A ~~Brief~~, Opinionated History of the API

Joshua Bloch
josh@bloch.us

A while back, some guy asked me “How did APIs develop in the history of programming?”

OK, so it was a Federal Court judge…

…but still, you have to admit, it's a good question

United States District Court
For the Northern District of California

IN THE UNITED STATES DISTRICT COURT

FOR THE NORTHERN DISTRICT OF CALIFORNIA

ORACLE AMERICA, INC.,
Plaintiff,
v.
GOOGLE INC.,
Defendant.
_______________________________/

No. C 10-03561 WHA

REQUEST FOR FURTHER BRIEFING

The Court has read the recent briefs and thanks counsel for the excellent work on both sides. By **THURSDAY AT NOON** both sides shall please file further briefing on whether computer programming languages have been held to be copyrightable or should be so held, developing the arguments much more fully than were done in the recent briefs. Also please summarize the expected trial evidence on the extent to which the 37 APIs should be or are deemed part of the Java programming language. What will be the expected trial evidence as to how APIs are regarded in other programming languages like C ++ . How did APIs develop in the history of programming, according to the trial evidence? Each side may have up to ten pages. Please don't simply point out that the other side cannot find case law to support its position.

IT IS SO ORDERED.

Dated: April 6, 2012.

WILLIAM ALSUP
UNITED STATES DISTRICT JUDGE

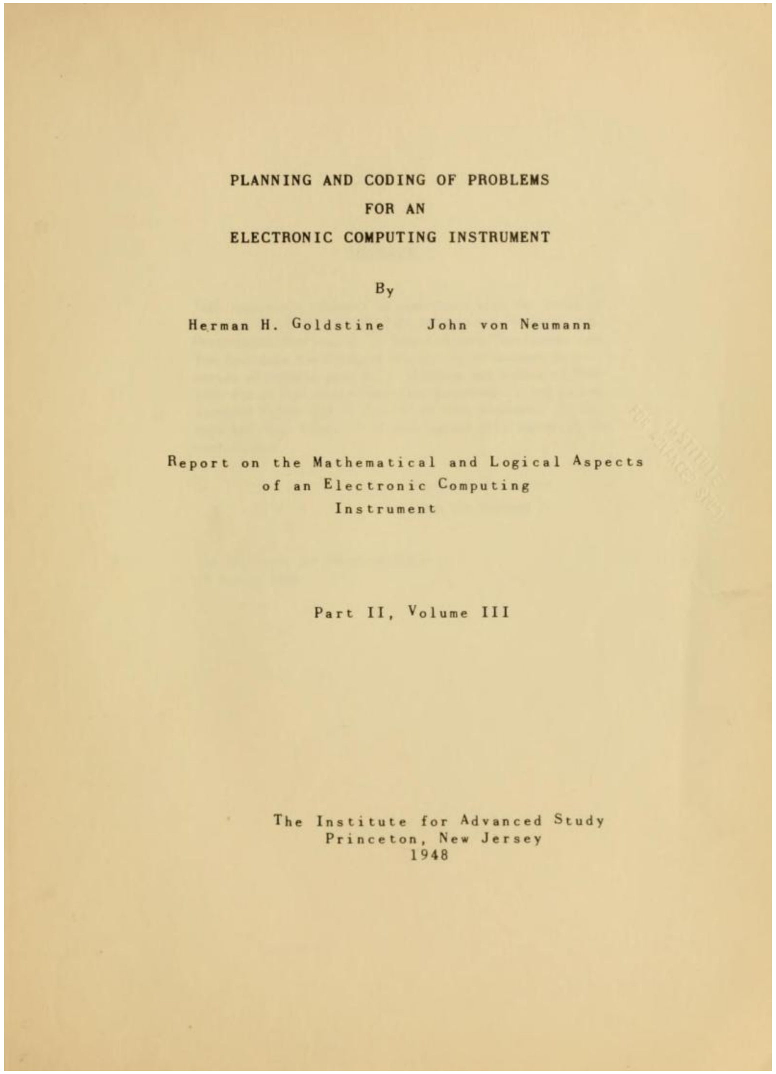
Outline

I. Who invented the API?
II. What exactly is an API? (A whirlwind tour)
III. How does an API come to be?
IV. What makes an API successful?
V. A legal digression
VI. Conclusion

Who invented the subroutine library?

- Term first appeared in Herman Goldstine and John von Neumann's “Planning and Coding of Problems for an Electronic Computing Instrument–Part II, Volume III” (Institute for Advanced Study, Princeton University, 1948)
  - First account of programming methodology for a stored-program computer (though none existed)
  - Made its way to every lab trying to build a computer
  - **Contains key idea: Most programs will make use of common operations. Library subroutines would reduce amount of new code and errors**

Here it is in black and (formerly) white



PLANNING AND CODING OF PROBLEMS
FOR AN
ELECTRONIC COMPUTING INSTRUMENT

By

Herman H. Goldstine   John von Neumann

Report on the Mathematical and Logical Aspects
of an Electronic Computing
Instrument

Part II, Volume III

The Institute for Advanced Study
Princeton, New Jersey
1948

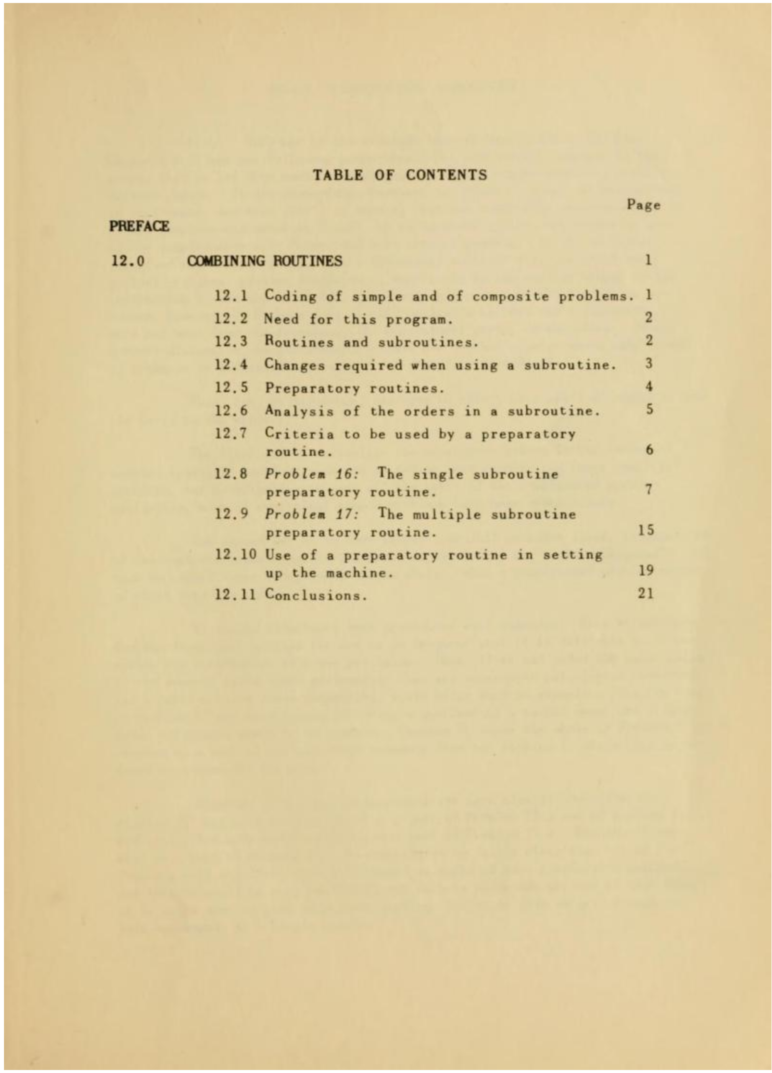


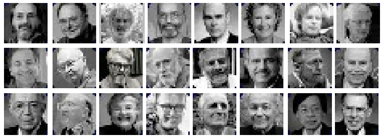
TABLE OF CONTENTS

A.M. TURING CENTENARY CELEBRATION WEBCAST

Search TYPE HERE

MORE ACM AWARDS

A.M. TURING AWARD



A.M. TURING AWARD WINNERS BY...

ALPHABETICAL LISTING YEAR OF THE AWARD RESEARCH SUBJECT

PHOTOGRAPHS

BIRTH:

26 June 1913, Dudley, England

DEATH:

29 November 2010, Cambridge, England

EDUCATION:

King Edward VI Grammar School, Stourbridge, England; BA (1934 - mathematics), MA (1936), PhD (1937 - physics) St Johns College, Cambridge University, England; Honorary Degrees: Newcastle-upon-Tyne, Hull, Kent, City of London, Bath, Amsterdam, Munich, Linköping, Cambridge University, University of Pennsylvania

EXPERIENCE:

Head of Computer Laboratory, Cambridge University, 1945-1980; Professor of Computer Technology, 1965-80; Fellow, St John's College, Cambridge, 1950 – 2010.

HONORS AND AWARDS:

Fellow, Royal Society, 1956; First President, British Computer Society, 1957-60, Distinguished Fellow, 1973; Foreign Honorary Member, American Academy of Arts and Sciences, 1974; Fellow, Royal Academy of Engineering, London, 1976; Foreign Associate, US National Academy of Engineering, 1977; Foreign Corresponding Member, Royal Spanish Academy of Sciences, 1979; Foreign Associate, US National Academy of Sciences, 1980; Foreign Corresponding Member, Spanish Academy of Engineering, 1999; Honorary Freeman, The Worshipful Company of Scientific Instrument Makers, 2000; Turing Lecturer, Association for Computing Machinery, 1967; Harry Goode Memorial Award, American Federation for Information Processing Societies, 1968; Eckert-Mauchly Award, Association for Computing Machinery and IEEE Computer Society, 1980; IEEE Computer

MAURICE V. WILKES

United Kingdom – 1967

CITATION

Professor Wilkes is best known as the builder and designer of the EDSAC, the first computer with an internally stored program. Built in 1949, the EDSAC used a mercury delay line memory. He is also known as the author, with Wheeler and Gill, of a volume on "Preparation of Programs for Electronic Digital Computers" in 1951, in which program libraries were effectively introduced.

SHORT ANNOTATED BIBLIOGRAPHY | ACM DL AUTHOR PROFILE | ACM TURING AWARD LECTURE | RESEARCH SUBJECTS | ADDITIONAL MATERIALS

Maurice Vincent Wilkes was born 26 June1913 in Dudley, in the county of Staffordshire in the English Midlands. His father was a financial officer for the estate of the Earl of Dudley which had extensive mining interests. His mother was a housewife. He was educated at King Edward VI Grammar School, Stourbridge. In his teens he built crystal sets, read *Wireless World*, and eventually gained a radio amateurs license – a background which proved useful when it came to building electronic computers two decades later. He entered St Johns College, Cambridge University, in 1931, where he read mathematics.

In October 1935 he became a research student at the Cavendish Laboratory, Cambridge University, working on the propagation of long radio waves. The following spring, he attended a lecture by Douglas Hartree, a computing expert and professor of mathematical physics at Manchester University. Hartree described the "differential analyzer" invented by Vannevar Bush at MIT. This was an analog computing machine for the integration of differential equations. Hartree had built a model differential analyzer from Meccano (a British constructor toy similar to the American *Erector Set*), which proved surprisingly useful. A copy of this machine was built at Cambridge under the direction of John Lennard-Jones, professor of theoretical chemistry, and Wilkes became an enthusiastic user. In early 1937, the University set up a Computing Laboratory under the direction of Lennard-Jones, and Wilkes was appointed assistant director from October 1937.

On the outbreak of war, the Computing Laboratory was taken over by the military. Wilkes joined the scientific war effort and worked on radar and operations research. This gave him an ideal background, and a network of contacts, for building computers after the war.

In October 1945, Wilkes returned to Cambridge to take full charge of what was now called the Mathematical Laboratory. In May 1946, he was visited by L. J. Comrie, a pioneer in mechanical computation, who brought with him a copy of the *First Draft of a Report on the EDVAC* written by John von Neumann, summarizing the deliberations of the computer group at the Moore School of Electrical Engineering, University of Pennsylvania. The Moore School had just completed the ENIAC, the world's first electronic computer for defense calculations, and the EDVAC was the design for a follow up machine. Wilkes had never seen the report before and stayed up late into the night reading it. He recognized it at once as "the real thing" and decided that the laboratory had to have one.

Later in 1946, Wilkes was invited to attend a summer school in computer design organized by the Moore School. Because of difficulties getting a transatlantic passage, he did not arrive on the course until mid-August, by which time he had missed more than half. Rarely short on confidence, Wilkes decided he had not missed much of consequence. Sailing home on the *Queen Mary* he began the design of a machine he called the Electronic Delay Storage Automatic Calculator – EDSAC for short, an acronym consciously chosen as a tribute to the EDVAC.

Work started on building the EDSAC in early 1947. Almost everything had to be done from first principles—memory technology, electronic arithmetic and logic, and control circuits. Cambridge University was at the center of UK computing at this time, in part because of the fortnightly colloquia Wilkes established, which were attended by members of almost every computer project in the country.

The EDSAC sprang into life on 6 May 1949, the world's first practical electronic computer. Manchester University had got there first in June 1948 with an experimental machine, but the EDSAC was the first capable of running realistic programs. By the beginning of 1950 the Laboratory was offering a regular computing service.

Why did Wilkes get the Turing Award if von Neumann & Goldstine had the idea?

