*The research program that I am going to describe to you today is quickly characterizable by saying: if in your office, you as an intellectual worker were supplied with a computer display backed up by a computer that was alive for you all day and was instantly responsive, err, responsive... how much value could you derive from that? Well that basically characterizes what we’ve been pursuing for many years...*”

1.2.3.1. The First Computer Mouse

Engelbart demonstrated the first computer mouse, a three-button pointing device with a tracking spot, or “bug,” on the screen. The mouse’s underside had two wheels that could roll or slide on a flat surface. Each wheel controlled a potentiometer. As the user moved the mouse, the respective rolling and sliding motions of the

¹³ “Online” refers to the interactive nature of NLS. In the 1960s, computing was typically batch-oriented, which made an interactive system very appealing.

¹⁴ Quoted from a video recording of Engelbart’s demonstration.

two wheels resulted in voltage changes that were translated to relative coordinate changes on the screen.

1.2.3.2. A 5-Chord Key Set

Another input device Engelbart used in his demonstration was a chord key set—a five-finger equivalent of a full-sized keyboard. The key set could be used to input up to 31 different characters (2^5 minus the one state when no keys are pressed.)

1.2.3.3. Document Processing

Engelbart showed that text could be entered, dragged, copied, pasted, formatted, scrolled, and grouped hierarchically in nested levels. Multiple lines of text could be collapsed into a single line. The text so created could be saved in files, with provision for storing metadata such as the file's owner and time of creation. The use of a mouse made these operations easier. Engelbart referred to the overall mechanism as *view control*.

The system was useful while editing code as well—blocks of code could be expanded and collapsed, and even auto-completion was supported.

Furthermore, documents could contain embedded statements for *markup*, which allowed formatting of documents for specific purposes such as printing.

1.2.3.4. Hypertext and Image Maps

Using hypertext, or text with hyperlinks, Engelbart could jump from one location to another. Hyperlinks could be used to facilitate access to search results, or could be explicitly used as visible or invisible “live” links to information.

The system also had picture-drawing capabilities. Even pictures could have live hyperlinks, like latter-day *image maps* in web pages.

Origins of Hypertext

Theodor Holm Nelson came up with the term “hypertext,” whereas the concept itself is ascribed to Vannevar Bush, who described it in the context of the memex machine, as we saw earlier in this chapter. Nelson is noted for his *Xanadu* project, which was to be a worldwide electronic publishing system. He coined the term “hyper-text” in 1965 to refer to a flexible, generalized, non-linear presentation of related information.

1.2.3.5. Searching

NLS provided powerful search facilities: keywords could be *weighted*, and search results were ordered accordingly. Results could also be presented as hyperlinks.

1.2.3.6. Windows

The computer screen could be split into a *frozen display* and a *scanning window*. While you were reading a manual, for example, and you needed to look up a term, you could split the screen and view the term’s definition in the dynamically changing scanning window—similar to latter-day frames in web pages.

1.2.3.7. Collaboration

The system also kept track of who you were and what you were doing. People could work collaboratively on files, annotate each other’s text, and leave notes for each other—akin to modern-day document versioning systems. It was also possible to leave messages for one or more specific people. A special language, essentially a programmable filter, would allow a test to be associated with pieces of text. Thereafter, readers could only view what they were allowed to, as determined by the result of the context-sensitive test.

1.2.3.8. *Live, Interactive Collaboration*

The SRI team demonstrated *live* audio-video conferencing. The communicating parties could even have collaborative screen sharing, with each party having independent capabilities. For example, two people could look at the same display, but one of them would have read-only privileges, whereas the other would be able to modify the display.

1.2.3.9. *The Result*

Engelbart stated that NLS was a vehicle to allow humans to *operate* (compose, study, and modify) within the domain of complex information structures, where *content* represents *concepts*. NLS was meant to be a tool to navigate complex structures: something linear text could not do well.

Perhaps no discussion of individual components of NLS can convey how awe-inspiring the overall system that resulted from the integration of these parts was. NLS is best understood and appreciated by watching a recording¹⁵ of the NLS demonstration.

Engelbart was also involved in the creation of the ARPANET,¹⁶ the precursor to the Internet. His team planned to create a special ARPANET service that would provide relevant network information. For example, the service would answer questions such as the following.

Who is providing what services?

What protocol do I use to get there?

Who is “down” (in the networking sense) today and who is “up”?

¹⁵ Recordings of the NLS demonstration are available online at a Stanford University web site (<http://sloan.stanford.edu/mousesite/1968Demo.html>).

¹⁶ The ARPANET began life in the late 1960s as a network consisting of only four computers, or *network nodes*. It was started by the U.S. Department of Defense Advanced Research Projects Agency (DARPA). SRI hosted one of the network nodes.

The inherent philosophy of the endeavors of Engelbart and his team was *bootstrapping*, which they defined as the recursive process of building tools that let you build better tools. A successful example of the bootstrapping philosophy is the UNIX operating system.

Even with such impressive innovations, NLS ran into misfortune. Several NLS team members went to the then nascent Xerox PARC, where they hoped to create a distributed (across the network)—rather than time-sharing—version of NLS. Worse still, SRI dropped the program, leaving no funding for the project. Engelbart went to a phone networking company called *Tymshare*, where he sat in a cubicle in an office building in Cupertino, very near to the birthplace of the Macintosh.

1.2.4. Smalltalk

The work done at Xerox PARC would greatly influence the face—and surely the *interface*—of computing. The 1970s saw the development and maturation at PARC's Computer Science Laboratory (CSL) of technologies such as high-quality graphical user interfaces, windowing systems, laser printing, and networking. Smalltalk emerged as both a programming language and a programming environment at PARC.

While a graduate student at the University of Utah in the late 1960s, Alan Kay had collaborated with Ed Cheadle on designing a personal computer, the FLEX machine, for those who were not computer professionals. The FLEX machine had a pointing and drawing tablet, a calligraphic display, and a multiwindowed graphical user interface. Its operating system was object-oriented. The user interacted with the computer—what Kay called a “*personal, reactive, minicomputer*”—using text and pictures. The machine's primary language was also called FLEX. It was a simple, interactive, programming language designed to run on a hardware interpreter. Kay's work had inspirations from many existing works of research, such as GRAIL,

LINC,¹⁷ LOGO, NLS, Simula, and Sketchpad. Kay described the design and implementation of the FLEX machine in his 1969 Ph. D. Thesis titled *The Reactive Engine*. Kay's doctoral work at Utah led to the development of Smalltalk, the first version of which was deployed at Xerox PARC in 1972; Kay was one of the founding members at PARC.

Besides Kay, Daniel H. H. Ingalls, Adele Goldberg, and others at PARC were involved in Smalltalk's development. Smalltalk was both a truly object-oriented programming language and an operating environment with an integrated user interface. Daniel H. H. Ingalls wrote the first Smalltalk evaluator in October 1972 as a thousand-line BASIC program. The first "program" to run on this evaluator was the summation $3 + 4$. Shortly afterwards, the Smalltalk-72 system appeared. It was implemented in Nova assembly language. Several versions followed, with perhaps the best known being Smalltalk-80.

Simula

Simula, which was one of the inspirations behind Smalltalk, was the first language to use object concepts. It originated at the Norwegian Computing Center, Oslo, in the mid 1960s, as a form of Algol 60 extended with classes and coroutines. It was intended to be suitable for discrete-event simulation, hence the name.

Later on, Smalltalk would be one of the inspirations behind the Objective-C programming language, which would be the language of choice on the NEXTSTEP platform. Apple would inherit NeXT's technologies, and Objective-C would be a key language for Apple as well. Many similarities can be readily seen between Smalltalk and Objective-C. Every Smalltalk variable refers to an object. Every

¹⁷ The LINC (Laboratory INstrument Computer) was a small stored-program computer built by Wesley Clark and Charles Molnar at MIT's Lincoln Laboratory in 1962. Digital Equipment Corporation subsequently manufactured the LINC. It is widely regarded as the first personal computer.

Smalltalk object is an instance of a class whose ancestor is a single base class named *Object*. Smalltalk uses a message-based model of computation. An operation involves sending a *message* to an object, which is the only way to interact with an object. The sender requests the receiver to perform an action named by a *selector*. A receiving object responds to a message by looking up in its class for a method with the same selector. If the method is not found, the object looks up in its superclass, and so on. The set of messages an object responds to constitutes the object's *protocol*. Figure 1–6 shows an example of Smalltalk code (note the similarities to modern-day Objective-C).

```
"The Towers of Hanoi"  
moveDisk: fromTower to: toTower  
    Transcript cr.  
    Transcript show: (fromTower: printString, '→', toTower printString).  
  
doHanoi: n from: fromTower using: usingTower to: toTower  
    (n > 0) ifTrue: [  
        self doHanoi: (n - 1) from: fromTower using: toTower to: usingTower.  
        self moveDisk: fromTower to: toTower.  
        Self doHanoi: (n - 1) from: usingTower using: fromTower to: toTower]  
  
(Object new) doHanoi: 3 from: 1 using: 2 to: 3.
```

FIGURE 1–6 The Towers of Hanoi implemented in Smalltalk

Perhaps the most consequential contribution of Smalltalk to personal computing was the Smalltalk environment's highly interactive user interface. As was the case with its predecessor, the FLEX machine, an important early goal of Smalltalk research had been to make computing systems more accessible to those who are not professional computer scientists. The hardware of a computer running Smalltalk consisted of a high-resolution bitmapped display screen, a mouse, and a keyboard. The user interface incorporated concepts such as the following.

- Overlapping and resizable windows that were essential to increasing the virtual real estate of the display screen, providing an illusion of multiple,

overlapping pieces of paper on an electronic desktop, with each “paper” containing an independently running activity

- Iconic and textual menus
- Scroll bars
- The use of the mouse as a single, uniform method of selecting, cutting, and pasting various types of objects
- The use of the mouse to perform operations, or *commands*, on objects

Smalltalk, along with its integrated environment, facilitated development of useful and interesting tools such as a WYSIWYG editor, a music capture and editing system, and an animation system. In turn, the availability of such tools was conducive to writing programs, editing text, drawing, real-time animation, and music synthesis. Smalltalk was meant to be a powerful language that experienced programmers could use, yet one that could still be easily grasped by children. Figure 1–7 shows a rendition of the Smalltalk-80 user interface.

Squeak

A modern, portable, open source implementation of the Smalltalk environment is available as *Squeak*, which was created by Kay,¹⁸ Ingalls, Kaehler, and others in 1995, while these people were employed at Apple. Squeak is available for Mac OS X.

Kay did not achieve all his goals with the FLEX machine. In the late 1960s, he came up with the idea of a powerful, easy to use, lap-sized personal computer that he called the *Dynabook*. It was so called because Kay envisioned the personal computer to be a dynamic conglomeration of all other media: animation, audio, graphics, text, and so on. Key described the Dynabook as a personal computer for “*children of all ages.*”

Kay’s work and ideas inspired an effort to create a personal computer at PARC in 1972: one that would be known as the *Alto*.

¹⁸ Kay would later become an Apple fellow.

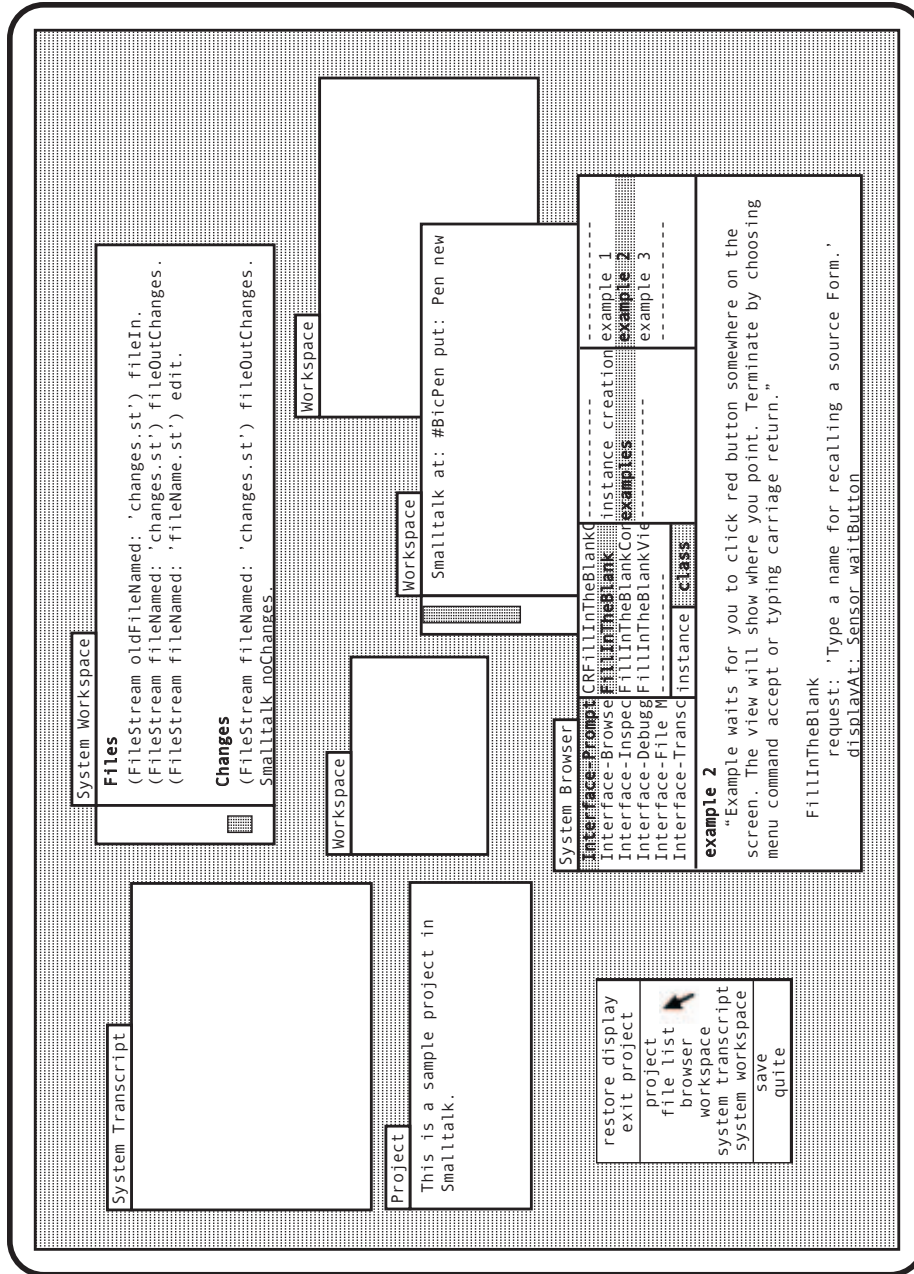


FIGURE 1-7 The Smalltalk-80 integrated user interface

Besides the Smalltalk group at PARC, the Pilot/Mesa and the Interlisp groups shared credit in developing or refining the concepts of bit-mapped graphics, windows, menus, and the mouse.

1.2.5. Xerox Alto

The “personal” in PARC’s personal computing effort referred to a non-shared system containing sufficient processing power, storage, and I/O capability to support the computing needs of a single user. The result was the Alto, which was originally designed by Charles P. Thacker and Edward M. McCreight. Other contributors to the Alto included Alan Kay, Butler Lampson, and various members of PARC’s Computer Sciences Laboratory and Systems Sciences Laboratory. Bob Metcalfe and David Boggs designed the Ethernet, an important technology used in the Alto.

The first Alto was functional in April 1973. It was named *Bilbo*. The very first bitmap picture it displayed was that of the Muppets’ Cookie Monster. Smalltalk was soon bootstrapped on this machine. Alan Kay referred to the Alto as the “Interim Dynabook.”

The Alto’s key hardware characteristics were the following.

- It consisted of a 16-bit medium-scale-integrated (MSI¹⁹) micro-programmed processor. The processor was *not* single-chip. It *emulated* a standard instruction set that was derived from the Data General Nova computer, thus aiding in portability. The corresponding emulation microcode—the *normal emulator*—resided in ROM. The instruction set could be extended through a small amount of microinstruction RAM.
- It had 64KB of error correcting code (ECC) 16-bit word semiconductor memory with a cycle time of 850 ns. The memory was expandable to 256KB.
- It had a bitmapped 606×808 point graphical display with a viewing area of 8.5”×11”. The display was oriented with the long tube dimension ver-

¹⁹ MSI refers to the number of electronic components on a chip. MSI has evolved to “very large” and “ultra large” scale integrated systems (VLSI and ULSI, respectively).

tical. It was implemented using a standard 875-line raster-scanned television monitor with a refresh rate of 60 fields per second (or 30 frames per second). The on-screen contents were refreshed from a bitmap in main memory, as bits were serially extracted from words fetched from memory, resulting in a video signal. There was a 16×16 hardware cursor whose bitmap was contained in 16 memory words at a specific address. The cursor was merged, or composited, with the video to present the final image. Displaying on the entire screen at full resolution (the screen-fill operation) required about 60% of the processor cycles.

- The Alto's input devices included a 61-key or 64-key unencoded keyboard, a five-finger key set, and a three-button mouse. The key set was programmatically visible as five bits of memory. The mouse contained a ball, as opposed to the wheels in the SRI mouse. Rather than numbering the mouse buttons, the Alto convention was to name them red, yellow, and blue, even though they were visually not of these colors.
- The Alto was housed in a small cabinet that contained the processor, one or more disks, and their power supplies. Other, larger I/O devices could be contained in their own cabinets that could be located away from the Alto, which was meant for desktop use. It had interfaces for local connection to printers and plotters, and a 2.94 Mbps Ethernet interface via which it could be connected to other Altos and laser printers. Note that the Alto's processor controlled the disk, the display, and the Ethernet.

The Alto could be booted from either a local disk, or the network. In the case of a disk boot, the user could specify a disk address from where to fetch the 256-word disk bootloader, with the default location being disk address 0. The first keyboard word was read to determine the boot type to perform. Pressing the <BS> key implied a network boot; otherwise, the microcode interpreted any reported keypresses as a disk address. The bootloader loaded a portion of Alto main memory from a boot file, eventually jumping to a known fixed location. The newly loaded program could then initialize still more of the Alto's state. In the case of a network boot, a special boot packet containing a 256-word Ethernet bootloader would have to arrive at the Alto for booting to commence. This packet—the *BreathOfLife packet*—was a raw Ethernet packet that was sent by a boot server periodically on each directly connected Ethernet. The boot microcode enabled the Ethernet receiver to

accept packets directed to a special host 377B. The received packets were copied into memory beginning at location 1. When a packet of type 602B was received without error, the Alto began executing instructions at location 3. The loader would establish further communication to continue booting.

Figure 1–8 shows a rendition of the Alto's interface.

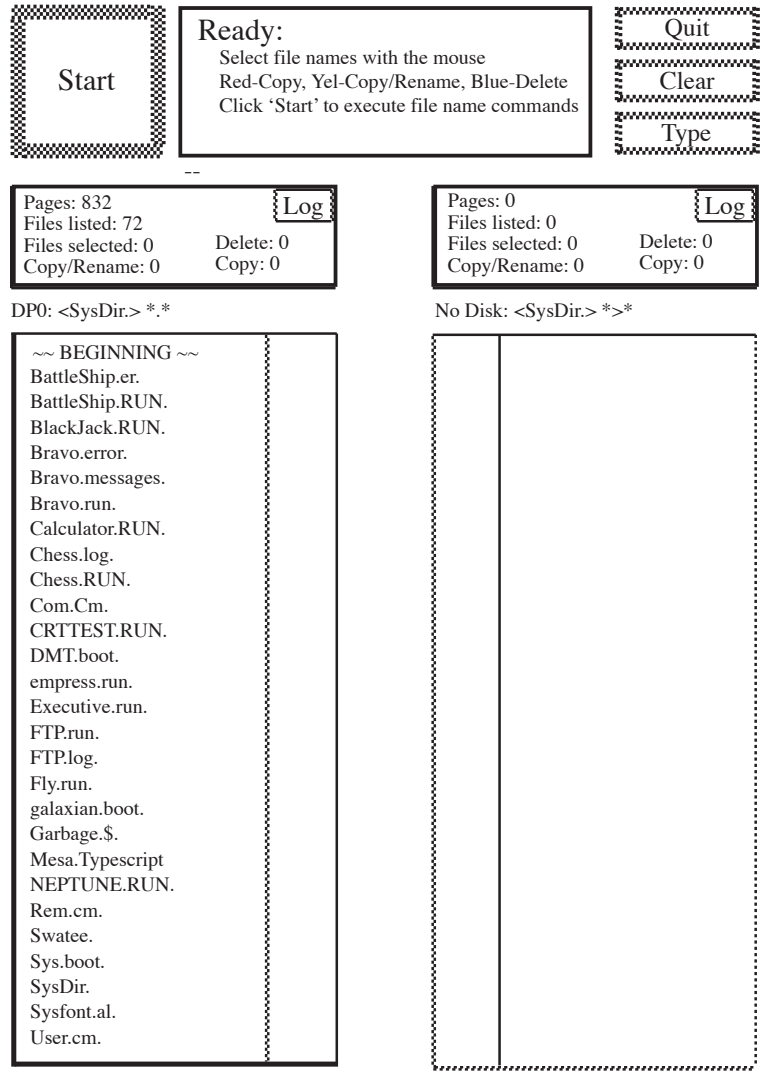


FIGURE 1–8 The Xerox Alto system

The Alto was later reengineered as the Alto II. By 1979, over 1500 Altos were in use, within and outside of Xerox.

As the number of Altos increased, many of the system aspects were standardized. Several standard facilities were made ROM-resident, including I/O device interfaces such as display, disk, Ethernet, keyboard, and mouse.

1.2.5.1. Alto OS

The Alto's operating system, which was stored in the file `sys.boot`, was implemented in BCPL. Its functionality included the following.

- Drivers for disk, keyboard, and display
- Management of memory, the clock, interrupts, and other events
- The file system
- The BCPL environment

BCPL was developed as a general-purpose recursive programming language, with emphasis on systems programming. It has similarities to ALGOL. It was used on a variety of computer systems such as CTSS (at Project MAC), PDP-11, GE635 (GECOS), and Nova.

The Alto ran its instruction set emulator in a task called *Emulator*, which ran with the lowest priority. It also was the 0th task. This task was always requesting wake-up, but could be interrupted by a wake-up request from any other task. In this sense, it ran in the background. Other standard tasks included tasks for the following subsystems.

- Disk (Disk Sector Task, Disk Word Task)
- Display (Display Word Task, Cursor Task, Display Horizontal Task, Display Vertical Task)
- Memory (Memory Refresh Task, Parity Task)
- Networking (Ethernet Task)

A technique called *Junta* allowed BCPL programs to eliminate layers of the Alto operating system that were not required by a particular subsystem.²⁰ Another technique called *Counter-Junta* could bring back the layers removed through Junta. Consequently, exceptionally large programs could run on the Alto with a clean return path to the operating system. To facilitate Junta, the operating system was divided into a series of levels. Each level had a known approximate size. Examples of levels include `levBcpl` (BCPL runtime routines), `levDisplay` (display driver), and `levMain` (the main operating system, including code for Junta itself). Figure 1–9 shows a pseudocode example of using Junta.

```
// All levels below levName will be de-activated, that is,
// levName will be the last level retained. Thereafter,
// ProcedureName() will be called; it should not return.
//
Junta(levName, ProcedureName)

...

// fCode could be fcOK or fcAbort
// This will also perform the CounterJunta()
//
OsFinish(fCode)
```

FIGURE 1–9 Using Junta

The Alto provided powerful and flexible graphics capabilities. For example, a set of BIT BLT²¹ routines was available as a BCPL driver. The BIT BLT algorithm was also implemented by a complex Alto instruction called BITBLT. Figure 1–10 shows an assembly-language excerpt from the source code of these routines (circa November 1975).

²⁰ The Alto's processor did not support virtual memory.

