Why is machine-readable text not as accessible as man-readable text? What can we do to make computer text more available? Does computable machine representation of graphic characters contribute anything to knowledge?

These questions open a fertile field for reflection and for further research. It is concluded that a combination of new systems programming disciplines, user procedures, and standards are needed to reach the same level of text accessibility that a library book offers. A major assertion is that a standard code for graphic characters is a scientific resource which must be guarded, if need be, by enforcement.

CHARACTER CODES: **who needs them?**

E. H. Clamons

FROM THE EDITOR

Ah, character sets! They seem so trivial. And so they are, until you try to input and run the same problem on different terminal and computer combinations and, much to your surprise, can't get it going. Or, if you do, you get wrong answers.

Mr. Clamons adds some weight to the argument that programs and recorded data are intellectual investments just like printed books in the library, and that (to conserve resources) persons with a right and need to use them should not be prevented by unintelligibility.

Away from the Middle Ages of computer usage, where the knowledge is cloistered by the priests we call programmers!

This paper was presented at the inaugural meeting of the Gesellschaft Für Informatik, Munich, Germany, 1971 Oct 12-14.

INTRODUCTION

In Pittsburgh in 1918, the proud owner of "one of those new-fangled" telephones discovered that he could not call a friend who lived only a mile away, who had also installed one. Two different telephone companies were involved, and they were not interested in providing service to customers of the competition! Their acts temporarily stifled the growth of an industry. Consumer pressure changed their attitudes. Standards provided the means of interconnecting the literally thousands of independent companies, and the phenomenal growth of the telephone industry is history.

The 1960s were marked by similar chaos in the computer industry. For example, variants of magnetic tape abounded. An entire industry sprang up to provide tape converters, and consultants determined the best mix of computers and converters to optimize performance. Tapes varied by raw material (metal, acetate, Mylar, etc.), by thickness (from 0.5 to 3 mils), by method of recording (e.g., RZ, NRZ, NRZI, etc.), and density (from 50 to 250 bits per inch).

The tape reels themselves varied in size, weight, and mounting requirements. But the disparity lay not only between different manufacturers; often reels of tape with nearly identical physical and recording characteristics could not be interchanged. Two units using identical tapes had to be "tuned" to each other before the tapes they produced could be interchanged. Standards were established to remedy this situation.

Incompatibility is still prevalent in data communications, but a new look is now being taken.

PRIVATE MESSAGES

A code-independent data transmission method is being proposed which will permit the user to communicate anything his heart desires from one device to another. Briefly, the method provides for synchronous data transmission. The first 8 bits of a transmission (message) are a flag (synchronization pattern), the next 8 an address, and the next 8 some control information; these 24 bits comprise the envelope. What follows then is the text; it is completely independent of format and code, being an arbitrary string of bits of equally arbitrary information content. A transmission is terminated by a 16-bit polynomial check character and a repetition of the flag character. Provision is made to modify data at the sender and restore it at the receiver whenever the flag character occurs adventitiously in the message. The method is not new in concept; it is possible now because of economic and technological breakthroughs. It will provide for data communications what standards provided for magnetic tape: a physical means of data transfer without regard to content. The interpretation of the information content of the transmitted text data is the receiver's problem, and it may be that he must be provided with some additional information, not contained in the message, in order to interpret it correctly. The system itself is concerned only with the orderly handling of message traffic on the basis of information contained in the envelope.

PUBLIC MESSAGES

Another tool for enhancing interchange of data was developed in 1963: The American Standard Code for Information Interchange (ASCII). (Note: the acronym ASCII is used here colloquially to mean a structure based upon ISO Recommendation 646, for lack of a better name, and not in deprecation of R646 or the other national codes based upon it).

The graphics of ASCII were chosen to fulfill the elementary needs of data and text processing, and were arranged in a logical and contiguous manner not evident in most other codes. ASCII is a dynamic code; it permits additions to satisfy our ever-increasing needs. Thus we see the introduction of "pages" of graphic and control characters, where each page assigns new meanings to the bit-patterns which normally represent ASCII characters. There will be pages of control characters, each tailored to the characteristics of the device or devices they control, and there will be pages of graphic characters to enrich the graphic repertoire of the code.