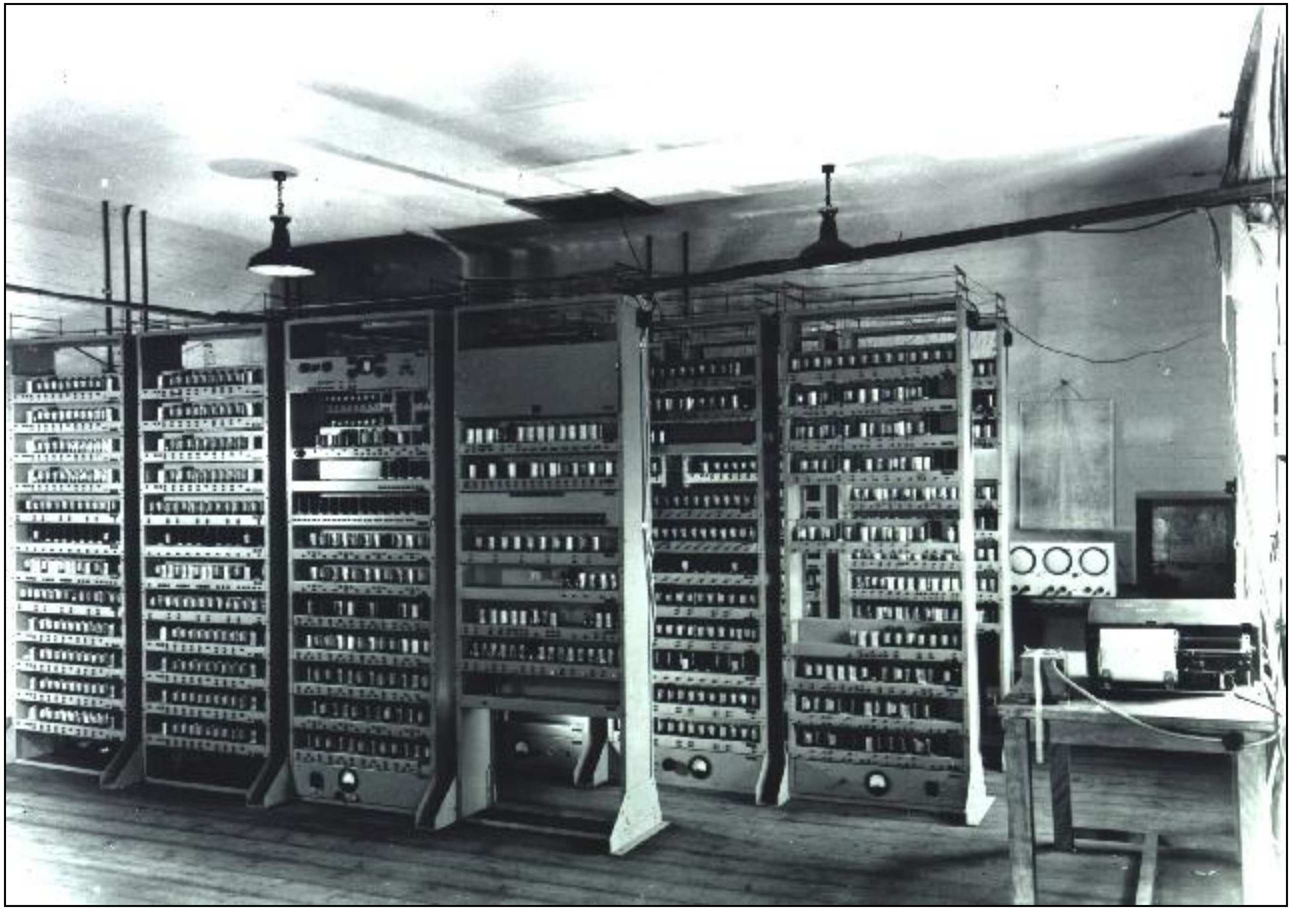
Goldstine & von Neumann were peddling vaporware!

“In ‘Planning and Coding’ a program consisted of a ‘main routine’ and a set of subroutines from the library; the routines were loaded in memory, starting at the low end. Because a subroutine might end up in any arbitrary location α, it would be coded relative to 0. Once in memory it would have to be adjusted by adding α to each address. This adjustment was to be performed by a ‘preparatory routine’ in the high end of memory. Goldstine and von Neumann's preparatory routine…would have required extensive operator intervention and it is difficult to imagine that it would ever have worked in practice.”

Martin Campbell-Kelly, “From Theory to Practice: The Invention of Programming, 1947-51”

Wilkes's machine was the real deal

EDSAC – University of Cambridge Mathematical Laboratory



EDSAC vital statistics

Electronic Delay Storage Automatic Calculator

- **World’s first stored-program computer**
  – Came to life on May 6, 1949; immediately useful
- 650 instructions per second
- 512 then 1024 17-bit (!) words of memory
  – Stored in mercury ultrasonic delay lines
- Input: paper tape, Output: teleprinter (6⅔ CPS)
- 3000 tubes, 12 kW power consumption
- Occupied a room 15' by 12'
- Name is homage to EDVAC (von Neumann & Goldstine)
  – EDVAC didn't run until 1951, on a limited basis

Why did Wilkes get done so much faster?

He kept it simple!

"The reason for the rapid completion, which was well ahead of any American computer, was that Wilkes wanted to have a machine as practical computing instrument rather than a machine of the highest technological performance. To this end he kept the EDSAC simple—conservative in its electronics and conventional in its architecture."

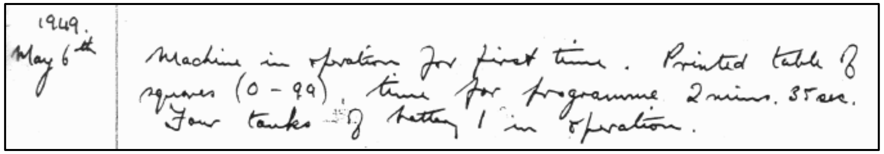
– Martin Campbell-Kelly, ibid.

"Importantly, the [recently rediscovered] drawings clearly show that the aim of EDSAC's designer, Sir Maurice Wilkes, was to produce a working machine quickly rather than to create a more refined machine that would take longer to build. The refinements could come later—and many did as the sequence of diagrams over the five-year period shows."

– Andrew Herbert, leader of the EDSAC Project at the [UK] National Museum of Computing

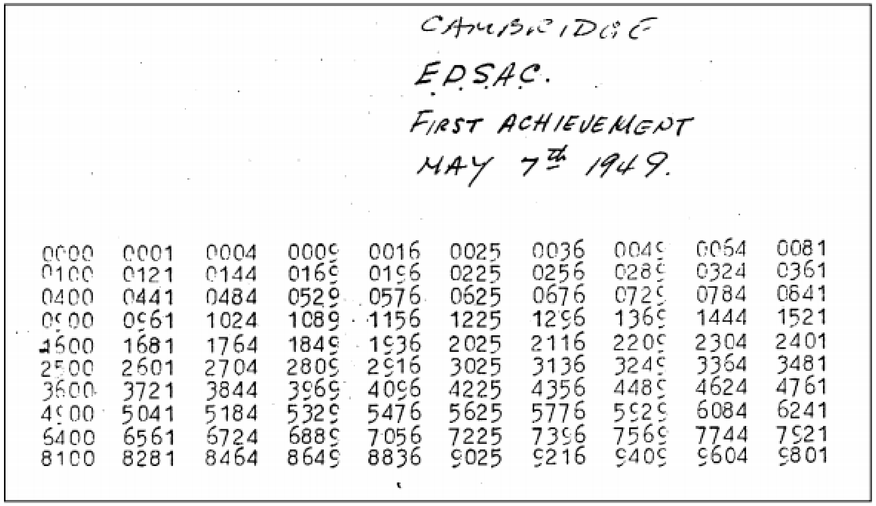
The first two EDSAC programs were toys

- First program printed the first 100 squares
  – Written by Wheeler, ran May 6, 1949



1949.
May 6th

Machine in operation for first time. Printed table of squares (0 – 99), time for programme 2 mins. 35 sec. Four tanks of battery 1 in operation.



CAMBRIDGE
EDSAC.
FIRST ACHIEVEMENT
MAY 7th 1949.

0000	0001	0004	0009	0016	0025	0036	0049	0064	0081
0100	0121	0144	0169	0196	0225	0256	0289	0324	0361
0400	0441	0484	0529	0576	0625	0676	0729	0784	0841
0900	0961	1024	1089	1156	1225	1296	1369	1444	1521
1600	1681	1764	1849	1936	2025	2116	2209	2304	2401
2500	2601	2704	2809	2916	3025	3136	3249	3364	3481
3600	3721	3844	3969	4096	4225	4356	4489	4624	4761
4900	5041	5184	5329	5476	5625	5776	5929	6084	6241
6400	6561	6724	6889	7056	7225	7396	7569	7744	7921
8100	8281	8464	8649	8836	9025	9216	9409	9604	9801

- Second program printed 170 primes
  – Written by Wilkes, ran May 10, 1949

A simple software architecture sufficed for these toy programs

- First 30 words were “initial orders” (boot loader)
  - Stored on electromechanical telephone switch
  - Pressing start button loaded initial orders to memory and began execution
- Loaded program from tape into memory
  - Program was loaded starting at location 30
- Technically it was a JIT assembler!
  - Programs were written in assembly language
  - Wilkes and Wheeler decreed that users would *never* have to cope with binary machine code
- Wheeler wrote the initial orders

Wilkes first *real* program: Airy's Integral

Solution to differential equation $y'' + xy = 0$

- *"By June 1949 ... I was trying to get working my first non-trivial program, which was for the numerical integration of Airy's differential equation. It was on one on my journeys between the EDSAC room and the punching equipment that 'hesitating at the angles of the stairs' the realization came over me that a good part of the remainder of my life was going to be spent in finding the errors in my own programs."* – Maurice Wilkes, Memoirs
- **Wilkes saw subroutines as a way out**
  – He gave problem to his Ph.D. student, Wheeler

Wheeler's architecture for subroutines

Finished September, 1949

- Wheeler devised “coordinating orders” to augment initial orders
  - “Pseudo-orders” for relocation of subrountines, parameter assignment, etc.
  - Initial orders ran coordinating orders interpretively
  - **Required no manual intervention**
- Program consisted of main, subroutines, and coordinating orders all on a single tape
  - Initial orders totaled a mere 42 instructions!
  - Constrained by capacity of telephone switches
  - Wilkes, not prone to overstatement, described Wheeler's work as “a *tour de force* of ingenuity”

Wheeler's subroutine linkage technique

"The Wheeler Jump"

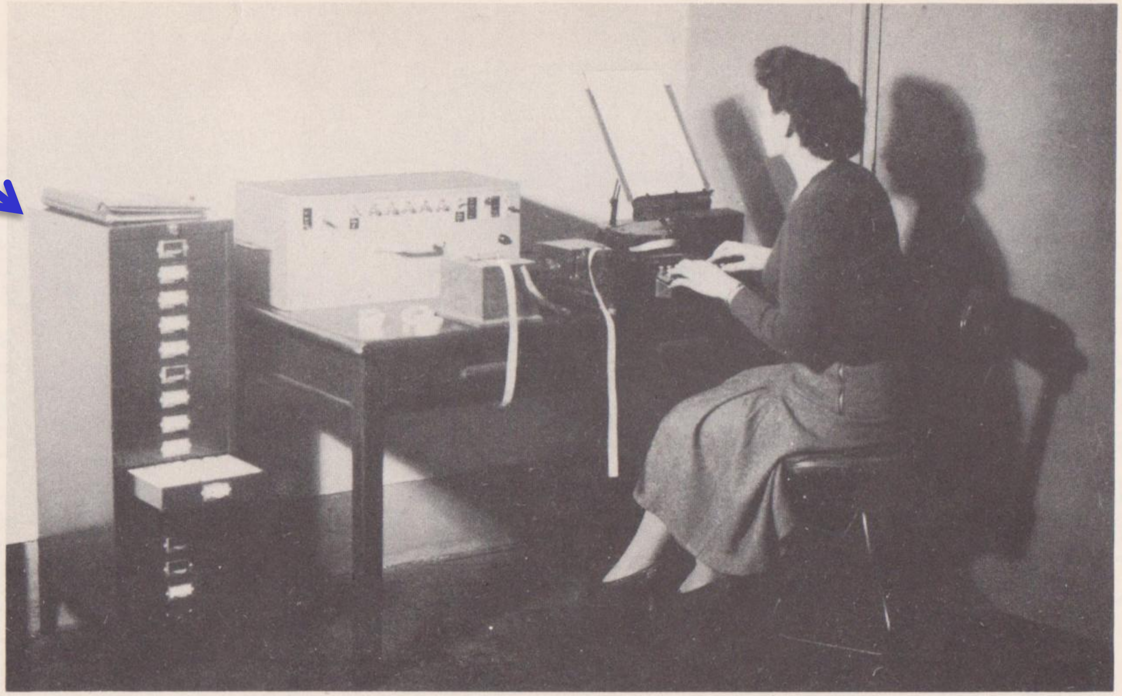
Number of storage location	Order	Explanation (Accumulator contains zero at this point)
m	A m F	adds number representing A m F into the accumulator (this is negative, since A corresponds to -4/16)
m+1	G n F	transfers control to n, since number in the accumulator is negative

The orders in the subroutine are as follows:

	G K	control combination; puts the value of n in 42
n	A 3 F	adds U 2 F to contents of accumulator (A m F) forming E m+2 F (link order) since A ≡ -4/16, U ≡ 7/16, whence A + U ≡ 3/16 ≡ E
n+1	T p+2 θ	plants link order in (n+p+2) (code letter θ causes C(42), i.e. n, to be added to address during input)
n+2 ... n+p+1		operational orders of the subroutine, p in number. These leave the accumulator empty
n+p+2	Z F	becomes E m+2 F (link order) as result of order in (n+1)

Fig. 4. The Wheeler Jump (from *WWG*, p. 22)

The EDSAC subroutine library



The library of tapes on which subroutines are punched is contained in the steel cabinet shown on the left. The operator is punching a program tape on keyboard perforator. She can copy mechanically tapes taken from the library on to the tape she is preparing by placing them in the tapereader shown in the center of the photograph.