²¹ "BIT BLT" is pronounced as *bit-blit*. The "BLT" stands for Block Transfer. The term is commonly used to refer to an algorithm for moving and modifying rectangular bitmaps from one area of memory to another on a bit-mapped device. Typically one of the areas resides in main memory and the other resides in display memory.


```

...
CLIPC:
    MOV 3,1      ; SUBR FOR WINDOW CLIPPING
                ; SAVE RETURN - BBSTABLE COMES IN AC2
    STA 1,TEMP1,2
    LDA 0,CHAR,2
    LDA 1,SPACE
    SNE 0,1
    JMP SPCIT
    ISZ TRLCHR,2 ; INDICATE NON_SPACE CHAR - HELP DEAL
                ; WITH MULTIPLE SPACES IN JUSTIFICATION
    NOP         ; SOMETIMES USED AS NIL FLAG - ARGHH!!
...

```

FIGURE 1-10 Assembly-language excerpt from the BIT BLT source for the Alto

1.2.5.2. Alto Executive

The program that a user interacted with just after the Alto booted was the Alto *Executive* (`Executive.Run`)—a command interpreter akin to the UNIX shell. It ran atop the Alto OS, and could theoretically be replaced by another program. It had built-in facilities for executing programs and command files, listing on-disk files, querying file sizes, and so on. Examples of Executive subroutines that could be invoked from the command line included the following.²²

- `BootFrom.~ FileName [...Sys.Boot]`
- `Chat.~`
- `Copy.~ DestFileName SourceFileName ...`
- `Delete.~ FileName ...`
- `Diagnose.~`
- `FileStat.~ FileName ...`
- `Ftp.~`
- `Login.~`
- `NetExec.~`

²² The “~” character, which was used for identifying Executive commands, was illegal in a filename.

- Quit.~
- Rename.~ OldFileName NewFileName
- Scavenger.~
- SetTime.~
- Type.~ FileName ...

The Executive supported powerful command-line editing and filename expansion. The user could refer to a set of files by specifying patterns containing special characters such as * and #, which the Executive expanded. The Executive displayed a digital clock and other useful status information, such as the versions of the operating system and the Executive itself, the owner's name, the disk's name; the Ethernet address of the Alto, and the number of free pages on the disk. When a user called another program from the Executive, the display was erased and replaced by that of the called program. The operating system invoked the Executive whenever a program finished running, or specifically, whenever the BCPL operator `finish` (or equivalent) was executed. Figure 1-11 shows an example of the Executive's prompt.

```
-- XEROX Alto Executive/11 ----- May 18, 1981 -- 786 Pages ----- OS Version
18/16 --- Alto 0#377# --- --- -----
> // eventBooted
> |
```

FIGURE 1-11 The Alto Executive's prompt

The Alto's *Find* subsystem allowed fast pattern-based file searches. Special-purpose files such as address or telephone lists, program source files, and library catalogs could be effectively searched through this subsystem.

1.2.5.3. NetExec

There also was a networked version of the Executive—the *NetExec*—that loaded programs from a boot server available via the Ethernet rather than from the local

disk. One could query available facilities by typing `<?>`. Examples of utilities typically available through the NetExec included the following.

- *Scavenger* was a program that rebuilt the file structure, but not the content, of an Alto disk. It was analogous to the UNIX `fsck` program. It checked disk packs while attempting to correct erroneous header blocks, checksum errors, and other problems. It prompted the user for confirmation in most cases. It could discover all well-formed files and all free pages, verify that the serial numbers of all well-formed files were distinct, and link all irrecoverably bad pages together as part of the file `Garbage.$`. Scavenger was available from the NetExec as an emergency remedial measure in case the local disk was rendered unbootable.
- *Chat* was a subsystem that allowed teletype-like interactive access to a remote computer on the network. It included an extension to support text-display control and graphics, similar to the latter-day `telnet` program.
- *CopyDisk* was a standalone program used to transfer the entire contents of a disk, either between computers, or between the multiple disks of a single computer.
- *FTP* was a file transfer program.

1.2.5.4. Programming Facilities

Many programming languages were available on the Alto, including BCPL, LISP, Smalltalk, Mesa,²³ and Poplar.²⁴

The system's debugger was called *Swat*. It saved machine state in a file named `swatee`. You could stop a running program and get back to the Executive by holding down the left `<SHIFT>` key and hitting the `<SWAT>` key.

²³ Mesa was a Pascal-like strongly-typed system programming language.

²⁴ Poplar was a simple, interactive, text-oriented programming language.

1.2.5.5. Applications

The Alto began with several productivity applications and went on to have many more. Examples of sophisticated Alto applications include the following.

- *Bravo* was a feature-rich, multiwindowed, text-processing application.
- *Draw* was an interactive illustrator program for creating pictures composed of lines, curves, and text captions. It divided the screen into multiple areas: brush menu, command menu, font menu, picture area, caption area, and a message area for displaying information, error, or prompting messages.
- *Laurel*²⁵ was a display-oriented mail messaging system that provided facilities to display, forward, classify, file, and print mail messages. It stored messages on the local Alto disk.
- *Markup* was a document illustration application.
- *Neptune* was a utility program for managing files and directories.
- *Officetalk* was an experimental forms-processing system that inspired the Xerox STAR system.

Alto used file extensions to conventionally indicate file content types. A file extension was a filename's trailing portion following a period. It could be null. Examples of such extensions include ".log" (program action history), ".mail" (Laurel mail file), ".run" (BCPL program executable), ".st" (Smalltalk source file), and ".syms" (BCPL symbol table file). Alto filenames could be at most 39 characters in length.

1.2.5.6. Networking

The development of inter-network communication facilities at PARC led to a packet format called PARC Universal Packet, or *Pup*. Besides the name of the abstract design of the packet format, *Pup* also represented the corresponding inter-network architecture, which included a hierarchy of protocols. The standard *Pup* format is shown in Figure 1–12. *Pup* influenced the creation of the TCP/IP protocol suite.

²⁵ A subsequent mail program for an Alto-successor machine was called *Hardy*.

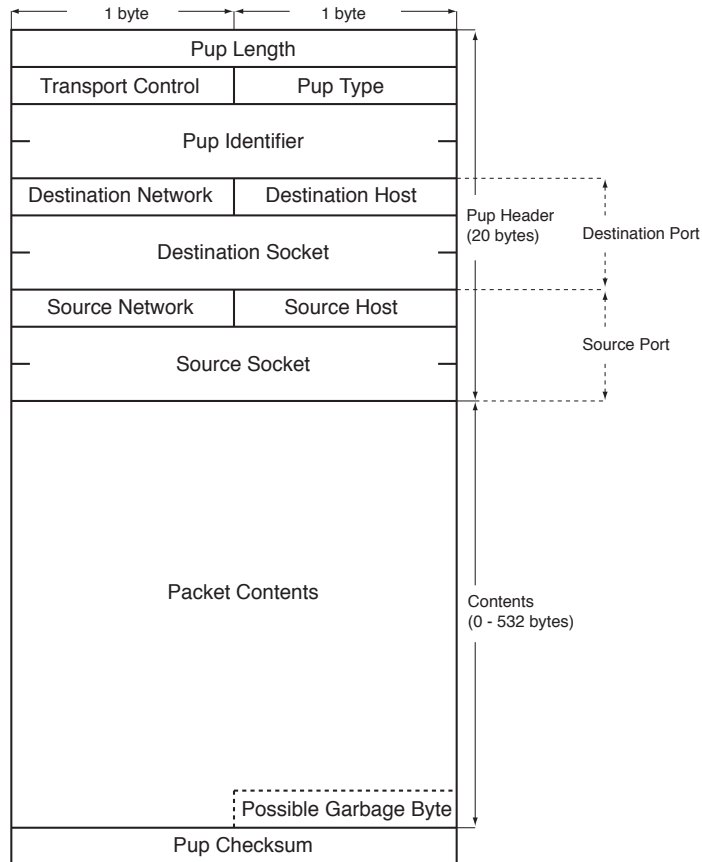


FIGURE 1-12 The standard Pup format

The Alto made heavy use of networking. As we saw earlier, it also included capabilities for file transfer and remote interactive access. A protocol called *Copy-Disk*, which was similar to FTP, allowed creation of a disk's bit-for-bit copy over the network. In all such protocols, the server *listened* for connection requests on a well-known *socket*. A client initiated a connection to the server using commands sent over an established connection to operate the server.

1.2.5.7. Worms

An interesting investigation involving networked Alto computers was with *worm* programs. In 1975, science fiction writer John Brunner had written about such programs in his book *The Shockwave Rider*. PARC researchers John F. Shoch and Jon A. Hupp experimented with worm programs in the early 1980s. The experimental environment consisted of over a hundred Ethernet-connected Altos. A worm was simply a multimachine computation, with each machine holding a *segment* of the worm. Segments on various machines could communicate with each other. If a segment was lost, say, because its machine went down, the remaining segments would search for an idle Alto on which to load a new copy—*self-repairing software*. It is important to note that the idea behind the worm experiments was not that of mischief. The researchers intended to create useful programs that would utilize otherwise idle machines—essentially a form of distributed computing.²⁶ Nevertheless, the aberrant potential of worms was clearly identified, although worms were still not perceived as a real security risk. Comparatively, viruses and self-replicating Trojan horse programs were considered bigger threats to security. Examples of applications that used such worms include the following.

- The *Existential* worm was a null worm that only contained logic for its own survival.
- The *Billboard* worm could distribute graphics images to multiple machines.
- The *Alarm Clock* worm was a distributed, fault-tolerant, computer-based alarm clock that could call a user on the telephone at a designated time.
- The *Multimachine Animation* worm was part of a distributed real-time animation mechanism involving multiple compute nodes and a single master node.

²⁶ The Alto worms were conceptually similar, in some aspects, to the controller and agent programs in a modern-day grid-computing environment such as Apple's Xgrid.

- The *Ethernet Diagnostic* worm conducted network tests to locate erroneous Ethernet interfaces. It reported the findings to a control host.

Other useful things that the worms were meant to do included reclaiming file space, shutting down idle workstations, and delivering mail.

Trek

A popular application for the Alto was the game *Trek*, which used a number of Alto computers on a single Ethernet to facilitate a distributed multiplayer game. The game's objective was to destroy the enemy and his base without being destroyed, and to become the "Master of the Universe." Each Alto user participating in the game could control one space ship. *Trek*'s display consisted of seven distinct areas: long range and galaxy scan; short range scan; acceleration and direction controls; energy and velocity indicators; damage information; fire control and supplies indicators; and the communication area.

Xerox used the Alto's technology to create a system designed for office professionals: the *STAR System*.

1.2.6. Xerox STAR System

Xerox introduced the 8010²⁷ STAR Information System at a Chicago trade show in April 1981. The STAR's hardware²⁸ was based on the Alto, with better components such as more memory, bigger disks, a higher resolution display, and faster Ethernet. An important difference from the Alto was that the STAR user interface was explicitly designed *before* actually building the hardware or software. Moreover, rather

²⁷ The Xerox STAR 8010 was also nicknamed the *Dandelion*.

²⁸ Even though the overall system was marketed as "STAR," the name STAR should only apply to the software. The computer itself was an 8000 series workstation.

than using an existing computer, the hardware was specially designed for the software. Figure 1–13 shows a mockup of the STAR user interface.

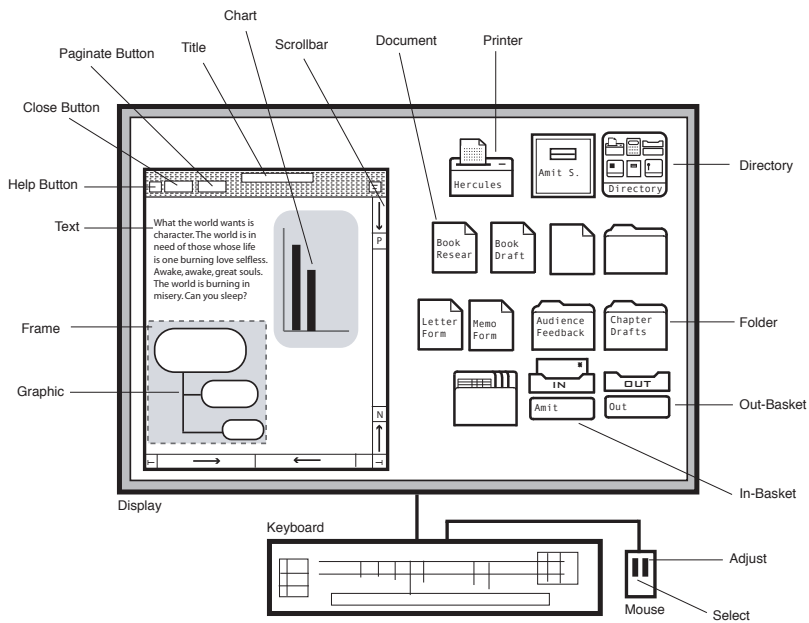


FIGURE 1–13 The Xerox STAR system

The STAR user interface provided the user with an electronic metaphor for the physical office. As Figure 1–13 shows, there were electronic analogs of common office objects: paper, folders, file cabinets, mailboxes, calculators, printers, and so on. It would be an understatement to say that the STAR interface greatly influenced many systems that came after it. Noteworthy aspects of the STAR user interface included the following.

- The user's first view of the working environment was the *Desktop*, which displayed *icons* (small pictures) of familiar objects such as documents, folders, file drawers, in-baskets, and out-baskets.

- The user could click on an icon and push the <OPEN> key to *open* the icon, which resulted in a window displaying the icon's contents. Icons could represent either *Data* (Document, Folder, or Record File) or *Function*. The user could copy, delete, file, mail, move, open, close, and print data icons. Function icons operated on data icons. Many of the function icons were analogous to modern-day application icons. Examples of function icons include File Drawer, In- and Out-Baskets, Printer, Floppy Disk Drive, User, User Group, Calculator, Terminal Emulator, and Network Resource Directory.
- Windows had *title bars* displaying the corresponding icon names and a context-sensitive command menu. Context-sensitive help was accessible via the ? button. Horizontal and vertical *scroll bars* provided page-up, page-down, and jump-to features. However, STAR windows could only be tiled²⁹ and were explicitly designed not to overlap; overlapping behavior was considered a nuisance (for the end user) in the Alto by STAR's designers.
- An abstraction called *Property Sheets* was analogous to today's control or property panels. A related abstraction called *Option Sheets* implemented a visual interface for providing *options* (arguments) to commands. For example, the "Find" option sheet was a powerful tool for searching text in a part or whole of a document or selection. Even textual properties such as case, face, font, position, and size could be used for searching. It also allowed optionally user-confirmed replacement of the found text and its properties.

The STAR system was rather expensive, with a base configuration initially costing \$16,500.

1.3. THE GRAPHICAL AGE AT APPLE

Apple was instrumental in making personal computing affordable for the masses with the Apple II. However, even after the Apple II's success, computers were difficult to learn for most people—a fact that was perceived by Apple and many others as a large impediment to computer use. The *Lisa* (Local Integrated Software Archi-

²⁹ Window overlapping was allowed, and in fact was turned on by default, in STAR's successor *ViewPoint*.

ecture) project began at Apple in 1979 to create an integrated, stand-alone, single-user, and easy-to-use microcomputer.

1.3.1. Lisa

Lisa's goals included the following.

- Be intuitive
- Be consistent
- Provide an integrated environment that was in line with people's day-to-day work
- Provide sufficient performance without being overly complex or exorbitantly priced
- Provide an open architecture to make it easy for Apple and third parties to develop additional hardware and software
- Be reliable
- Be aesthetically appealing and not out of place in a typical work environment

As we noted at the beginning of Section 1.2, Lisa was greatly inspired by the work done at Xerox PARC: especially by the Smalltalk environment, and to a lesser extent, by the Xerox STAR System. Apple was made privy to many details of PARC's technologies, thanks to a deal in which Xerox received Apple stock in return for allowing Apple to visit Xerox to observe and understand some of Xerox's work. The marketing requirements for the Lisa project, which was already underway when Apple visited PARC for the first time, included heavy incorporation of Smalltalk concepts. Apple also adopted the STAR system's Desktop metaphor, along with STAR's use of icons. Apple would refine and augment these concepts and use them to create a pragmatic and efficient user interface for Lisa.

Apple released Lisa at an annual shareholder meeting on January 19, 1983—a year before the Macintosh was introduced. Lisa's price was \$9995, making it five

times more expensive than the originally planned price of \$2000.³⁰ Apple proclaimed that Lisa would revolutionize the way work was done in office environments. Apple also emphasized Lisa's small learning curve by claiming that a first-time user could do productive work with Lisa in less than 30 minutes. Apple had earlier estimated that existing computers required twenty to thirty hours of training and practice before a user could start being productive.

1.3.1.1. Packaging

The first Lisa system consisted of the following discrete parts.

- A patented compact desktop unit weighing 48 lbs; it housed the computer itself, a CRT screen, two floppy disk drives, and the power supply
- A one-button mouse
- A keyboard with a numeric keypad
- An Apple ProFile hard disk drive unit

Various peripherals such as mouse, keyboard, hard disk drives, printers, and serial devices could be plugged into the main unit. Moreover, it was possible to disassemble the unit without tools.

1.3.1.2. Processor and Memory

Lisa had a 5 MHz Motorola MC68000 processor, which had a 16-bit external data path with a 32-bit internal architecture. The system had a memory management unit (MMU), but no floating-point unit (FPU). Lisa initially included 1MB of RAM, but could support up to 2MB. However, the processor was capable of handling a 16MB address space, and supported multiple memory addressing modes. A physical address could lie in one of following three address spaces:

³⁰ When introduced, the Lisa project had consumed about 200 man-years and 50 million dollars.

- The main memory (RAM), where Lisa programs and data were stored
- The I/O address space, which was used for accessing peripheral controllers, status registers, and control registers
- The special I/O address space, which was used for accessing the boot ROM and internal MMU registers

Lisa's MMU provided four (numbered 0 through 3) logical address space contexts for programs to run, of which context 0 was reserved for use by the operating system.

1.3.1.3. Display

Lisa had a 12" CRT display with active dimensions of 8.5"×6". Apple referred to it as a "half-page"³¹ display, given that a Letter-sized sheet of paper measures 8.5"×11". The bitmapped display's maximum resolution was 720×364. It allowed for up to 45 lines of 144 characters each. The horizontal and vertical pixel densities were not the same, being 90 and 60 dots per inch, respectively. Therefore, a mathematical square did not appear as a square on Lisa's screen. Like the Xerox STAR system, Lisa strived to present a physical office metaphor. It mimicked the appearance of real-life paper by displaying black text on white background. Since a white screen flickers more than a black screen, Lisa required a higher refresh rate for its display, thus adding to its price.

1.3.1.4. Storage

Each of Lisa's two built-in floppy disk drives used special high-density double-sided 5.25" diskettes, with a formatted capacity of 851KB. The 5MB Apple ProFile was a self-contained external hard disk drive unit that connected to Lisa's built-in

³¹ You can get a feel for the size of Lisa's display by folding a Letter-sized sheet of paper into half.

parallel port. It was possible to connect multiple such disk unit, for a maximum of seven units, using optional parallel interface cards.

1.3.1.5. Expansion

Lisa had a built-in parallel port, two built-in serial ports, and three expansion slots on the motherboard that could be accessed by removing the main unit's back panel. It also had a composite video output allowing it to be connected to compatible external monitors.

1.3.1.6. Lisa OS

Lisa was introduced with a proprietary operating system (the Lisa Office System, or Lisa OS) and a suite of office applications. Several aspects of Lisa's software would become part of Apple's systems to come, and in fact, many such concepts exist in Mac OS X in some form.

Lisa was not tied to a particular operating system. Unlike its successor, the Macintosh, Lisa did not have portions of the operating system in ROM. This allowed it to support multiple operating systems. It presented the user with an interactive screen called the *Environments Window* if multiple bootable environments (for example, Lisa OS and Lisa Workshop) were found on attached storage devices.

At the time of Lisa's release, Apple announced that Microsoft had been working on a version of Xenix for Lisa. Eventually, SCO Xenix was available for Lisa.

Most of Lisa's system and application software was written in an extended version of Pascal (Lisa Pascal). During Lisa's development, Apple even considered using a p-Chip that would run p-Code natively.

Lisa OS was designed to support Lisa's graphical user interface. Figure 1-14 shows a logical view of Lisa's software architecture.

Lisa OS was designed to support Lisa's graphical user interface. Figure 1-14 shows a logical view of Lisa's software architecture.

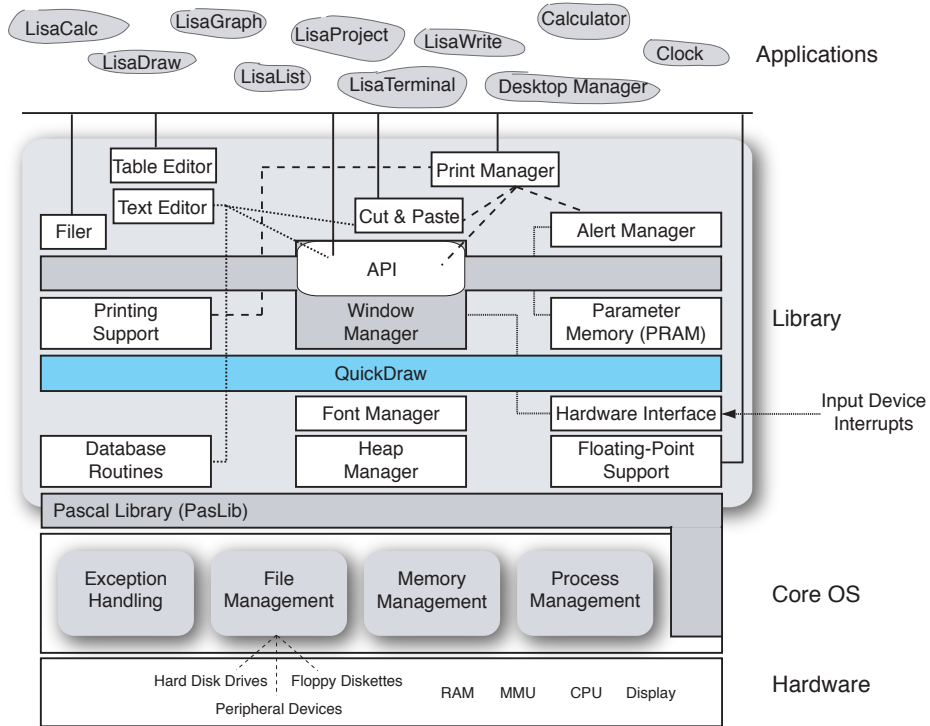


FIGURE 1-14 Lisa's software architecture

Process Management

Lisa OS implemented non-preemptive multitasking. Although the scheduling algorithm was simple in the absence of preemption, it supported process priorities.

A process could only be created by another process, except the initial process, which was created by the operating system as the “shell” process upon booting. The shell process ran the *Desktop Manager* application by default. The system's process management API included calls for creating, terminating, suspending, and resuming

processes. Terminating a process also resulted in the termination of all its descendants.

Examples of Lisa process-management system calls included the following.

- `make_process`
- `kill_process`
- `activate_process`
- `suspend_process`
- `info_process`
- `setpriority_process`
- `yield_process`
- `sched_class`

System-level exceptions resulted in the termination of a process—a side effect of the execution of default exception handlers. Processes could install custom exception handlers, which were invoked with detailed exception context. Examples of Lisa exception-management system calls included the following.

- `enable_excep`
- `disable_excep`
- `declare_excep_hdl`
- `signal_excep`

Interprocess Communication

By default, a process was not allowed to access the logical address space of another process. Interprocess communication was possible through multiple mechanisms such as events, shared files, and shared memory. *Events* were structured messages consisting of a system-attached header and a sender-provided data block, transmitted between processes over named *channels*. A process could listen on a channel, waiting for messages to arrive. Alternatively, a process could register an exception handler and arrange for an exception to be generated upon message arrival.

Examples of Lisa event-channel management system calls included the following.

- `make_event_chn`
- `kill_event_chn`
- `open_event_chn`
- `close_event_chn`
- `wait_event_chn`
- `send_event_chn`

Memory Management

Lisa OS supported segmented virtual memory wherein read-only code segments could be swapped in as needed. A program had to be accordingly compiled and linked for this to work. For example, the programmer could divide a program into independent parts, with the size of each part being limited to 128KB. When an instruction attempted to access code that was not in physical memory, a bus error occurred. The consequent system trap was handled by the memory manager, which brought the required segment into physical memory and restarted the instruction.