A standard (common to the public) code is much needed for peripheral devices and terminals. Theoretically these devices could be made code-insensitive by providing them with the ability to change the code they will recognize and the graphics they will display But, with few exceptions, economics do not yet justify even partial code independence as a realistic design goal. Some devices offer some freedom in the initial selection of a code, but, once selected, the code is fixed. As a rule, the simpler and less expensive a device is, the less modularity and flexibility one can expect. However, economics of these devices cannot and should not be the sole factor in the determination of the code which they use. If every device sported its own code, the processor would be swamped with conversions, leaving little time for productive work. Devices which cannot use ASCII because of design limitations will have to seek their place in the market place against the combined competitive forces of ASCII-coded devices.

EFFECT UPON PROGRAMS AND DATA

Programming languages have become more amenable to human communication. FORTRAN, COBOL, and ALGOL are now standardized and in common usage. But alas, they are not as machine-independent as they were once envisioned. Peripheral devices with varying capabilities prevented commonality of input-output operations, and the painfully slow econo-technological progress of mass storage prevented successful standardization of data management. The end of further improve-

	ISO	ECMA	ANSI	US FIPS PUB	JIS	USSR
Binary code for characters (plus control meanings)	R646 (67 Dec)	ECMA-6 (67 Jun)	X3.4 – 1968	1	C6220 – 1969	GOST 13052-67
Hollerith card code	R2021 (71 May)	ECMA-25 (70 Aug)	X3.26 – 1969	14		
Graphics for the controls	DR2047	ECMA-17 (68 Nov)	X3L2/987 (70 Jul 29)			
National Usage	97/2/525					
Additional controls						
Serial transmission	R1177 (70 Jan)		X3.15 – 1966			
Track assignment on 1″ paper tape	R1113 (69 Sep)	ECMA-10 (65 Nov)	X3.6 – 1955	2		
Track assignment on 0.5″ magnetic tape	R962 (69 Feb) R1863 (71 May)	ECMA-12 (67 Nov)	X3.22 – 1967	3		

SOURCE DOCUMENTS – STANDARDS AND DRAFT STANDARDS

3-char control (7-bit code)
alternate 2-char mnemonic
ANSI mnemonic (8-bit code)
ECMA mnemonic (if different)

ACK	(AK)			Acknowledge
BEL	(BL)			Bell
BS				Backspace
CAN	(CN)			Cancel
		CD	DCL	Character Delete
		CI	ICL	Character Insert
		CIF		Character Insert Off
		CIN		Character Insert On
		CLC	EEL	Clear Line from Cursor
CR				Carriage Return
		CSC	EED	Clear Screen from Cursor
DC1	(D1)			Device Control 1
DC2	(D2)			Device Control 2
DC3	(D3)			Device Control 3
DC4	(D4)			Device Control 4
DEL	(DT)			Delete
DLE	(DL)			Data Link Escape
EM				End of Medium
ENQ	(EQ)			Enquiry
		EO		Eight Ones
EOT	(ET)			End of Transmission
ESC	(EC)			Escape
		ESI		Extended Shift In
		ESO		Extended Shift Out
ETB	(EB)			End of Transmission Block
ETX	(EX)			End of Text
FF				Form Feed
FS				File Separator
GS				Group Separator
		HF	RPM	Highlight Off
		HLF	FHL	Half Line Feed
		HLR	RHL	Half Line Reverse Feed
		HN	APM	Highlight On
HT				Horizontal Tab
		HTC		Horizontal Tab Clear
		HTS		Horizontal Tab Set
		LCF		Local Copy Off (full duplex)
		LCN		Local Copy On (half duplex)
		LD	DL	Line Delete
LF				Line Feed
		LI	IL	Line Insert
NAK	(NK)			Negative Acknowledge
		NP		Next Page
NUL	(NU)			Null
		PD	CUD	Cursor (Pointer) Down
		PFF		Protect Format Off
		PFN		Protect Format On
		PH	CUH	Cursor (Pointer) Home
		PL	CUB	Cursor (Pointer) Left
		PM		Cursor (Pointer) Return
		PP		Previous Page
		PR	CUF	Cursor (Pointer) Right
		PT		Cursor (Pointer) Tab
		PU	CUU	Cursor (Pointer) Up
		RLF		Reverse Line Feed
RS				Record Separator
		SD		Scroll Down
SI				Shift In
SO				Shift Out
SOH	(SH)			Start of Heading
SP				Space (a blank)
STX	(SX)			Start of Text
		SU		Scroll Up
SUB	(SB)			Substitute
SYN	(SY)			Synchronous Idle
US				Unit Separator
VT				Vertical Tab
		VTC		Vertical Tab Clear
		VTS		Vertical Tab Set