The EDSAC subroutine library

Another view

PART II

SPECIFICATIONS OF LIBRARY SUBROUTINES

Each subroutine is distinguished by a letter denoting its category and a serial number within that category. The categories are as follows.

Category	Subject
A	Floating point arithmetic.
B	Arithmetical operations on complex numbers.
C	Checking.
D	Division.
E	Exponentials.
F	General routines relating to functions.
G	Differential equations.
J	Special functions.
K	Power series.
L	Logarithms.
M	Miscellaneous.
P	Print and layout.
Q	Quadrature.
R	Read (i.e., Input).
S	nth root.
T	Trigonometrical functions.
U	Counting operations.
V	Vectors and matrices.

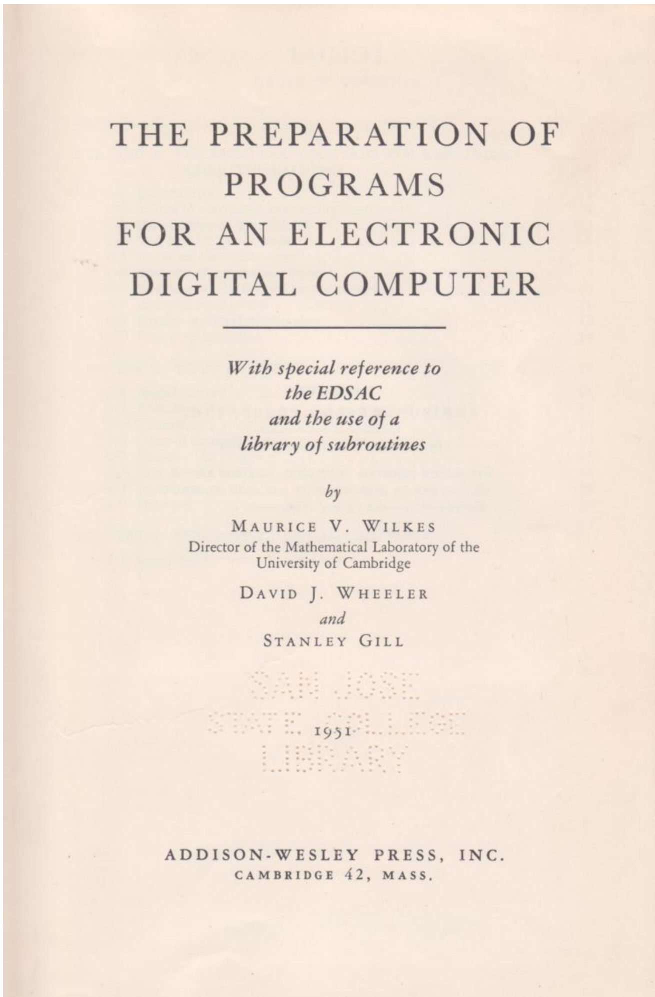
In the specifications on succeeding pages the following information is given in abbreviated form immediately beneath the title of each subroutine:

1. Type of subroutine, i.e., whether open, closed, interpretive, or special.
2. Restriction on address of first order. If the word "even" appears it denotes that the first order must have an even address; if no note appears it indicates that the address may be either odd or even.
3. Total number of storage locations occupied by the subroutine.
4. Addresses of any storage locations needed as working space by the subroutine.
5. Approximate operating time (not possible to state in all cases).

The gaps in the numbering within each category correspond to subroutines which have become obsolete.

72

This book introduced the world to subroutine libraries – WWG



- The world's first text on computer programming
- The definitive work on programming until high level languages arose
- Contained entire API
  - Meticulously documented
- Specifically cited in Wilkes's Turing Award
- Tech Report Sept. 1950
- Published spring 1951

Wheeler presented key ideas in 1952 paper

"The Use of Sub-routines in Programmes."
ACM National Meeting, Pittsburgh, Pa., May 2-3, 1952

- The paper described these concepts
  - The subroutine
  - The subroutine library
  - Generality vs. performance tradeoffs
  - The importance & difficulty of library documentation
  - Information hiding
  - The *interpretive routine*
  - The interpretive debugger (Gill)
  - Higher-order functions(!)
- And they actually implemented all this stuff!

A remarkable passage from the paper

"It should be pointed out that the preparation of a library sub-routine requires a considerable amount of work. This is much greater than the effort merely required to code the sub-routine in its simplest possible form. It will usually be necessary to code it in the library standard form and this may detract from its efficiency in time and space. It may be desirable to code it in such a manner that the operation is generalized to some extent. However, even after it has been coded and tested there still remains the considerable task of writing a description so that people not acquainted with the interior coding can nevertheless use it easily. This last task may be the most difficult."

42 years later, David Parnas wrote this

…and it was news to many

"Reuse is something that is far easier to say than to do. Doing it requires both good design and very good documentation. Even when we see good design, which is still infrequently, we won't see the components reused without good documentation."

– D. L. Parnas, Software Aging. Proceedings of the 16th International Conference on Software Engineering, 1994

Another remarkable passage

The conclusion of Wheeler's 1952 paper

"The prime objectives to be borne in mind when constructing sub-routine libraries are simplicity of use, correctness of codes and accuracy of description. All complexities should—if possible—be buried out of sight."

54 Years later, I wrote this

*…and it was **still** news to many*

“APIs should be easy to use and hard to misuse. It should be easy to do simple things; possible to do complex things; and impossible, or at least difficult, to do wrong things.

Documentation matters. No matter how good an API, it won’t get used without good documentation. Document every exported API element: every class, method, field, and parameter.

Minimize accessibility; when in doubt, make it private. This simplifies APIs and reduces coupling.”

– J. Bloch, How to Design a Good API and Why it Matters. Proceedings of OOPSLA 2006.

Wheeler’s paper was only two pages long!

THE USE OF SUB-ROUTINES IN PROGRAMMES

D. J. Wheeler
Cambridge & Illinois Universities

A sub-routine may perhaps best be described as a self-contained part of a programme, which is capable of being used in different programmes. It is an entity of its own within a programme. There is no necessity to compose a programme of a set of distinct sub-routines; for the programme can be written as a complete unit, with no divisions into smaller parts. However it is usually advantageous to arrange that a programme is comprised of a set of sub-routines, some of which have been made specially for the particular programme while others are available from a 'library' of standard sub-routines. The reasons for this will be discussed below.

When a programme has been made from a set of sub-routines the breakdown of the code is more complete than it would otherwise be. This allows the coder to concentrate on one section of a programme at a time without the overall detailed programme continually intruding. Thus the sub-routines can be more easily coded and the tested in isolation from the rest of the programme. When the entire programme has to be tested it is with the foreknowledge that the incidence of mistakes in the sub-routines is zero (or at least one order of magnitude below that of the untested portions of the programme!)

If library sub-routines exist for the major part of a code then the task of constructing the remaining part of the programme is naturally very much less than if the code had to be written from the very beginning. However, one will rarely have available sub-routines to do exactly what is required and thus a certain amount of manipulation may be necessary before a given sub-routine can be used. Even so, it is usually far easier to use a sub-routine which will meet the specifications with a small amount of manipulation than to make one specially for the purpose.

It should be pointed out that the preparation of a library sub-routine requires a considerable amount of work. This is much greater than the effort merely required to code the sub-routine in its simplest possible form. It will usually be necessary to code it in the library standard form and this may detract from its efficiency in time and space. It may be desirable to code it in such a manner that the operation is generalized to some extent. However, even after it has been coded and tested there still remains the considerable task of writing a description so that people not acquainted with the interior coding can nevertheless use it easily. This last task may be the most difficult.

·Besides the organization of the individual sub-routines there remains the method of the general organization of the library. How are the sub-routines going to be stored? Are they going to be stored on punched paper tape or are they going to be available in the auxiliary store of the machine? Usually it will be found that it is not possible to write the sub-routines such that they may be put into arbitrary positions in the store-although in certain machines this is now possible. Usually some translation process will have to be arranged so that an invariant form of sub-routine stored on some medium such as paper tape can be translated to the form required in a particular application. This translation is possible because fixed rules can be set up for adjusting a sub-routine so that it becomes correct in the set of locations in which it is put and used.

Page 235

One next considers the methods by which sub-routines can be used. There are a number of different ways of transferring control to sub-routines and arranging that control is returned to the appropriate point to which it is required. One of the simpler methods was that used for the closed sub-routines of the EDSAC in which it was arranged that when the sub-routine had performed its part of the computation then control was returned to a point in the main programme immediately after the orders which had called it into use. This has been described in detail by Goldstine. This perhaps facilitates thinking of a sub-routine as an 'order' of the machine although it is usually of a more complicated kind than that wired in the circuits of the machine.

A second more interesting type of sub-routine is an interpretive routine. In this type of routine it is arranged that a sequence of operations is performed each time the sub-routine is called into action, each operation being determined by one parameter or 'order' in a list of such 'orders'. This type of sub-routine is particularly useful for coding certain special types of arithmetic for the machine, for example, floating point arithmetic in which numbers are expressed as $X \times 10^{p}$.
Thus the sub-routine executes the 'orders' in the list in a similar fashion to the way that the machine obeys ordinary orders. However, the orders that it does are determined by the parts of the sub-routine, and so can be made to do any kind of operation or arithmetic.

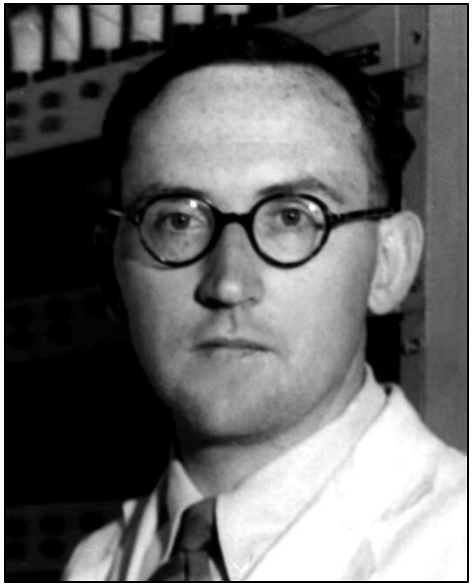
One extension of an interpretive routine is a checking routine which is so arranged that the 'orders' that are obeyed are identical with those of the machine. However, the interpretive routine retains control and so it is possible to print out extra information about the course of the programme. This extra information makes it possible to follow the meanderings of the program in detail thus helping to locate the errors of a programme. This is not a good method of finding errors in programmes as it takes a long time and the programmers knowledge of the programme is not utilized - as it should be - in tracing the fault. However, it is a useful last resort and can quite often give out information about a code which would be difficult to find in any other way.

Sub-routines seem to have two distinct uses in programmes. The first and most obvious use is for the evaluation of functions, a simple example being the evaluation of sine x given x. The second use is for the organization of processes such as the integration of a function given f(x). This second type requires more consideration to make it useful and general. For instance how should f(x) be specified for the sub-routine? One obvious and useful way is to allow the integrating sub-routine access to an auxiliary sub-routine which is capable of evaluating f(x).

The above remarks may be summarized by saying sub-routines are very useful-although not absolutely necessary-and that the prime objectives to be born in mind when constructing them are simplicity of use, correctness of codes and accuracy of description. All complexities should-if possible-be buried out of sight.

Page 236

The inventors of the subroutine library



“Adviser”
Sir Maurice V. Wilkes
1913 – 2010

“Primary Inventor”
David J. Wheeler FRS
1927 – 2004

“Minor Contributor”
Stanley Gill
1926 – 1975

Why didn't Wilkes and Wheeler discuss the API as distinct from library?

