DLS 5-MAR-74 13:57 30180

Outline of steps for COMming JOVIAL Manual

To document work in progress for IS managers and to ask SRI for a little help.

Outline of steps for COMming JOVIAL Manual

After a few false starts and some overestimation of my L-10 programming talents, I have hit upon the following scheme for transforming the JOVIAL Language Specification document from its current ragged form into a thing of beauty. The steps listed below seem necessary in light of the requirement for ths document to be as error free as is humanly possible. 1

INPUT TYPING 1a

The typing task is not a trivial one, since the text is sprinkled with "metalinguistic" terms, which must eventually be set in an italicized font. After some experimention, I became convincd that the flagging of these terms was best accomplished during initial input, rather than during later editing. In the source text, they are either typed in italics, underlined with a solid line or with a /////// line. After the first few pages, the typist was able to pick this up with no mistakes by the second chapter. An ↑ was used to flag these terms since this symbol is not used in the JOVIAL language. Less frequently there are instances of examples which must be set in monospaced font. These are flagged with a ← for the same reason. 1a1

The tables and syntatic equations are being done seperately, by someone very familiar with tabs etc. on an IMLAC. This task is almost impossible to do on a TI. 1a2

One problem, which has increased the burden on the editor, is missing characters on input. It seems that the typist goes so fast, that either the TIP or the NLS buffer becomes full and characters are dropped. I would be interested in discovering if anyone else on the ARPANET or at SRI has experienced similar problems. 1a2a

INITIAL EDITING 1b

Initial editing consists of overall structure editing, which is necessary to translate between NLS structure and the document structure and to correct any level problems introduced by the input typist. This can be done very rapidly with the IMLAC and level viewspecs. Then follows a paragraph-by-paragraph visual inspection of the text at the IMLAC. Metalinguistic terms are found fairly easily from context and most typos can be picked up and corrected on the spot. Minor grammatical editing is also done at this time. 1b1

NEWWORD editing 1c

I have made a copy of the user program INDEX and made some changes to use it as an aid in detecting and correcting mispelled words. The program is contained in <STONE>NEWWORD. It includes in its vocabulary, "words" containing -'s, :'S and 's. It excludes words in a list contained in <STONE>MASTTTER. These words are obtained from the result of previous NEWWORD runs as described below. 1c1

The general procedure is to run NEWWORD and at the display, scroll down through the plex it creates. Mispelled words tend to stand out like sore thumbs in a sorted plex. When one is found, I just jump on the link, find the word and correct it, Jump to Return, copy the link to the correctly spelled word if it exists and delete the statement. Invariably I will find a half dozen words which were overloked in the first editing pass. 1c2

The only problem with this approach is the list of words generated is very long..6-800 words on the size chapters I am dealing with. The next step is designed to reduced the length of this list, and hence make the NEWWORD list more useful as an editing aid. 1c3

UPDATE MASTER 1d

A list of correctly spelled words is contained in <STONE>MASTER. My original idea was to have a program that automatically added words from the edited NEWWORD list to the MASTER list. I overestimated my L-10 programming ability, however, and have not been able to make it work yet. However, I think that this would still be a valid approach, if someone at SRI (Dean maybe, since the program would be similar to the userprogram INDEX) could handle it. The idea is that as the document grows, the NEWWORD list would become shorter and shorter, eventually yielding a high percentage of misspelled words. 1d1

The file structure of the list would have to be changed, since the max statement size would soon be exceeded. I found this to happen for several letters of the alphabet on the second pass. 1d2

The procedure I now use is this: 1d3

I create a TEMP file and do an Execute Assimilate of the edited NEWWORD list to the TEMP file with a content analyzer patern, [',SP$PT',]; turned on. This picks up only those words which have occured in 3 or more statements, by virtue of the fact that the NEWWORD program inserts commas after each link. 1d3a

I then run a program <STONE>FIRSTWORD against the TEMP file, which strips off the links and leaves me with a word followed by a space in each statment. 1d3b

I then run <USER-PROGS>APPEND and group the words starting with the same letter into one statement. 1d3c

I then append the statements in TEMP to those in MASTER. 1d3d

I have only run three chapters this way, and so far there is still plenty of room left in each statement in MASTER. I haven't kept track of the rate of word acquisition. 1d4

CONTENT PROOFING 1e

An ODP is done and an independent person, who is knowledgeable in the JOVIAL Language Spec area does a final proofing for content and font indicators. Limited experience indicates that a two person operation goes smoother. One person reading the orginal text and the other following the NLS printout. After this stage is completed and the necessary changes are made, the content and font indications are considered 100% accurate. They will not be reviewed again in a systematic manner. 1e1

TERM INDEX 1f

This document will be subject to sporadic updates, as new constructs are added to the language and old ones are redefined or renamend. The process of defining changes is a slow one involving several committees and levels of coordination within AF and DOD. To facilitate changing and republishing of the document, it is desireable to have an index to all occurances of the "metalinguistic" terms used in the document. The ↑ then serves a double function of font change indicator and of index flag. <STONE>JMINDEX is used to extract the words preceded by an ↑ and append links pointing to which NLS statements they ocurr in. (I might want to change this program slightly, to reflect the number of times the word ocurrs in each statement.) The chapter is then Jounaled as the "offical" reference copy for future updating purposes. 1f1

INSERT COM DIRECTIVES 1g

The final fonts, sizes and styles awaits the results of our first experimental run, where we will be running the same sample pages with different mixes of fonts, margins, page layout etc. If the body of the document is all in the same font, then we can write an L-10 CA program to replace strings beginning with ↑ with *string*. Likewise the strings beginning with ← will be replaced with .Mono=On;*string*.Mono=Off;. This will take care of 90% of the font changes. If it becomes necessary to change fonts as well as style, we can use the V1, V2, V3 constructs. There is a problem with this however, in the cases where a metalinguistic or example term is immediately followed by a punctuation mark. It might be better to use the Font construct. I will have to see how often this ocurrs and make a judgement on how much hand editing is required. 1g1

We will probably want to redefine the most commonly used font directives to make them as compact as possible, since some of the statements are already near the max. limit. We should also redefine the directive delimiters, to speed up the ODX process. It appears that ↑ might be good for this also. Any comments? 1g2