Examples of Lisa memory-management system calls included the following.

- `make_dataseg`
- `kill_dataseg`
- `open_dataseg`
- `close_dataseg`
- `mem_info`
- `info_address`

File System

Lisa OS used a hierarchical file system that incorporated both UNIX-like aspects and some hitherto new concepts. The `mount` system call performed a similar function as on UNIX: it was used to introduce a new object into the file system's name

space. Besides files and folders, the file system name space also contained disk volumes, printer devices, and serial devices—again, like UNIX. The system performed I/O to these objects uniformly, regardless of the underlying device, although one could perform device-specific “control” operations on files representing devices.

The file system stored redundant information for each file to reduce the likelihood of data loss in a crash. The volume format had a central disk catalog that contained critical information about each file. This information was replicated in a special block belonging to the file. Moreover, each used block on disk was *tagged*, containing information indicating that block’s logical position in the contents of the file that it belonged to. Thus, critical redundant information was distributed throughout the volume, making it possible to recover or reconstruct information after a system crash in many cases. The standard file system repair program was called the *scavenger*.³²

The Lisa file system supported per-file attributes. System-generated metadata, such as a file’s size and creation date, were stored as attributes. There were APIs using which applications could define, create, and access their own attributes, which were stored in a per-file *label*—an early form of the resource fork in the latter-day Macintosh Hierarchical File System (HFS). It was also possible for a program to preallocate contiguous file system space.

Examples of Lisa file system calls included the following.

- `open`
- `make_pipe`
- `read_data`
- `flush`
- `lookup`
- `allocate`
- `truncate`

³² Recall that the Alto’s file system repair program was also called scavenger.

- `read_label`
- `get_working_dir`
- `device_control`

Other sets of system calls included those for timers and for manipulating system configuration. The system call Pascal interface was implemented in the Lisa Operating System library.

The Graphical User Interface

Perhaps the best-known aspect of Lisa is the graphical user interface (GUI) of the Lisa OS. The system's shell—the *Desktop Manager*—used icons of documents and folders to act as an electronic analog of a real desk with a filing system. Some noteworthy aspects of the user interface included the following.

- Multiple, overlapping windows
- Hierarchical pull-down menus
- A menu bar³³ that was always visible at the top of the screen and changed automatically as the active application changed, revealing options that were relevant to that application
- Dialog boxes for prompts, error messages, and other interactions with the user
- Horizontal and vertical scroll bars
- Use of the mouse for pointing, clicking, double-clicking, dragging, and selecting – optionally in conjunction with key-presses
- A cursor whose shape changed depending on its current function
- Special or *modifier* keys on the keyboard (including the “Apple key”), with support for direct invocation of frequently used menu commands from the keyboard
- Editing commands such as copy, cut, paste, and undo
- A “scrap folder” called the *Clipboard*—for use by copy-paste and cut-paste operations

³³ Lisa's menu bar did not have the Apple menu found in latter systems. However, menu commands did use an Apple symbol. Apple later started using the cloverleaf symbol (the `cmd` key) for menu commands.

The Clipboard was implemented as a globally shared data segment accessible by all processes. Data was stored in the Clipboard in multiple formats, including a generic format that could be used by a process that did not understand the application-specific version of that data.

There were graphic representations (icons) of objects typically found in an office environment, such as files, folders, blank stationery, clipboard, and trash can (or wastebasket). The icons were active in that they could be clicked. They were used to represent both objects and tasks. Double-clicking on a folder icon would display the folder's contents in a window, whereas double-clicking on a document icon would launch the appropriate application to open the document (each document could have an associated application with it). The resultant windows came up animated. An object could be renamed by pointing at its icon and simply typing the desired name. New folders and documents were created by “tearing off” items from *pads* of empty folders and documents, respectively. Items were deleted by dragging them to a trash can icon. Figure 1–15 shows a screenshot of Lisa's user interface.

Advanced Features

In addition to bringing graphical user interfaces to mainstream computing, Lisa was among the first to provide software controls for hitherto mechanical ones. It had instant-on capability: pressing the power switch while Lisa was operating sent a reset signal to the CPU. The handler routine for this signal caused open documents and the state of the desktop to be saved to disk, after which the computer went into a low-power state. Powering Lisa on would restore the desktop state.

Other software controls allowed ejecting a diskette; and adjusting preferences such as screen brightness, key repeat rate, and the tone generator volume.

Lisa also had a hardware serial number that could be used by its software for multiple purposes, including a rudimentary form of digital content protection.

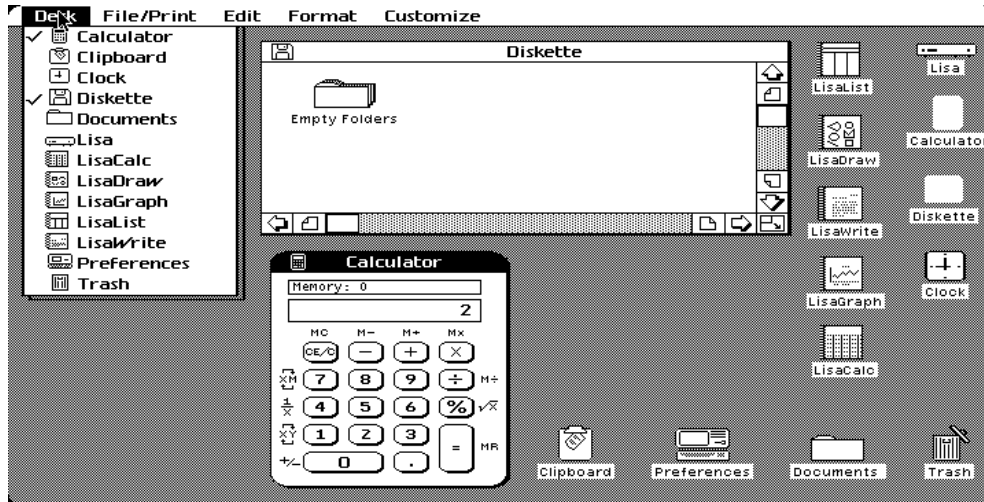


FIGURE 1-12 Lisa's graphical user interface

Lisa came with a suite of the following GUI-based office applications.

- *LisaCalc*—spreadsheet and financial modeling tool with support for up to a 255×255 worksheet
- *LisaDraw*—drawing and illustration application
- *LisaGraph*—application for making bar, line, pie, and scatter graphs
- *LisaList*—database for creating and maintaining various types of lists
- *LisaProject*—visual project-management and scheduling application based on Project Evaluation and Review Technique (PERT)
- *LisaTerminal*—asynchronous communications application that supported emulation of teletype (TTY), VT52, and VT100 terminals
- *LisaWrite*—WYSIWYG word processor

Lisa applications had consistent user interfaces so that a user was not required to learn every application from scratch. It was also possible to transfer data between applications.

Lisa's competition included the IBM PC XT (introduced on March 8, 1983),

the DEC family of personal computers (the Rainbow 100, the DECmate II, and the PC300 line), the Corvus Concept Network Workstation (which also used the M68000 processor), the Fortune 32:16 Hard Disk System, the Xerox STAR System, and the Xerox 820-II Personal Computer.

Multiple programming languages were available³⁴ for Lisa, such as Pascal, BASIC-Plus, C, and COBOL. Lisa's software library, many of whose components are shown in Figure 1–14, provided a variety of primitives for building complex applications. Pascal language units in the software library included the Alert Manager, the Font Manager, the Print Manager, QuickDraw, and the Window Manager.

QuickDraw was a high-performance bitmap graphics technology that used regions, both for clipping and for implementing shapes with efficient use of memory. A region was an arbitrary area that could include multiple groups of disjoint areas.

1.3.1.7. Lisa WorkShop

Lisa WorkShop was Lisa's development environment. Strongly influenced by UCSD Pascal, it was primarily meant for Pascal-based development, although it supported other languages such as BASIC-Plus, C, and COBOL-74. It provided a multitude of tools for software development: a Pascal compiler, a 68K assembler, a linker, a low-level debugger, a file comparison utility, a mouse-based graphical text-editor, and so on. It also provided a command-line shell, along with subsystems for accessing files on storage devices (the *File Manager*) and for accessing low-level features of Lisa (the *System Manager*). The WorkShop could even copy-protect software, after which copies of such protected software would only run on one machine (the first machine on which the copy was made).

³⁴ The Pascal, Basic-Plus, and COBOL products could be purchased from Apple at the time of Lisa's introduction at the suggested retail price of \$595, \$295, and \$995, respectively.

The WorkShop was extensively used for the development of Macintosh software.

1.3.1.8. Lisa's Fate

Despite Lisa's technical sophistication and Apple's advertising,³⁵ Lisa was a commercial failure, partly owing to its high cost. The addition of more memory and a disk drive pushed Lisa's price well above \$10,000. The Macintosh, which was under development, was perceived by many as a far more affordable mini-Lisa. In 1984, Apple released a second version of the computer, Lisa 2, at half the price of the original. The Lisa 2/5 variant came with a 5MB external ProFile disk drive, whereas the Lisa 2/10 variant came with an internal 10MB "Widget" drive. A year after the Macintosh was introduced, Lisa 2 was re-branded as the Macintosh XL. It ran the Macintosh operating system courtesy of the MacWorks XL software, which implemented Macintosh ROM emulation.

Lisa was discontinued in 1985. In September 1989, Apple buried about 2700 Lisa computers in the Logan landfill in Utah. The value of the computers had depreciated so much that the tax break received from scrapping the computers resulted in more money for Apple than could be obtained by selling them.

Although Lisa failed to become the perfect computer it was designed to be, it introduced several aspects that would become part of Apple's systems to come. In this sense, Lisa was a technological success.

³⁵ Hollywood actor Kevin Costner appeared as a businessman in a 1983 Lisa advertisement.

1.3.2. The Macintosh

At the turn of the 1980s, there was a project called *Annie* inside Apple. Apple employee Jef Raskin³⁶ was not happy with names such as “Lisa” and “Annie,” which represented a sexist approach according to him. He changed the project’s name to *Macintosh*, a deliberate misspelling of “McIntosh,” which is a variety of Apples.

McIntosh was also part of the name of a stereo manufacturer called McIntosh Labs. The name was brought under contention when Apple tried to trademark it, but Apple eventually managed to buy the trademark. During the legal battle, Apple considered acronyms such as MAC, for Mouse Activated Computer. There were alleged jokes within Apple that “MAC” actually was an acronym for Meaningless Acronym Computer. For a short while, there were even efforts to change the project’s name to Bicycle, which alluded to a quote from Steve Jobs about personal computers being “bicycles for the mind.”

Jef Raskin had written *The Book of Macintosh*, an Apple-internal document on personal computing that described a cheap, user-friendly computer designed for the *Person In The Street* (PITS). Some of the desirable features of the computer were the following.

- The computer “system” must not consist of myriad external components. It must be all in one, with components such as the display, the keyboard, disks, and others all integrated into one package. Moreover, the package must be portable, with a handle, and must not weigh more than 20 lbs.
- The computer’s internals should not be visible.
- The PITS should never be required to open the computer, nor even see its interior. The only reason to open the computer should be for repair.
- The PITS should not be required to deal with components in sockets, whether inside or outside of the computer. Any additional boards, RAMs, ROMs, or other accessories should be allowed only if they exist as standalone appliances that can be trivially connected to the computer.

³⁶ Jef Raskin was Apple employee number 31.

- There should be no external cables besides a power cord. In the ideal world, there would not even be a power cord!
- If there are multiple models of a computer system, the models should not have differences that require documentation in a user's manual.
- There should not be too many keys on the keyboard.
- There should be as few manuals as possible, and even those should be small. The manuals should not use computer jargon.
- The end-user should be able to purchase the computer for no more than \$500.

Besides documenting his vision for an affordable, appliance-like computer, Jef Raskin also put together a capable initial team in late 1979 to pursue his Macintosh project. However, Raskin left the project—and some time later, the company—before Macintosh could become substantial. Steve Jobs, who took over, would be the driving force behind the Macintosh *product*.

Steve Jobs unveiled the Macintosh on January 24, 1984, at the Flint Center in De Anza College, Cupertino. The occasion was Apple's annual shareholders meeting. Jobs opened the meeting by reciting lines from Bob Dylan's song *The Times They Are A-Changin'*.

*Come writers and critics
Who prophesize with your pen
And keep your eyes wide
The chance won't come again
And don't speak too soon
For the wheel's still in spin
And there's no tellin' who
That it's namin'.
For the loser now
Will be later to win
For the times they are a-changin'.*

Besides being a vehicle of Lisa's technology of windows, icons, pull-down menus, mouse commands, and software integration, the Macintosh was a compelling execution of marketing wizardry. Apple hailed it as "*the computer for the rest of us.*" After demonstrating the Macintosh's capabilities, Jobs said "*We've done a lot of talking about Macintosh recently, but today, for the first time ever, I'd like the Macintosh to speak for itself.*" A program running on the Macintosh "spoke" an introductory message:

"Hello, I am Macintosh. It sure is great to get out of that bag. Unaccustomed as I am to public speaking, I'd like to share with you a maxim I thought of the first time I met with an IBM mainframe: NEVER TRUST A COMPUTER YOU CAN'T LIFT! Obviously, I can talk, but right now I'd like to sit back and listen. So, it is with considerable pride that I introduce a man who's been like a father to me... STEVE JOBS."

An early Macintosh sales brochure had the following blurb: "*For the first time in recorded computer history, hardware engineers actually talked to software engineers in moderate tones of voice, and both were united by a common goal: to build the most powerful, most transportable, most flexible, most versatile computer not-very-much-money could buy. And when the engineers were finally finished, they introduced us to a personal computer so personable it can practically shake hands. And so easy to use most people already know how. They didn't call it the QZ190, or the Zipchip 5000. They called it Macintosh.*"

Later known as the Mac 128K due to the 128KB of built-in RAM, the Macintosh debuted at a price of \$2,495. It had the following key specifications.

- 7.83 MHz Motorola MC68000 processor
- No memory management unit (MMU), no floating-point unit (FPU), and no L1 or L2 caches
- 32-bit internal data bus
- 64KB ROM

- 128KB RAM
- 20 bytes of parameter memory (PRAM) on a CMOS custom chip with 4.5 V user-replaceable backup battery
- Internal single-sided 3.5" floppy disk drive that accepted 400KB *hard shell* floppy disks
- An external drive port with a DB-19 connector that allowed attachment of a second drive
- Mouse port (DE-9 connector), and a mechanical tracking mouse with optical shaft encoding at 3.54 pulses per mm (90 pulses per inch) of travel
- Synchronous serial keyboard bus with an RJ-11 connector, and a software-mapped 58-key keyboard
- Two RS-232/RS-422 serial ports (DE-9 connectors) for connecting modems, printers, and other peripherals
- Four-voice sound generator with 8-bit digital/analog conversion and 22 kHz sampling rate
- 512×342-pixel bit-mapped black and white display on a 9-inch diagonal CRT screen
- Physical dimensions of 13.6"×9.6"×10.9", with a weight of 16 lb 8 oz (7.5 kg), and a logic board area of 80 square inches
- No internal fan

The Macintosh ran a single-user, single-tasking operating system, initially known as *Mac System Software*, which resided on a single 400KB floppy disk. Many Macintosh programs were either conversions of, or influenced by, Lisa programs. Examples include MacPaint, MacProject, and MacWrite.

Unlike Lisa, the Macintosh was *not* designed to run multiple operating systems. The Macintosh ROM contained both low-level and high-level code. The low-level code was for hardware initialization, diagnostics, drivers, and so on. The higher-level *Toolbox* was a collection of software routines meant for use by applications, quite like a shared library. Toolbox functionality included the following.

- Management of dialog boxes, fonts, icons, pull-down menus, scroll bars, and windows
- Event handling

- Text entry and editing
- Arithmetic and logical operations

The Lisa-derived QuickDraw portion of the Toolbox contained highly optimized primitives for drawing shapes and user-interface elements. The use of common system-provided user-interface routines ensured a consistent and standard user interface. With time, the Toolbox would have an incredible amount of functionality (and associated APIs) packed into it, obstructing Apple's attempts to create a modern operating system while maintaining backwards compatibility. Figure 1–16 shows a screenshot of the first Macintosh operating system.

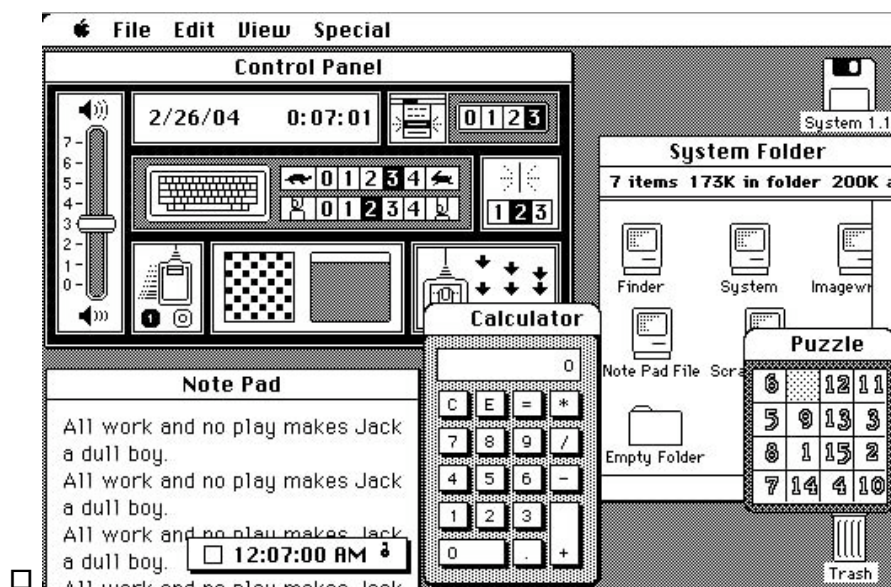


FIGURE 1–16 Macintosh System 1: the first Macintosh operating system

The *Finder* was the default application that ran as the system came up. It was an interface for browsing the file system and launching applications. Owing to the

single-tasking operating system, the user had to quit any running application to work in the Finder.

The Macintosh File System (MFS) was a *flat* file system: all files were stored in a single directory. However, the system software presented a hierarchical view that *showed* nested folders. Each disk contained a folder called *Empty Folder* at its root level. Renaming this folder created a new folder, with a replacement Empty Folder appearing as a side effect.

There was a menu-bar at the top, like in the case of Lisa, but with an Apple menu. There was also an iconic trash can that was automatically emptied every time the system booted. The Macintosh also heralded Apple's Human Interface Guidelines, which partially evolved from Lisa's user-interface standards.

An Icon Named Trash

The Macintosh trash can is sometimes criticized for being poorly designed, as it is not only meant to *destroy* files, but also for ejecting disks so that they can be *safely* put away. Apple's interface designers once explained the rationale behind this design.

Since the original Macintosh only had a single floppy disk drive, and no hard disk, it was expected that users would typically use several diskettes while working on the Macintosh. A convenience feature of the system was that it cached—in memory—the list of files on a diskette. The cache was retained even after the diskette had been ejected. A grayed-out Desktop icon for that diskette indicated this fact. Clicking on such an icon prompted the user to insert the appropriate diskette in the drive. Dragging a grayed-out icon to the trash freed up the memory used by that diskette's cache.

Thus, even if a user intended to permanently eject a diskette, two actions were required: the eject command, and dragging an icon to the trash. The redundancy was removed by combining these actions to a single action: dragging an “active” (non-grayed-out) icon to the trash caused the diskette to be ejected *and* its cache to be deleted.

At its introduction, the Macintosh was targeted for two primary markets: knowledge-workers and students. Referring to the telephone as the first “desktop appliance,” Steve Jobs hoped that the Macintosh would become the second desktop appliance. Bill Gates of Microsoft stated, “*To create a new standard takes something that’s not just a little bit different. It takes something that’s really new, and captures people’s imaginations. Macintosh meets that standard.*”

In 1984, Apple also introduced AppleTalk, a self-configuring, multilayered local area network (LAN) protocol whose features include dynamic addressing, router discovery, and name binding.

1.4. MANY SYSTEMS FOR MANY APPLES

After the Macintosh’s release, Apple spent the next few years improving the Macintosh operating system and creating some other noteworthy systems.

1.4.1. System Software Releases 2 - 6

For a long time, there were multiple, independent versioning schemes in effect for Macintosh system components: a System Software Release, a System Version, a Finder Version, a MultiFinder Version, a LaserWriter Version, and so on. Eventually there were attempts to unify these versions.

Notable improvements made during this time included the following.

- Continued speed improvements for the Finder, including a disk cache and an additional *minifinder* to make applications launch faster
- Commands for common tasks such as shutting down, creating new folders, and ejecting disks
- A hierarchical file system (HFS) that supported true hierarchy, allowing folders to be nested without illusory aid
- Support for multiple monitors

- Support for larger disk drives
- AppleShare client features

Figure 1–17 shows a screenshot of System 6.

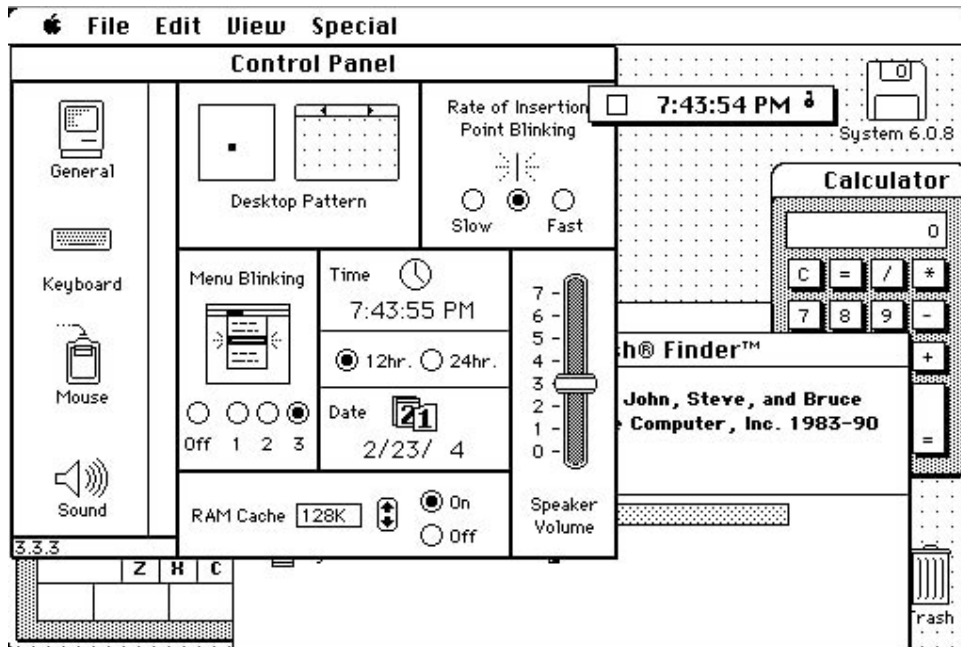


FIGURE 1–17 Macintosh System 6

An important improvement came when Apple incorporated cooperative multi-tasking through the *MultiFinder*. Initially included as a separate piece of software along with the original Finder, MultiFinder soon became mandatory. It allowed the user to have several programs open simultaneously and to assign specific amounts of RAM to these programs. Usability improvements included providing a progress bar with cancel button for “copy file” and “erase disk” operations. Until this point, the Finder did not use color even on color capable systems. This was remedied with the introduction of Color QuickDraw.

1.4.2. What Color is Your System?

In March 1988, after the Macintosh had been around for four years, some Apple engineers and managers had an off-site meeting. As they brainstormed to come up with future operating system strategies, they noted down their ideas on three sets of index cards that were colored blue, pink, and red.

Blue would be the project for improving the existing Macintosh operating system. It would eventually form the core of System 7.