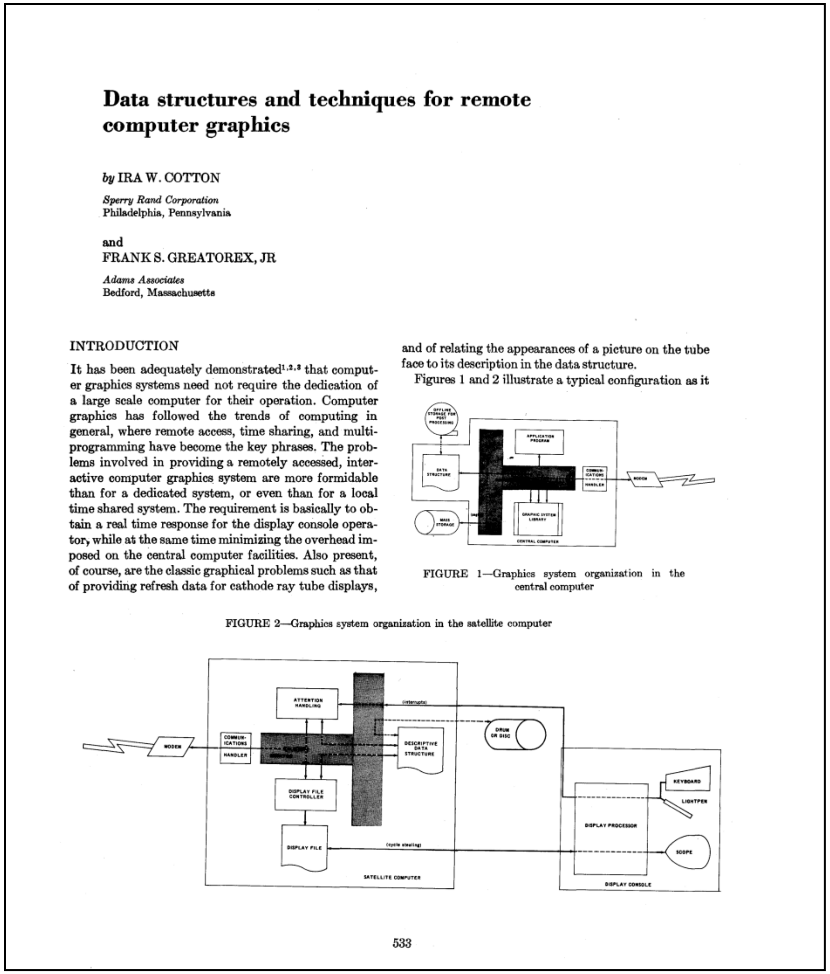
- **Because the two were largely isomorphic**
- There was only one machine architecture
  - The notion of portability didn't exist
- There were no legacy programs
  - The notion of backward compatibility didn't exist
- Little reason for them to discuss API separately
  - But they clearly understood API design principles

The field progressed, and existing subroutine libraries were *reimplemented*

- New hardware was built
  - Reimplementing existing APIs enabled *portability*
    - Preserved investment in code and education
- New algorithms were devised
  - Reimplementing existing APIs improved performance of existing application programs
- Gave life to APIs, independent of libraries

I *think* this 1968 paper is the first to use the term *Application Program Interface*

December 1968 AFIPS Fall Joint Computer Conference



Data structures and techniques for remote computer graphics

by IRA W. COTTON

Sperry Rand Corporation
Philadelphia, Pennsylvania

and
FRANK S. GREATOREX, JR

Adams Associates
Bedford, Massachusetts

INTRODUCTION

It has been adequately demonstrated[1,2,3] that computer graphics systems need not require the dedication of a large scale computer for their operation. Computer graphics has followed the trends of computing in general, where remote access, time sharing, and multiprogramming have become the key phrases. The problems involved in providing a remotely accessed, interactive computer graphics system are more formidable than for a dedicated system, or even than for a local time shared system. The requirement is basically to obtain a real time response for the display console operator, while at the same time minimizing the overhead imposed on the central computer facilities. Also present, of course, are the classic graphical problems such as that of providing refresh data for cathode ray tube displays, and of relating the appearances of a picture on the tube face to its description in the data structure.

Figures 1 and 2 illustrate a typical configuration as it

FIGURE 1—Graphics system organization in the central computer

FIGURE 2—Graphics system organization in the satellite computer

533

- I don’t know for sure
  - Earliest I could find
- Merriam-Webster says first use is 1974

The paper first dances around the term…

“Normally, the interface between application programs and the system is desired via FORTRAN-type subroutine calls.”

“The system has been designed to be essentially hardware independent in the sense that the implementation either in the central or satellite computers may be recoded for different or improved hardware, while still maintaining the same interface with each other and with the application program.”

…And then runs with it

“Finally, hardware independence at the central computer means that a consistent application program interface could be maintained if that computer were replaced; this also applies to the satellite processor if hardware independence can be achieved there also. Eventual replacement of at least a portion of the hardware is almost a certainty, given the rapid rate of new developments in computer technology. A sufficiently flexible, hardware independent system guarantees that technological advances will not make the system prematurely obsolete.”

What’s going on here?

- Authors understood the importance of APIs
  - **They allow implementations to be replaced without harm to clients**
- So the API has life of its own, apart from library
  - And the concept deserves a name
- Many other people understood this too
  - Not a great intellectual achievement
- Libraries naturally give rise to APIs
  - **APIs weren’t invented so much as discovered**
  - Arguably Wheeler & Wilkes were “latent inventors”

II. What exactly constitutes an API?

- Wikipedia says this (as of 10/16/2014)
  - In computer programming, an application programming interface (API) specifies a software component in terms of its operations, their inputs and outputs ~~and underlying types~~. Its main purpose is to define a set of functionalities that are independent of their ~~respective~~ implementation, allowing both definition and implementation to vary without compromising each other.

I propose this simple definition

But I acknowledge that it still isn't perfect

An application programming interface (API) specifies a component in terms of its operations, their inputs, and outputs. Its main purpose is to define a set of functionalities that are independent of their implementation, allowing the implementation to vary without compromising the users of the component.

Definition suggests a two-part test

If you can answer yes to these two questions it's an API

1. Does it provide a set of operations defined by their inputs and outputs?
2. Does it admit reimplementation without compromising its users?

FORTRAN II standard library (1958)

28 math functions. Defined in the language manual

APPENDIX B

The chart summarizes the 20 built-in functions at present available as open subroutines on the FORTRAN II system tape.

Type of Function	Definition	No. of Args.	Name	Mode of Argument	Mode of Function
Absolute value	|Arg|	1	ABSF	Floating	Floating
			XABSF	Fixed	Fixed
Truncation	Sign of Arg times largest integer $\leq$ |Arg|	1	INTF	Floating	Floating
			XINTF	Floating	Fixed
Remaindering (see note 1 below)	Arg_1 (mod Arg_2)	2	MODF	Floating	Floating
			XMODF	Fixed	Fixed
Choosing largest value	Max (Arg_1, Arg_2, ...)	≥ 2	MAX0F	Fixed	Floating
			MAX1F	Floating	Floating
			XMAX0F	Fixed	Fixed
			XMAX1F	Floating	Fixed
Choosing smallest value	Min (Arg_1, Arg_2, ...)	≥ 2	MIN0F	Fixed	Floating
			MIN1F	Floating	Floating
			XMIN0F	Fixed	Fixed
			XMIN1F	Floating	Fixed
Float	Float fixed number	1	FLOATF	Fixed	Floating
Fix	Same as XINTF	1	XFIXF	Floating	Fixed
Transfer of sign	Sign of Arg_2 times |Arg_1|	2	SIGNF	Floating	Floating
		2	XSIGNF	Fixed	Fixed
Diminishing (see note 2 below)	Arg_1 (dim Arg_2)	2	DIMF	Floating	Floating
			XDIMF	Fixed	Fixed

NOTES: 1. The function MODF (Arg_1, Arg_2) is defined as $Arg_1 - [Arg_1/Arg_2]\ Arg_2$, where $[x]$ = integral part of x.
2. The function DIMF (Arg_1, Arg_2) is defined as Arg_1 - Min (Arg_1, Arg_2).

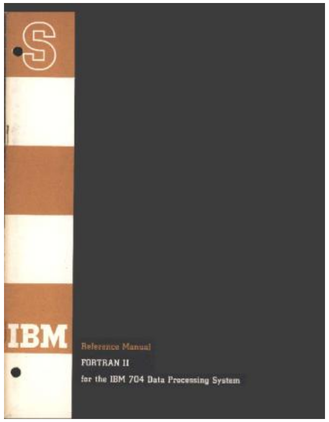
Library Functions

The Library functions are pre-written and may exist on the library tape or in prepared card decks. These functions constitute "closed" subroutines, i.e., instead of appearing in the object program for every reference that has been made to them in the source program, they appear only once, regardless of the number of references.

Hand-coded Library functions may be added to the library. Rules for coding these subroutines are given in Appendix D; those for adding them to the library are included in the FORTRAN II Operations Manual, Form C28-6066-4.

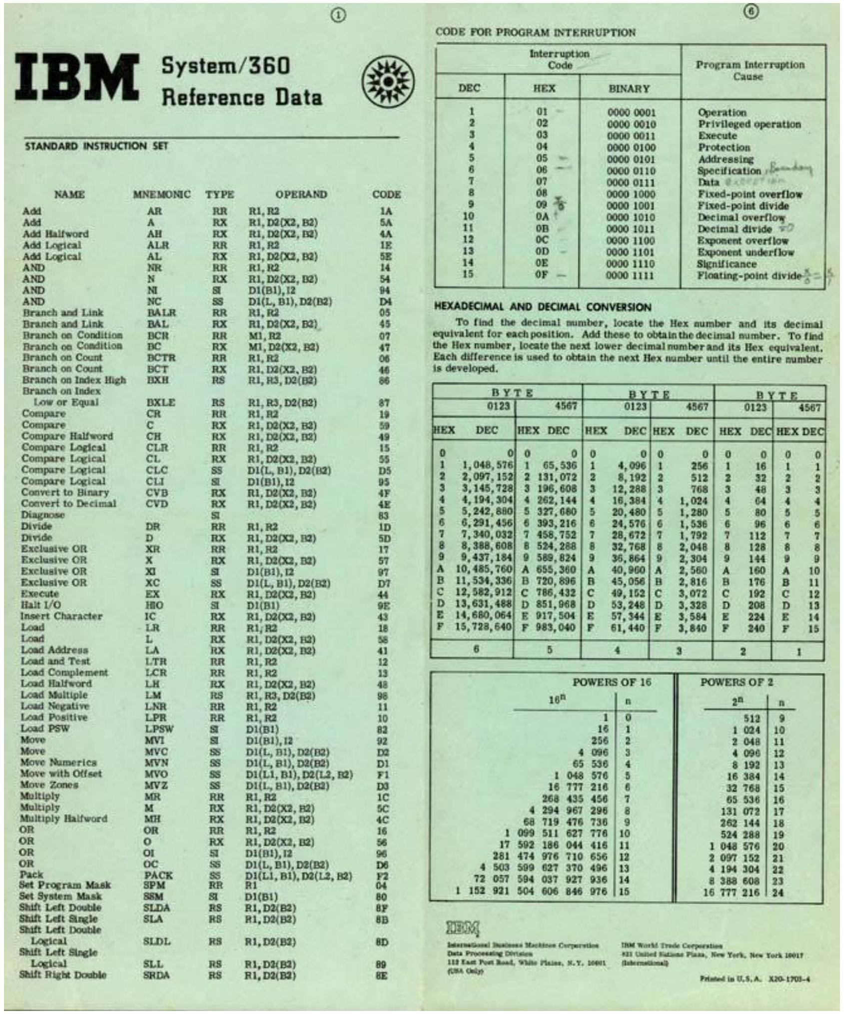
Seven Library functions are included in the FORTRAN II System. These are:

Name	Function
LOGF	Natural Logarithm
SINF	Trigonometric Sine
COSF	Trigonometric Cosine
EXPF	Exponential
SQRTF	Square Root
ATANF	Arctangent
TANHF	Hyperbolic Tangent



The IBM S/360 instruction set (1964)

Is an instruction set an API?



①

IBM System/360 Reference Data

STANDARD INSTRUCTION SET

NAME	MNEMONIC	TYPE	OPERAND	CODE
Add	AR	RR	R1, R2	1A
Add	A	RX	R1, D2(X2, B2)	5A
Add Halfword	AH	RX	R1, D2(X2, B2)	4A
Add Logical	ALR	RR	R1, R2	1E
Add Logical	AL	RX	R1, D2(X2, B2)	5E
AND	NR	RR	R1, R2	14
AND	N	RX	R1, D2(X2, B2)	54
AND	NI	SI	D1(B1), I2	94
AND	NC	SS	D1(L, B1), D2(B2)	D4
Branch and Link	BALR	RR	R1, R2	05
Branch and Link	BAL	RX	R1, D2(X2, B2)	45
Branch on Condition	BCR	RR	M1, R2	07
Branch on Condition	BC	RX	M1, D2(X2, B2)	47
Branch on Count	BCTR	RR	R1, R2	06
Branch on Count	BCT	RX	R1, D2(X2, B2)	46
Branch on Index High	BXH	RS	R1, R3, D2(B2)	86
Branch on Index Low or Equal	BXLE	RS	R1, R3, D2(B2)	87
Compare	CR	RR	R1, R2	19
Compare	C	RX	R1, D2(X2, B2)	59
Compare Halfword	CH	RX	R1, D2(X2, B2)	49
Compare Logical	CLR	RR	R1, R2	15
Compare Logical	CL	RX	R1, D2(X2, B2)	55
Compare Logical	CLC	SS	D1(L, B1), D2(B2)	D5
Compare Logical	CLI	SI	D1(B1), I2	95
Convert to Binary	CVB	RX	R1, D2(X2, B2)	4F
Convert to Decimal	CVD	RX	R1, D2(X2, B2)	4E
Diagnose		SI		83
Divide	DR	RR	R1, R2	1D
Divide	D	RX	R1, D2(X2, B2)	5D
Exclusive OR	XR	RR	R1, R2	17
Exclusive OR	X	RX	R1, D2(X2, B2)	57
Exclusive OR	XI	SI	D1(B1), I2	97
Exclusive OR	XC	SS	D1(L, B1), D2(B2)	D7
Execute	EX	RX	R1, D2(X2, B2)	44
Halt I/O	HIO	SI	D1(B1)	9E
Insert Character	IC	RX	R1, D2(X2, B2)	43
Load	LR	RR	R1, R2	18
Load	L	RX	R1, D2(X2, B2)	58
Load Address	LA	RX	R1, D2(X2, B2)	41
Load and Test	LTR	RR	R1, R2	12
Load Complement	LCR	RR	R1, R2	13
Load Halfword	LH	RX	R1, D2(X2, B2)	48
Load Multiple	LM	RS	R1, R3, D2(B2)	98
Load Negative	LNR	RR	R1, R2	11
Load Positive	LPR	RR	R1, R2	10
Load PSW	LPSW	SI	D1(B1)	82
Move	MVI	SI	D1(B1), I2	92
Move	MVC	SS	D1(L, B1), D2(B2)	D2
Move Numerics	MVN	SS	D1(L, B1), D2(B2)	D1
Move with Offset	MVO	SS	D1(L1, B1), D2(L2, B2)	F1
Move Zones	MVZ	SS	D1(L, B1), D2(B2)	D3
Multiply	MR	RR	R1, R2	1C
Multiply	M	RX	R1, D2(X2, B2)	5C
Multiply Halfword	MH	RX	R1, D2(X2, B2)	4C
OR	OR	RR	R1, R2	16
OR	O	RX	R1, D2(X2, B2)	56
OR	OI	SI	D1(B1), I2	96
OR	OC	SS	D1(L, B1), D2(B2)	D6
Pack	PACK	SS	D1(L1, B1), D2(L2, B2)	F2
Set Program Mask	SPM	RR	R1	04
Set System Mask	SSM	SI	D1(B1)	80
Shift Left Double	SLDA	RS	R1, D2(B2)	8F
Shift Left Single	SLA	RS	R1, D2(B2)	8B
Shift Left Double Logical	SLDL	RS	R1, D2(B2)	8D
Shift Left Single Logical	SLL	RS	R1, D2(B2)	89
Shift Right Double	SRDA	RS	R1, D2(B2)	8E