The biggest problem comes in inserting the tables and syntatic equations. So far I have just put in directives to GYES enough lines to allow for their manual insertion. I am tempted to prepare the text of the tables and equations using the system and copy them into the positon where they belong. This would assure a uniform treatement of tables and running text. The syntatic equations are repeated at the end of the document anyway, in an appendix used for crossreferencing. If they were insereted in the text, then the only mannual process would be to draw in the equation box boundaries and table row and column division lines by hand. If I go this route I will need to turn off the Justification before each graphic. In any event, THE GRAPICS PORTION OF THE JOB SHOULD BE STARTED NOW, since the approach will determine the size of the graphic, which in turn determins the values of the directives used, where pagination ocurrs and the content of the footer directive for each page. I have a couple of questions, which make a difference even now in this area: 1g3

Can the COM right and left justify a line with different fonts in it? With different Sizes? With different styles? With mixes of the above? 1g3a

How are Tabs treated by COM in Justified mode? Unjustified mode? Does the Point size of the previous line effect absolute tabbing distance? ie do I have to explicity reset size at the beginning of each table? Could <USER-PROGS>NOTABS be changed easily to pay attention to TABSTOPS directives? 1g3b

Similar guestion for GYES, etc...if I have set a default YBS in the orgin statement will the GYES always give me the same absolute spacing per line? Some of the tables immediately follow a section heading which will be set in a larger size than the subparagraphs under it. I'm concerned that the lines following the headings might be larger than those in the subparagraphs. 1g3c

XCOM PROOF 1h

An CDX run will be made. Some one familiar with the JOVIAL Manual will have to review this and make decisions on what to do with tables that are divided on two pages. Should another statement be brought over and the table put on the next page? Should the whole table be forced to the next page and the current page be left partially blank? Several runs per chapter may be necessary to get the desired placement. At least by breaking the document up into chapters, an early change will not effect the entire rest of the document. 1h1

The final job under this task will be to put the proper values, corresponding to the last section number on the page, in the Footer directive for each page. 1h2

COM PROOFING 1i

A final proof of the COM proofs will have to be made, to see if we got what we thought we were going to get. A check should be made to see if the graphics fit properly. Again this will have to be made by someone outside the NLS team. 1i1

CAMERA READY COPY 1j

Once the proofs have ben reviewed and any necessary changes made, quality copies will be made from the microfilm at DDSI. 1j1

Question..can DDSI make camera ready copy from pieces of a COM run. I am thinking of the possibility that a minor mistake is discovered, which will require rerunning a few pages out of a chapter. Do we have to rerun the whole chapter through the process or can they make up camera copy from pieces of several runs? Extra charge? 1j1a

INSERTING GRAPHICS 1k

Depending on the route chosen for tables and syntatic equations, the lines, brackets, braces and other continuation symbols will have to be inserted or the entire graphic wil have to be inserted. The art work is a job for the Arts and Drafting group. If the inserts are to be treated as graphics, someone will have to retype them to get quality improvements over what we now have. 1k1

DLS 5-MAR-74 13:57 30180

Outline of steps for COMming JOVIAL Manual

FINAL PAGE PROOF
**1l

A final proof, check of footer section numbers, crosscheck of syntax indexes, etc. will have to be made before sending the document to the printer. 1l1

GOTO PRINTER 1m

Who does the printing? How many copies? 1m1

DISTRIBUTION 1n

To which offices? To individuals? Keep a list of who has copies for revision, updating, ammendments, etc. Good job for NLS. 1n1

Rough estimates of the amount of time it will take for each task, based on our experience to date with the first four chapters (121 typewritten pages). 2

TASK	MIN/PAGE (PERSON)	SEC/PAGE (CPU)	
			2a
			2b
INPUT TYPING........................	20	20	2c
INITIAL EDITING.....................	10	10	2d
NEWWORD EDITING.....................	5	35	2e
UPDATE MASTER.......................	1	5	2f
CONTENT PROOFING....................	5	5	2g
TERM INDEX..........................	0	10	2h
INSERT COM DIRECTIVES...............	1	5	2i
XCOM PROOF..........................	5	30	2j
COM PROOFING........................			2k
CAMERA READY COPY...................			2l
INSERTING GRAPHICS..................			2m
FINAL PAGE PROOF....................			2n
GOTO PRINTER........................			2o

DLS 5-MAR-74 13:57 30180

Outline of steps for COMming JOVIAL Manual

DISTRIBUTION........................ 2p

Estimates above are based on the following factors: 3

2:1 ratio of typewritten pages to COM pages. 3a

ODP--4sec/page, SENDPRINT--1sec/page 3b

ODX--10sec/page, SENDPRINT--4sec/page. 3c

INPUT TYPING 3d

Includes time for 1 ODP 3d1

INITIAL EDITING 3e

NEWWORD EDITING 3f

30 sec/page for run time (16 more if you have to compile). 3f1

UPDATE MASTER 3g

CONTENT PROOFING 3h

TERM INDEX 3i

runtime of jmindex. 3i1

INSERT COM DIRECTIVES 3j

2 ODX runs . 3j1

XCOM PROOF 3k

COM PROOFING 3l

CAMERA READY COPY 3m

INSERTING GRAPHICS 3n

FINAL PAGE PROOF 3o

GOTO PRINTER 3p

DISTRIBUTION 3q

The overall cost of the project then can be estimated at: 4

PEOPLE--1hr/pg X 400pgs X $7.00/hr = $2800 4a

$7.00 is the average of 1 technical and 1 clerk. 4a1

COMPUTER--2min/pg X 400pgs / 60min/hr X $100/hr = $1300 4b

$100/hr comes from $500K for facility for a year, which contains 52weeks, 6days/week, 16hrs/day. 4b1

COM--$3.50/pg X 400pgs = $1400 4c

Rough guess at the total then is $6K, which compares with $50K quoted by one contractor. 5

Outline of steps for COMming JOVIAL Manual

(J30180) 5-MAR-74 13:57; Title: Author(s): Duane L. Stone/DLS;
Distribution: /EJK JLM FJT ARB NDM DVN RJC; Sub-Collections: RADC;
Clerk: DLS;