Note 1

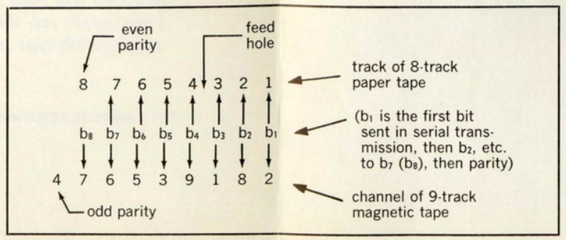
These 12 positions are variable for national usage — 2 for currency, 7 primary national usage, and 3 secondary usage which are diacritical marks when preceded by BSP. The presently-known assignments are given in the table below.

	currency		1st 7 national				dia	dia	1st 7 national				dia
	2/3	2/4	4/0	5/11	5/12	5/13	5/14	6/0	7/11	7/12	7/13	7/14	
Netherlands—A	#	$	@	[	\	]	^	`	{	|	}	~	
Australia	#	$	@	[	\	]	↑	`	{	|	}	~	
Belgium—A	£	$	§	°	Ç	§	^	`	é	ù	è	¨	
W. Germany—A	#	$	§	Ä	Ö	Ü	^	`	ä	ü	ö	ß	
US	#	$	@	[	\	]	^	`	{	|	}	~	
Japan	¥	$	@	[	¥	]	^	`	{	|	}	~	
UK	£	$	@	[	\	]	^	`	{	|	}	~	
Italy—A	£	$	@	[	\	]	^	`	{	|	}	~	
Switzerland—A	¤	¤	à	é	ç	^	ü	ä	ö	è	ï	~	
France—A	#	$	à	°	ç	§	^	ù	é	ù	è	¨	
USSR	¤	$	@	[	\	]	↑	`	{	|	}	~	
Netherlands—B	£	$	à	°	Ç	§	^	`	é	ij	è	¨	
Belgium—B	£	$	à	°	ç	é	^	ù	é	ij	è	¨	
France—B	£	$	à	°	ç	§	^	ù	é	ù	è	¨	
Switzerland—B	#	$	à	°	ç	é	^	ù	ä	ù	ê	¨	
Italy—B	£	$	§	°	Ç	é	^	ù	é	ù	è	¨	
Switzerland—C	#	Ft	§	É	Ö	Ü	Å	ü	ä	ö	ï	ä	
Hungary	#	$	§	Á	Ö	Ü	Á	ü	é	ö	ü	ß	
W. Germany—B	£	$	§	Ä	Ö	Ü	^	`	ä	ö	ü	ß	
Switzerland—D	¤	¤	à	ä	ö	ü	^	`	é	ö	ü	ä	
Sweden	#	¤	É	Ä	Ö	Å	Ü	é	ä	ö	å	ü	
Finland				Ä	Ö	Å			ä	ö	å	ü	
Denmark				Æ	Ø	Å			æ	ø	å	ü	
Norway				Æ	Ø	Å			æ	ø	å	ü	
Spain					Ñ								

(ESC) (CHAR.) SOFT COPY CONTROLS (SINGLE CHAR.)