⑥

CODE FOR PROGRAM INTERRUPTION

Interruption Code			Program Interruption Cause
DEC	HEX	BINARY	
1	01	0000 0001	Operation
2	02	0000 0010	Privileged operation
3	03	0000 0011	Execute
4	04	0000 0100	Protection
5	05	0000 0101	Addressing
6	06	0000 0110	Specification
7	07	0000 0111	Data
8	08	0000 1000	Fixed-point overflow
9	09	0000 1001	Fixed-point divide
10	0A	0000 1010	Decimal overflow
11	0B	0000 1011	Decimal divide
12	0C	0000 1100	Exponent overflow
13	0D	0000 1101	Exponent underflow
14	0E	0000 1110	Significance
15	0F	0000 1111	Floating-point divide

HEXADECIMAL AND DECIMAL CONVERSION

To find the decimal number, locate the Hex number and its decimal equivalent for each position. Add these to obtain the decimal number. To find the Hex number, locate the next lower decimal number and its Hex equivalent. Each difference is used to obtain the next Hex number until the entire number is developed.

BYTE				BYTE				BYTE			
0123		4567		0123		4567		0123		4567	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0	0	0	0	0
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15
6		5		4		3		2		1	

POWERS OF 16

16^n	n
1	0
16	1
256	2
4 096	3
65 536	4
1 048 576	5
16 777 216	6
268 435 456	7
4 294 967 296	8
68 719 476 736	9
1 099 511 627 776	10
17 592 186 044 416	11
281 474 976 710 656	12
4 503 599 627 370 496	13
72 057 594 037 927 936	14
1 152 921 504 606 846 976	15

POWERS OF 2

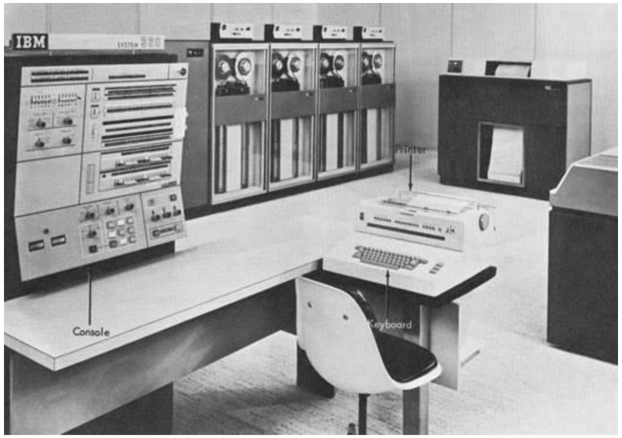
2^n	n
512	9
1 024	10
2 048	11
4 096	12
8 192	13
16 384	14
32 768	15
65 536	16
131 072	17
262 144	18
524 288	19
1 048 576	20
2 097 152	21
4 194 304	22
8 388 608	23
16 777 216	24

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
(USA Only)

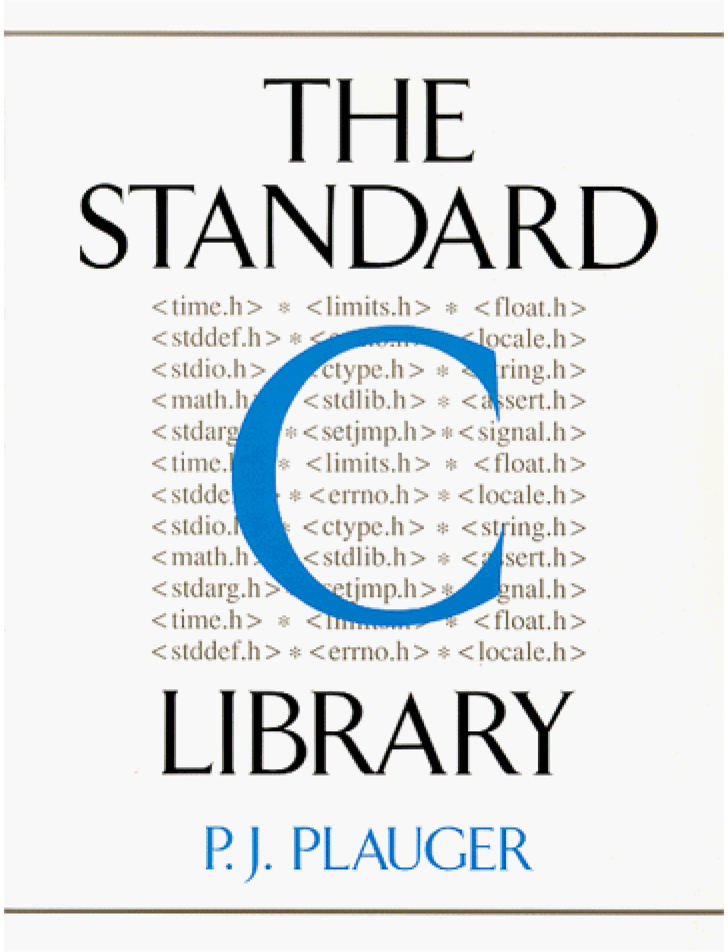
IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

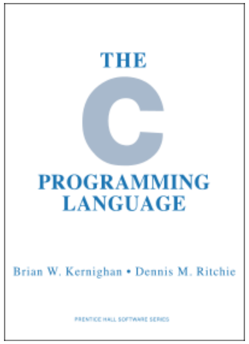
Printed in U.S.A. X20-1703-4



The C Standard Library (1975)

Newer language, bigger API (but not huge)





K & R on the C standard libraries (1978)

They understood the role of APIs in ensuring portability

Input and output facilities are not part of the C language. Nonetheless, real programs do interact with their environment. In this chapter we describe "the standard I/O library," a set of functions designed to provide a standard I/O programming interface, yet reflect only operations that can be provided on most modern operating systems. ***The routines are meant to be "portable," in the sense that they will exist in compatible form on any system where C exists, and that programs which confine their system interactions to facilities provided by the standard library can be moved from one system to another essentially without change.*** – *The C Programming Language* (1st Ed., p. 143)

Unix (6th Edition) system calls (1975)

Operating system kernels have APIs

UNIX PROGRAMMER'S MANUAL

Sixth Edition

K. Thompson

D. M. Ritchie

May, 1975

INTRO (II) 11/5/73 INTRO (II)

INTRODUCTION TO SYSTEM CALLS

Section II of this manual lists all the entries into the system. In most cases two calling sequences are specified, one of which is usable from assembly language, and the other from C. Most of these calls have an error return. From assembly language an erroneous call is always indicated by turning on the c-bit of the condition codes. The presence of an error is most easily tested by the instructions *bes* and *bec* ("branch on error set (or clear)"). These are synonyms for the *bcs* and *bcc* instructions.

From C, an error condition is indicated by an otherwise impossible returned value. Almost always this is –1; the individual sections specify the details.

In both cases an error number is also available. In assembly language, this number is returned in r0 on erroneous calls. From C, the external variable *errno* is set to the error number. *Errno* is not cleared on successful calls, so it should be tested only after an error has occurred. There is a table of messages associated with each error, and a routine for printing the message. See *perror (III).*

The possible error numbers are not recited with each writeup in section II, since many errors are possible for most of the calls. Here is a list of the error numbers, their names inside the system (for the benefit of system-readers), and the messages available using *perror.* A short explanation is also provided.

0 – (unused)

1 EPERM Not owner and not super-user
Typically this error indicates an attempt to modify a file in some way forbidden except to its owner. It is also returned for attempts by ordinary users to do things allowed only to the super-user.

2 ENOENT No such file or directory
This error occurs when a file name is specified and the file should exist but doesn't, or when one of the directories in a path name does not exist.

3 ESRCH No such process
The process whose number was given to *signal* does not exist, or is already dead.

4 EINTR Interrupted system call
An asynchronous signal (such as interrupt or quit), which the user has elected to catch, occurred during a system call. If execution is resumed after processing the signal, it will appear as if the interrupted system call returned this error condition.

5 EIO I/O error
Some physical I/O error occurred during a *read* or *write.* This error may in some cases occur on a call following the one to which it actually applies.

6 ENXIO No such device or address
I/O on a special file refers to a subdevice which does not exist, or beyond the limits of the device. It may also occur when, for example, a tape drive is not dialled in or no disk pack is loaded on a drive.

7 E2BIG Arg list too long
An argument list longer than 512 bytes (counting the null at the end of each argument) is presented to *exec.*

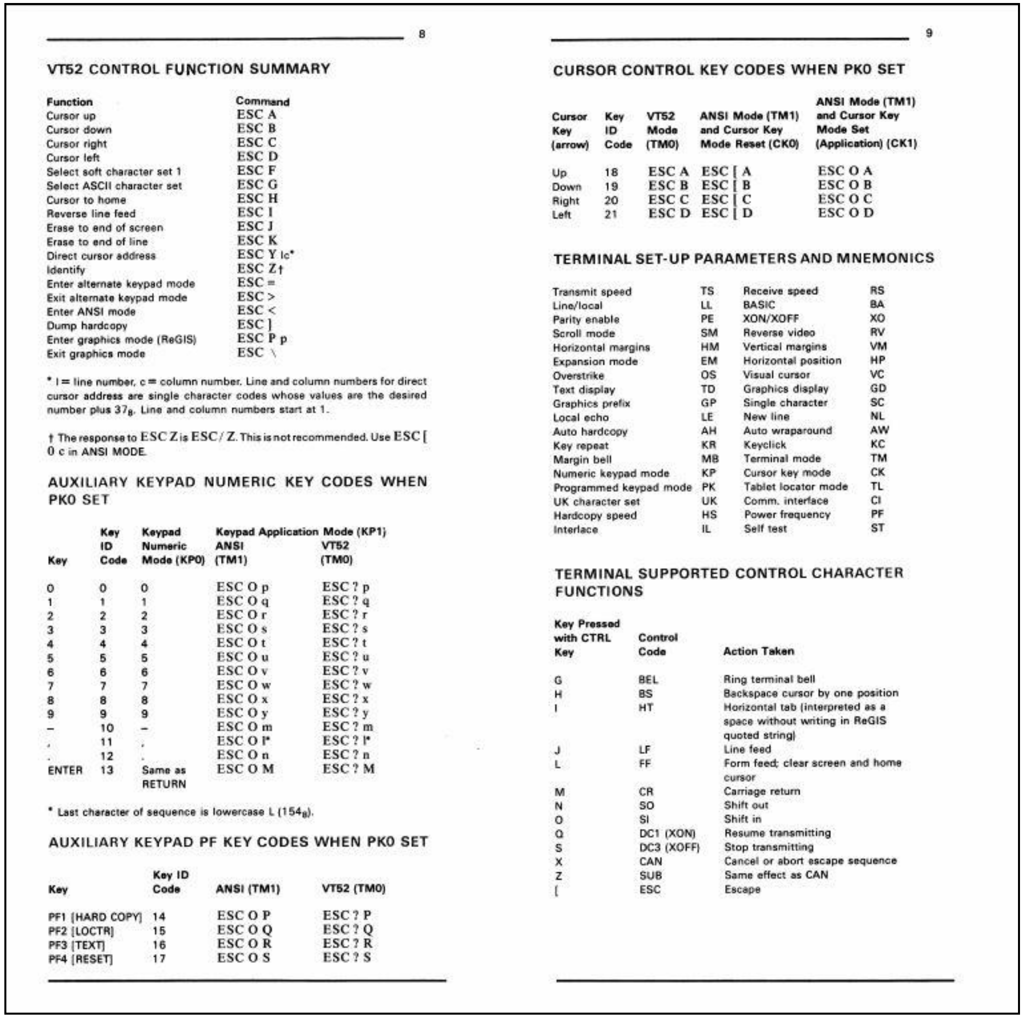
8 ENOEXEC Exec format error
A request is made to execute a file which, although it has the appropriate permissions, does not start with one of the magic numbers 407 or 410.

9 EBADF Bad file number
Either a file descriptor refers to no open file, or a read (resp. write) request is made to a file which is open on-

- 1 -

The DEC VT100 escape sequences (1978)

Is a peripheral control interface an API?