ARPA Users on OFFICE-1

Jim: 1

I just had a meeting with Connie McLindon regarding ARPA users on OFFICE-1. We started with the list of ARPA people who had accounts at ISI, and tried to think of a rationale why they should NOT have directories at OFFICE-1. I couldn't think of any compelling reasons, so Connie is asking you to set them all up. For the immediate future, we're not thinking in terms of moving all their usage to OFFICE-1; rather, we're just anxious to offer it as an option for periods when ISI is down or overloaded. I get the feeling that this is going to upset you, so let me put forward some reasons: 1a

1. ARPA is paying the major share of OFFICE-1's costs. In return, the only visible resource it is getting is the block of ARPA slots. Simple justice and good management therefore demands that these slots be used for ARPA's benefit. 1a1

2. Of all the uses to which ARPA could put these slots, the least demanding and disruptive to other users is the type of activity typical of ARPA office use of ISI, i.e. SNDMSG, READMAIL, RD, etc. 1a2

3. If ARPA does not take immediate steps to use these slots, then they will -- through the group allocation scheme -- be swallowed up by others such as RADC, Bell, NIC. From ARPA's point of view, any and all of these are of substantially lower priority than ARPA office use. 1a3

4. The hours of operation of OFFICE-1 coincide with the most overloaded and frustrating period for ISI. Thus ARPA management use of OFFICE-1 will be of greater benefit in relieving network pressure than any single other act I can think of. In particular, it may enable us to move some computational users back to ISI rather than having to foist them on unwilling hosts. 1a4

5. We want to gradually expand the universe of software used by ARPA management people to include the fantastic planet of NLS. Getting them on OFFICE-1 now is a good start. 1a5

6. ISI has been notably unreliable during the primary hours of ARPA management use, while OFFICE-1 has been rock-solid. It is therefore highly attractive to open OFFICE-1 as an option for these users. 1a6

So we're not kidding; we really do want to make OFFICE-1 available to the ARPA office right now. There are other issues we'd like you to address at the same time: 1b

1. These users are accustomed to using the following services and systems: 1b1

SNDMSG 1b1a

READMAIL 1b1b

RD 1b1c

TECO 1b1d

These should be provided so that we can make a smooth transition to NLS later. (I'm sorry about TECO, but that's the way it is.). 1b2

2. Within this group of users, we really must make some sort of reasonable provision for priority use by a sub-group. Within the cloistered computer science community this no doubt seems arbitrary, capricious, and profoundly undemocratic. But it is in fact an inescapable element of any real-world environment into which you will introduce your technology. So we might as well get used to the idea right now, and let the ARPA Director's office serve as a model for a broad class of priority users. 1b3

I suggest therefore that arrangements be made that Lukasik and Tachmindji are never denied access. How this is to be done is up to you folks, but that's what we'd like to see. 1b4

I can well understand that these steps may produce some apprehension on your part, in that dissatisfaction with OFFICE-1 might produce a halo of dissatisfaction with SRI/ARC. I see little cause for fear: 1c

First, reliability of the hardware is perceived as TYMSHARE's responsibility, not ARC's. 1c1

Second, the users will be employing initially software produced elsewhere, so that bugs will be firmly related to other culprits. 1c2

Thirdly, the competition is so miserable that even performance substantially below your usual high standards will look pretty impressive. 1c3

I hope that you'll see these desires as a tribute to the fine management and excellent service the OFFICE-1 project has demonstrated thus far, and not as a callous attempt to wring blood out of a stone. We're motivated by a desire to keep moving toward the goal of a computer-augmented office and to get full utilization of the expensive resources we've procured with so much difficulty. We're willing to discuss a lot, but compromise only a little. 1d

Sincerely, 1e

John. 1f

ARPA Users on OFFICE-1

(J30181) 5-MAR-74 14:41; Title: Author(s): John S. Perry/JSP;
Distribution: /JCN(action) CKM(fyi) JCRL(fyi) CF(fyi) DCR2(fyi);
Sub-Collections: NIC; Clerk: JSP;

(J30182) 6-MAR-74 05:11; Title: Author(s): Duane L. Stone/DLS;
Distribution: /RJC; Sub-Collections: RADC; Clerk: DLS;
Origin: <PETELL>C3.NLS;1, 5-MAR-74 11:16 DLS ;

JOVIAL Manual--Chapter 3

edited text, contains↑ & ←

Chapter 3 1

↑VARIABLES 2

3.1 Concept of ↑Variables
A JOVIAL ↑program:declaration consists of a string of ↑statements and ↑declarations that specify rules for performing computations with sets of data. The basic elements of data are items. Items are named to distinguish one from another. Sometimes, a ↑name applies to a group of items, requiring indexing to tell one member of the group from another. Several named groups may be subsumed under another group, which is known as a table and which is itself named. Tables and items may in turn be collected in another group called a data block which, again, is named. Space may be allocated these data structures either statically at compile time or dynamically at execution time. 3

.1 The value of items and other data can be changed in various ways. A data element whose value can be changed by means of an ↑assignment:statement is known as a variable. Items, then, are variables. Table entries can function as variables, as can parts of items under the influence of the ↑primitives .BIT and .BYTE. 3a

.2 A ↑variable is the designation, within a ↑program:declaration, of a variable to be manipulated within the computer. The two syntax equations for ↑variable (above) indicate, first, the type of data involved, and second, the grammatical form of the ↑variable related to the kind of data structure in which the variable exists. 3b

3.2 Named:Variable 4

A ↑named:variable is a reference to a variable by means of a ↑name associated with the variable through a ↑data:declaration. A ↑simple:variable is a reference (for the purpose of using or changing its value) to a variable declared to be a simple variable; one not declared as a constituent of a table. No ↑index is involved in a ↑simple:variable because the reference is to a variable that is one of a kind, not part of a matched set. Use of the ↑pointer:formula is explained in Section 7.8 4a

A ↑table:variable is a reference to a variable declared to be part of a table. A table consists of a collection of entries and there is an occurrence of each table item in each entry. An ↑entry:variable is a reference to the entire entry as a single variable. An ↑indexed:variable (a ↑table:variable or ↑entry:variable) generally includes an ↑index to select the particular occurrence of the variable being referenced. 4b