b8 b7 b6 b5	b4 b3 b2 b1	COL / ROW	0000 0	0001 1	0010 2	0011 3	0100 4	0101 5	0110 6	0111 7	1000 8	1001 9	1010 10	1011 11	1100 12	1101 13	1110 14	1111 15
	0000	0	NUL □	DLE ⊟	SP △	0	NOTE 1 @ CI	P CD	NOTE 1 `	p	CI	CD		—	タ ユ	ミ ン		
	0001	1	SOH ⌜	DC1 ◷	!	1 HTS	A PU	Q CIN	a	q	PU	CIN	。	ア	チ a	ム я	А	Я
	0010	2	STX ⊥	DC2 ⊕	"	2 HTC	B PD	R CIF	b	r	PD	CIF	「	イ	ツ б	メ р	Б	Р
	0011	3	ETX ⌟	DC3 ⊖	NOTE 1 £ #	3 HN	C PR	S SU	c	s	PR	SU	」	ウ	テ ц	モ с	Ц	С
	0100	4	EOT ⟋	DC4 ◴	NOTE 1 $	4 HF	D PL	T SD	d	t	PL	SD	、	エ	ト д	ヤ т	Д	Т
	0101	5	ENQ ⊠	NAK ⟀	%	5 VTS	E	U NP	e	u		NP	・	オ	ナ е	ユ у	Е	У
	0110	6	ACK ✓	SYN ⊓	&	6 VTC	F	V PP	f	v		PP	ヲ	カ	ニ ф	ヨ ж	Ф	Ж
	0111	7	BEL ⌓	ETB ⊣	'	7 RLF	G PM	W PFN	g	w	PM	PFN	ァ	キ	ヌ г	ラ в	Г	В
	1000	8	BS ↶	CAN ⧖	(	8 HLR	H PH	X PF	h	x	PH	PF	ィ	ク	ネ х	リ ъ	Х	Ъ
	1001	9	HT →	EM ⏴	)	9 HLF	I PT	Y	i	y	PT		ゥ	ケ	ノ и	ル ы	И	Ы
	1010	10	LF ☰	SUB ⍰	*	: LCF	J CSC	Z	j	z	CSC		ェ	コ	ハ й	レ з	Й	З
	1011	11	VT ↓	ESC ⊖	+	; LCN	K CLC	NOTE 1 [	k	NOTE 1 {	CLC		ォ	サ	ヒ к	ロ ш	К	Ш
	1100	12	FF ↡	FS ◰	,	<	L LI	NOTE 1 \	l	NOTE 1 ¦	LI		ャ	シ	フ л	ワ э	Л	Э
	1101	13	CR ←	GS ◱	−	=	M LD	NOTE 1]	m	NOTE 1 }	LD		ュ	ス	ヘ м	ン щ	М	Щ
	1110	14	SO ⊗	RS ◲	.	>	N	NOTE 1 ^	n	NOTE 1 ~			ョ	セ	ホ н	゛ ч	Н	Ч
	1111	15	SI ⊙	US ◳	/	?	O	—	o	DEL ▨			ッ	ソ	マ о	° —	О	EO

Alternate controls in these 5 columns are achieved by preceding the regular character with an ESCape.



JISCII (Japanese Industrial Standard Code for Information Interchange) is an 8-bit code consisting of the ISO characters plus the Katakana characters shown in the upper row positions of columns 10-13 (columns 8 and 9 are reserved for additional controls, 14 and 15 for additional graphics).

GOST 13052-7 defines the USSR set, shown in the lower row entry positions of columns 12-15. Actually, the standard defines these characters for columns 4-7 of a 7-bit set (SO = Russian register, SI = Latin register). Columns 8-11 are identical to 0-3.

The Hollerith card code for 256 characters is constructed from
1 or 2 or 3 . . . or 7 or blank (no punch) 8
and any combination of 12, 11, 0, 8, and 9 (from none to all) × 32 (2^5)
= 256

For historical reasons, the assignments present little in the way of a regular pattern, but they are the key to translate to and from IBM EBCDIC.

REFERENCE CHART ISO CODE AND ASSOCIATED RELATIONSHIPS

Note — this is not a standard in itself. Refer to the appropriate documents (see reverse side). Screened characters in columns 3, 4, 5, 8 and 9 are under consideration.