8

VT52 CONTROL FUNCTION SUMMARY

Function	Command
Cursor up	ESC A
Cursor down	ESC B
Cursor right	ESC C
Cursor left	ESC D
Select soft character set 1	ESC F
Select ASCII character set	ESC G
Cursor to home	ESC H
Reverse line feed	ESC I
Erase to end of screen	ESC J
Erase to end of line	ESC K
Direct cursor address	ESC Y lc*
Identify	ESC Z†
Enter alternate keypad mode	ESC =
Exit alternate keypad mode	ESC >
Enter ANSI mode	ESC <
Dump hardcopy	ESC]
Enter graphics mode (ReGIS)	ESC P p
Exit graphics mode	ESC \

* l = line number, c = column number. Line and column numbers for direct cursor address are single character codes whose values are the desired number plus 37_8. Line and column numbers start at 1.

† The response to ESC Z is ESC / Z. This is not recommended. Use ESC [0 c in ANSI MODE.

AUXILIARY KEYPAD NUMERIC KEY CODES WHEN PK0 SET

Key	Key ID Code	Keypad Numeric Mode (KP0)	Keypad Application Mode (KP1) ANSI (TM1)	VT52 (TM0)
0	0	0	ESC O p	ESC ? p
1	1	1	ESC O q	ESC ? q
2	2	2	ESC O r	ESC ? r
3	3	3	ESC O s	ESC ? s
4	4	4	ESC O t	ESC ? t
5	5	5	ESC O u	ESC ? u
6	6	6	ESC O v	ESC ? v
7	7	7	ESC O w	ESC ? w
8	8	8	ESC O x	ESC ? x
9	9	9	ESC O y	ESC ? y
–	10	–	ESC O m	ESC ? m
,	11	,	ESC O l*	ESC ? l*
.	12	.	ESC O n	ESC ? n
ENTER	13	Same as RETURN	ESC O M	ESC ? M

* Last character of sequence is lowercase L (154_8).

AUXILIARY KEYPAD PF KEY CODES WHEN PK0 SET

Key	Key ID Code	ANSI (TM1)	VT52 (TM0)
PF1 [HARD COPY]	14	ESC O P	ESC ? P
PF2 [LOCTR]	15	ESC O Q	ESC ? Q
PF3 [TEXT]	16	ESC O R	ESC ? R
PF4 [RESET]	17	ESC O S	ESC ? S

9

CURSOR CONTROL KEY CODES WHEN PK0 SET

Cursor Key (arrow)	Key ID Code	VT52 Mode (TM0)	ANSI Mode (TM1) and Cursor Key Mode Reset (CK0)	ANSI Mode (TM1) and Cursor Key Mode Set (Application) (CK1)
Up	18	ESC A	ESC [A	ESC O A
Down	19	ESC B	ESC [B	ESC O B
Right	20	ESC C	ESC [C	ESC O C
Left	21	ESC D	ESC [D	ESC O D

TERMINAL SET-UP PARAMETERS AND MNEMONICS

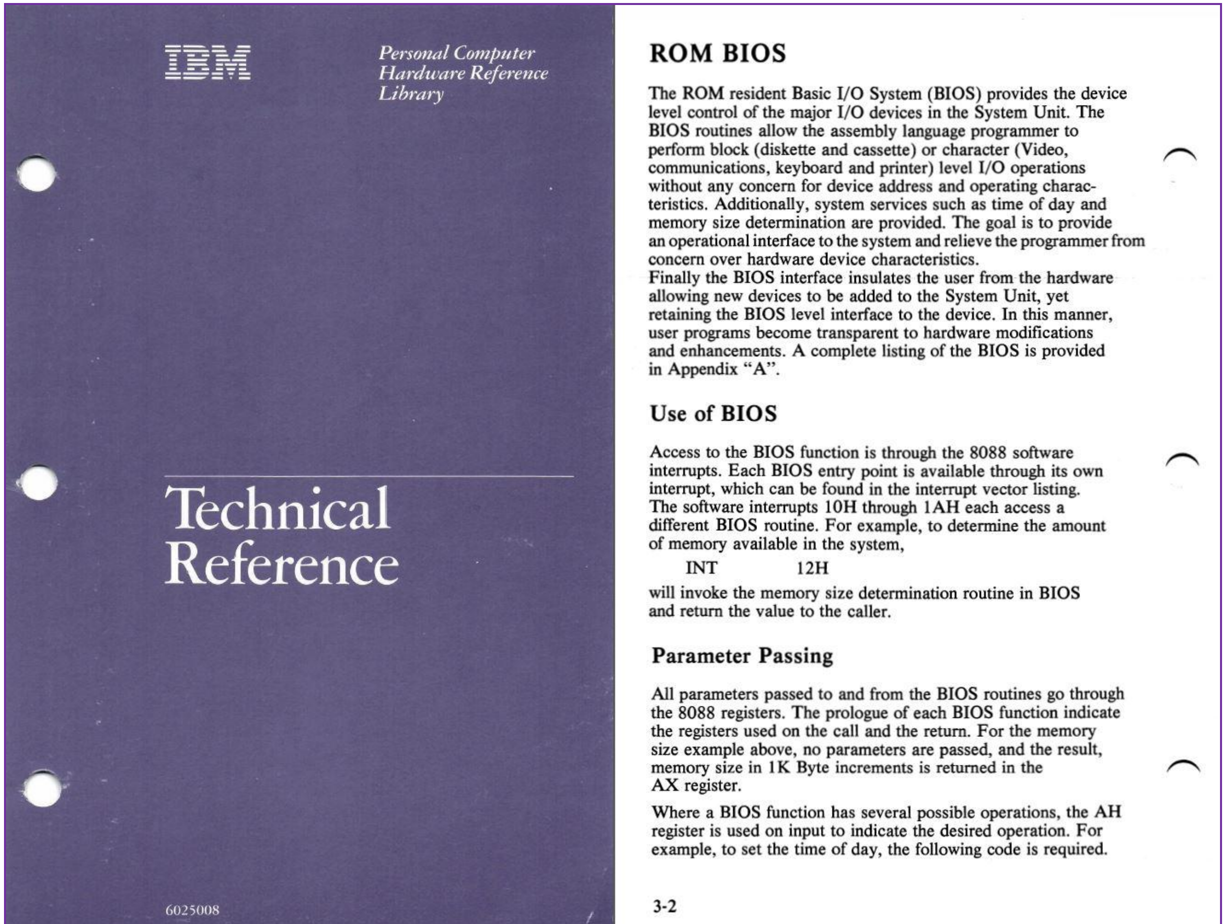
Transmit speed	TS	Receive speed	RS
Line/local	LL	BASIC	BA
Parity enable	PE	XON/XOFF	XO
Scroll mode	SM	Reverse video	RV
Horizontal margins	HM	Vertical margins	VM
Expansion mode	EM	Horizontal position	HP
Overstrike	OS	Visual cursor	VC
Text display	TD	Graphics display	GD
Graphics prefix	GP	Single character	SC
Local echo	LE	New line	NL
Auto hardcopy	AH	Auto wraparound	AW
Key repeat	KR	Keyclick	KC
Margin bell	MB	Terminal mode	TM
Numeric keypad mode	KP	Cursor key mode	CK
Programmed keypad mode	PK	Tablet locator mode	TL
UK character set	UK	Comm. interface	CI
Hardcopy speed	HS	Power frequency	PF
Interlace	IL	Self test	ST

TERMINAL SUPPORTED CONTROL CHARACTER FUNCTIONS

Key Pressed with CTRL Key	Control Code	Action Taken
G	BEL	Ring terminal bell
H	BS	Backspace cursor by one position
I	HT	Horizontal tab (interpreted as a space without writing in ReGIS quoted string)
J	LF	Line feed
L	FF	Form feed; clear screen and home cursor
M	CR	Carriage return
N	SO	Shift out
O	SI	Shift in
Q	DC1 (XON)	Resume transmitting
S	DC3 (XOFF)	Stop transmitting
X	CAN	Cancel or abort escape sequence
Z	SUB	Same effect as CAN
[	ESC	Escape

IBM PC BIOS (1981)

BIOS firmware provides API to underlying hardware



IBM

Personal Computer Hardware Reference Library

Technical Reference

6025008

ROM BIOS

The ROM resident Basic I/O System (BIOS) provides the device level control of the major I/O devices in the System Unit. The BIOS routines allow the assembly language programmer to perform block (diskette and cassette) or character (Video, communications, keyboard and printer) level I/O operations without any concern for device address and operating characteristics. Additionally, system services such as time of day and memory size determination are provided. The goal is to provide an operational interface to the system and relieve the programmer from concern over hardware device characteristics.
Finally the BIOS interface insulates the user from the hardware allowing new devices to be added to the System Unit, yet retaining the BIOS level interface to the device. In this manner, user programs become transparent to hardware modifications and enhancements. A complete listing of the BIOS is provided in Appendix “A”.

Use of BIOS

Access to the BIOS function is through the 8088 software interrupts. Each BIOS entry point is available through its own interrupt, which can be found in the interrupt vector listing. The software interrupts 10H through 1AH each access a different BIOS routine. For example, to determine the amount of memory available in the system,

```
INT        12H
```

will invoke the memory size determination routine in BIOS and return the value to the caller.

Parameter Passing

All parameters passed to and from the BIOS routines go through the 8088 registers. The prologue of each BIOS function indicate the registers used on the call and the return. For the memory size example above, no parameters are passed, and the result, memory size in 1K Byte increments is returned in the AX register.

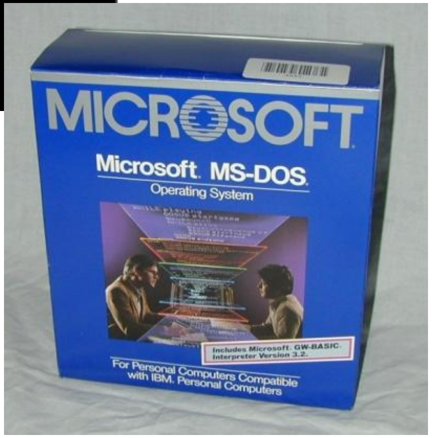
Where a BIOS function has several possible operations, the AH register is used on input to indicate the desired operation. For example, to set the time of day, the following code is required.

3-2

MS-DOS command line interface (1981)

Are CLIs APIs?

```
For more information on a specific command, type HELP command-name.
APPEND   Allows programs to open data files in specified directories as if
         they were in the current directory.
ASSIGN   Redirects requests for disk operations on one drive to a differen
         drive.
ATTRIB   Displays or changes file attributes.
BACKUP   Backs up one or more files from one disk to another.
BREAK    Sets or clears extended CTRL+C checking.
CALL     Calls one batch program from another.
CD       Displays the name of or changes the current directory.
CHCP     Displays or sets the active code page number.
CHDIR    Displays the name of or changes the current directory.
CHKDSK   Checks a disk and displays a status report.
CLS      Clears the screen.
COMMAND  Starts a new instance of the MS-DOS command interpreter.
COMP     Compares the contents of two files or sets of files.
COPY     Copies one or more files to another location.
CTTY     Changes the terminal device used to control your system.
DATE     Displays or sets the date.
DEBUG    Runs Debug, a program testing and editing tool.
DEL      Deletes one or more files.
DIR      Displays a list of files and subdirectories in a directory.
---More---
```



The Hayes modem AT command set (1982)

Is a peripheral control interface an API?

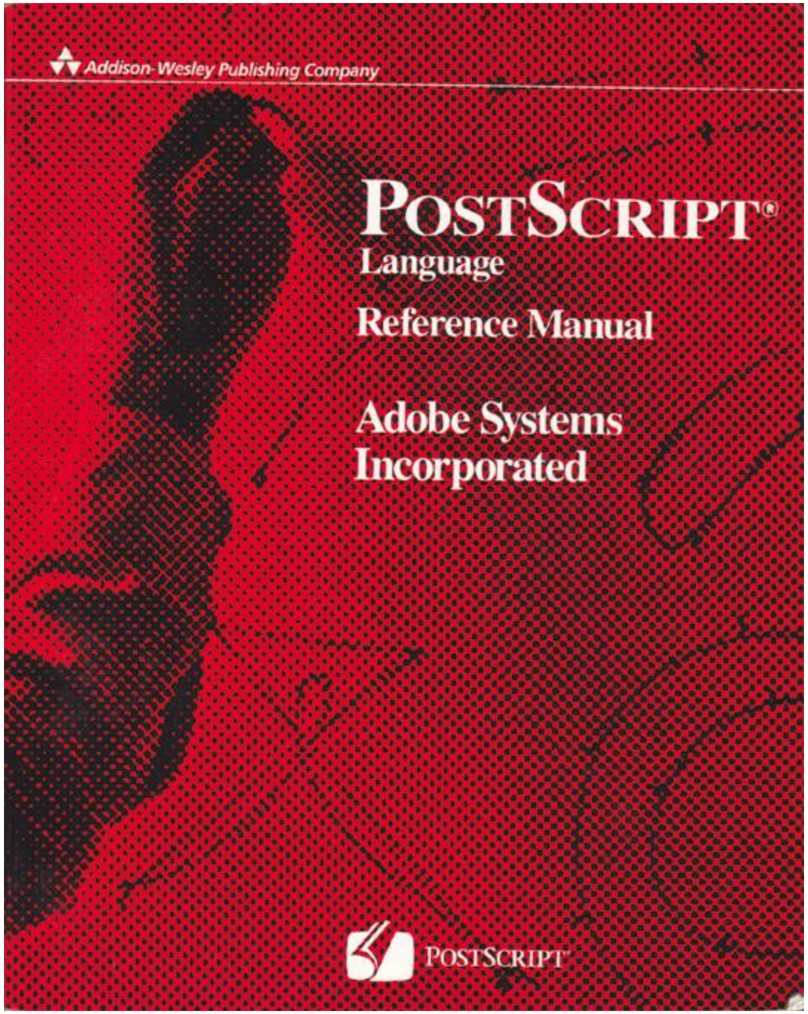
The basic Hayes command set [edit]

The following commands are understood by virtually all modems supporting an AT command set, whether old or new.