Pink would soon become a revolutionary operating system project at Apple. The operating system was planned to be object-oriented. It would have full memory protection, multitasking with lightweight threads, a large number of protected address spaces, and several other modern features. After languishing for many years at Apple, *Pink* would move out to *Taligent*, a company jointly run by Apple and IBM. We will briefly discuss Taligent in Section 1.6.4.

Since the color red is “pinker than pink,” ideas considered too advanced even for *Pink* were made part of the *Red* project.

As the 1980s were drawing to an end, the system software was at major version 6. System 7, a result of the *Blue* project, would be Apple’s most significant system yet, both relatively and absolutely. However, that would not be until 1991. Apple would come out with two interesting operating systems before that: GS/OS and A/UX.

Gestalt

In 1989, Apple introduced a system call named “Gestalt” in version 6.0.4 of the operating system. Gestalt allowed applications to dynamically query the capabilities that were present in a running system configuration. It would go on to become a widely used system call, and continues to exist in Mac OS X as a Carbon function.

“Gestalt” is originally a German word that means *wholeness, shape, or form*. In one of its connotations, it is used to denote a structure or configuration integrated to form a functional unit in such a way that the properties of the whole are not derivable by summation of its parts.

1.4.3. GS/OS

As noted earlier, the Apple II had a rather long life span. After the release of the Macintosh in 1984, the Apple II still existed as a product. The Apple II_{GS} was introduced in 1986, almost as a bridge between the old and the new. It was the first and only 16-bit Apple II, and had impressive multimedia abilities (the “GS” stood for *graphics and sound*). Its notable features included the following.

- A 6502-compatible³⁷ 65C816 processor. The firmware-resident monitor allowed assembling and disassembling instructions for both processors.
- Support for 24-bit addressing, which allowed memory expansion up to 8 MB. The monitor could handle both 16-bit and 24-bit addresses.
- Two very high-resolution graphics modes: 320×200 with a 16-color palette and 640×200 with a 4-color palette.
- RGB and NTSC video outputs.
- A 32-voice Ensoniq Digital Oscillator chip that could be driven by firmware to produce up to 15 musical instruments.
- A mouse-driven, color desktop interface with windows and menus. A built-in control panel desk accessory allowed the user to set machine parameters for display, disk drives, processor speed, serial ports, and so on.
- Two standard serial ports that could be used with AppleTalk.

The Apple II_{GS} had several other additions or improvements over previous Apple II machines.

Apple ProDOS was forked into 8- and 16-bit versions to accommodate the Apple II_{GS}. After using ProDOS 16 as the computer’s operating system for a short

³⁷ The user could select either the 1 MHz processor clock speed of the 6502, or a faster 2.8 MHz.

time, Apple replaced it with GS/OS: a new 16-bit native-mode system that significantly improved performance on many fronts such as boot time, disk access time, and program launch time. Figure 1–18 shows a screenshot of GS/OS.

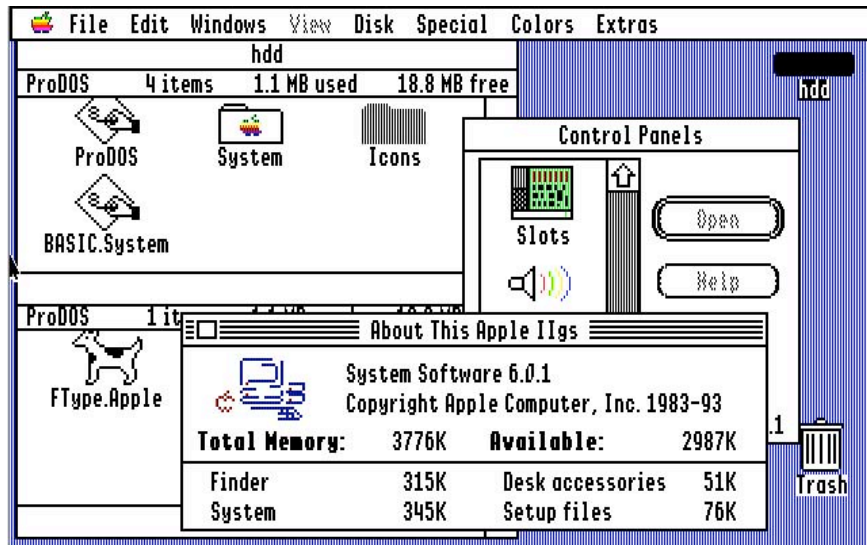


FIGURE 1–18 GS/OS

GS/OS had several modern features. It had the concept of *file system translators* (FSTs)—a generic file interface that allowed different file systems to be read from and written to. The concept was along similar lines as AT&T’s *file system switch*, Sun Microsystems’ *vnode/vfs*, and DEC’s *gnode* that were being introduced in the mid-1980s to allow multiple file systems to coexist. GS/OS eventually went on to have FSTs for various file systems: Macintosh HFS, ISO/High Sierra, AppleShare, and native file systems of Apple DOS 3.3, Apple Pascal, MS-DOS, and ProDOS. Since the AppleShare FST allowed GS/OS to access an AppleShare file server using AppleTalk networking, the GS/OS Finder could browse over the network. GS/OS could even be network booted.

The graphical control panel in GS/OS was a facility for controlling various system settings. Third-party developers could add their own control panel devices (CDEVs), thus extending the set of entities that the control panel could access.

The last version of GS/OS—4.02—shipped with Apple][gs System 6.0.1.

1.4.4. A/UX

A/UX was Apple's own version of POSIX compliant UNIX. It was released in late 1988. It only ran on 68K-based Apple machines³⁸ with both a floating-point unit (FPU) and a paged memory-management unit (PMMU). The earliest A/UX was based on 4.2BSD and AT&T UNIX System V Release 2, but it would later derive from 4.3BSD and various subsequent System V releases. A/UX features included the following.

- 4.3BSD extensions such as groups, signals, and job control
- System V IPC (semaphores, messages) and System V-style signals
- Networking (AppleTalk, STREAMS, TCP/IP over Ethernet as well as over a serial connection, sockets, domains, subnets, NFS with Yellow Pages, and so on)
- The Berkeley Fast File System (ffs)
- Multiple Unix command interpreters such as the Bourne, Korn, and C shells
- A comprehensive set of development tools such as lint, lex, yacc, debuggers (adb, sdb, MacsBug), various editors, assembler, linker, C compiler (cc), FORTRAN compiler (f77), make, and SCCS
- The X Window System
- AppleTalk printing and file sharing client services via LocalTalk or EtherTalk

³⁸ For example, A/UX ran on a Macintosh SE/30.

Besides POSIX compatibility, A/UX supported the BSD and System V APIs. In particular, A/UX was compliant with the System V Interface Definition (SVID), and passed the System V Verification Suite (SVVS).

Figure 1–19 shows a screenshot of A/UX.

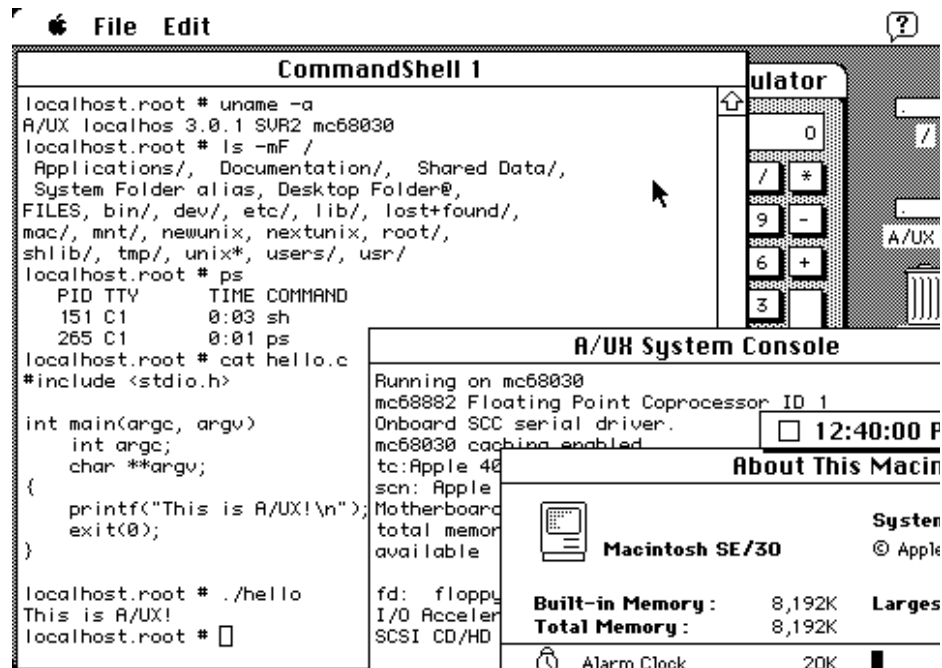


FIGURE 1–19 The A/UX operating system

More interestingly, A/UX combined various features of the Macintosh operating system with Unix. A/UX 2.x used Macintosh System 6, whereas A/UX 3.x combined a Unix environment with System 7. The default user environment consisted of the Macintosh Finder, which was essentially a graphical shell hosted by A/UX. The A/UX file system appeared as a disk drive icon in the Finder. Files could be accessed and manipulated through their icons. It was possible to run Macintosh

applications, Unix command-line or X Window applications, and even MS-DOS applications³⁹ side-by-side. The *MacX* display server allowed X Window System client applications to be displayed on the A/UX Desktop.

The OSF/Motif toolkit for the X Window system was available as a third-party addition. A/UX supported hybrid applications that made use of both the Unix system call interface and the Macintosh Toolbox. All types of applications could be launched from within the Finder.

Whereas Unix processes ran with preemptive multitasking under A/UX, the Macintosh Finder (the MultiFinder specifically) still supported only cooperative multitasking. When 32-bit addressing was in effect, the `startmac` application was responsible for creating the Macintosh environment under A/UX. The `startmac24` variant was used with 24-bit addressing. Many aspects of this environment were customizable, including which application to run as the Finder. Macintosh-style menu-driven login, logout, system startup, and shutdown procedures were supported. A conceptual diagram of the A/UX architecture is shown in Figure 1–20.

Many proponents of A/UX regarded it as the holy grail of Unix systems. Given compatible hardware, the A/UX installation procedure was incredibly simple for a Unix system, being essentially one-click.

From trivialities such as similar commands (such as `appleping`) to the more elaborate marriage of the Unix and Macintosh environments, vestiges of insights gained through A/UX can be seen in Mac OS X. Examples include the following.

- A/UX used `/.mac/<host>/Desktop Folder/` and `/.mac/<host>/Trash/` as the Unix pathnames for directories containing items visible on the Macintosh Desktop and in the trash can, respectively.
- Unix uses the `'/'` character to separate path components, whereas Macintosh file systems use `':'`. Invisible translation was done while accessing or moving files from one environment to the other.
- Home directories for user accounts were located in `/users/`.

³⁹ Running MS-DOS applications required the SoftPC product.

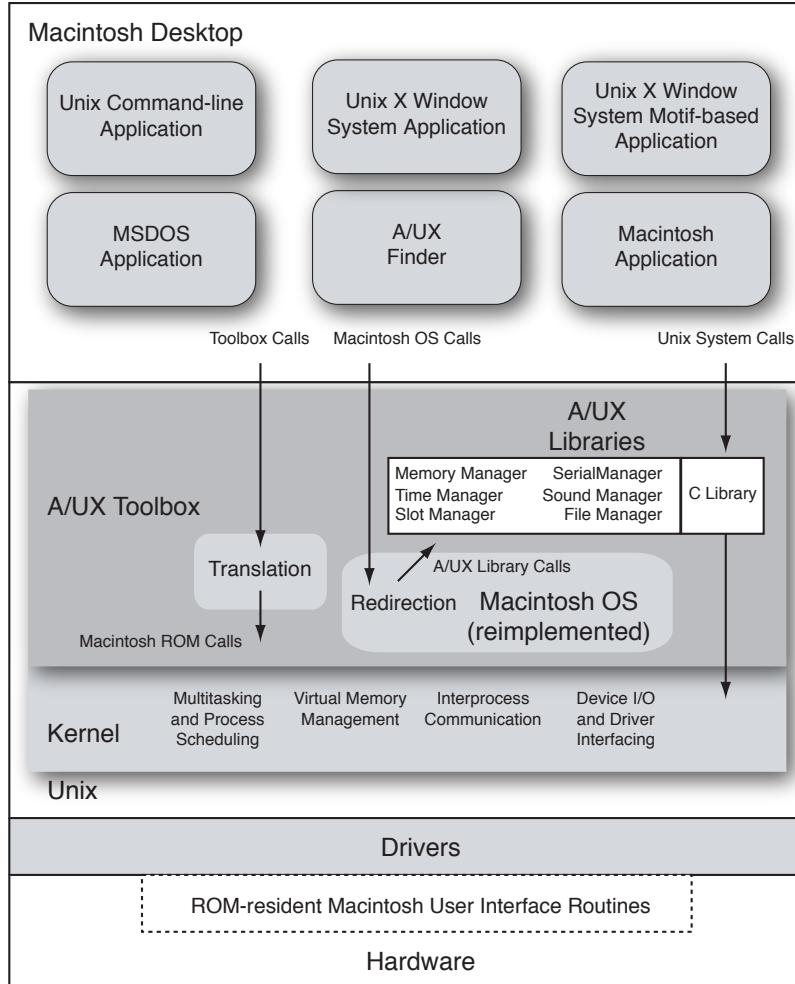


FIGURE 1-20 A/UX architecture

The last version of A/UX—3.1.1—was released in 1995.

1.5. SEEKING POWER

As the 1990s began, Apple was making great efforts to overhaul its operating system. Of the three “colorful” projects mentioned earlier, Blue, would emerge as System 7.

1.5.1. System 7

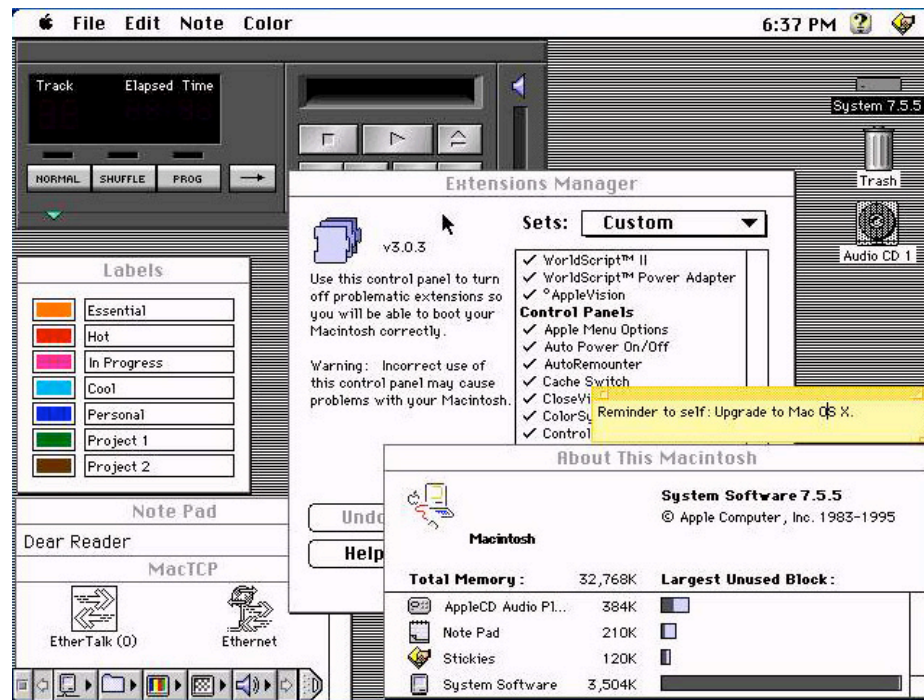
When released in 1991, System 7 represented a gigantic leap forward in comparison to earlier Macintosh systems. Some of its key features included a built-in Multi-Finder, built-in networking (AppleTalk Phase 2⁴⁰), built-in file sharing (AppleShare), support for 32-bit memory addressing, a virtual memory implementation, and a multitude of new technologies⁴¹ such as the following.

- *AppleScript*—a system-level macro language for automating tasks
- *ColorSync*—a color management system
- *PowerTalk*—a collaboration and email software
- *QuickTime*—a cross-platform multimedia software for viewing and manipulating video, animations, images, and audio
- *TrueType*—a font technology
- *WorldScript*—a multilingual text-rendering engine and programming interface

The first Macintosh with an MMU was introduced in 1987: the Macintosh II. Whereas A/UX required an MMU, it was not until System 7 that a Macintosh operating system would use the MMU. However, the virtual memory support was only preliminary—features such as protected address spaces, memory-mapped files, page locking, and shared memory, were not present.

⁴⁰ AppleTalk Phase 2 was introduced in 1989. It was based on the original version of AppleTalk, but included several improvements and performance enhancements.

⁴¹ Some of these technologies were not bundled with the first release of System 7.

**FIGURE 1–21** System 7

System 7 also had several usability improvements such as the following.

- Users could view, and switch between, running applications via a menu.
- The trash can now had the same status as any other folder. It now had to be proactively emptied instead of the system automatically emptying it.
- In addition to cut-and-paste, text-selections could be dragged between applications.
- Aliases could be created for access to documents and applications from two or more locations.
- Control Panels and Extensions were organized hierarchically on disk.

Certain machines such as the Macintosh II, IIx, IIcx, and the SE/30 could

have 32-bit support and a larger virtual memory capability through a 32-bit “system enabler” program (called MODE32) on System 7. The standard ROMs of these machines were not 32-bit clean, and therefore were only compatible with 24-bit addressing. MODE32 allowed selecting and changing between 24-bit and 32-bit addressing modes. With 32-bit addressing, it was possible to use more than 8 MB of contiguous physical memory. With virtual memory, it was possible to use hard disk space as swap space to run programs.

Even with the aforementioned improvements, System 7 only performed cooperative multitasking, and lacked memory protection.

Around this time, Apple formed an alliance with IBM and Motorola—a move that put the PowerPC on Apple's hardware roadmap. The advent of the PowerPC required fundamental changes in the design of the Macintosh operating system.

1.5.2. AIM for POWER

The emphasis on making the semantics of computer architecture close to those of higher level programming environments had led to very complex processors. However, people like Seymour Cray understood the benefits of simplicity in computer architecture design even in the early 1960s. Cray's CDC 6600 supercomputer and the CRAY-1 computer were both RISC machines, although the term “RISC” had not been coined yet.

1.5.2.1. A RISCy Look Back

RISC stands for *Reduced Instruction Set Computer*, although it does not only imply fewer instructions. RISC architectures are predominantly load-store and register-centric, usually employ fixed-format instructions, have efficient pipelining, require relatively fewer clock cycles per instruction, and so on.

The line between RISC and CISC has been growing fuzzier over the years,

particularly as focus of microprocessor companies has shifted to micro-architecture. It is common to see companies attempting to optimize superscalar, out-of-order execution. For example, Intel's Pentium Pro (1995) translated complex x86 instructions into RISC-like micro-operations during instruction decoding. A superscalar engine executed these micro-operations in a speculative, out-of-order fashion, with register renaming. Intel's P6 architecture had various other RISC-like features.

The 801 Minicomputer Project, so called after the name of the building it was housed in, started at IBM in 1975. John Cocke, who is regarded as the “father of RISC architecture,” and others explored a simplified instruction set along with compiler code-generation strategies for improving performance and reducing cost. An early RISC tenet was an instruction-processing rate of 1 per clock-cycle. The 801 *was* able to achieve a clock-cycles-per-instruction (CPI) ratio of 1 for contrived code, but not for general-purpose code. It had 120 instructions. As instruction pipelining and cache memories evolved and improved, it became increasingly possible to meet the goal of having a CPI of 1.

IBM's first RISC-based product was the IBM Personal Computer RT (RISC Technology), which was announced in January 1986. It ran an operating system called the Advanced Interactive Executive (AIX). Further work led to a new design called “AMERICA,” which led to “RIOS,” and eventually the “POWER” architecture. POWER is a contraction for “Performance Optimized With Enhanced RISC.” It combined the original RISC concepts with some traditional (CISC) concepts resulting in a more balanced architecture, representing the second generation of IBM's RISC technology. The associated product family included the RISC System/6000 (RS/6000), along with the AIX 3 operating system. The first RS/6000 systems were announced on February 15, 1990.

The POWER architecture defined 184 instructions, which, although a few too many from a RISC purist's perspective, performed well with the independent execution units available in the RS/6000, as multiple instructions could be executed in a

single cycle. As the POWER architecture evolved, the earliest version of the architecture came to be known as POWER1. Even in the POWER1 era, there were multiple implementations of POWER, such as the low-end RISC Single Chip (RSC), the mid-end RS .9, and the high-end RS 1.0. The RSC had a shared data and instruction cache. It was a low-cost shrinkage of POWER onto a single chip, whereas the others were multiple-chip. The lowest-end RS/6000 model—the 33 MHz Model 220—was released in January 1992.

In the early 1980s, Berkeley and Stanford Universities were working on the RISC and MIPS projects, respectively. By 1990, there were several competing RISC architectures in the market: MIPS, HP Precision Architecture (PA-RISC), SPARC V8, Motorola 88K, and IBM RS/6000. The Intel i860 was introduced in 1989 as a general-purpose, 64-bit RISC processor with 3D graphics capabilities. The Alpha AXP from Digital Equipment Corporation joined the RISC crowd in 1992 as another 64-bit RISC processor.

1.5.2.2. Apple Wants RISC

As part of a project code-named Jaguar, Apple had briefly considered using a Motorola 88K variant as their future RISC-based hardware platform. They turned to the POWER architecture next.

In 1991 Apple, IBM, and Motorola joined forces to form the “AIM” Alliance with the goal of creating a Common Hardware Reference Platform (CHRP). The collaboration resulted in the PowerPC Architecture—a derivative of POWER. PowerPC included most of the POWER instructions, while adding some new ones and excluding some rarely used instructions. Important PowerPC improvements included the following.

- It supported both 32-bit and 64-bit computing, with an implementation being free to only implement the 32-bit subset. An implementation supporting both would be able to dynamically switch between them.

- It had a cleaner and simplified superscalar design.
- It had a cleaner separation between architecture and implementation. The resulting architecture was flexible enough to permit a broad range of implementations.
- It supported symmetric multiprocessor (SMP) systems.

CHRP was also aimed at companies other than Apple so that they could sell PowerPC-based systems. Microsoft Windows NT 3.51 and 4.0 ran on PowerPC Reference Platform (PReP) compliant systems, until Microsoft announced in early 1997 that it would phase out NT development on the PowerPC architecture. A version of Solaris (2.5.1) was also released as Solaris PowerPC Edition.

The first PowerPC processor was the 601. It was jointly developed by the AIM companies at the Somerset Design Center in the Northwest Hills of Austin, Texas. When introduced in September 1993, the 601 ran at 66 MHz. It implemented the 32-bit subset of the PowerPC Architecture, but without the full PowerPC instruction set. It was regarded as a bridge that would allow vendors to transition to PowerPC. IBM's RS/6000 Model 250 Workstation was the first PowerPC-based system to ship.

From PowerPC to x86

After Apple adopted the PowerPC, it remained the mainstay of Apple hardware for over a decade—until the year 2005, specifically. Steve Jobs announced in his Worldwide Developer Conference 2005 keynote that Apple had decided to transition from PowerPC to the x86 platform within the next two years. Apple also announced its partnership with Intel to facilitate the transition. Mac OS X was demonstrated to be already running on x86 hardware during the keynote.

Apple has used many PowerPC generations over the years. Until Apple introduced the Power Macintosh “G5” in June 2003, all PowerPC processors used by

Apple had been 32-bit implementations. The first 64-bit G5 chips—namely, the PowerPC 970 and 970FX—are very similar to the POWER4 chips since the G5 is based on the POWER4 architecture. A key difference is that the G5 contains the VMX⁴² vector-processing unit (VPU). Moreover, a single POWER4 chip has two processor cores, whereas the 970 and the 970FX only have one each. Apple added the dual-core 970MP processor to its PowerPC offerings in the second half of 2005.