Reprints of this chart are available from the Honeywell Computer Journal (P.O. Box 6000, Phoenix, AZ 85005) at $1 each postpaid.

ments in these devices is not in sight; indeed, it represents the greatest potential and challenge in cost reduction and performance improvement. However, the prospect of new hardware developments should no longer be a deterrent in the progress of programming languages. One cannot be emphatic enough in urging further efforts toward machine independent programming. Unless progress is made in the areas of data description and data management, data processing will be mired in the chaos it has created.

Standardization of data systems was born from necessity. It is an undertaking unlike any other in the history of formal standardization. It is forward-looking, it is highly intellectual, and it is characterized by a high degree of cooperation among scientists, users, and manufacturers. Because it has been going on at a feverish pace, it will be useful to reflect what it is we have accomplished and where we are going. We have seen that in media and communications standardization we have succeeded in interchanging "bits" between systems; in language standardization, we have almost succeeded up to the point of data management. Rather than tackle the whole spectrum of machine independent programming, let us examine only one aspect: code independence.

DILUTION OF THE STANDARD

ASCII is not used as universally as its developers once hoped it woud be. This is due to the IBM EBCDIC, which is technically inferior to ASCII because the graphics are scattered through the tableau of bit-patterns to carry on the tradition of punch cards, and because it caters to a typewriter which was never intended nor designed for computer applications. But all is not wasted. Both ASCII and EBCDIC are related through the punch card code, which establishes a one-to-one correspondence between the three possible code pairs.

The need for coexistence of the two codes, and the need to provide a transition to the ASCII standard, call for a re-examination of the philosophy of written communication as it was known before the advent of the computer, for an examination of the analogous building blocks in data processing systems, and for a determination from the analogies as to whether it is reasonable to expect that the written word (once so readily accessible in written, typed, or printed form) will ever be equally accessible when recorded in machine readable form. An associated examination is to determine whether the factors which led to the differences between the availability from the two storage media might not be useful in furthering the state of the art.

The beginnings of writing and electronic computing are surprisingly similar. The analog between a chiseled (or scribed) line and a bit is all too obvious: both were used initially to count. Here the analogy diverges. Man discovered curved lines, and drew pictures to separate the items he could count — three cows, four sheep. Curved bits are not now available in computers, however; instead, we group bits into bytes of n bits each, and use the property that there are 2^n different bit patterns for any chosen byte size. Now we can represent "three" "cows", "four" "sheep", etc., by assigning a particular bit pattern to each of these four words.

A further divergence occurred when the value representation of a number (the Roman numeral) was replaced by the graphic representation (the Arabic numeral). The computer ascribes an implied numeric value to the graphics when representing them by bit-patterns (the value that was previously used in counting operations). Here is where the chaos started. Man had thousands of years to develop the numerals, the alphabet, the abstract symbols, the Chinese symbols, and at most 35 years to find electronic equivalents. The fever and the excitement of the computer age preoccupied us with the immediate use of the new tool and blinded us to the future. We were too busy solving the problems that hung over our heads to worry about the consequences of arbitrary assignment of bit-patterns to characters. Instead, we used a binary representation of the existing punch card code; thus the BCD code was born.

As soon as the computer had proven itself as a viable tool for solving the problems of science and technology, it was applied to business, and a new element was added: sorting, merging, and collating. The concept of an ordered number set was equated to a collating sequence, and the bit-patterns of the alphabet were chosen so that their values would rank in the same order as the collating sequence. But not everyone had the same idea; no two value assignments were the same, and idiosyncrasies crept in. For instance, anybody making assignments without prior knowledge of the computer industry would assume that the value of I + 1 represents J. But many of us know that it is not always so; in the BCD code I + 7 equals J! Here is a simple example of the grave consequences harbored in the assignment of a value to a graphic character. The arbitrariness of assignment of bit-patterns (codes), combined with the property that in a computer a graphic representation also has an implied value, allows operations upon data which could not be practiced when only visual representation was available.