.2 An †index is correlated with a †dimension:list. Every †table:declaration contains a †dimension:list which prescribes the number of dimensions of the table and the extent of the table in each of these dimensions in terms of its †lower:bound and its †upper:bound. (Some of the detailed specifications can be omitted; the defaults are explained elsewhere.) Each †index:component must evaluate to an integer value (†numeric:formulas are explained in Sec 5) not less than the †lower:bound and not greater than the †upper:bound in the corresponding position of the relevant †dimension:list. The relevant †dimension:list is, of course, the one in the †table:declaration bearing the †table:name beginning the †entry:variable or in the †table:declaration containing the †item:declaration bearing the †item:name starting the †table:variable. The rightmost †index:component selects the element, of the row selected by the †index:component second from the right, from the plane selected by the index:component third from the right, etc. 4c

.3 If the †index is omitted from an †indexed:variable, whether or not the empty †brackets remain, the meaning is the same as if the complete †index were present and each †index:component were equal to its corresponding †lower:bound. In fact, a legitimate form of †indexed:variable is to omit one or more †index:components, marking their positions of necessary with †commas. The meaning of such a form is the same as if each missing †index:component were present with a value equal to its corresponding †lower:bound. The following example shows an †ordinary:table:declaration and three †entry:variables, all with exactly the same meaning: 4d

←TABLE ALPHA [3:7, 9, 100:157, 0:50]; NULL; 4d1

←ALPHA [3, 3, 100,0] 4d2

←ALPHA [, 3,, 0] 4d3

←ALPHA [,3] 4d4

3.3 †Letter:Control:Variable, †Functional:Variable 5

A †letter:control:variable is a reference to a variable designated within a †loop:statement to aid in control of execution of the †controlled:statement and to have meaning only within the †loop:statement. It is explained in Section 5.8 in conjunction †loop:statements. 5a

.1 †Format:variable is a special form that enables a list of values to be converted to character type and assembled into a character value. The details are given in Section 6.1.7 5a1

INSERT BOX 5a2

.2 The above construct selects a string, of the characters denoted by the †named:character:variable, to be considered as the variable to be given a new value. The †named:character:variable can be any †simple:variable or †indexed:variable of character type. The bytes of the †named:character:variable are considered to be numbered, starting with zero at the left. The †numeric:formula following the first †comma is evaluated as an integer and used to select the byte of the †named:character:variable to be considered the leftmost byte of the †functional:variable. If there is no second †comma and no second †numeric:formula, the leftmost byte of the †functional:variable is its only byte. Otherwise, the second †numeric:formula is evaluated and tells how many bytes there are including the leftmost byte, in the †functional:variable. 5a3

.3 The †named:variable in the above metalinguistic formula can be of any type. The construct selects a string of bits, from the bits denoted by the †named:variable, and treats that string of bits as a bit variable. The bits of the †named:variable are considered to be numbered, starting with zero at the left. The †numeric:formula following the first †comma selects the bit to be considered the first bit of the derived variable. The †numeric:formula following the second †comma (if there is one) determines the number of bits in the derived string (one bit if there is no such †numeric:formula). In signed variables, the sign bit is bit zero and the leftmost magnitude bit is bit one. In unsigned numeric variables, the leftmost magnitude bit is bit zero. In entries, the leftmost bit of the first word is bit zero. In character variables, the number of bits per byte is system dependent. In floating variables, the sign bits of the significand and exrad are included in the bit count, but the arrangement of bits is system dependent. 5a4

3.4 †Format:Variable, †Bit:Variable, †Character:Variable 5b

†Format variable is explained in Section 6.1.7. 5c

.1 The construct using ←BIT is explained in Section 3.3.3. A †bit:variable denotes a string of bits without consideration of any numeric or other meaning associated with those bits. Almost all †named:variables carry an implication of some data type other than "bit". However, an †entry:variable, if the †table:name is not declared so as to imply some specific data type, denotes only the string of bits constituting the entry. 5c1

.2 The construct using ←BYTE is explained in Section 3.3.2.

The †named:character:variable is a †named:variable using a †name declared to denote a variable (an item or an entry) of character type. 5c2

3.5 Numeric:Variable 5d

Any †numeric:variable can be used as a †pointer:variable. The details of the use of †pointer:variables are given in Chapter 7 in conjunction with discussion of controlled allocation. †Letter:control:variable is explained fully in connection with †loop:statements. Without being explicitly declared, it becomes an †integer:variable through its usage. All †names that can be used as †named:variables are declared as explained in Chapter 7. Some †entry:variables may use †names not associated with any data type. All other †named:variables use †names that are associated with †item:descriptions. These †item:descriptions give the data type among other things (see Section 7.16 for details). One data type is "character" as mentioned above in Section 3.4.2. Another data type is "floating". †Floating:variables use †names declared to be of floating type. The other descriptive terms in †item:descriptions denote "signed" and "unsigned", but we are interested here in other attributes. Signed and unsigned data are also associated with one or two †numbers. The first †number declares the size of the datum, the number of bits in its magnitude. If this is the only †number in its †item:description, the datum is an integer value and the †named:variable denoting it is an †integer:variable. The second †number in the †item:description for a signed or unsigned value declares the precision of the value, the number of bits in its magnitude after the point. If this second †number is present, even if its value is zero, the datum is a fixed value and the †named:variable denoting it is a †fixed:variable. 5e

PAN 6-MAR-74 11:24 30183

Sign-on problem

Every time I sign on the system, it asks me for my ident. Is there anyway that this could be done automatically - if so, it would be greatly appreciated. 1

Sign-on problem

(J30183) 6-MAR-74 11:24; Title: Author(s): Penny A. Napke/PAN;
Distribution: /FEED IMM PAN; Sub-Collections: NIC; Clerk: PAN;

From: HHughes.MAC at MIT-Multics
Date: 03/06/74 1602-edt 1

Title: Abstract - TR 52 thru 55 1a

Implementing Multi-Process Promitives in a Multiplexed Computer System 1b

Rappaport, Robert L. 1b1

November 1968 MAC-55 1b2

A.B.S.T.R.A.C.T. 1b2a

In any computer system, primitive functions are needed to control the actions of processes in the system. This thesis discusses a set of six such process-control primitives which are sufficient to solve many of the problems involved in parallel processing, as well as in efficient multiplexing of system resoruces among the many processes in a system. In particular, the thesis documents the work performed in implementing these primitives in a particular computer system - the Multics system - which is being developed at M.I.T.'s Project MAC. 1c