1.5.2.3. Apple Likes RISC: ARM

As Apple was turning to RISC computing in the early 1990s, it collaborated with other partners besides IBM. In the mid 1980s, the Cambridge-based British company Acorn Computer Group⁴³ had developed the world's first commercial RISC processor. At that time, ARM stood for "Acorn RISC Machine." The first version of the ARM architecture (ARMv1) had 26-bit addressing, with no onboard multiplier or coprocessor. The first ARMv1 processor, the ARM1, saw limited use in prototypes of the *Archimedes* workstation and as a low-cost secondary component in the BBC microcomputer. It is noteworthy that ARM eschewed some key features of the prevailing Berkeley RISC architecture: delayed branching, register windows, and requiring all instructions to execute in a single cycle each.

Even before the AIM alliance was formed, Apple had teamed with Acorn to fund a new company called Advanced RISC Machines (ARM) Limited. The company's goal was to create a new RISC microprocessor standard. VLSI Technology was an investor and technology partner in this endeavor. It was also ARM's first licensee. ARM Limited's first processor was an embeddable RISC core called the ARM6. Based on version 3 of the ARM processor architecture (ARMv3), the ARM6 had full 32-bit code and data addressing. An ARM6 processor—a 20 MHz

⁴² VMX is the same as AltiVec, which is a Motorola trademark.

⁴³ One of Acorn's best-known computers was the BBC micro, which, at its launch in 1981, was based on a 2 MHz 6502 processor.

610—was used in Apple’s MessagePad hand-held that ran the Newton operating system⁴⁴.

1.5.3. Mac OS for PowerPC

System 7.1.2 was the first Apple operating system to run on the PowerPC, even though much of the code was not PowerPC-native. Given Apple’s roadmap, porting all components of the operating system to a new architecture would have taken prohibitively long. Moreover, it was extremely important for Apple to provide a way so that 68K-based applications would continue to run even on the PowerPC platform. The system architecture devised to address these issues included a hardware abstraction layer (HAL) and a 68K emulator.

A *nanokernel*⁴⁵ was used to “drive” the PowerPC. Executing in supervisor mode, the nanokernel acted as the HAL. It exported low-level interfaces for interrupt management, exception handling, memory management, and so on. Only the system software and possibly debuggers could use the nanokernel API.

The 68K emulator was initialized by the nanokernel during boot time. It only emulated a 68LC040⁴⁶ user-mode instruction set, without emulating the paged MMU (PMMU) or the FPU. Exception stack frames were created as per a slightly older processor (the 68020) for better compatibility. There were other caveats and limitations as compared to a real 68LC040. In particular, since A/UX used the PMMU directly, it did not run on this emulator.

Since two instruction-set architectures were simultaneously in effect, a system level component called the *Mixed Mode Manager* was used to handle context

⁴⁴ Keywords often associated with ARM processors include “embedded,” “high-performance,” “low-cost,” “power-efficient,” and “RISC.” Such features are especially appealing for low-power devices.

⁴⁵ A term sometimes used to refer to a kernel that is even smaller than a microkernel.

⁴⁶ The 68LC040 was a low-cost version of the 68040—it lacked a floating-point unit.

switches between the two types of code. Code pieces belonging to the two architectures could also call each other. The manager was transparent to 68K code, but PowerPC code *was* aware of it.

The initial PowerPC ports of Mac OS had little native PowerPC code—most existing applications, device drivers, system extensions, large parts of the Toolbox, and the operating system itself were not PowerPC-native. In fact, even though Apple's PowerPC introduction occurred in 1994, most of the operating system was still 68K-based in 1996. It would not be until 1998 that Mac OS was mostly PowerPC-native. Figure 1–22 is a conceptual depiction of part of the transition to PowerPC.

It was hoped that the nanokernel would form the basis for future Apple systems, perhaps as a robust microkernel.

1.5.4. MAE

A Mixed Mode Manager was also used in Apple's *Macintosh Application Environment* (MAE) product. MAE was an X Window application available for Sun and Hewlett-Packard workstations. It ran on SunOS (SPARC) and HP-UX (HP9000/700), providing an emulated Macintosh environment. It also contained a 68K emulator—for example, MAE 3 had a 68LC040 emulator. The Mixed Mode Manager was used to optimize execution by translating 68K code to native instructions if possible.

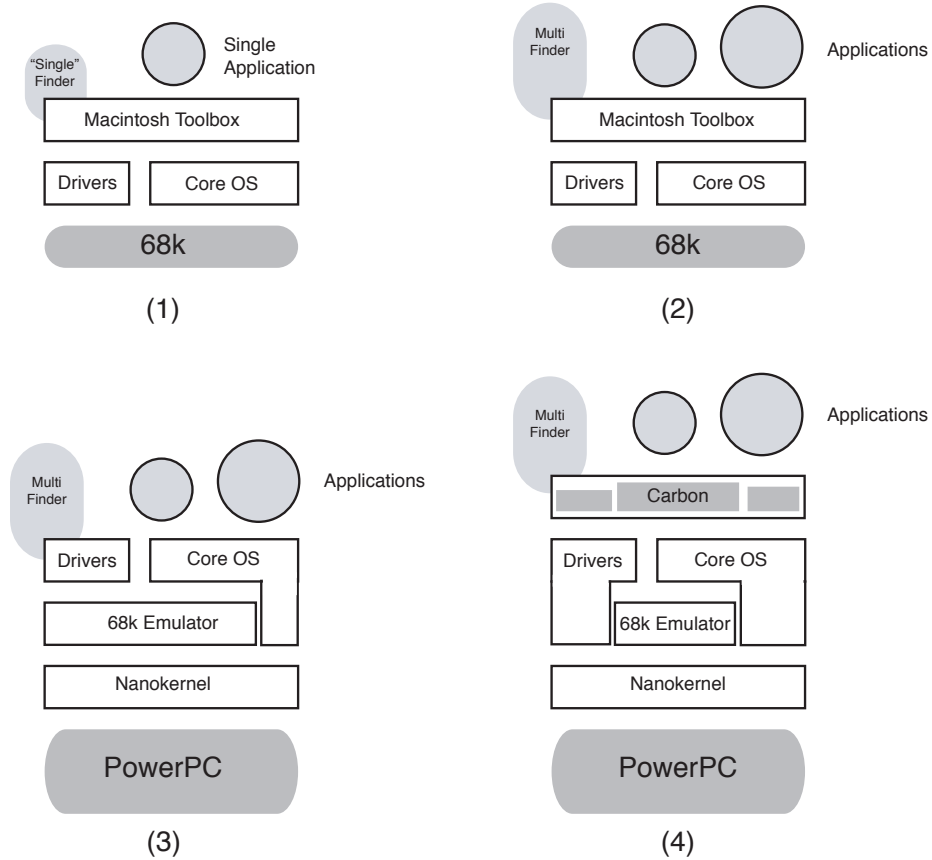


FIGURE 1-22 Transitioning to the PowerPC

MAE provided an emulated System 7. Depending on the MAE version, it supported features such as aliases, AppleEvents, AppleTalk, NFS, QuickDraw, QuickTime, and TrueType. MAE ran as a user-space process called `mae`, along with an auxiliary daemon process called `maed`, which was the Macintosh environment daemon. Multiple copies of MAE could run simultaneously on a workstation within the constraints of available resources.

MAE contained an implementation of the Macintosh Process Manager, which was responsible for loading and unloading all Macintosh applications used from within MAE. When the `mae` process started, it reserved a memory region for use by the Process Manager and the virtual Macintosh system. The default size of this region was 16MB, although the user could configure it based on resources available on the host, the number of MAE instances that could be launched, and the amount of “physical memory” desired in the virtual environment. Even though MAE itself ran on a UNIX system, applications within MAE ran with cooperative multitasking—as they would on a real Macintosh. Moreover, there was no memory protection. MAE had several behavioral similarities to A/UX. It made UNIX file systems visible as Mac OS volumes. UNIX symbolic links were represented as aliases. A MAE user could launch UNIX commands from the Finder.

Since MAE had neither a real Macintosh-compatible graphics card nor a real monitor, its graphics component had to provide an illusion to the Macintosh software running within it. It implemented a virtual Macintosh monitor as a framebuffer, which was updated as a result of applications making QuickDraw calls, with QuickDraw, in turn, notifying MAE of the parts of the bitmap that had changed. In early versions of MAE, the entire framebuffer was passed to the X Window system. In version 3.0 of MAE, updates to the framebuffer were translated to Xlib calls that were sent to the X server.

MAE included the following primary components.

- The main executable (`apple/bin/mae`), the auxiliary daemon, other native UNIX code, and related data
- The MAE *engine* (`apple/lib/engine`), which included code for the Mac OS ROM (Toolbox), 680x0 “glue” code, and related data
- Mac OS system code, application code, and data
- The MAE X Window graphics buffer

The glue code mapped Mac OS operations to UNIX system calls. The MAE engine and associated data were mapped into memory using the `mmap()` system call. If the host platform supported copy-on-write, substantial portions of MAE could be shared between its multiple instances.

MAE made heavy use of processor caches, performing best on machines with large caches. It cached both instructions and data. Examples of instruction cache use by MAE included the following.

- Native code to support the 680x0 glue and dynamic compilation
- Native code to emulate instructions
- Dynamically compiled code to emulate instructions

Examples of data cache use by MAE included the following.

- Native data to support the 680x0 glue and dynamic compilation
- Dispatch table⁴⁷ for mapping 680x0 opcodes to native emulation code
- 680x0 instructions emulated by MAE
- 680x0 data for Mac OS system code and the Toolbox
- Graphics framebuffer

MAE was discontinued on May 14, 1998.

MAS

Apple had announced another emulation-based Mac-on-Unix solution called Macintosh Application Services (MAS). Unlike MAE, MAS supported both 680x0-based and PowerPC-based Macintosh software on PowerPC-based Unix systems such as AIX.

⁴⁷ The dispatch table was indexed by the 16-bit 680x0 opcode. It was perhaps the most aggressive cache user among all of MAE's components.

Meanwhile, Apple continued to develop System 7, adding features and improving existing features. The Internet was burgeoning, rapidly making networking capabilities an essential component of even end-user operating systems. System 7.5 and later included *OpenTransport*, a networking subsystem based on Mentat Portable STREAMS.

Other noteworthy features added to Systems 7.5.x included *QuickDraw 3D*, a Java runtime, and the *OpenDoc* component software architecture that could be likened to Microsoft's Object Linking and Embedding (OLE). Visual enhancements such as a startup screen with a Mac OS logo and a progress bar were also added around this time.

In the 1990s, Apple also made some forays into server computing. Let us briefly look at some of these endeavors.

1.5.5. Apple Workgroup Server

Apple Workgroup Server was a line of servers intended for use as file, print, and database servers for workgroups. These systems were initially 68K-based, but PowerPC-based systems were also available eventually. The following are some examples of systems from this line.

- **WGS 60** (50/25 MHz Motorola 68040) used System 7.1, ran AppleShare 4.0, and was targeted at small businesses and classroom lab environments.
- **WGS 80** (66 MHz Motorola 68040) used System 7.1, ran AppleShare 4.0, and was targeted at medium-sized business environments where it could act as a communications server – for example, as an Internet router and a SNA*ps, X.25, or X.400 gateway.
- **WGS 95** (66 MHz Motorola 68040) used A/UX 3.1 as the server operating system, ran AppleShare Pro, and was targeted at large or data-intensive workgroups—for example, as a high-performance AppleShare file and print server. It was also intended to run relational database (RDBMS) products such as Oracle7 Cooperative Server.

- **WGS 9650** (233 MHz PowerPC 604e) ran Mac OS 7.6.1.

The 60, 80, and 95 were introduced in 1993, whereas the 9650 was introduced in 1997.

1.5.6. NetWare for PowerPC

Apple partnered with Novell in the mid 1990s to port NetWare to the PowerPC. The jointly funded port was designed to be much easier to configure than NetWare for x86. It was intended to run on an Apple-designed server called *Shiner*.

Although the port was more or less completed, the project was killed before it ever shipped. *Shiner* would lead to the *Apple Network Server*, which ran AIX.

1.5.7. AIX for PowerPC

A few years later—in 1996—Apple had a short-lived product called Apple Network Server. The PowerPC-based server had hardware features such as the following.

- Processor resident on a daughterboard for easy upgradeability
- An easy-to-replace logic board
- Up to 1MB of L2 cache
- Up to 512MB of parity RAM
- A 4MB ROM and an 8KB NVRAM
- Multiple hot-swappable and RAID-capable drive bays, for a total approximate disk capacity of 340GB
- Hot-swappable power supplies and fans
- Provisions for adding several SCSI devices and PCI cards
- An external LCD display for system diagnostics and status messages
- Several physical security features

The Network Server came with the *AIX for Apple Network Servers* operating system, which was based on IBM's AIX. The server did *not* support Mac OS. With AIX, Apple had an advanced operating system with features such as the following.

- Memory protection
- Preemptive multitasking
- Multithreading
- Support for various networking protocols including AppleTalk and AppleTalk services
- RAID
- Journal File System (JFS)
- Logical Volume Manager (LVM), which supported multiple file systems up to 256GB in size, and could handle files up to 2GB in size

The user could work with the command-line interface, or could have one of the *AIXwindows* or *Common Desktop Environment* (CDE) graphical interfaces. The operating system included several applications to ease administrative tasks. A menu-driven *System Management Interface Tool* (SMIT) was used for configuration, installation, maintenance, and troubleshooting. A graphical *Visual System Management* (VSM) interface allowed system tasks to be performed by clicking on icons.

Apple revised the Network Server line shortly after its introduction, but discontinued it in 1997. The next serious server offering from Apple would not be until 2002, when the Xserve would be released.

1.6. QUEST FOR *THE*⁴⁸ OPERATING SYSTEM



FIGURE 1–20 Microsoft Windows NT 3.1

Microsoft’s Windows 3.x had been extremely successful since its release in 1990. Microsoft had been working on a new operating system code-named “Chicago.” Initially slated for 1993 release, “Chicago” kept slipping. It would be eventually be released as Windows 95. Microsoft did, however, release Windows NT in 1993. NT was an advanced operating system meant for high-end client-server applications. It had various important features such as symmetric multiprocessing support, a pre-

⁴⁸ Whereas the word “the” is used here to designate prominence and desirability, it is an interesting coincidence that “THE” was the name of a multiprogramming system described by Edsger W. Dijkstra in a 1968 paper.

emptive scheduler, integrated networking, subsystems for OS/2 and POSIX, virtual machines for DOS and 16-bit Windows, a new file system called NTFS, and support for the Win32 API.

Apple needed an answer to Microsoft's onslaught, particularly in the face of the upcoming Windows 95, which was to be an end-user operating system.

The Pink and the Red projects would turn out to be rather unsuccessful. Apple would continue to make attempts to solve the "OS problem" one way or another.

1.6.1. Star Trek

Star Trek was a bold project that Apple ran jointly with Novell to port Mac OS to run on the x86 platform. A team consisting of engineers from both Apple and Novell actually succeeded in creating a very reasonable prototype in an incredibly short amount of time. The project was cancelled, however, for various reasons: Apple had already committed to the PowerPC; many within and outside of Apple thought that doing so would disrupt Apple's existing business model; and vendor feedback was not encouraging.

Many years later, *Darwin*—the core of Apple's far more successful Mac OS X—would run on both the PowerPC and the x86. Whereas the *Star Trek* prototype showed the "Happy Mac" logo while booting up, Darwin/x86 prints the message "*Welcome to Macintosh*" during boot.

Star Trek was finally vindicated with Apple's mid-2005 announcement of transitioning Mac OS X to the x86 platform. The first x86-based Macintosh computers—the iMac and the MacBook Pro (the successor to the PowerBook)—were unveiled at the San Francisco Macworld Conference & Expo in January 2006.

1.6.2. Raptor

Raptor was in many respects the Red project. It was supposed to provide Apple with a next-generation microkernel that would run on any architecture. As the Star Trek project was being cancelled, it was considered for absorption by Raptor, which itself would die due to budgetary limitations and employee attrition, among other reasons.

1.6.3. NuKernel

NuKernel was a kernel project at Apple that was meant to result in a modern operating system kernel on more than one occasion.

1.6.4. TalOS

Apple and IBM formed a company called *Taligent* in early 1992 to continue work on the Pink project. Pink originally aimed to be an object-oriented operating system, but later morphed into an object-oriented environment called *CommonPoint* that ran on many modern operating systems such as AIX, HP-UX, OS/2, Windows 95, and Windows NT. It was also meant to run on Apple's NuKernel. *Taligent Object Services* (TalOS) was the name given to a set of lower-level technologies that were to be built around Mach 3.0. TalOS was meant to be an extensible and portable operating system, with a small footprint and good performance.

TalOS was object-oriented from the kernel up, with even device drivers and network protocols implemented in an object-oriented fashion. Taligent's object-oriented libraries were known as *frameworks*. There were frameworks for user interfaces, text, documents, graphics, multimedia, fonts, printing, and low-level services such as drivers. These, along with the TalOS development tools, explicitly

strived to shift the burden of programming from application developers to application system engineers.

Note that even though there existed other commercial systems such as NEXTSTEP that had object-oriented *application frameworks*, Taligent aimed to build its entire programming model around objects. In NEXTSTEP, the developers who created frameworks had to map object behavior to the underlying libraries, Unix system calls, Display PostScript, and so on—all of which had procedural APIs. In contrast, Taligent's CommonPoint applications were not meant to use the host OS APIs *at all*.

In 1995, Taligent became a wholly owned subsidiary of IBM. The Pink project did not give Apple the next-generation operating system that Apple had been seeking.

1.6.5. Copland

Apple made an announcement in early 1994 that it would channel more than a decade of experience into the next major release of the Macintosh operating system, Mac OS 8. The project was codenamed "Copland." It was expected that Copland would be Apple's *real* response to Microsoft Windows. With Copland, Apple hoped to achieve several goals, many of which had been long elusive, such as the following:

- Adopt RISC as a key foundation technology by making the system fully PowerPC-native.
- Integrate, improve, and leverage existing Apple technologies such as ColorSync, OpenDoc, PowerShare, PowerTalk, QuickDraw 3D, and QuickDraw GX.
- Retain and improve the ease-of-use of the Mac OS interface, while making it multiuser and fully customizable. In particular, Copland's implementation of themes allowed customization of most user-interface elements on a per-user basis.

- Extend interoperability with DOS and Windows.
- Make Mac OS systems the best network clients.
- Incorporate active assistance that works across applications and networks—that is, make it very easy to automate a wide variety of tasks.
- Release Copland as a system that may be openly licensed to foster development of Mac OS compatible clones by third parties.

To achieve these goals, Copland was supposed to have a comprehensive set of system-level features, for example:

- A hardware abstraction layer (HAL) that would also help vendors in creating compatible systems
- A microkernel (the NuKernel) at its core
- Symmetric multiprocessing with preemptive multitasking
- Improved virtual memory with memory protection
- A flexible and powerful system extension mechanism
- Critical subsystems such as I/O, networking, and file systems running as services on top of the kernel
- Built-in low-level networking facilities such as X/Open Transport Interface (OTI), System V STREAMS, and Data Link Provider Interface (DLPI)
- File searching based on both metadata and content
- The ability to perform “live upgrades” on a system without affecting the performance of other running programs

Work on Copland gained momentum during the early 1990s, and by the mid 1990s, Copland was heavily counted on to do wonders for Apple. Apple dubbed it as *“The Mac OS Foundation for the Next Generation of Personal Computers.”* However, the project kept slipping. A few prototypical Driver Development Kit (DDK) releases went out, but a 1996 release, as had been planned and hoped, did not seem feasible. Due to numerous pressures, full memory protection had not been included after all. Apple’s CEO Gil Amelio described the state of Copland as *“... just a collection of separate pieces, each being worked on by a different team... that were expected to magically come together somehow...”*

A conceptual view of Copland is shown in Figure 1–24.

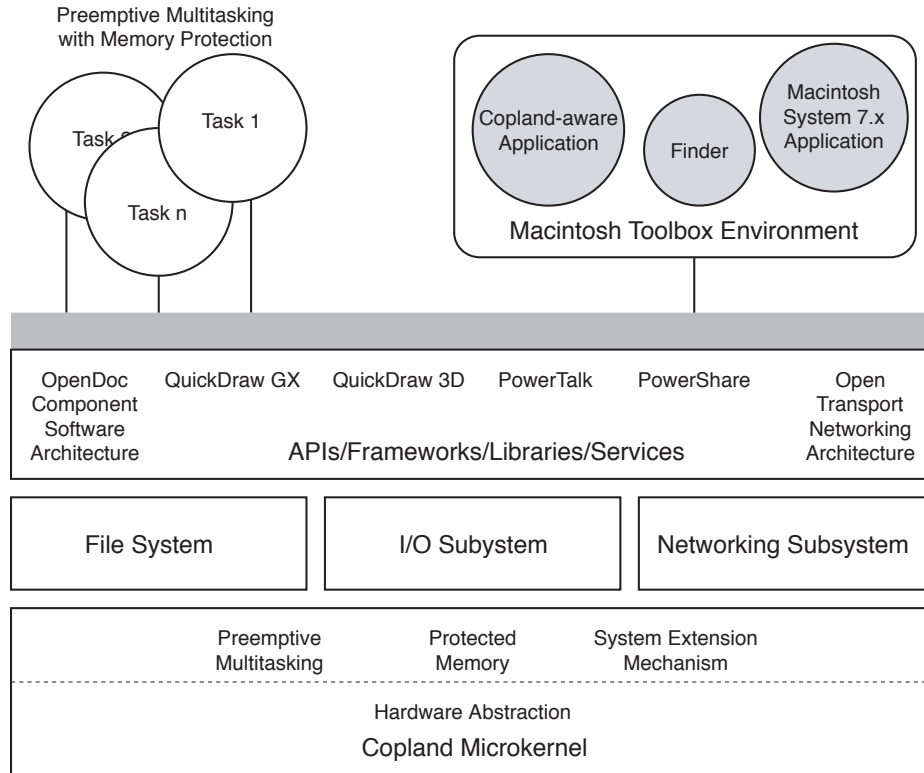


FIGURE 1–24 Copland architecture

Apple eventually decided to cancel Copland in May 1996. Amelio announced that Copland's best pieces would be shipped with future releases of their existing system, beginning with the upcoming System 7.6, whose name was formally changed to *Mac OS 7.6*.

1.6.6. Gershwin

After the Copland debacle, Apple's need for a new operating system was direr than ever. Focus shifted briefly to a project named *Gershwin*, which was to include the painfully elusive memory protection, among other things. However, it was apparently nothing more than a codename, and it is believed that nobody ever worked on Gershwin.

1.6.7. BeOS

Apple briefly considered partnering with Microsoft to create an Apple OS based on Windows NT. Other systems under consideration were Solaris from Sun Microsystems and BeOS from Be. In fact, Apple's acquisition of Be came rather close to materializing.

Be was founded in 1990 by Jean-Louis Gassée, Apple's former head of Product Development. Be's capable engineering team had created an impressive operating system in BeOS. It had memory protection, preemptive multitasking, and symmetric multiprocessing. It even ran on the PowerPC,⁴⁹ thus fitting with Apple's hardware agenda. BeOS was designed to be especially adept at handling multimedia. It had a metadata-rich file system called BeFS that allowed files to be accessed via multiple attributes. However, BeOS was still an unfinished and unproven product. For example, it did not yet support file sharing or printing, and only a few applications had been written for it. Figure 1–25 shows a screenshot of BeOS.

⁴⁹ BeOS initially ran on Be's own PowerPC-based machine called the *BeBox*. It was later ported to the x86 platform.