AVOIDING CODE DEPENDENCE

Code sets containing different graphics cannot be used in code independent systems. A graphic character which is not a member of the set in use cannot be represented by one of its bit-patterns. If the sets are of equal size, graphic equivalences may be assigned, but such homomorphism is dangerous because of what may result when operating upon the implied value of graphic representations. It must be evident that graphic substitutions are not a sound basis for ensuring the accessibility of arbitrary data.

When forced to work with more than two codes of dissimilar set content, one soon finds out that graphic substitution results in contradictions: two different bit-

patterns representing the same graphic character. The practical effect of this phenomenon is that the most popular character set still in use today consists of 26 letters of the alphabet, 10 numerals, 6 symbols and space. Contradictions are avoided, and syntax description facilitated, by confining oneself to the most universally available characters. One might try transliteration, the analog of romanizing cyrillic text. Unfortunately, transliteration is not necessarily a reversible process; further, it is far beyond the capability of existing business and scientific practice. A minimum requirement for code-independence is a common character set; a wisdom our forebears developed over thousands of years.

A standard set of bit-patterns for the graphics is not as important as set content (the analog is a comparison between Gothic and Roman cursive script). There exists a one-to-one correspondence between the two. Human beings can easily learn to transpose one into the other. There seems to be no evidence that code sets which are related in one-to-one correspondence cannot be used in code-independent programming. A thorough study is needed to verify this statement.

Avoiding code dependence is relatively simple, but somewhat difficult to implement. "We must learn to resist the temptation to use the binary value of a code, which represents a graphic character, for anything else"!

- All permissible arithmetic or comparison operations must be defined explicitly.
- Nothing is permissible which is not defined.
- The internal representations must not be visible.

  For example, the collating sequence must be defined explicitly, and binary values must be defined by 0's and 1's or by "binary value of a decimal number", but never by a literal (which is what programmers call a graphic representation).

This simple rule is difficult to implement because the burden of compliance is now enforced only in part by the system. Some of it may have to be enforced administratively. This paper solicits a study to determine the feasibility of total enforcement of code independence in systems using code sets which are in one-to-one correspondence.

It is not clear that operation on the values of characters might be useful in advancing knowledge, even though some generalized arguments have been made for it. Here, too, is an area worthy of study. One conclusion stands out: code-independence is difficult to implement if one operates on the values of characters. Should it turn out that knowledge would be advanced by such operations, it might be necessary to standardize the graphic repertoire more rigidly than it now is, and to enforce the standard as a scientific resource.

THE CASE FOR DATA DESCRIPTION

It was probably more instinct than insight that prompted the ISO and its member bodies to adopt the same punch card code for ASCII as was adopted for EBCDIC. That one standard defines one-to-one correspondence for the most used codes; the use of other codes is fading gradually. Unfortunately, the ruse of using the punch card code to establish the correspondence between the other two codes backfired, for one-to-one correspondence is violated. Certain bit-patterns in the punch card code have two interpretations, for instance, the bit-pattern for J also represents the numeric value of −1. This punch card tradition necessitates the introduction of data description to handle the heteromorphic data representations. If computer text is to become as easy to use as visible text, then the specific representation ascribed to a bit-pattern must be evident from the data, not from an associated program or from an implied field definition (i.e., one defined on paper but not part of the machine record). Of course, a need for data description would have arisen even without the dual meaning of the punch card codes. Most computers use binary, fixed point, floating point data forms, etc., and they too need to be described properly.

Data description as introduced by Dr. Grace M. Hopper in FLOW-MATIC survives today in COBOL. It is essentially code-independent. However, the technique does not lend itself to nonformatted data; these must be studied further. Here, in addition to recognizing mixed data forms, we must learn to distinguish between the format of text and the meaning which is ascribed to it by the format. Thus, if the same text is typed and printed, it may have different formats but the meaning of the text would remain the same. This is a complex aspect of code-independence, the preservation of content without relying on a specific method of converting from computer storage to the printed page. This subject deserves much attention because it, too, has a heavy bearing on the ability to read computer-stored data as readily as a book from a library.