During the course of work that went into the implementation of these primitives, design problems were encountered which caused the overall rpgram design to go through two iterations before program performance was deemed acceptable. The thesis discusses the way the design of these programs evolved during the course of this work. 1d

The Graphic Display as an Aid in the Monitoring of A Time-shared Computer System 1e

Grochow, Jerrold M. 1e1

October 1968 MAC-TR-54 AD-689-468 1e2

A.B.S.T.R.A.C.T. 1e2a

The Graphical Display Monitoring System was developed as a medium for dynamic observation of the state of a time-shared computer system. The system is integrated to create graphic displays, dynamically retrieve data from the Multics' Time-Sharing System supervisor data bases, and allow on-line viewing of this data via the graphic displays. On-line and simulated experiments were performed with various members of the Project MAC Multics staff to determine the most relevant data for dynamic monitoring, the most meaningful display formats, and the most desirable sampling rates. The particular relevance of using a graptic display as an output medium for the monitoring system is noted. 1f

As a guide to other designers, a generalized description of the priciples involved in the design of this on-line, dynamic monitoring device includes special mention of those areas of particular hardware or software system dependence. Several as yet unsolved problems relating to time-sharing system monitoring, including those of security and data base protection, are discussed. 1g

The Flow Graph Schemata Model of Parallel Computation 1g1

Slutz, Donald R. 1g1a

September 1968 MAC-TR-53 AD-683-393 1g1b

A.B.S.T.R.A.C.T. 1g1b1

Flow Graph Schemata are introduced as uninterpreted models of parallel algorithms, operating asynchronously and reflecting physical properties inherent to any implementation. Three main topics are investigated: (1) determinacy, (2) equivalence, and (3) equivalence-preserving transformations on the control structure of a Flow Graph Schemata. A model is determinate if the results of a computation depend only on the initial values and not on any timing constraints withing the model. Equivalence is undecidable in general, but for a large class of determinate Flow Graph Schemata which are in a maximum parallel form, equivalence is shown decidable. In equivalence-preserving transformations, sufficient tested conditions for equivalence are formulated that depend only on the portion of the structure to be transformed. 1h

Current and future computational systems are evaluated in terms of results obtained for Flow Graph Schemata. A number of interesting extensions of the work are suggested. 1i

Absentee Computations in a Multiple-Access Computer System 1i1

Deitel, Harvey M. 1i1a

August 1968 MAC-TR-52 AD-684-738 1i1b

A.B.S.T.R.A.C.T. 1i1b1

In multiple-access computer systems, emphasis is placed upon servicing serveral interactive users simultaneiously. However, many computations do not require user interaction, and the user may therefore want to run these computations "absentee" (or, user not present). A mechansim is presented which provides for the handling of absentee computations in a multiple-access computer system. The design is intended to be implementation-indepent. Some novel features of the system's design are: a user can switch computations from interactive to absentee (and vice versa), the system can temporarily suspend and then continue absentee computations to aid in maintaining an efficient absentee-interactive workload on the system, system administrative personnel can apportion system resources between interactive and absentee computations in order to place emphasis upon a particular mode during certain periods of operation, and the system's multiple-computation-stream facility which allows the user to attach priorities to his absentee computations by placing the computations in ether low-, standard-, or high-priority streams. 1j

2

(J30184) 6-MAR-74 13:03; Title: Author(s): Herb S. Hughes/HSH;
Distribution: /HSH MAP ; Sub-Collections: NIC; Clerk: HS;

ARPA Users on OFFICE-1

Jim: 1

I just had a meeting with Connie McLindon regarding ARPA users on OFFICE-1. We started with the list of ARPA people who had accounts at ISI, and tried to think of a rationale why they should NOT have directories at OFFICE-1. I couldn't think of any compelling reasons, so Connie is asking you to set them all up. For the immediate future, we're not thinking in terms of moving all their usage to OFFICE-1; rather, we're just anxious to offer it as an option for periods when ISI is down or overloaded. I get the feeling that this is going to upset you, so let me put forward some reasons: 1a

1. ARPA is paying the major share of OFFICE-1's costs. In return, the only visible resource it is getting is the block of ARPA slots. Simple justice and good management therefore demands that these slots be used for ARPA's benefit. 1a1

2. Of all the uses to which ARPA could put these slots, the least demanding and disruptive to other users is the type of activity typical of ARPA office use of ISI, i.e. SNDMSG, READMAIL, RD, etc. 1a2

3. If ARPA does not take immediate steps to use these slots, then they will -- through the group allocation scheme -- be swallowed up by others such as RADC, Bell, NIC. From ARPA's point of view, any and all of these are of substantially lower priority than ARPA office use. 1a3

4. The hours of operation of OFFICE-1 coincide with the most overloaded and frustrating period for ISI. Thus ARPA management use of OFFICE-1 will be of greater benefit in relieving network pressure than any single other act I can think of. In particular, it may enable us to move some computational users back to ISI rather than having to foist them on unwilling hosts. 1a4

5. We want to gradually expand the universe of software used by ARPA management people to include the fantastic planet of NLS. Getting them on OFFICE-1 now is a good start. 1a5

6. ISI has been notably unreliable during the primary hours of ARPA management use, while OFFICE-1 has been rock-solid. It is therefore highly attractive to open OFFICE-1 as an option for these users. 1a6

So we're not kidding; we really do want to make OFFICE-1 available to the ARPA office right now. There are other issues we'd like you to address at the same time: 1b

1. These users are accustomed to using the following services and systems: 1b1

SNDMSG 1b1a

READMAIL 1b1b

RD 1b1c

TECO 1b1d

These should be provided so that we can make a smooth transition to NLS later. (I'm sorry about TECO, but that's the way it is.). 1b2

2. Within this group of users, we really must make some sort of reasonable provision for priority use by a sub-group. Within the cloistered computer science community this no doubt seems arbitrary, capricious, and profoundly undemocratic. But it is in fact an inescapable element of any real-world environment into which you will introduce your technology. So we might as well get used to the idea right now, and let the ARPA Director's office serve as a model for a broad class of priority users. 1b3

I suggest therefore that arrangements be made that Lukasik and Tachmindji are never denied access. How this is to be done is up to you folks, but that's what we'd like to see. 1b4