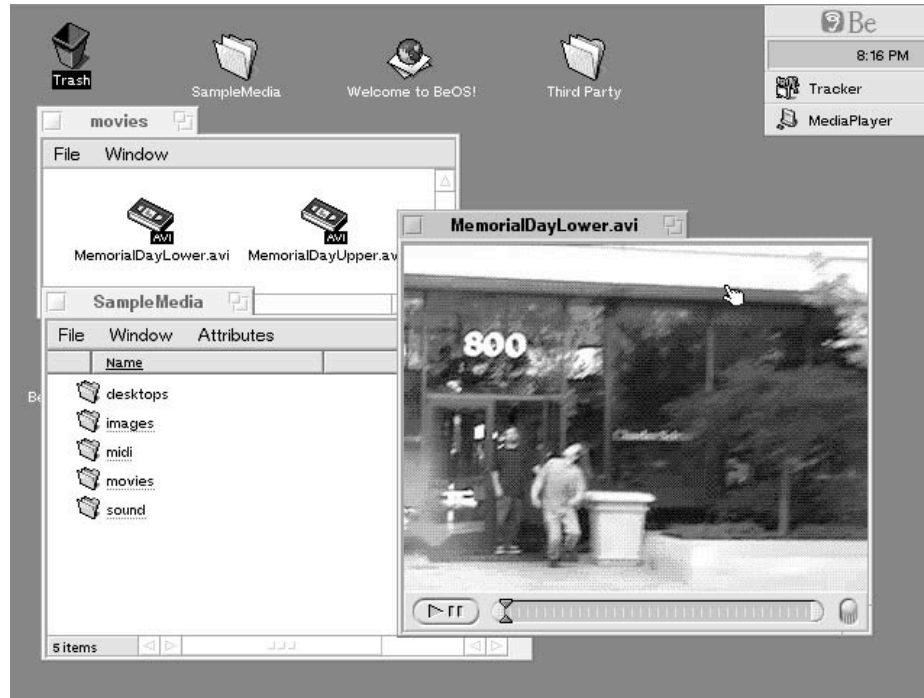


FIGURE 1–25 BeOS

Gassée and Apple negotiated back and forth over Be's acquisition. The total investment in Be at that time was estimated to be about \$20 million, and Apple valued Be at \$50 million. Gassée sought over \$500 million, being confident that Apple would buy Be. Apple negotiated up to \$125 million, and Be negotiated down to \$300 million. When things still did not work out, Apple offered \$200 million, and even though it is rumored that Gassée was actually willing to accept this offer, it is also said that he came back with a "final price" of \$275 million, hoping Apple would bite the bullet. The deal did not happen. In any case, Be had a tough contender in NeXT, a company founded and run by another one-time Apple employee: *Steve Jobs*.

Be would eventually fail as a company—its technological assets were acquired by Palm, Inc. in 2001.

1.6.8. Plan A

Unlike Be, NeXT's operating systems had at least been proven in the market, despite NeXT not having any resounding successes. In particular, OPENSTEP had been well received in the enterprise market. Moreover, Steve Jobs pitched NeXT's technology strongly to Apple, asserting that OPENSTEP was many years ahead of the market. The deal with NeXT did go through: Apple acquired NeXT in February 1997 for over \$400 million. Amelio later quipped, *"We choose plan A instead of Plan Be."*

NeXT's acquisition would prove pivotal to Apple, as NeXT's operating system technology would be the basis for what would become Mac OS X. Let us now look at the background of NeXT's systems.

1.7. THE NEXT CHAPTER

All of Steve Jobs' operational responsibilities at Apple were "taken away" on May 31, 1985. Around this time, Jobs had come up with an idea for a startup for which he pulled in five other Apple employees. The idea was to create the perfect research computer for universities, colleges, and research labs. Jobs had even attempted to seek the opinion of Nobel laureate biochemist Paul Berg on using such a computer for simulations. Although interested in investing in Jobs' startup, Apple sued Jobs upon finding out about the Apple employees joining him. After some mutual agreements, Apple dropped the suit the year after. The startup was NeXT Computer, Inc.

NeXT's beginnings were promising. Jobs initially used \$7 million of his personal money. Several larger investments would be made in NeXT, such as \$20 million from Ross Perot and \$100 million from Canon Inc. a few years later. True to its original goal, NeXT strived to create a computer that would be perfect in form and function. The result was the NeXT cube.

The cube's motherboard had a clever, visually appealing design. Its magnesium case was painted black with a matte finish. The monitor stand required an astonishing amount of engineering (for a monitor stand). An onboard digital signal-processing chip allowed the cube to play stereo quality music. The machines were manufactured in NeXT's own state-of-the-art factory.

1.7.1. NEXTSTEP

Jobs unveiled the NeXT cube on October 12, 1988, at the Davies Symphony Hall in San Francisco. The computer ran an operating system called NEXTSTEP, which used as its kernel a port of CMU Mach 2.0 with a 4.3BSD environment.⁵⁰ NEXTSTEP's window server was based on Display PostScript—a marriage of the PostScript page-description language and window-system technologies.

In 1986, Sun Microsystems had announced their own Display Postscript Window System called NeWS.

NEXTSTEP offered both a graphical user-interface and a Unix-style command-line interface. The NEXTSTEP graphical user interface had multilevel menus, windows whose contents were shown while being dragged, and smooth scrolling. A *dock* application always stayed on top and held frequently used applications. Other NEXTSTEP features included the following.

- The ability to “hide” applications instead of quitting them

⁵⁰ The Mach implementation in NEXTSTEP included NeXT-specific features, as well as some features from later versions of CMU Mach.

- CD-quality sound
- A versatile mail application that supported voice annotation of messages, inline graphics, and dynamic lookup of email addresses over the network
- Drag and drop of complex objects between applications
- A *services* menu that could be accessed from various applications to provide services such as dictionary and thesaurus
- A *Digital Librarian* application that could build searchable indexes of content dragged to it
- A file viewer that extended across the network
- An object-oriented device driver framework called the *Driver Kit*

NEXTSTEP used drag and drop as a fundamental, powerful operation. It was possible to drag an image from, say, the mail application, to a document editing application such as WordPerfect. Conversely, you could drag a spreadsheet to the mail application to attach it with a message. Since the file viewer was network capable, a remote directory could be dragged as a short cut on the user's desktop (specifically, on the *shelf*).

NEXTSTEP used Objective-C as its native programming language. It included *Interface Builder*, a tool for designing application user interfaces graphically. A number of *software kits* were provided to aid in application development. A software kit was a collection of reusable classes (or object templates). Examples include the Application Kit, the Music Kit, and the Sound Kit.

Objective-C

Objective-C is an object-oriented, compiled programming language invented by Brad Cox and Tom Love in the early 1980s. It is an object-oriented superset of C, with dynamic binding and a messaging syntax inspired by Smalltalk. It aims to be a simpler language than C++. Consequently, it does not have many features of C++, such as multiple inheritance and operator overloading.

Cox and Love founded StepStone Corporation, from which NeXT licensed the language and created its own compiler. In 1995, NeXT acquired all rights to StepStone's Objective-C related intellectual property.

Apple's Objective-C compiler used in Mac OS X is a modified version of the GNU compiler.

At the time of the cube's announcement, NEXTSTEP was at version 0.8. It would be another year before a 1.0 mature release would be made.

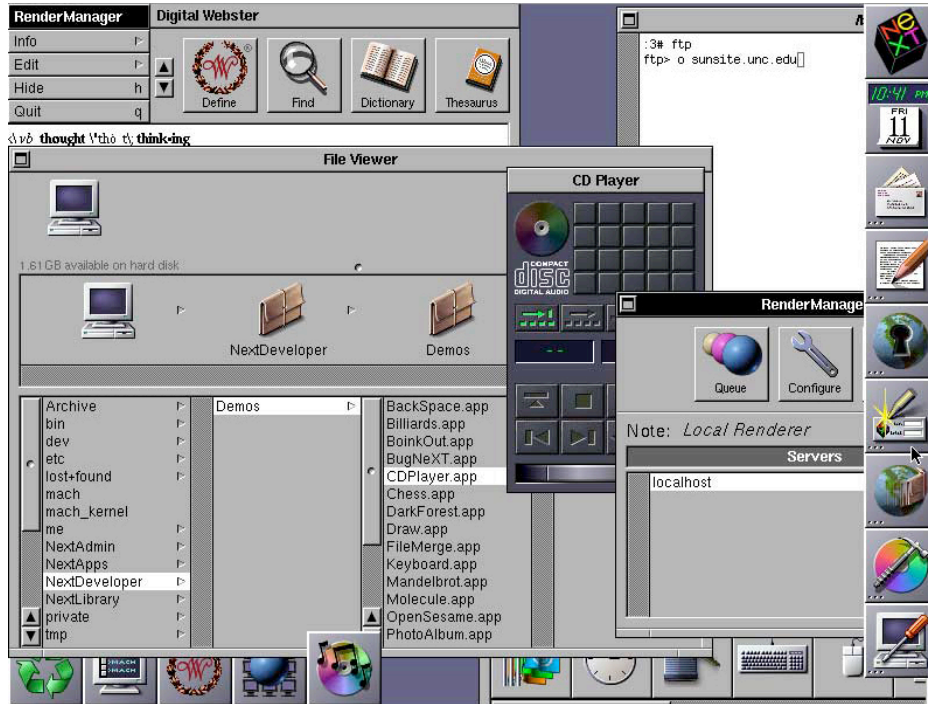
NEXTSTEP 2.0 was released a year after 1.0, with improvements such as support for CD-ROMs, color monitors, NFS, on-the-fly spell checking, and dynamically-loadable device drivers.

In the fall of 1990, Timothy John "Tim" Berners-Lee at CERN created the first web browser. It offered WYSIWYG browsing and authoring. The browser was developed on a NeXT computer. Tim's collaborator, Robert Cailliau, later said that "... *Tim's prototype implementation on NEXTSTEP is made in the space of a few months, thanks to the qualities of the NEXTSTEP software development system...*"

At the 1992 NeXTWORLD Expo, NEXTSTEP 486—a \$995 version for the x86—was announced.

The last version of NEXTSTEP—3.3—was released in February 1995. By that time NEXTSTEP had very powerful application development facilities courtesy of tools such as the Project Builder and the Interface Builder. There existed an extensive collection of libraries for user interfaces, databases, distributed objects, multimedia, networking, and so on. NEXTSTEP's object-oriented device driver toolkit was especially helpful in driver development.

Figure 1-26 shows a screenshot of NEXTSTEP.

**FIGURE 1-26** NEXTSTEP

NEXTSTEP ran on the 68K, x86, PA-RISC, and SPARC platforms. It was possible to create a single version of an application containing binaries for all supported architectures. Such multiple-architecture binaries are known as “fat” binaries⁵¹.

Despite the elegance of NeXT’s hardware and the virtues of NEXTSTEP, the company had proven to be economically unviable over the years. In early 1993, NeXT announced its plans to leave the hardware business but continue development

⁵¹ Mac OS X supports fat binaries. In particular, a fat binary can be used to contain 32-bit and 64-bit versions of a program on Mac OS X 10.4 and later. The so called “Universal Binaries” on the x86 version of Mac OS X are simply fat binaries.

of NEXTSTEP for the x86 platform. Figure 1–27 shows the timeline of NeXT's operating systems.

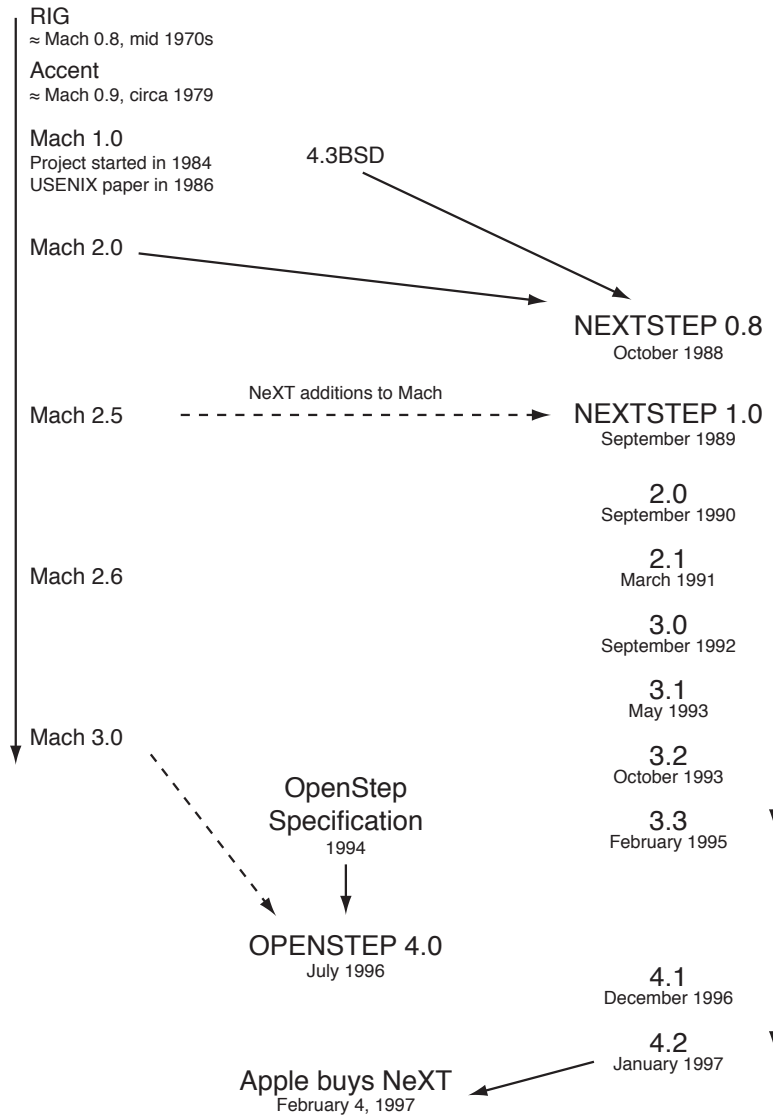


FIGURE 1–27 The timeline of NeXT's operating systems

Canon had a personal workstation, the *object.station 41*, which was designed to run NEXTSTEP. The system's 100 MHz Intel 486DX4 processor was upgradeable to an Intel Pentium OverDrive processor. Besides NEXTSTEP as the operating system, the machine included Insignia Solutions' SoftPC.

1.7.2. OPENSTEP

NeXT partnered with Sun Microsystems to jointly release specifications for OpenStep, an open platform comprised of several APIs and frameworks that anybody could use to create their own implementation of an object-oriented operating system—running on any underlying core operating system. The OpenStep API was implemented on SunOS, HP-UX, and Windows NT. NeXT's own implementation—essentially an OpenStep compliant version of NEXTSTEP—was released as OPENSTEP 4.0 in July 1996, with 4.1 and 4.2 following shortly afterwards.

The OpenStep API and the OPENSTEP operating system did not seem to turn things around for NeXT, even though they caused some excitement in the business, enterprise, and government markets. NeXT started to shift focus to its WebObjects product, which was a multiplatform environment for rapidly building and deploying web-based applications.

As we saw earlier, NeXT was purchased by Apple in early 1997. Mac OS X would be based on NeXT's technology. WebObjects would keep up with advancements in its domain, as exemplified by its support for Web Services and Enterprise Java. Apple uses WebObjects for its own web sites, such as the Apple Developer Connection (ADC) site, the online Apple Store, and the .Mac offering.

Figure 1–28 shows a screenshot of OPENSTEP.

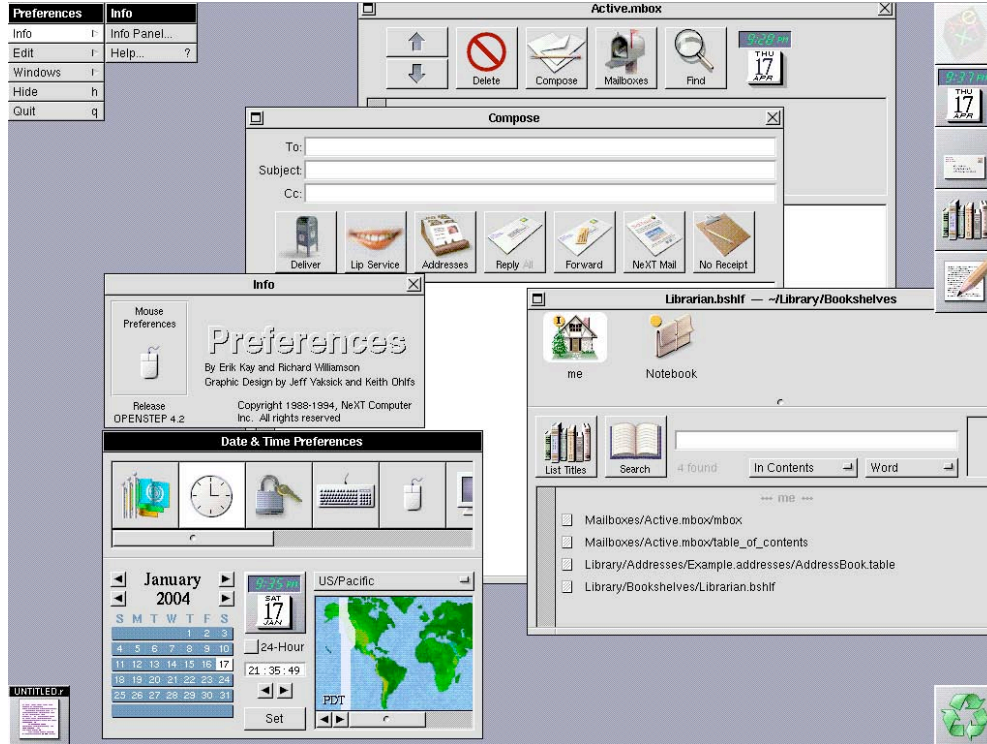


FIGURE 1-28 OPENSTEP

1.8. THE MACH FACTOR

Along with NeXT's operating system came its kernel, which became the kernel foundation of Apple's future systems. Let us now briefly discuss the origins and evolution of Mach—a key component of the NEXTSTEP kernel, and in turn, of the Mac OS X kernel.

1.8.1. Rochester's Intelligent Gateway

A group of researchers at the University of Rochester, New York, began development of an “intelligent” gateway system named *RIG* (Rochester's Intelligent Gateway) in 1975. Jerry Feldman, who coined the name RIG, largely did the system's initial design. RIG was meant to provide uniform access—say, via terminals—to a variety of local and remote computing facilities. Local facilities could be locally connected disks, magnetic tapes, printers, plotters, batch processing or time-sharing computers, and so on. Remote facilities could be available through a network such as the ARPANET. RIG's operating system—called *Aleph*—ran on a Data General Eclipse minicomputer.

The Aleph kernel was structured around an interprocess communication (IPC) facility. RIG processes could send messages to each other, with a *port* specifying the destination. A port was an in-kernel message queue that was globally identified by a dotted pair of integers: a process number and a port number. A process could have several ports defined within itself, each of which could be used to wait for a message to arrive on. A process X could *shadow* or *interpose* another process Y. In the case of shadowing, X received a copy of every message sent to Y. While interposing, X intercepted all messages sent to, or originating from Y. This IPC facility based on messages and ports was a basic building block of the operating system.

RIG was killed a few years later due to several fundamental shortcomings in its design or in the underlying hardware, for example:

- The lack of paged virtual memory
- A 2KB limit on the size of a message due to the limited address space provided by the underlying hardware
- Inefficient IPC due to limited message size
- No protection for ports
- No way to notify the failure of a process to a dependent process without explicit registration of such dependencies

- Networking not an area of emphasis in the original design

RIG port numbers were global, allowing any process to create or use them. Therefore, any process could send a message to any other process. However, RIG processes, which were single threaded, did have protected address spaces.

1.8.2. Accent

Richard Rashid was one of the people who worked on RIG. In 1979, Rashid moved to Carnegie Mellon University, where he would work on *Accent*, a network operating system kernel. Active development of Accent began in April 1981. Like RIG, Accent was also a communication-oriented system that used IPC as the basic system-structuring tool, or “glue.” However, Accent addressed many of RIG’s shortcomings, for example:

- Processes had large (4GB), sparse virtual address spaces that were linearly addressable.
- There was flexible and powerful virtual memory management that was integrated with IPC and file storage. The kernel itself could be paged, although certain critical parts of the kernel, such as I/O memory and the virtual memory table, were “wired” in physical memory.
- Copy-on-write (COW) memory mapping was used to facilitate large message transfers. Based on experience with RIG, it was expected that most messages would be simple. There were optimizations for the common case.
- Ports had the semantics of *capabilities*.
- Messages could be sent to processes on another machine through an intermediary process, thus providing location transparency.

Memory-related API calls in Accent included functions for creating, destroying, reading, and writing memory *segments*, with support for copy-on-write. One may think of Accent as RIG enhanced with virtual memory and network-transparent messaging.

Accent was developed to support two distributed computing projects: SPICE (distributed personal computing) and DSN (fault-tolerant distributed sensor network). Accent was also the name of a food product (a spice) sold by Accent International, Inc. The only ingredient of this product was monosodium glutamate (MSG). In computing, one often abbreviates “message” as “msg”.

Accent ran on PERQ computers, which were commercial graphics workstations. Three Rivers Corporation delivered the first PERQ in 1980. QNIX was a UNIX environment based on AT&T System V UNIX that ran under Accent on PERQ machines. Developed by Spider Systems, QNIX used its own microcode,⁵² but ran in an Accent window managed by Accent’s *Sapphire* window manager, with other Accent programs running alongside. A LISP machine (SPICE LISP) was also available for Accent, along with other languages such as Ada, PERQ Pascal, C, and Fortran. PERQ could interpret bytecode in hardware, akin to latter-day mechanisms for Java.

Within a few years, the future of Accent did not look promising as well. It needed a new hardware base, support for multiprocessors, and portability to other kinds of hardware. Accent also had difficulty supporting UNIX software.

Matchmaker

The *Matchmaker* project was started in 1981 as part of the SPICE project. Matchmaker was an interface-specification language intended for use with existing programming languages. Using the Matchmaker language, object-oriented remote procedure call (RPC) interfaces could be specified. The specification would be converted into interface code by a multitarget compiler. Matchmaker is readily comparable to the *rpcgen* protocol compiler and its language. The *Mach Interface Generator* (MIG) program, which is also used in Mac OS X, was derived from Matchmaker.

⁵² The PERQ had soft-microcode, allowing its instruction set to be extended.

1.8.3. Mach

The sequel to Accent was called *Mach*, which was conceived as a UNIX-compatible Accent-inspired system. In retrospect, with respect to the first version (1.0) of Mach, one could consider Accent and RIG to be Mach versions 0.9 and 0.8 respectively.

When Mach was developed, UNIX had been around for over fifteen years. Although the designers of Mach subscribed to the importance and usefulness of UNIX, they noted that UNIX was no longer as simple or as easy to modify as it once had been. Richard Rashid called the UNIX kernel a “*dumping ground for virtually every new feature or facility.*” Mach’s design goals were partially a response to the inexorably increasing complexity of UNIX.

The Mach project started in 1984 with an overall goal of creating a microkernel that would be the operating system foundation for other operating systems. The project’s specific goals included the following.

- Provide full support for multiprocessing.
- Exploit other features of modern hardware architectures that were emerging at that time. Mach aimed to support diverse architectures, including shared memory access schemes such as Non-Uniform Memory Access (NUMA) and No-Remote Memory Access (NORMA).
- Support transparent and seamless distributed operation.
- Reduce the number of features in the kernel to make it less complex, while giving the programmer a very small number of abstractions to work with. Nevertheless, the abstractions would be general enough to allow several operating systems to be implemented on top of Mach.
- Provide compatibility with UNIX.
- Address shortcomings of previous systems such as Accent.

Mach was intended to primarily implement processor and memory management, but no file system, networking, or I/O. The “real” operating system was to run

as a user-level Mach task. Written in C, the Mach kernel was also meant to be highly portable.

Mach's implementation used 4.3BSD as the starting code base. Its designers had RIG and Accent as references in the area of message-passing kernels. DEC's TOPS-20 operating system⁵³ provided some ideas for Mach's virtual memory subsystem. As Mach evolved, portions of the BSD kernel were replaced by their Mach equivalents, and various new components were added.