CONCLUSION

During the coming decade we must dedicate ourselves to the task of creating code-independent software and to support it in hardware. We should do this not only because it will make data and programs transferable, but because it will return to human beings the things they can handle: graphic characters, words, paragraphs, pages, books, and libraries. We will thus return to ourselves the control of the processes which the programming community has usurped by confusing them with machine-dependent binary representations and their manipulation. The last decade has seen a preoccupation with the encoding of data. Top corporate officers and even the United States Government were called upon to resolve the choice of a code. And rightly so; the choice of a character set is not one to be left unguarded. It is a resource of mankind, one of the elements which, properly nurtured, will enable us to further explore the unknown.

STANDARD ABBREVIATIONS for THE UNITED STATES

Some standards seem to be more fun than others, especially if they make our jobs easier. This one provides names and abbreviations of the 50 states, and the District of Columbia. Now we need no longer decide whether to abbreviate Pennsylvania as "Penna." or "Pa."; they are both wrong. The correct form is PA (both capital letters, no period). The correct abbreviation for the U.S.A. is US.

Following are the standard abbreviations for the 50 states of the US and the District of Columbia recognized both for information exchange and by the US Postal Service.

The full standard (which also gives a 2-digit numeric code for machine use) is the Federal Information Processing Standards Publication 5-1, dated June 15, 1970, SD Catalog No. C 13.52:5-1. It is available from the Superintendent of Documents, US Government Printing Office, Washington, DC 20402.

Name	Abbrev	Name	Abbrev
ALABAMA	AL	MISSOURI	MO
ALASKA	AK	MONTANA	MT
ARIZONA	AZ	NEBRASKA	NE
ARKANSAS	AR	NEVADA	NV
CALIFORNIA	CA	NEW HAMPSHIRE	NH
COLORADO	CO	NEW JERSEY	NJ
CONNECTICUT	CT	NEW MEXICO	NM
DELAWARE	DE	NEW YORK	NY
DISTRICT OF COLUMBIA	DC	NORTH CAROLINA	NC
		NORTH DAKOTA	ND
FLORIDA	FL	OHIO	OH
GEORGIA	GA	OKLAHOMA	OK
HAWAII	HI	OREGON	OR
IDAHO	ID	PENNSYLVANIA	PA
ILLINOIS	IL	RHODE ISLAND	RI
INDIANA	IN	SOUTH CAROLINA	SC
IOWA	IA	SOUTH DAKOTA	SD
KANSAS	KS	TENNESSEE	TN
KENTUCKY	KY	TEXAS	TX
LOUISIANA	LA	UTAH	UT
MAINE	ME	VERMONT	VT
MARYLAND	MD	VIRGINIA	VA
MASSACHUSETTS	MA	WASHINGTON	WA
MICHIGAN	MI	WEST VIRGINIA	WV
MINNESOTA	MN	WISCONSIN	WI
MISSISSIPPI	MS	WYOMING	WY

authors

ANDREW A. AINES

Office of Science Information Service, National Science Foundation, Washington, DC, US

After seven years as a member, Office of Science and Technology, Executive Office of the President, and Chairman of the Committee on Scientific and Technical Information (COSATI) for the past five years, Andrew A. Aines is now a Senior Staff Associate, Office of Science Information Service, National Science Foundation. Most of his military career was devoted to research and development and high-level logistics work. Prior to retirement as a colonel in the United States Army, he was in the Office of the Director of Defense Research and Engineering of the Department of Defense. Earlier, Mr. Aines was Director of Army Technical Information, located in the Office of the Chief of Research and Development, Department of the Army. His military honors include the Legion of Merit, Bronze Star, and Army Commendation Medal with Oak Leaf Cluster. Mr. Aines holds degrees in experimental psychology, business administration, and international affairs. He is the author of a number of technical papers, and is a member of the American Psychological Association, the American Society for Information Science, the Association for Computing Machinery, the American Society for Cybernetics, and other professional societies. He has been awarded several patents. Mr. Aines was appointed by the President of the United States as a member of the recently formed National Commission on Libraries and Information Sciences.