I can well understand that these steps may produce some apprehension on your part, in that dissatisfaction with OFFICE-1 might produce a halo of dissatisfaction with SRI/ARC. I see little cause for fear: 1c

First, reliability of the hardware is perceived as TYMSHARE's responsibility, not ARC's. 1c1

Second, the users will be employing initially software produced elsewhere, so that bugs will be firmly related to other culprits. 1c2

Thirdly, the competition is so miserable that even performance substantially below your usual high standards will look pretty impressive. 1c3

I hope that you'll see these desires as a tribute to the fine management and excellent service the OFFICE-1 project has demonstrated thus far, and not as a callous attempt to wring blood out of a stone. We're motivated by a desire to keep moving toward the goal of a computer-augmented office and to get full utilization of the expensive resources we've procured with so much difficulty. We're willing to discuss a lot, but compromise only a little. 1d

Sincerely, 1e

John. 1f

ARPA Users on OFFICE-1

(J30185) 6-MAR-74 13:48; Title: Author(s): John S. Perry/JSP;
Distribution: /JCN CKM JCRL; Sub-Collections: NIC; Clerk: JSP;
Origin: <ARPA>ARPAUSERS.NLS;1, 6-MAR-74 13:43 JSP ;

JBL 6-MAR-74 19:08 30186

WHEEEEEEE

This is an answer, since I forgot to mention at work that I got your test message. 1

WHEEEEEEE

(J30186) 6-MAR-74 19:08; Title: Author(s): Joel B. Levin/JBL; Distribution: /SEJ JBL; Sub-Collections: NIC; Clerk: JBL;

lynn:
hi, how are things with you and all our friends? i think it may be spring here, things are warming up. we are moving to richmond, joann has found a townhouse type 2 bedroom place for us to live in, i will be staying with somebody in washington 4 days a week then spending weekends in richmond with joann, it sounds like a real drag, but it will cut down on the amount of time we spend commuting. also we should use much less gasoline.
lynn, could you find a copy of network measurement note 18 and send it to me? i am also missing network measurement notes 12 13 14 15 17, if you can get me copies of those i would be appreciative.
say hello to the dinner night group for us.
--jon.

(J30187) 7-MAR-74 05:41; Title: Author(s): Jonathan B. Postel/JBP;
Distribution: /LYNN; Sub-Collections: NIC; Clerk: JBP;

Implementing Multi-Process Promitives in a Multiplexed Computer System 1

Rappaport, Robert L. 1a

November 1968 MAC-55 1b

A.B.S.T.R.A.C.T. 1b1

In any computer system, primitive functions are needed to control the actions of processes in the system. This thesis discusses a set of six such process-control primitives which are sufficient to solve many of the problems involved in parallel processing, as well as in efficient multiplexing of system resoruces among the many processes in a system. In particular, the thesis documents the work performed in implementing these primitives in a particular computer system - the Multics system - which is being developed at M.I.T.'s Project MAC. 2

During the course of work that went into the implementation of these primitives, design problems were encountered which caused the overall rggram design to go through two iterations before program performance was deemed acceptable. The thesis discusses the way the design of these programs evolved during the course of this work. 3

The Graphic Display as an Aid in the Monitoring of A Time.shared Computer System 4

Gro 4a

Abstract - TR 52 thru 55

(J30188) 7-MAR-74 07:08; Title: Author(s): Herb S. Hughes/HSH;
Distribution: /HSH MAP ; Sub-Collections: NIC; Clerk: HS;

Abstract - TR 52 thru 55

Implementing Multi-Process Promitives in a Multiplexed Computer System 1

Rappaport, Robert L. 1a

November 1968 MAC-55 1b

A.B.S.T.R.A.C.T. 1b1

In any computer system, primitive functions are needed to control the actions of processes in the system. This thesis discusses a set of six such process-control primitives which are sufficient to solve many of the problems involved in parallel processing, as well as in efficient multiplexing of system resoruces among the many processes in a system. In particular, the thesis documents the work performed in implementing these primitives in a particular computer system - the Multics system - which is being developed at M.I.T.'s Project MAC. 2

During the course of work that went into the implementation of these primitives, design problems were encountered which caused the overall rpgram design to go through two iterations before program performance was deemed acceptable. The thesis discusses the way the design of these programs evolved during the course of this work. 3

The Graphic Display as an Aid in the Monitoring of A Time-shared Computer System 4

Grochow, Jerrold M. 4a

October 1968 MAC-TR-54 AD-689-468 4b

A.B.S.T.R.A.C.T. 4b1

The Graphical Display Monitoring System was developed as a medium for dynamic observation of the state of a time-shared computer system. The system is integrated to create graphic displays, dynamically retrieve data from the Multics' Time-Sharing System supervisor data bases, and allow on-line viewing of this data via the graphic displays. On-line and simulated experiments were performed with various members of the Project MAC Multics staff to determine the most relevant data for dynamic monitoring, the most meaningful display formats, and the most desirable sampling rates. The particular relevance of using a graphic display as an output medium for the monitoring system is noted. 5

As a guide to other designers, a generalized description of the priciples involved in the design of this on-line, dynamic monitoring device includes special mention of those areas of particular hardware or software system dependence. Several as yet unsolved problems relating to time-sharing system monitoring, including those of security and data base protection, are discussed. 6

The Flow Graph Schemata Model of Parallel Computation 6a

Slutz, Donald R. 6a1

September 1968 MAC-TR-53 AD-683-393 6a2

A.B.S.T.R.A.C.T. 6a2a

Flow Graph Schemata are introduced as uninterpreted models of parallel algorithms, operating asynchronously and reflecting physical properties inherent to any implementation. Three main topics are investigated: (1) determinacy, (2) equivalence, and (3) equivalence-preserving transformations on the control structure of a Flow Graph Schemata. A model is determinate if the results of a computation depend only on the initial values and not on any timing constraints withing the model. Equivalence is undecidable in general, but for a large class of determinate Flow Graph Schemata which are in a maximum parallel form, equivalence is shown decidable. In equivalence-preserving transformations, sufficient tested conditions for equivalence are formulated that depend only on the portion of the structure to be transformed. 7

Current and future computational systems are evaluated in terms of results obtained for Flow Graph Schemata. A number of interesting extensions of the work are suggested. 8