When published in 1986, the original Mach paper hailed Mach as "*A New Kernel Foundation For UNIX Development.*" While not everybody saw or sees it that way, Mach went on to become a rather popular system. From Apple's standpoint, the paper's title might as well have been "*A NuKernel Foundation...*"

Nomenclature

Avadis Tevanian, one of the inventors of Mach and Apple's Chief Software Technology Officer, told me the following history about how Mach was named. (Tevanian qualified the account as his best memory of an event that occurred two decades ago.) On a rainy day in Pittsburgh, Tevanian and some others were on their daily trek to lunch. As they were thinking of names for the yet unnamed Mach kernel, Tevanian, navigating around one of the numerous mud puddles, suggested the name "MUCK" in jest. MUCK was to stand for "Multi-User Communication Kernel" or "Multiprocessor Universal Communication Kernel." As a joke, Richard Rashid passed the name along to a colleague, Dario Giuse, who was Italian. Giuse inadvertently pronounced MUCK as "Mach," and Rashid liked it so much that the name stuck.

Initially the Mach designers presented four basic abstractions in the kernel.

⁵³ TOPS-20 was a descendant of the TENEX operating system.

A *task* is a container for the resources of one or more threads.⁵⁴ Examples of resources include virtual memory, ports, processors, and so on.

A *thread* is a basic unit of execution in a task. The task provides an execution environment for its threads, whereas the threads actually run. The various threads of a task share its resources, although each has its own execution state, which includes the program counter and various other registers. Thus, unlike a process in Accent, a Mach “process” is divided⁵⁵ into a task and multiple threads.

A *port* is similar to an Accent port—it is an in-kernel message queue with capabilities. Ports form the basis for Mach’s interprocess communication facilities. Mach implements ports as simple integral values.

A *message* is a collection of data that threads in different tasks, or in the same task, can send to each other using ports.

Another basic Mach abstraction is that of a *memory object*, which could be thought of as a container for data (including file data) mapped into a task’s address space. Mach requires a paged memory-management unit (PMMU). Through its *pmap* (physical map) layer, Mach provides an excellent interface to the machine-dependent MMU facilities. Mach’s virtual memory subsystem was designed to support large, sparse virtual address spaces, and was integrated with IPC. In traditional UNIX, contiguous virtual memory space was implied, with the heap and the stack growing towards each other. In contrast, Mach allowed for sparse address spaces. Regions of memory could be allocated from anywhere in the address space. Memory could be shared for reading and writing in a structured manner. Copy-on-write techniques were used both to optimize copy operations and for sharing physical memory between tasks. The generalized memory object abstraction allowed for *ex-*

⁵⁴ It is possible to have a Mach task with zero threads, although such a task would not be very useful.

⁵⁵ Certain subsequent versions of Mach further subdivided a thread into an “activation” and a “shuttle.”

*ternal*⁵⁶ memory pagers to handle page faults and page-out data requests. The source or target data could even reside on another machine.

FreeBSD's virtual memory architecture is based on Mach's.

As noted earlier, Mach was neither meant to provide, nor provided, any file system, networking, or I/O capabilities. It was to be used as a *service operating system* to create other operating systems from. It was hoped that this approach would maintain simplicity and promote portability of operating systems. One or more operating systems could run on top of Mach as user-level tasks. However, real-life implementations deviated from this concept. Release 2.0 of Mach, as well as the rather successful Release 2.5, had monolithic implementations in that Mach and BSD resided in the same address space.

One of CMU's important decisions was to provide all Mach software with unrestrictive licensing—free of distribution fees or royalties. The Open Software Foundation⁵⁷ (OSF) used Release 2.5 of Mach for providing many of the kernel services in the OSF/1 operating system. Mach 2.x was also used in Mt. Xinu, Multimax (Encore), Omron LUNA/88k, NEXTSTEP, and OPENSTEP.

The Mach 3 project was started at CMU and continued by OSF. Mach 3 was the first *true microkernel* version—BSD ran as a user-space Mach task, with only fundamental features being provided by the Mach kernel. Other changes and improvements in Mach 3 included the following.

- Kernel preemption and a real-time scheduling framework to provide real-time support

⁵⁶ Implies external to the kernel—that is, in user-space

⁵⁷ The OSF was formed in May 1988 to develop core software technologies and supply them to the entire industry on fair and reasonable terms. It went on to have several hundred members from among commercial end users, software companies, computer manufacturers, universities, research laboratories, and so on. The OSF later became the Open Group, and then became Silicomp.

- Low-level device support wherein devices were presented as ports to which data or control messages could be sent, with support for both synchronous and asynchronous I/O
- A completely rewritten IPC implementation
- System call redirection that allowed a set of system calls to be handled by user-space code running within the calling task
- Use of *continuations*, a kernel facility that gives a thread the option to block by specifying a function (the *continuation function*) that is called when the thread runs again

Historically, arguments in favor of “true” microkernels have emphasized a greater degree of system structure and modularity, improved software engineering, ease of debugging, robustness, software malleability (for example, the ability to run multiple operating system personalities), and so on. The intended benefits of microkernel-based operating systems such as Mach 3 were offset by the significant real-life performance problems that occurred due to reasons such as the following.

- The cost of maintaining separate protection domains, including the cost of context switching from one domain to another (often, simple operations resulted in many software or hardware layers to be crossed)
- The cost of kernel entry and exit code
- Data copies in MIG-generated stub routines
- The use of semantically powerful, but implementation-heavy IPC mechanisms, even for same-machine RPC

Many operating systems were ported to the conceptual virtual machine provided by the Mach API, and several user-mode operating system interfaces were demonstrated to execute on top of Mach. The *Mach-US* symmetric multiserver operating system contained a set of server processes that provided generic system services such as local interprocess communication; networking; and management of devices, files, processes, and terminals. Each server typically ran in a separate Mach task. An emulation library, which was loaded into each user process, provided an operating system personality. Such libraries used generic services to emulate different operating systems by intercepting system calls and redirecting them to the ap-

propriate handlers. Mach emulators existed for BSD, DOS, HP-UX, OS/2, OSF/1, SVR4, VMS, and even the Macintosh operating system.

Richard Rashid went on to become the head of Microsoft Research. Mach coinventor Avie Tevanian would be the Chief Software Technology Officer at Apple.

1.8.4. MkLinux

Apple and OSF began a project to port Linux to run on various Power Macintosh platforms, with Linux hosted on top of OSF's Mach implementation. The project led to a core system called *osfmk*. The overall system was known as *MkLinux*. The first version of MkLinux was based on Linux 1.3. It was released as MkLinux DR1 in early 1996. Subsequent releases moved to Linux 2.0 and beyond. One of the releases was incorporated into Apple's Reference Release.

MkLinux used a single server approach: the monolithic Linux kernel ran as a single Mach task. Mac OS X uses a kernel base derived from *osfmk*, and includes many MkLinux enhancements. However, all kernel components in Mac OS X, including the BSD portions, reside in the same address space.

Mach^{Ten}

Besides A/UX, there was another avenue—a third party one—on which the Macintosh had close encounters of the Unix kind. The Mach^{Ten} product from Tenon Systems was introduced as an unobtrusive Unix solution for MacOS: it ran as an application atop Apple's operating system. In contrast, A/UX ran directly on top of hardware.

Mach^{Ten} was based on the Mach kernel with a BSD environment. It provided preemptive multitasking for Unix applications running within it, although the MacOS execution environment remained cooperative multitasking.

Although the marriage of Mach, BSD, and Macintosh in Mach^{Ten} sounds similar to the latter-day Mac OS X, there is a critical difference in design and philosophy. Mac OS X was a continuation of NEXTSTEP technology in several ways. Apple provided legacy compatibility and ease of transition at two primary levels: through APIs such as Carbon, and through the Classic virtualizer. In contrast, Mach^{Ten} was a logical opposite: MacOS remained the first class citizen, whereas Unix ran in a virtual machine (UVM) that was implemented within a standard Macintosh application. The UVM provided a preemptive multitasking execution environment with a set of Unix APIs (such as POSIX, including the standard C library and POSIX threads), a BSD-style networking stack, file systems such as UFS and FFS, RPC, NFS, and so on. Mach^{Ten} also included an implementation of the X Window System.

Although confined within a single application, Mach^{Ten} consisted of various subsystems similar to a full-fledged operating system. At the logically lowest level, an interface layer talked to MacOS. The Mach kernel resided above this layer, providing services such as memory management, interprocess communication, tasks, and threads. Other Mach^{Ten} subsystems that directly talked to the MacOS interface layer included the window manager and the networking stack's ARP layer.

1.8.5. Musical Names

Apple's operating system strategy after acquiring NeXT was two-pronged: it would keep improving Mac OS for the consumer desktop market, and would create a high-end operating system based on NeXT technology. The new system, called *Rhapsody*, would mainly be targeted towards the server and enterprise markets.

In contrast to the chromatic aberrations such as Pink and Red, Apple also had a string of musically inspired code names for its operating system projects. Copland

and Gershwin were named after Aaron Copland and George Gershwin,⁵⁸ both American composers. *Rhapsody in Blue* is a famous work of Gershwin.

1.9. STRATEGIES

The first release of an Apple operating system after NeXT's purchase was in late 1996 with version 7.6. This release represented the initial stage of Apple's new operating system roadmap. It was the first system to be called *Mac OS*. Apple's plan was to release full standalone installations once a year, with updates in between. Many Power Macintosh and PowerBook models that were not supported by Mac OS 7.6 were supported by the 7.6.1 incremental update. The system originally slated to be version 7.7 would eventually become Mac OS 8.

Mac OS 7.6 required a compatible computer that was 32-bit clean, with at least a 68030 processor. It offered performance enhancements in several areas such as virtual memory, memory management, PowerPC Resource Manager routines, system startup, and the File Manager's caching scheme. It also integrated key Apple technologies such as Cyberdog, OpenDoc, Open Transport, and QuickTime.

Two phenomena were sweeping the computer world at that time: the Internet and Microsoft Windows 95. Apple emphasized compatibility of Mac OS 7.6 with Windows 95 and highlighted the system's Internet prowess. Mac OS 7.6 included built-in support for TCP/IP, PPP, and Apple Remote Access (ARA). Its integrated Cyberdog technology could be used to incorporate Internet features into documents that used "Live Objects." For example, live web links and email addresses could reside on the Desktop, and could be activated from the Finder.

⁵⁸ George Gershwin's brother Ira actually came up with the title *Rhapsody in Blue*.

OpenDoc

OpenDoc was a cross-platform component software architecture for Mac OS, OS/2, Windows, and UNIX. It started as collaboration between several companies, including Apple, IBM, and WordPerfect. An independent association called Component Integration Laboratories (CI Labs) was founded to act as a forum for the “open” evolution of OpenDoc.

OpenDoc was implemented as a set of shared libraries that allowed construction and sharing of compound documents. An OpenDoc *document* was composed of building blocks of content called *components*, which could be interactively edited. A component was a relatively small software unit containing a well-defined focused functionality. OpenDoc aimed to replace large, monolithic applications with applications constructed by mixing and matching various components. Examples of OpenDoc component types include graphics, Internet, spreadsheet, text, and video components. OpenDoc's implementation of such functionality, which was identified as hitherto being redundant across complex applications, yielded reusable building blocks that could be embedded into OpenDoc-aware documents. A document could have features such as editable portions, live data feed from an Internet source, user-interface elements that linked one part of the document to another, and hot areas where objects could be dragged and dropped.

OpenDoc was supported by several key technologies: Document Level Services, Component Level Services, Open Scripting Architecture⁵⁹ (OSA), System Object Model (SOM), and Open Linking and Embedding of Objects (interoperable with Microsoft's OLE).

The OLE-inspired COM and DCOM were OpenDoc's competitors. Whereas OpenDoc failed, COM is heavily used by modern versions of Microsoft Windows.

⁵⁹ OSA is an automation and scripting API that also exists in Mac OS X. It supports application-independent scripting, allowing multiple scripting systems and languages to coexist. AppleScript is the primary language that supports OSA.

1.9.1. Mac OS 8 and 9

As we saw earlier, Copland and Pink were potential candidates for Mac OS 8 at one time or another. Similarly, Gershwin was a candidate for Mac OS 9. Over the years, some important features that were either created or improved for Copland were added to Mac OS 8 and 9, as was originally intended. The following are examples of such features.

- A search engine that could search on local drives, network servers, and the Internet (released as Sherlock)
- The Copland API, which gradually evolved into Carbon
- The Platinum-look user interface
- Multiple users, with support for per-user preferences

Mac OS 8 had a multithreaded Finder that allowed several file-oriented operations simultaneously. Other notable features included the following.

- The Mac OS Extended file system (HFS Plus), which was introduced with Mac OS 8.1
- Contextual menus activated by a control-click
- Spring-loaded folders⁶⁰
- Personal web hosting
- Web browsers (Microsoft Internet Explorer and Netscape Navigator) bundled with the system
- Macintosh Runtime for Java (MRJ—Apple’s implementation of the Java environment) part of the system
- Enhancements to power-management, USB, and FireWire

Figure 1–29 shows a screenshot of Mac OS 8.

⁶⁰ Spring-loaded folders are a feature of the Finder’s user interface. If the user pauses briefly while dragging an item onto a folder icon, a window springs open displaying the folder’s contents. This allows the user to choose where to put the item. Continuing to hold the item causes a subfolder to spring open, and so on.

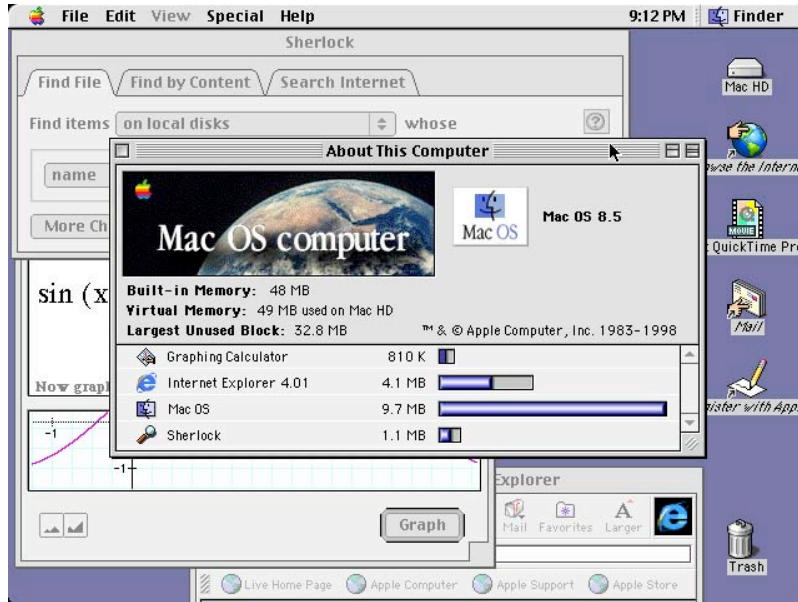


FIGURE 1-29 Mac OS 8

Mac OS 8.5 was PowerPC-only. The nanokernel was overhauled in Mac OS 8.6 to integrate multitasking and multiprocessing. It included a preemption-safe memory allocator. The multiprocessor (MP) API library could now run with virtual memory enabled, although virtual memory was still optional.

When Mac OS 9 was released in 1999, it was hailed by Apple as the “*best Internet operating system ever*”. It was the first Mac OS version that could be updated over the Internet. It could also use the AppleTalk protocol over TCP/IP. Its useful security features included file encryption and the *Keychain* mechanism for storing passwords securely.

An important component of Mac OS 9 was a mature installation of the Carbon APIs, which at the time represented about 70% of the legacy Mac OS APIs. Carbon provided compatibility with Mac OS 8.1 and later.

The last release of Mac OS 9 was released in late 2001 as version 9.2.2. With the advent of Mac OS X, this “old” Mac OS would eventually be referred to as *Classic*. Figure 1–30 shows a screenshot of Mac OS 9.



FIGURE 1–30 Mac OS 9

1.9.2. Rhapsody

We saw that after acquiring NeXT, Apple based its next-generation operating system called Rhapsody on NeXT’s OPENSTEP. Rhapsody was first demonstrated at the 1997 World Wide Developers Conference (WWDC). Figure 1–31 shows a screenshot of Rhapsody.



FIGURE 1-31 Rhapsody

Rhapsody consisted of the following primary components.

- The kernel and related subsystems that were based on Mach and BSD
- A Mac OS compatibility subsystem (the *Blue Box*)
- An extended OpenStep API implementation (the *Yellow Box*)
- A Java virtual machine
- A Display PostScript-based windowing system
- A user interface that was Mac OS-like, but also had features from OPENSTEP

Apple had plans to port to Rhapsody most key Mac OS frameworks, for example, QuickTime, QuickDraw 3D, QuickDraw GX, and ColorSync. Rhapsody was also to support numerous file systems such as Apple Filing Protocol (AFP), FAT, HFS, HFS Plus, ISO9660, and UFS.

There were two developer releases of Rhapsody, dubbed DR1 and DR2. These were released both for the PowerPC and the x86 platforms.

1.9.2.1. Blue Box

Shortly after Rhapsody DR1 was released, Apple extended the PowerPC version with a Mac OS compatibility environment called the Blue Box. Implemented by a Rhapsody application (`MacOS.app`), Blue Box was a virtual environment that appeared as a new Macintosh hardware model. `MacOS.app` loaded a Macintosh ROM file from disk and created an environment within which Mac OS ran mostly unchanged. Blue Box initially ran Mac OS 8.x, full-screen, with the ability to switch between Rhapsody and Mac OS using the `<cmd-return>` key combination. It placed certain restrictions on the applications that ran within it. For example, an application could neither access the hardware directly, nor could use undocumented Mac OS APIs. The implementers' initial goal was to achieve 90% to 115% of native Mac OS performance. Blue Box beta 1.0 used Open Transport—rather than BSD sockets—for networking. Support for newer versions of Mac OS, as well as for running the Blue Box windowed, was added later. The Blue Box environment would be known as the *Classic environment* in Mac OS X, provided by an application named “`Classic Startup.app`”.⁶¹

The Blue Box environment is a virtualization layer, and not an emulation layer. “Harmless” instructions execute natively on the processor, whereas “harmful” instructions, such as those that can affect the hardware, are trapped and handled appropriately.

⁶¹ The application was called `Classic.app` in earlier versions of Mac OS X.

1.9.2.2. Yellow Box

Rhapsody's development platform was called the Yellow Box. Besides being hosted on the Power Macintosh and x86 versions of Rhapsody, it was also available independently for Microsoft Windows. Figure 1–32 shows a screenshot of Yellow Box running under Windows XP.

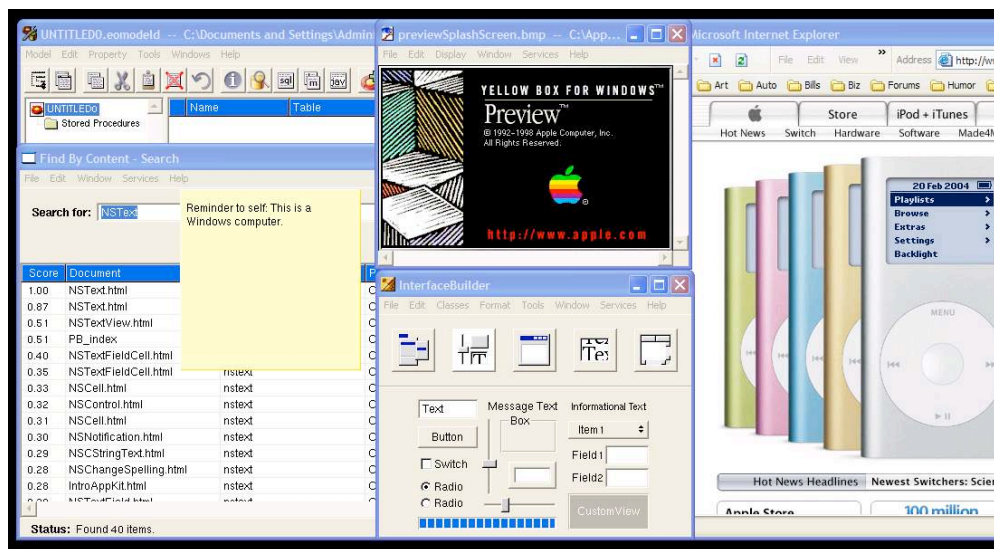


FIGURE 1–32 Yellow Box

Yellow Box included most of OPENSTEP's integrated frameworks, which were implemented as shared object libraries. These were augmented by a runtime and development environment. There were three core object frameworks whose APIs were available in Objective-C and Java.

- *Foundation* was a collection of base classes with APIs for allocating, deallocating, examining, storing, notifying, and distributing objects.

- *Application Kit* was a set of APIs for creating user interfaces; managing and processing events; and using services such as color and font management, printing, cut-and-paste, and text-manipulation.
- *Display PostScript* was a set of APIs for drawing in PostScript, compositing images, and performing other visual operations. It could be considered as a subset of Application Kit.

Yellow Box included NeXT's Project Builder integrated development environment and the Interface Builder visual tool for creating graphical user-interfaces. The Windows NT implementation of Yellow Box provided a very similar environment through a combination of the following Apple provided Windows system services and applications:

- The Mach Emulation Daemon (the `machd` service)
- The Netname Server (the `nmserver` service)
- The Window Server (the `windowServer` application)
- The Pasteboard Server (the `pbs` application)

Earlier implementations of the OpenStep API for platforms such as Solaris used a similar architecture. Yellow Box evolved into the Mac OS X Cocoa APIs.

1.10. TOWARDS MAC OS X

After Rhapsody's DR2 release, Apple would still alter its operating system strategy, but would finally be on its way towards achieving its goal of having a *new* system. During the 1998 Worldwide Developer Conference, Adobe's Photoshop ran on what would be Mac OS X. However, the first shipping release of Mac OS X would take another three years. Figure 1-33 shows an approximation of the progression from Rhapsody towards Mac OS X.

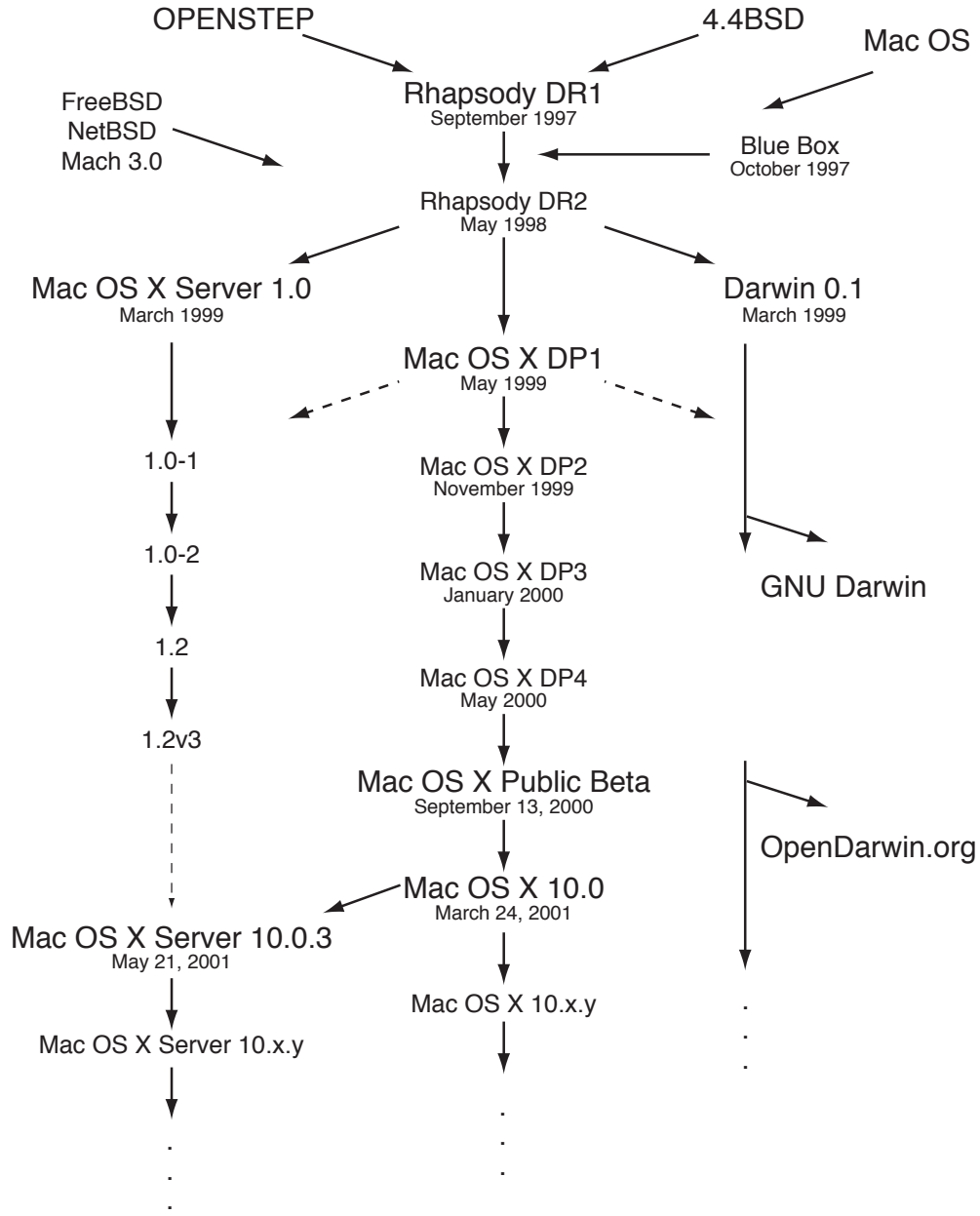


FIGURE 1-33 An approximation of the Mac OS X timeline

1.10.1. Mac OS X Server 1.x

As people were expecting a DR3 release of Rhapsody, Apple announced Mac OS X Server 1.0 in March 1999. Essentially an improved version of Rhapsody, it was bundled with WebObjects, the QuickTime streaming server, a collection of developer tools, the Apache web server, and facilities for booting or administering over the network.