Command	Description	Comments
A0 or **A**	Answer incoming call	
A/	Repeat last command	Don't preface with **AT**, don't follow with carriage return. Enter usually aborts.
D	Dial	Dial the following number and then handshake P - Pulse Dial T - Touch Tone Dial W - Wait for the second dial tone R - Reverse to answer-mode after dialing @ - Wait for up to 30 seconds for one or more ringbacks , - Pause for the time specified in register S8 (usually 2 seconds) ; - Remain in command mode after dialing. **!** - Flash switch-hook (Hang up for a half second, as in transferring a call.) L - Dial last number
E0 or **E**	No Echo	Will not echo commands to the computer
E1	Echo	Will echo commands to the computer (so one can see what one types)
H0	Hook Status	On hook. Hangs up the phone, ending any call in progress.
H1	Hook status	Off hook. Picks up the phone line (typically you'll hear a dialtone)
I0 to **I9**	Inquiry, Information, or Interrogation	This command returns information about the model, such as its firmware or brand name. Each number (0 to 9, and sometimes 10 and above) returns one line of modem-specific information, or the word ERROR if the line isn't defined. Today, Windows uses this for Plug-and-play detection of specific modem types.
L0 or **Ln** (n=1 to 3)	Speaker Loudness. Supported only by some modems, usually external ones. Modems lacking speakers, or with physical volume controls, or ones whose sound output is piped through the sound card will not support this command.	Off or low volume
M0 or **M**	Speaker off, completely silent during dialing	**M3** is also common, but different on many brands
M1		Speaker on until remote carrier detected (i.e. until the other modem is heard)
M2		Speaker always on (data sounds are heard after CONNECT)
		Returns the modem back to the normal connected state after being

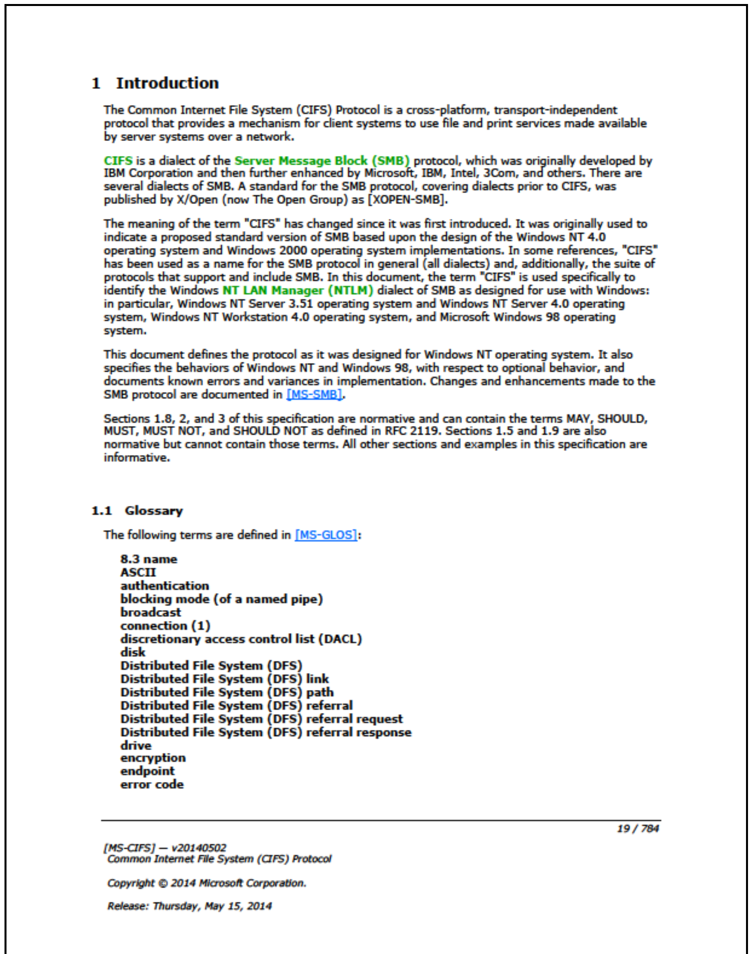
Adobe PostScript (1985)

Is it a language, an API or both?



Server Message Block (SMB) (1990)

Are wire-level protocols APIs?



1 Introduction

The Common Internet File System (CIFS) Protocol is a cross-platform, transport-independent protocol that provides a mechanism for client systems to use file and print services made available by server systems over a network.

CIFS is a dialect of the **Server Message Block (SMB)** protocol, which was originally developed by IBM Corporation and then further enhanced by Microsoft, IBM, Intel, 3Com, and others. There are several dialects of SMB. A standard for the SMB protocol, covering dialects prior to CIFS, was published by X/Open (now The Open Group) as [XOPEN-SMB].

The meaning of the term "CIFS" has changed since it was first introduced. It was originally used to indicate a proposed standard version of SMB based upon the design of the Windows NT 4.0 operating system and Windows 2000 operating system implementations. In some references, "CIFS" has been used as a name for the SMB protocol in general (all dialects) and, additionally, the suite of protocols that support and include SMB. In this document, the term "CIFS" is used specifically to identify the Windows **NT LAN Manager (NTLM)** dialect of SMB as designed for use with Windows: in particular, Windows NT Server 3.51 operating system and Windows NT Server 4.0 operating system, Windows NT Workstation 4.0 operating system, and Microsoft Windows 98 operating system.

This document defines the protocol as it was designed for Windows NT operating system. It also specifies the behaviors of Windows NT and Windows 98, with respect to optional behavior, and documents known errors and variances in implementation. Changes and enhancements made to the SMB protocol are documented in [MS-SMB].

Sections 1.8, 2, and 3 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in RFC 2119. Sections 1.5 and 1.9 are also normative but cannot contain those terms. All other sections and examples in this specification are informative.

1.1 Glossary

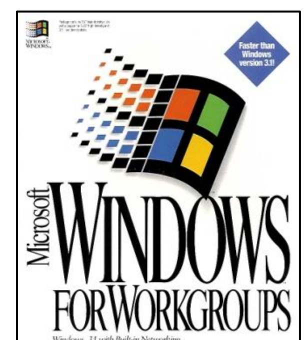
The following terms are defined in [MS-GLOS]:

8.3 name
ASCII
authentication
blocking mode (of a named pipe)
broadcast
connection (1)
discretionary access control list (DACL)
disk
Distributed File System (DFS)
Distributed File System (DFS) link
Distributed File System (DFS) path
Distributed File System (DFS) referral
Distributed File System (DFS) referral request
Distributed File System (DFS) referral response
drive
encryption
endpoint
error code

19 / 784

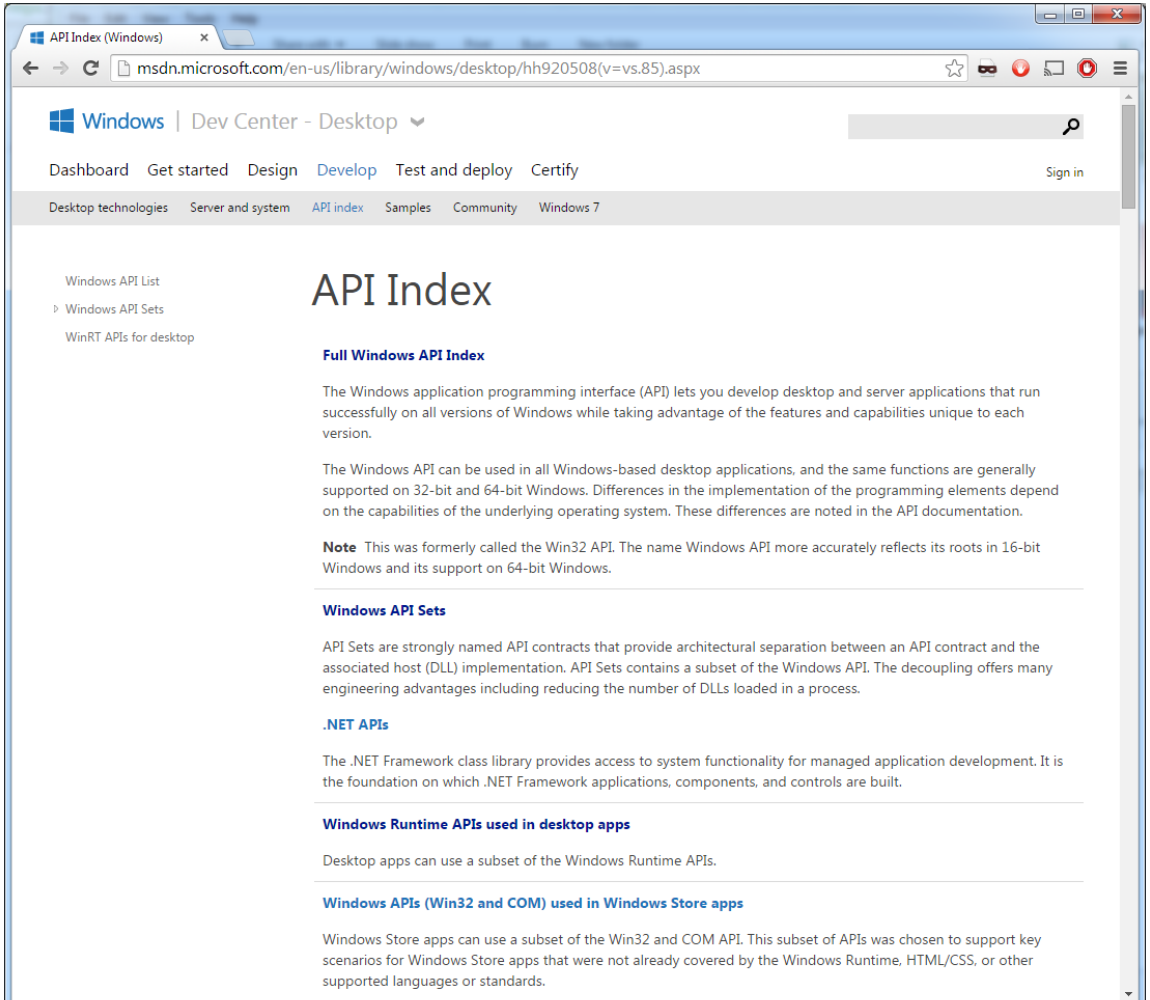
[MS-CIFS] — v20140502
Common Internet File System (CIFS) Protocol

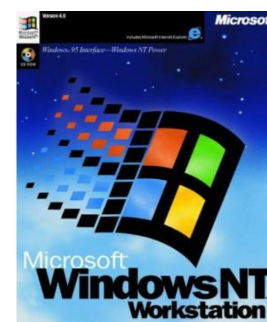
Release: Thursday, May 15, 2014



Windows (née Win32) API (1993)

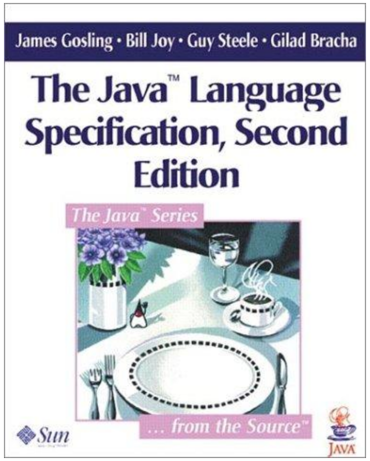
Newer operating systems, bigger API





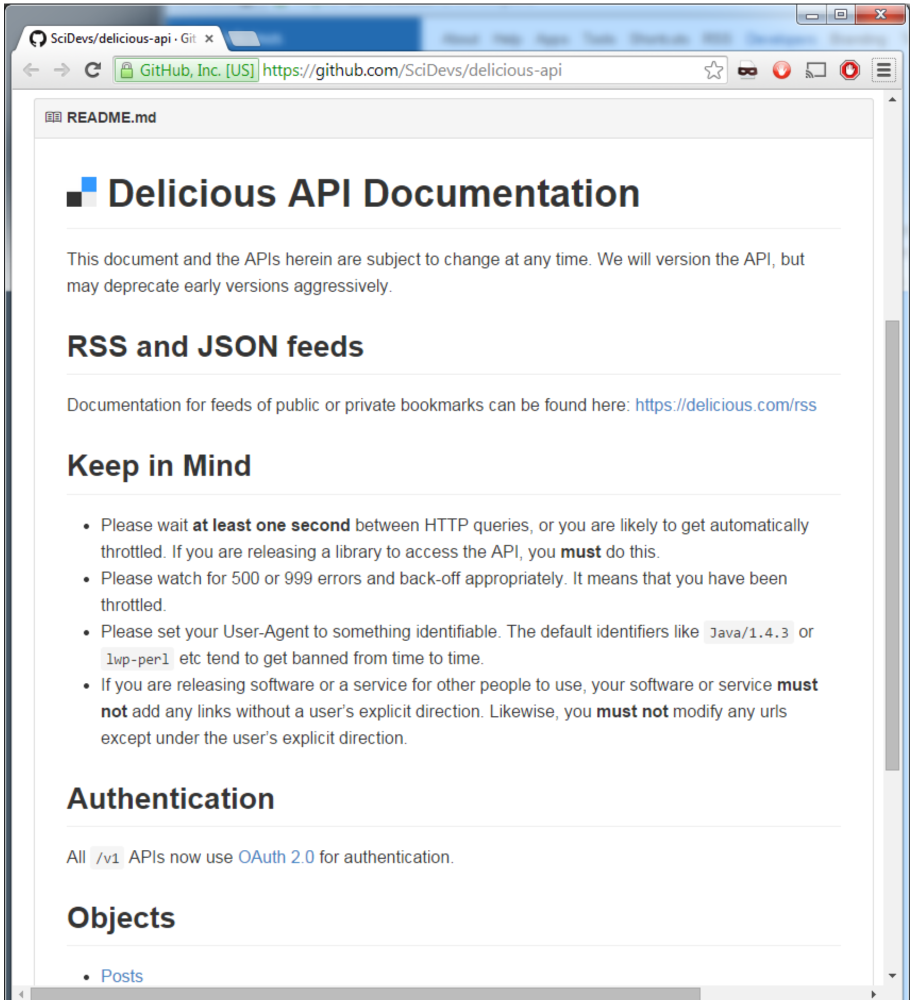
The Java class libraries (Version 2, 1998)