Absentee Computations in a Multiple-Access Computer System 8a

Deitel, Harvey M. 8a1

August 1968 MAC-TR-52 AD-684-738 8a2

A.B.S.T.R.A.C.T. 8a2a

In multiple-access computer systems, emphasis is placed upon servicing serveral interactive users simultaneiously. However, many computations do not require user interaction, and the user may therefore want to run these computations "absentee" (or, user not present). A mechansim is presented which provides for the handling of absentee computations in a multiple-access computer system. The design is intended to be implementation-indepent. Some novel features of the system's design are: a user can switch computations from interactive to absentee (and vice versa), the system can temporarily suspend and then continue absentee computations to aid in maintaining an efficient absentee-interactive workload on the system, system administrative personnel can apportion system resources between interactive and absentee computations in order to place emphasis upon a particular mode during certain periods of operation, and the system's multiple-computation-stream facility which allows the user to attach priorities to his absentee computations by placing the computations in ether low-, standard-, or high-priority streams. 9

10

Abstract - TR 52 thru 55

(J30189) 7-MAR-74 07:38; Title: Author(s): Herb S. Hughes/HSH;
Distribution: /HSH MAP ; Sub-Collections: NIC; Clerk: HS;

DVN 7-MAR-74 08:57 30190

Sndmessage to Someone with Directories on More Than One Machine

I normally work as a user on SR-ARC. Most days I log in to Office-1 atleast once, but usualy not more often. 1

Ofcourse journal items go to me automatically at SRI-ARC. If you want to reach me with a sendmessage, however, The chances are I will get it sooner if you address i to vanNouhuys@office-1. 2

DVN 7-MAR-74 08:57 30190

Sndmessage to Someone with Directories on More Than One Machine

(J30190) 7-MAR-74 08:57; Title: Author(s): Dirk H. Van Nouhuys/DVN; Distribution: /ECW SJM RJ; Sub-Collections: SRI-ARC DEIS; Clerk: DVN;

Implementing Multi-Process Promitives in a Multiplexed Computer System 1

Rappaport, Robert L. 1a

November 1968 MAC-55 1b

A.B.S.T.R.A.C.T. 1b1

In any computer system, primitive functions are needed to control the actions of processes in the system. This thesis discusses a set of six such process-control primitives which are sufficient to solve many of the problems involved in parallel processing, as well as in efficient multiplexing of system resoruces among the many processes in a system. In particular, the thesis documents the work performed in implementing these primitives in a particular computer system - the Multics system - which is being developed at M.I.T.'s Project MAC. 2

During the course of work that went into the implementation of these primitives, design problems were encountered which caused the overall rpgram design to go through two iterations before program performance was deemed acceptable. The thesis discusses the way the design of these programs evolved during the course of this work. 3

The Graphic Display as an Aid in the Monitoring of A Time-shared Computer System 4

Grochow, Jerrold M. 4a

October 1968 MAC-TR-54 AD-689-468 4b

A.B.S.T.R.A.C.T. 4b1

The Graphical Display Monitoring System was developed as a medium for dynamic observation of the state of a time-shared computer system. The system is integrated to create graphic displays, dynamically retrieve data from the Multics' Time-Sharing System supervisor data bases, and allow on-line viewing of this data via the graphic displays. On-line and simulated experiments were performed with various members of the Project MAC Multics staff to determine the most relevant data for dynamic monitoring, the most meaningful display formats, and the most desirable sampling rates. The particular relevance of using a graphic display as an output medium for the monitoring system is noted. 5

As a guide to other designers, a generalized description of the priciples involved in the design of this on-line, dynamic monitoring device includes special mention of those areas of particular hardware or software system dependence. Several as yet unsolved problems relating to time-sharing system monitoring, including those of security and data base protection, are discussed. 6

The Flow Graph Schemata Model of Parallel Computation 6a

Slutz, Donald R. 6a1

September 1968 MAC-TR-53 AD-683-393 6a2

A.B.S.T.R.A.C.T. 6a2a

Flow Graph Schemata are introduced as uninterpreted models of parallel algorithms, operating asynchronously and reflecting physical properties inherent to any implementation. Three main topics are investigated: (1) determinacy, (2) equivalence, and (3) equivalence-preserving transformations on the control structure of a Flow Graph Schemata. A model is determinate if the results of a computation depend only on the initial values and not on any timing constraints withing the model. Equivalence is undecidable in general, but for a large class of determinate Flow Graph Schemata which are in a maximum parallel form, equivalence is shown decidable. In equivalence-preserving transformations, sufficient tested conditions for equivalence are formulated that depend only on the portion of the structure to be transformed. 7

Current and future computational systems are evaluated in terms of results obtained for Flow Graph Schemata. A number of interesting extensions of the work are suggested. 8

Absentee Computations in a Multiple-Access Computer System 8a

Deitel, Harvey M. 8a1

August 1968 MAC-TR-52 AD-684-738 8a2

A.B.S.T.R.A.C.T. 8a2a

In multiple-access computer systems, emphasis is placed upon servicing serveral interactive users simultaneiously. However, many computations do not require user interaction, and the user may therefore want to run these computations "absentee" (or, user not present). A mechansim is presented which provides for the handling of absentee computations in a multiple-access computer system. The design is intended to be implementation-indepent. Some novel features of the system's design are: a user can switch computations from interactive to absentee (and vice versa), the system can temporarily suspend and then continue absentee computations to aid in maintaining an efficient absentee-interactive workload on the system, system administrative personnel can apportion system resources between interactive and absentee computations in order to place emphasis upon a particular mode during certain periods of operation, and the system's multiple-computation-stream facility which allows the user to attach priorities to his absentee computations by placing the computations in ether low-, standard-, or high-priority streams. 9

10

Abstract - TR 52 thru 55

(J30191) 7-MAR-74 09:09; Title: Author(s): Herb S. Hughes/HSH;
Distribution: /HSH MAP ; Sub-Collections: NIC; Clerk: HS;

New FTP codes

Jon and Ken--
The first version of the new ftp code spec is done. You will find it in directory <BBN-NET> in both NLS and text form; the former is (bbn-net, ftpcodes,0:w) and the latter is <BBN-NET>FTPCODES.TXT. Please go over the choice of code numbers and text fairly carefully, to see what I have left out, where I was too ambiguous, or too verbose. Thanks, Nancy 1