Apple also announced an initiative called *Darwin*: a fork of Rhapsody's developer release. Darwin would become the open-source core of Apple's systems.

Over the next three years, as updates would be released for the server product, development of the desktop version would continue, with the server sharing many of the desktop improvements.

1.10.2. Mac OS X Developer Previews

There were four Developer Preview releases of Mac OS X: named DP1 through DP4. Substantial improvements were made during these "DP" releases.

1.10.2.1. DP1

An implementation of the Carbon API was added in DP1. Carbon represented an overhaul of the "classic" Mac OS APIs, which were pruned, extended, and modified to run in the more modern Mac OS X environment. Carbon was also meant to help Mac OS developers in transitioning to Mac OS X. A Classic application would require an installation of Mac OS 9 to run under Mac OS X, whereas Carbon applications could be compiled to run as native applications under both Mac OS 9 and Mac OS X.

1.10.2.2. DP2

The Yellow Box evolved into Cocoa, originally alluding to the fact that besides Objective-C, the API would be available in Java. A version of the Java Development Kit (JDK) was included, along with a just-in-time (JIT) compiler. The Blue Box environment was provided via `Classic.app` (a new version of `MacOS.app`) that ran as a process called `TruBlueEnvironment`. The Unix environment was based on 4.4BSD. DP2 thus contained a multitude of APIs: BSD, Carbon, Classic, Cocoa, and Java. There was widespread dissatisfaction with the existing user interface. The Aqua user interface had not been introduced yet, although there were rumors that Apple was keeping the “real” user interface a secret.⁶²

Carbon is sometimes perceived as “the old” API. Although Carbon indeed contains modernized versions of many old APIs, it also provides functionality that may not be available through other APIs. Parts of Carbon are complementary to “new” APIs such as Cocoa. Nevertheless, Apple has been adding more functionality to Cocoa so that dependencies on Carbon can be eventually eliminated. For example, much of the QuickTime functionality was only available through Carbon in Mac OS X versions prior to 10.4. Apple introduced the QTKit framework for Cocoa in Mac OS X 10.4, which reduces or eliminates Carbon dependencies for QuickTime.

1.10.2.3. DP3

The Aqua user interface was first demonstrated during the San Francisco Macworld Expo in January 2000. Mac OS X DP3 included Aqua along with its distinctive elements: “water-like” elements, pinstripes, pulsating default buttons, “traffic-light” window buttons, drop shadows, transparency, animations, sheets, and so on. The

⁶² Apple had referred to the Mac OS X user interface as “Advanced Mac OS Look and Feel”.

DP3 Finder was Aqua-based as well. The *Dock* was introduced with support for photorealistic icons that were dynamically scalable up to 128×128 pixels.

1.10.2.4. DP4

The Finder was renamed the *Desktop* in DP4. The System Preferences application (`Preferences.app`—the precursor to “`System Preferences.app`”) made its first appearance in Mac OS X, allowing the user to view and set a multitude of system preferences such as Classic, ColorSync, Date & Time, Energy Saver, Internet, Keyboard, Login Items, Monitors, Mouse, Network, Password, and others. Prior to DP4, the Finder and the Dock were implemented within the same application. The Dock was an independent application (`Dock.app`) in DP4. It was divided into two sections: the left side for applications and the right side for the trash can, files, folders, and minimized windows. Other notable components of DP4 included an integrated development environment and OpenGL.

The Dock’s visual indication of a running application underwent several changes. In DP3, an application’s Dock icon had a few pixels high bottom edge that was color-coded to indicate whether the application was running. This was replaced by an ellipsis in DP4, and was followed by a triangle in subsequent Mac OS X versions. DP4 also introduced the smoke cloud animation that ensues after an item is dragged off the Dock.

1.10.3. Mac OS X Public Beta

Apple released a beta version of Mac OS X at the Apple Expo in Paris on September 13, 2000. Essentially a publicly available preview release for evaluation and development purposes, the “Mac OS X Public Beta” was sold for \$29.95 at the Apple Store. It was available in English, French, and German. The software’s packaging contained a message from Apple to the beta testers: “*You are holding the future*”

of the Macintosh in your hands.” Apple also created a Mac OS X tab on its web site that would contain information on Mac OS X, including updates on third-party applications, tips and tricks, and technical support. Figure 1–34 shows a screenshot of Mac OS X Public Beta.



FIGURE 1–34 Mac OS X Public Beta

Although the beta release was missing important features and ostensibly lacked in stability and performance, it demonstrated several important Apple technologies at work, particularly to those who had not been following the DP releases. The beta's key features were the following.

- The Darwin core with its xnu kernel that offered “true” memory protection, preemptive multitasking, and symmetric multiprocessing
- The PDF-based Quartz 2D drawing engine
- OpenGL support
- The Aqua interface and the Dock
- Apple’s new mail client, with support for IMAP and POP
- A new version of the QuickTime player
- The Music Player application for playing MP3s and audio CDs
- A new version of the Sherlock Internet searching tool
- A beta version of Microsoft Internet Explorer

With Darwin, Apple would continually leverage a substantial amount of existing open source software by using it for, and often integrating it with Mac OS X. Apple and Internet Systems Consortium, Inc. (ISC) jointly founded the *OpenDarwin* project in April 2002 for fostering cooperative open source development of Darwin. *GNU-Darwin* is an open source Darwin-based operating system.

The New Kernel

Darwin’s kernel is called “xnu.” It is unofficially an acronym for “X is Not Unix.” It is also a coincidental tribute to the fact that it is indeed the NuKernel for Mac OS X. xnu is largely based on Mach and FreeBSD, but includes code and concepts from various sources such as the formerly Apple supported MkLinux project, the work done on Mach at the University of Utah, NetBSD, and OpenBSD.

1.10.4. Mac OS X 10.x

The first version of Mac OS X was released on March 24, 2001 as Mac OS X 10.0 “Cheetah.” Soon afterwards, the versioning scheme of the server product was revised to synchronize it with that of the desktop system. Since then, the trend has

been that a new version of the desktop is released first, soon followed by the equivalent server revision.

The first few major Mac OS X releases are listed in Table 1–1. Note that the code names are all taken from felid taxonomy.

TABLE 1–1 Mac OS X Versions

Version	Codename	Release Date
10.0	Cheetah	March 24, 2001
10.1	Puma	September 29, 2001
10.2	Jaguar	August 23, 2002
10.3	Panther	October 24, 2003
10.4	Tiger	April 29, 2005
10.5	Leopard	2006/2007?

Let us look at some notable aspects of each major Mac OS X release.

1.10.4.1. Mac OS X 10.0

Apple dubbed “Cheetah” as “*the world’s most advanced operating system.*” Finally, Apple had shipped an operating system with features that it had long sought. However, it was clear that Apple had a long way to go in terms of performance and stability. Key features of 10.0 included the following.

- The Aqua user interface, with the Dock and the Finder as the primary user-facing tools
- The PDF-based Quartz 2D graphics engine
- OpenGL for 3D graphics
- QuickTime for streaming audio and video (shipping for the first time as an integrated feature)
- Java 2 Standard Edition (J2SE)
- Integrated Kerberos

- Mac OS X versions of the three most popular Apple applications available as free downloads: iMovie 2, iTunes, and a preview version of AppleWorks
- Free IMAP service for .Mac email accounts

When Mac OS X 10.0 was released, there were approximately 350 applications available for it.

1.10.4.2. Mac OS X 10.1

“Puma” was a free update released six months after 10.0’s release. It offered significant performance enhancements, as indicated by Apple’s following claims.

- Up to 3× improvement in application launch speed
- Up to 5× improvement in menu performance
- Up to 3× improvement in window resizing
- Up to 2× improvement in file copying

There were substantial performance boosts in other areas such as system startup, user login, Classic startup, OpenGL, and Java. Other key features of this release included the following.

- The ability to move the Dock from its usual place at the bottom to the left or right
- System status icons on the menu bar to provide easier access to commonly used functions such as volume control, display settings, date and time, Internet connection settings, wireless network monitoring, and battery charging
- iTunes and iMovie as part of system installation, and the introduction of iDVD
- A new DVD player with a simplified interface
- Improved iDisk functionality based on WebDAV
- A built-in image capturing application to automatically download and enhance pictures from digital cameras

- The ability to burn over 4GB of data to a DVD, with support for burning recordable DVD discs directly in the Finder
- An integrated SMB/CIFS client

The Carbon API implementation in 10.1 was complete enough to allow important third party applications to be released. Carbonized versions of Microsoft Office, Adobe Photoshop, and Macromedia Freehand were released soon after 10.1's release.

1.10.4.3. Mac OS X 10.2

“Jaguar” was released at 10:20 pm to emphasize its version number. Its important feature additions included the following.

- *Quartz Extreme*—an integrated hardware acceleration layer for rendering on-screen objects by compositing them using primarily the Graphics Processing Unit (GPU) on supported graphics cards
- *iChat*—an AOL Instant Messaging (AIM) compatible “IM” client
- An enhanced mail application (`Mail.app`) with built-in adaptive spam filtering
- A new Address Book application with support for vCards, Bluetooth, and iSync synchronization with .Mac servers, PDAs, certain cell phones, and other Mac OS X computers (the Address Book's information was accessible to other applications)
- QuickTime 6, with support for MPEG-4
- An improved Finder with quick file searching from the toolbar and support for spring-loaded folders
- *Inkwell*—a handwriting recognition technology integrated with the text system, allowing text input using a graphics tablet
- *Rendezvous*,⁶³ which was Apple's implementation of *ZeroConf*—a zero-configuration networking technology allowing enabled devices to find one another on the network
- Better compatibility with Windows networks
- Version 3 of the Sherlock Internet services tool

⁶³ Rendezvous was later renamed Bonjour.

Hereafter, Apple introduced new applications and incorporated technologies in Mac OS X at a bewildering pace. Other notable additions to Mac OS X after the release of “Jaguar” included the iPhoto digital photo management application, the Safari web browser, and an optimized implementation of the X Window system.

1.10.4.4. Mac OS X 10.3

“Panther” added several productivity and security features to Mac OS X, besides providing general performance and usability improvements. Notable 10.3 features included the following.

- An enhanced Finder, with a sidebar and support for labels
- Audio and video conferencing through the iChat AV application
- *Exposé*—a user-interface feature that can “live shrink” each on-screen window such that no windows overlap, allowing the user to find a window visually, after which each window is restored to its original size and location
- *FileVault*—encryption of a user’s home directory
- Secure deletion of files in a user’s trash can via a multipass overwriting algorithm
- Fast user switching
- Built-in faxing
- Improved Windows compatibility courtesy of better support for SMB shares and Microsoft Exchange
- Support for HFSX—a case-sensitive version of the HFS Plus file system

The BSD component in “Panther” was based on FreeBSD 5.

1.10.4.5. Mac OS X 10.4

Besides providing typical evolutionary improvements, “Tiger” introduced several new technologies such as *Spotlight* and *Dashboard*. Spotlight is a search technology consisting of an extensible set of metadata importer plug-ins and a query API for

searching files based on their metadata, even immediately after new files are created. Dashboard is an environment for creating and running lightweight desktop utilities called *widgets*, which normally remain hidden and can be summoned by a key-press. Other important “Tiger” features include the following.

- Improved 64-bit support, with the ability to compile 64-bit binaries, and 64-bit support in the libSystem shared library
- *Automator*—a tool for automating common procedures by visually creating workflows
- *Core Image*—a media technology employing GPU-based acceleration for image processing
- *Core Video*—a media technology acting as a bridge between QuickTime and the GPU for hardware-accelerated video processing
- *Quartz 2D Extreme*—a new set of Quartz layer optimizations that use the GPU for the entire drawing path (from the application to the framebuffer)
- *Quartz Composer*—a tool for visually creating compositions using both graphical technologies (such as Quartz 2D, Core Image, OpenGL, and QuickTime) and non-graphical technologies (such as MIDI System Services and Rich Site Summary)
- Support for resolution-independent user interface
- Improved iChat AV, with support for multiple simultaneous audio and video conferences
- *PDF Kit*—a Cocoa framework for managing and displaying PDF files from within applications
- Improved Universal Access, with support for an integrated spoken interface
- Improved Sync Services
- An embeddable SQL database engine (SQLite) allowing applications to use SQL databases without running a separate RDBMS⁶⁴ process
- *Core Data*—a Cocoa technology that integrates with Cocoa bindings and allows visual description of an application's data entities, whose instances can persist on a storage medium
- An improved Search Kit

⁶⁴ Relational Database Management System

- *Fast Logout and Autosave* for improved user experience
- Support for Access Control Lists (ACLs)
- New formalized and stable interfaces, particularly for kernel programming
- Improvements to: the Web Kit (including support for creating and editing content at the DOM level of an HTML document), the Safari web browser (including RSS support), QuickTime (including support for the H.264 code and a new QuickTime Kit Cocoa framework), the Audio subsystem (including support for OpenAL, the Open Audio Library), the Mac OS X installer application, Xcode, and so on

The first shipping x86-based Macintosh computers used Mac OS X 10.4.4 as the operating system.

1.11. OTHERS

Besides desktop and server computers, Apple has made various other devices running operating systems, such as the Pippin multimedia device, hand-held computers (the MessagePad and the eMate), and the iPod portable music player.

1.11.1. Mac OS on the Pippin

Apple announced the *Pippin* platform in Tokyo on December 13, 1994. The Pippin was dubbed as a multimedia device, a set-top box, and a network computer. It was a multimedia player platform based on Apple's second generation Power Macintosh hardware and software.

The nomenclature has fruity connotations and connections. Pippin is also a variety of apples (the fruit). Pippin apples are smaller than the McIntosh variety of apples.

The Pippin was meant for activities such as playing back CDs, surfing the Internet, reading email, and playing games—but not for full-fledged computing. Its primary display was meant to be a television screen. It was therefore positioned as a

device that was “more than” a video game console, but “less than” a personal computer. Apple’s plan was to license the Pippin platform to third parties, allowing manufacturers to build and sell their own versions. The license terms were to include a per-title royalty.

The Pippin’s technical specifications are shown in Table 1–2. Note that of the 6MB RAM, approximately 2MB was used for system software and video, and the rest was available for use by titles. Memory could be added via expansion cards in increments of 2MB, 4MB, or 8MB. The few devices actually made also had 128KB of built-in NVRAM, of which the system software used 8KB. The NVRAM was represented as a small HFS volume.

TABLE 1–2 The Pippin’s technical specifications

Area	Details
Processor	66 MHz PowerPC 603e processor, with 3 instructions per clock cycle, 8KB data cache, 8KB instruction cache, and IEEE standard compliant single/double precision floating-point unit
Memory	6MB total RAM and 64KB SRAM
Physical Storage	4X CDROM drive, floppy disk drives and disk drives attachable through an expansion bus
Video	Support for NTSC, PAL, S-Video, and VGA (640×480); up to 16.7 million colors; support for 8-bit and 16-bit video
Audio	Stereo 16-bit 44 kHz sampled input and output
Input Devices	Support for up to four simultaneous game controllers over the Apple Desktop Bus, support for standard ADB keyboard and mice connected through adapters

Area	Details
I/O	Serial port and telephony support via an optional GeoPort

The Pippin also offered a proprietary digital filtering technique for improving text visibility on a standard television screen. Planned connectivity features for the Pippin included file sharing and other communication with home computers. Internet connectivity was to be provided using an additional adapter or an external modem.

Apple announced in early 1995 that the Pippin would ship by the end of that year.⁶⁵ There were no plans to ship an Apple-branded Pippin. The first third party to license the Pippin was Bandai⁶⁶ Digital Entertainment Corporation—a Japanese toy manufacturer and CDROM game title publisher. Bandai’s Pippin-based *Power Player* product was expected to sell for approximately \$500. A “cheap” multimedia computer’s cost was over \$1000 at that point.

The Pippin ran a custom version of Mac OS that had various dedicated features. Several Macintosh computer specific features had been removed to minimize memory footprint. However, highly used or otherwise necessary features *were* included, for example: the Macintosh Toolbox; a built-in 680x0 emulator; integrated QuickTime; and an integrated, PowerPC-native version of QuickDraw. Since the Pippin did not include a disk drive by default, it typically booted from system software residing on the same disc as a Pippin title. Developers could create their own system software bundle by choosing the appropriate system version and configuration.

The Pippin’s intended software library was to include games and software for reference, learning, and interactive music. It was believed that the Pippin would

⁶⁵ Apple eventually provided a Pippin Developer SDK in early 1996.

⁶⁶ Bandai’s product line included the Power Rangers action figures.

even run simple versions of financial software, spreadsheets, and word processors. In fact, Apple expected many Macintosh applications to run unchanged on the Pippin,⁶⁷ with the only requirement being a remastering step to include system software on application discs. Conversely, Pippin titles were expected to work on the Macintosh too.

The Pippin met with a positive reception from the media. An October 1996 article in a Microsoft developer column called Pippin “*A Real Network Computer.*” The article pointed out that the Pippin’s appeal was that for the price of a high-end VCR, it delivered an almost complete low-end Macintosh. Apple’s licensing plans were also praised:

“Apple’s go-it-alone attitude has achieved legendary status. That’s why it’s amazing that the company that once vowed never to license their crown jewels is neither manufacturing nor marketing the Pippin itself. The Network Computer may be a joke so far, but Pippin clearly is not.” BYTE magazine proclaimed “*Bandai Digital’s @WORLD Web-browsing system may one day be the Mac network computer for corporations.*”

However, the Pippin was a failed product—it did not reach most of its intended markets. Bandai sold some systems, marketing them as ATMARK in Japan and @WORLD in USA. The @WORLD system came with a slightly modified version of the Spyglass Internet browser. A few other variants of Pippin existed as samples or prototypes. Some Pippins had a PowerPC sticker and another stating “*Advanced Technology By Apple Computer.*”

Apple also had a set top box product: the Apple Interactive TV Box. It supported the MPEG-2 video decompression standard, and a variety of input and output ports such as ADB, dual 21-pin EURO-SCART, RF IN and RF OUT (either NTSC or PAL), RJ-45 connector for either E1 or T1 data streams, Apple System/Peripheral 8 Cable (serial), S-Video output, RCA

⁶⁷ Nevertheless, the Pippin was a fundamentally more limited runtime environment as compared to a Macintosh.

composite video output, RCA stereo audio output, and SCSI HDI-30.

1.11.2. Newton OS

Newton was a software and hardware technology that Apple created for a family of PDAs and PDA-like products. Apple's line of hand-held computing devices, such as models of the MessagePad and the eMate 300, ran the Newton operating system. Newton also ran on clone devices.

The MessagePad was physically similar to a latter-day Personal Digital Assistant (PDA), with an active LCD screen for on-screen tapping, writing, and drawing. The original model was introduced in 1993 with Newton OS 1.0. It had a 20 MHz ARM 610 processor, 640KB RAM, and 4MB ROM. The last model—the MP2100—was introduced in late 1997. It had a 161.9 MHz StrongARM processor, 8MB RAM, and 8MB ROM. It ran Newton OS 2.1.

The eMate was a portable⁶⁸ computer with a 480×320 pixel active LCD screen and a keyboard. Its screen had a 0.30 mm dot pitch, supported 16 levels of gray, and had a yellow-green luminescent backlight.

The Newton System Software was logically divided into three parts: Newton OS at the lowest level, System Services, and Application Components.

1.11.2.1. *Newton OS*

The operating system was preemptive and multitasking. It could be considered as a modular set of tasks, each dedicated to specific functionality such as scheduling, task management, inter-task communications, memory management, power management, and various interactions with hardware. A low-level, extensible communi-

⁶⁸ The eMate 300 measured 289.6×305.0×53.3 mm, and weighed 4.2 lbs.

cations subsystem managed serial hardware, infrared,⁶⁹ and AppleTalk networking. This subsystem was extensible in that new protocols could be dynamically added and removed.

1.11.2.2. System Services

Many system services ran atop the operating system, such as Book Reader, Endpoint Communications, Filing, Find, Imaging and Printing, Intelligent Assistant, Object Storage System, Routing and Transport, Stationery, Text Input and Recognition, and View System.

1.11.2.3. Application Components

These included the NewtonScript Application Program and the user interface that ran atop System Services. Newton applications, both built-in and third party, ran in a single operating system task.

Newton used a sophisticated, modeless input recognition system that could recognize text, shapes, and gestures. The text recognizer could handle printed, cursive, or mixed handwriting. The shape recognizer could recognize both simple and complex geometric shapes. A descendant of this recognition technology exists as *Inkwell* in Mac OS X.

Dylan

A programming language called *Dylan* was considered as a candidate for being the primary language for developing Newton applications. Dylan is an object-oriented, Lisp-like, dynamic language that combines the features of static (such as C and C++) and dynamic (such as Lisp) languages. It was originally developed at Apple in collabo-

⁶⁹ Newton devices could exchange information with each other using infrared wireless transmission.

ration with Carnegie Mellon University and Harlequin Inc. One of Dylan's primary design goals was to be a suitable language for commercial software development. Apple abandoned Dylan in 1995.

1.11.3. The iPod's Operating System

Apple's successful iPod portable music player runs a proprietary operating system. When the first iPod was released in 2001, its software's "About" section mentioned *PortalPlayer*, a company that offers platform suites to manufacturers developing portable digital entertainment devices. A small company called Pixo was also credited. Pixo's focus was on developing a wireless software platform and services for phone manufacturers. The Pixo software platform consisted of components such as the following:

- Pixo Kernel
- Pixo Toolbox
- Pixo Application Framework
- Pixo User-interface Builder
- Pixo Platform Applications
- Pixo Partner Applications
- Pixo Internet Microbrowser

Pixo was acquired by Sun Microsystems in 2003.

The iPod uses PortalPlayer's *Digital Media Platform*, which is marketed as a turnkey solution consisting of System-On-Chip integrated circuits (ICs), a customizable firmware suite, integrated third party services, PC software, and other components. The iPod uses PortalPlayer's PP50xx chip, which contains two ARM7TDMI processor cores. Its embedded operating system, along with its encoding and decoding components, also come from PortalPlayer.

Pixo's software, particularly the Toolbox, provided the foundation on which the iPod's user interface was originally designed and implemented by Apple. The Pixo Toolbox provided modules for memory management, low-level graphics such as bitmaps, boxes, lines, and text, Unicode, collection classes, resource database, and standard libraries. Pixo's range of applications included Address Book, Calculator, Calendar, Email, Graphical World Clock, Memo Maker, Todo List, and PC Synchronization.