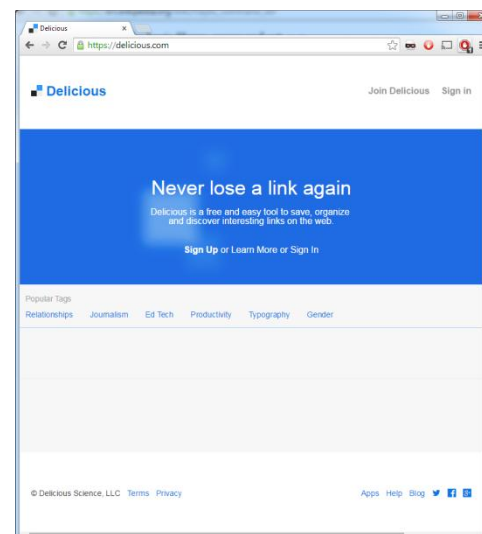
Even newer language, even bigger API



Delicious web API (2003)

Web services have APIs





Proposed API definition may be too broad

It admits ISAs, CLIs, wire-level protocols, etc.

If this bothers you, we can add to the definition:

> An API augments a programming language (or set of languages with an interoperable calling convention). Alternatively, an API may be described in an *interface definition language.*

Lessons from the whirlwind tour

- APIs come in all shapes and sizes
  - And they just keep getting bigger
- Many APIs live forever
  - Outlive the platforms for which they were created
- APIs can create entire industries
  - Both above and beneath
- **APIs are the methods of operation by which components in a system use one another**
  - **The glue that connects our digital universe**

III. How does an API come to be?

I was going to base this part on the story of Stu Feldman and **`make`**. *But the talk is too long, so I'll stick to one quote.*

"The tool took off and because it took off so fast, I never went back and fixed many of the design errors… because I didn't want to screw up my fifteen person user community."

– Stuart Feldman apologia for **`make`**'s quirks, 1989 interview by the late Michael S. Mahoney, Professor of the History of Science at Princeton

In this case, I'd argue that an incompatible API change might have been in order…

Where do APIs come from, Mommy?

- Software components are usually designed to address a specific pain point
  - Typically, you aren't the only one feeling the pain
  - So others use component too (if it soothes the pain)
  - And its interface becomes a bona fide API
  - Ready or not
- **Necessity is the mother of the API**

IV. What makes an API successful?

By "successful" I mean "widely adopted"

- **Right thing** – It has to address a real problem
  - Technology for its own sake tends to lose
- **Right place** – successful language or platform
  - "Best" language is seldom the most widely adopted
- **Right time** – market windows can be small
  - If someone else gets there first, API may be ignored
- **Good enough** – it has to solve the problem
  - See Dick Gabriel's *The Rise of Worse is Better* (1991)

If you want your API to succeed

- Make sure solves a real problem
- Do it on a successful (or soon to be successful) language/platform
- Get it out quickly
- *Subject to these constraints*, make it good

Success isn’t everything

- If an API is successful and not good
  - it will cause real pain
- If it’s good and not successful
  - it may influence a successful API in the future

Moral of this section

- You can’t know which APIs will take off, or when
- Design all interfaces as if they were public APIs
  – They might just be
- Don't wait to design in the quality
  – It’s not generally possible after the fact
  – Your API may take off while it still sucks
- Principles of good API design are well known
- So are the costs of ignoring them

V. A legal digression

Disclaimer: I am not a lawyer, nor do I play one on television. Offer void where prohibited by law. Not responsible for hurricane, lightning, tornado, tsunami, volcanic eruption, earthquake, flood, and other Acts of God, misuse, neglect, unauthorized repair, damage from improper installation. Reg. Penna. Dept. Agr. Cash value 1/20¢.

We've *always* had the freedom to reimplement each others' APIs

Every *API in Part II was reimplemented, many repeatedly*

API	Creator	Year	Reimplementer	Year
FORTRAN library	IBM	1958	Univac	1961
IBM S/360 ISA	IBM	1964	Amdahl Corp.	1970
C standard library	AT&T / Bell Labs	1976	Mark Williams Co.	1980
Unix system calls	AT&T / Bell Labs	1976	Mark Williams Co.	1980
VT100 Esc Seqs	dec	1978	Heathkit	1980
IBM PC BIOS	IBM	1981	Phoenix Technologies	1984
MS-DOS CLI	Microsoft	1981	FreeDOS Project	1998
Hayes AT cmd set	Hayes Micro	1982	Anchor Automation	1985
PostScript	Adobe	1985	GNU/GhostScript	1988
SMB	Microsoft	1992	Samba Project	1993
Win32	Microsoft	1993	Wine Project	1996
Java 2 class libs	Sun	1998	Google/Android	2008
Delicious web API	Delicious	2003	Pinboard	2009

Even the EDSAC ISA was reimplemented!

- Tokyo Automatic Computer (TAC)
- Built by Toshiba Corp. and Tokyo University
- Transistorized machine, completed 1959
- Reimplemented ISA so they could use library
  - Which they got from the appendix of WWG
- No contact with Cambridge University
  - Wilkes learned of TAC many years later

API reimplementation is under attack

- 8/10 – Oracle sues Google in Federal Court for reimplementing Java APIs in Android
  – Alleges patent and copyright infringement
- 5/12 – Jury rules no patent infringement
- 5/12 – Judge William Alsup rules that the Java APIs are not copyrightable
- 2/13 – Oracle appeals (only) copyright ruling
- **5/14 – US Court of Appeals for the Federal District reverses Judge Alsup's ruling**
- 10/14 - Google petitions US Supreme Court

What does it mean for you if the Federal Circuit ruling stands?

- You can't reimplement an API without permission from its author
  - May require payment of hefty licensing fees
  - May require adherence to field-of-use restrictions, e.g., you may not implement API on a mobile device
- Grants author near-perpetual monopoly on API
  - Life + 70 years, or 95 years for works-for-hire
- Do you use GNU, a PC, Samba, Wine, Android?
  - None could have been built if their API's creators could assert copyright and didn't want them built

The right to reimplement APIs is crucial

- New entrants can compete against incumbent with products that conform to established API
  - Increases competition, which is good for everyone
  - Even the incumbent (who can't become lazy)
- Software is more interoperable, less expensive
- Software can outlive original implementation of underlying API
- Geeks get to spend more time hacking
  - less time talking to lawyers
- Companies spend more time building products
  - less time negotiating with or suing each other

A bit of US copyright law

Key tenet since 1879: idea-expression dichotomy

17 U.S. Code § 102 - Subject matter of copyright

(a) Copyright protection subsists…in original works of authorship fixed in any tangible medium of expression…Works of authorship include the following categories: literary, musical, dance, graphic, audiovisual, audio, and architectural.

(b) In no case does copyright protection for an original work of authorship extend to any idea, procedure, process, system, **method of operation**, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied in such work.

Judge Alsup's ruling (short excerpt)

"Each command [in an API] calls into action a pre-assigned function. The overall name tree, of course, has creative elements but it is also a precise command structure — a utilitarian and functional set of symbols, each to carry out a pre-assigned function. ***This command structure is a system or method of operation under Section 102(b) of the Copyright Act and, therefore, cannot be copyrighted. Duplication of the command structure is necessary for interoperability.****"*

Judge Alsup's ruling (another excerpt)

"So long as the specific code used to implement a method is different, anyone is free under the Copyright Act to write his or her own code to carry out exactly the same function or specification of any methods used in the Java API. It does not matter that the declaration or method header lines are identical."

To paraphrase Judge Alsup, copyright protects implementation, not interface.

Sadly, the Federal Circuit disagreed

- Ruled that API was part of the “structure, sequence & organization” (SSO) of the library, and so entitled to copyright protection
  - SSO concept from *Whelan v. Jaslow* (1986)
  - Meant to protect against “non-literal copying”
- I’m not sure they understood what an API is
  - Read the opinion and decide for yourself
- Hopefully, case is headed to Supreme Court

VI. Conclusion

- APIs date back to dawn of the computer age
  - Wilkes and Wheeler invented them in 1949–1950
  - It just took us a while to realize they existed
- They’re the glue that connects digital universe
- The magic of APIs: they can be reimplented
- We’ve been free to do so since time of EDSAC
- I sincerely hope that we don’t lose this freedom

A ~~Brief~~, Opinionated History of the API

Joshua Bloch
josh@bloch.us

Bibliography (1)

David Barron. Pioneer Profiles - Stan Gill. A memorial to Stan Gill. - http://www.cs.man.ac.uk/CCS/res/res46.htm#e

Campbell-Kelly, Martin (2011). From Theory to Practice: The Invention of Programming, 1947-51. in Dependable and Historic Computing: Essays Dedicated to Brian Randell on the Occasion of his 75th Birthday (Lecture Notes in Computer Science / Theoretical Computer Science and General Issues) by Cliff Jones (Editor), John L. Lloyd (Editor). p. 30. A great history of the early EDSAC era. http://goo.gl/VTVtvj

Campbell-Kelly, Martin (2011). In Praise of *Wilkes, Wheeler, and Gill*. Communications of the ACM, Vol. 54 No. 9, Pages 25-27, 10.1145/1995376.1995386. http://cacm.acm.org/magazines/2011/9/122802-in-praise-of-wilkes-wheeler-and-gill/fulltext

Campbell-Kelly, Martin (2001?)., The Edsac Simulator. Download the software, read the tutorial, and you too can have a go at programming the EDSAC. - http://www.dcs.warwick.ac.uk/~edsac

Bibliography (2)

Campbell-Kelly, Martin (1967)., Turing Award for Maurice V. Wilkes. - http://amturing.acm.org/award_winners/wilkes_1001395.cfm

Cotton, Ira W., and Frank S. Greatorex Jr. "Data structures and techniques for remote computer graphics." Proceedings of the December 9-11, 1968, fall joint computer conference, part I. ACM, 1968. The first reference I could find to the term application program[ming] interface. - http://goo.gl/7rwnB8

Goldstine, Herman Heine, and John Von Neumann. Planning and coding of problems for an electronic computing instrument. Institute for Advanced Study, 1948. Generally regarded as the first printed reference to subroutine and subroutine library. - https://archive.org/details/planningcodingof0103inst

Google, EDSAC - A Cultural Shift in Computing. A wonderful short film celebrating the EDSAC computer on its 64th anniversary. - https://www.youtube.com/watch?v=76OhF3kR2MA

Bibliography (3)

IEEE. David Wheeler 1985 Computer Pioneer Award (1985). A nice summary of Wheeler's contributions. - http://www.computer.org/portal/web/awards/cp-wheeler

National Museum of Computing (2014). Rediscovered EDSAC diagrams reveal secrets. - http://goo.gl/GJkGk3

University of Cambridge Computer Laboratory (1999). EDSAC 99. The Computer Laboratory celebrated the 50th Anniversary of the EDSAC 1 computer in April 1999. This web site contains a thorough record of the event, with lots of good information. - http://www.cl.cam.ac.uk/events/EDSAC99

Wilkes, Maurice Vincent, David J. Wheeler, and Stanley Gill. "The preparation of programs for an electronic digital computer." Cambridge, Mass (1951). The first programming text, and the book that introduced the world to subroutine libraries. Long out of print, as is the reprint, but it’s available for print-on-demand. - http://goo.gl/s63Cna

Bibliography (4)

Wheeler, D. J. (1952). "The use of sub-routines in programmes". "Proceedings of the 1952 ACM national meeting (Pittsburgh) on - ACM '52". p. 235. doi:10.1145/609784.609816. - http://www.laputan.org/pub/papers/wheeler

Naur, Peter. "Impressions of the early days of programming." BIT Numerical Mathematics 20.4 (1980): 414-425. A wonderful paper by Peter Naur describing his experience using EDSAC to solve real problems between 1951 and 1953. - http://datamuseum.dk/site_dk/rc/naur/naur3.pdf

Mahoney, Michael S.(1989). Interview with Stu Feldman, 9-20-89. Princeton University Dept. of History, Program in History of Science. - http://www.princeton.edu/~hos/mike/transcripts/feldman.htm

Bloch, Joshua. "How to design a good API and why it matters." Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications. ACM, 2006. Gratuitous Citation of my OOPSLA talk on API design. - http://dl.acm.org/citation.cfm?id=1176622