New FTP codes

(J30192) 7-MAR-74 09:18; Title: Author(s): Nancy J. Neigus/NJN;
Distribution: /JBP KTP; Sub-Collections: NIC; Clerk: NJN;

Tickler for week of 11 March - 15 March

oIn case, you are interested, Frank Tomaini will be on travel the week of 18 March (THE WHOLE WEEK)

Tickler for week of 11 March - 15 March

(mm3) 11 March - Monday 1

0830 hrs. Branch Chief's Meeting 1a

(mt3) 12 March - Tuesday 2

Due Date - ISIS - Names Submitted for those interested in attending General Electric IR&D Review of Proposed FY-74 Program to be held 21 March. 2a

(mw3) 13 March - Wednesday 3

Due Date - LaForge & Liuzzi - TWX - WWMCCS Standard Software Impact 3a

ISF Confessions 0830 hrs. 3b

FY-75 DSF Submission - ISIS/D. Nelson - AF Form 111 w/AF Form 725 and RADC Form 7...due in DORP NLT 15 Mar 3c

Due Date - ALL DOCUMENTATION CLERKS - Emergency Change to AFM 12-50 3d

(mth3) 14 March - Thursday 4

0830 hrs. Branch Chief's Meeting 4a

Laboratory Activity Reports due today: Bucciero must have them by 1000, ISM must have them by 1100, and DOT must have them by 1600. 4b

(mf3) 15 March - Friday 5

Timecards due today 5a

Bobbie: Travel figures due by noon. 5b

Tickler for week of 11 March - 15 March

(J30193) 7-MAR-74 13:04; Title: Author(s): Roberta J. Carrier/RJC; Distribution: /RADC; Sub-Collections: NIC RADC; Clerk: RJC;

JOVIAL Manual--Chapter 4

Contains ↑ & ←, structured

Chapter 4 1

↑FORMULAS 1a

4.1 Concept of ↑Formulas 1a1

Chapter 3 discusses ↑variables, the constructs standing for elements of data whose values may be changed. ↑Formulas are the means for specifying the new values for ↑variables. ↑Formulas also generally supply values for any purpose--such as comparisons and other selections of courses of action. Since ↑constants and ↑variables denote values they are also ↑formulas. 1a1a

.1 Any ↑numeric:formula can be used as a ↑pointer:formula. The details of the use of ↑pointer:formulas are given in Section 7.8. ↑Value:formulas and ↑numeric:value:formulas can occur only in ↑loop:controls. The details of their use are explained in section 5.8. 1a1a1

4.2 ↑Constant:Formula 1a2

A ↑constant:formula is a ↑formula whose value can be determined at compile time, once and for all. That particular criterion is somewhat system dependent. In places in this language specification where a ↑formula is called for, it is only a matter of efficiency whether a ↑constant:formula is evaluated at compile time or execution time. A ↑constant:formula, however, can be used in places where this manual calls explicitly for a ↑constant. The ↑constant:formula must then be evaluated at the time it is encountered in order properly to compile the ↑program:declaration. The same consideration applies to a place where a ↑number is required, but not as part of another ↑symbol such as a ↑floating:constant. When a ↑constant:formula is used to represent a number, it must evaluate to an appropriate integer value. In general, parts of this document which require ↑constants or ↑numbers do not reiterate this permission to use ↑constant:formulas. A ↑constant:formula is not permitted as part of a ↑form:list, which is, after all, a second level syntax equation applied to that which is first the value of a ↑character:formula. 1a2a

4.3 ↑Conditional:Formula 1a3

There is no data type that is intrinsically conditional; however, any ↑formula can be considered a ↑conditional:formula in the appropriate setting. A

↑conditional:formula is the ↑formula following any of the three ↑primitives ←IF, ←WHILE, ←UNTIL (see sections 5.7 and 5.8 on ↑conditional:statements and ↑loop:statements) or the ↑directive:key ←!TRACE. A ↑formula of any type can be used in these positions. After all operations are performed as called forth in the ↑formula --bit or byte extraction, shifting, concatenation, function evaluation, comparisons, arithmetic, logical combination, attribute guidance, etc.--the rightmost bit of the result is examined without further conversion. If that rightmost bit is ←0 the ↑conditional:formula represents the logical predicate "false". If the rightmost bit is ←1 the ↑conditional:formula represents the logical predicate "true". This can, of course, lead to machine dependencies if ↑conditional:formulas contain any operands other than unsigned integers except in ↑comparisons. For example, a negative integer as a ↑conditional:formula will lead to a result on a one's complement machine opposite to the result on a two's complement machine. The following table indicates the action to take, depending on the value of the ↑conditional:formula 1a3a

4.4 ↑Character:Formula 1a4

↑Character:constant is explained in Section 2.8.1. ↑Character:variable is explained in Section 3.4.2. ↑Character:form is one of the two types of form, explained in Section 4.17.2. A ↑function:call is the invocation of a certain kind of ↑procedure:declaration as explained in Section 4.18. A ↑character:function:call is the invocation of one of these special ↑procedure:declarations having its effective output parameter of character type. One of the ↑intrinsic:function:calls (see Section 4.19), the ↑byte:string:function:call, is a ↑character:function:call. 1a4a

.1 Any ↑character:formula represents a value having a size measured in bytes. For its use in the ↑byte:string:function:call, the bytes of the ↑character:formula (any ↑character:formula can be used where indicated as the first ↑actual:input:parameter in the metalinguistic equation) are numbered starting with zero on the left. With respect to this numbering, the first ↑numeric:formula (the second ↑actual:input:parameter) tells which byte of the stated ↑character:formula is to become the first (leftmost) byte of the derived ↑character:formula. The second ↑numeric:formula, if present, tells how many bytes (following consecutively to the right) are to be included in the derived ↑character:formula. If the second

↑numeric:formula is missing, just one byte is used. The ↑numeric:formulas must yield non-negative values. Only the integer parts of these values are used--the fractions are truncated. The sum of the two values must not exceed the size of the first ↑actual:input:parameter. If the second ↑numeric:formula (the third ↑actual:input:parameter) has a value of zero, then the ↑byte:string:function:call represents a character value of zero size. Such a value as an operand in concatenation leaves the other operand unchanged. It can be appropriately padded in any context in which it might occur. For instance, as a ↑conditional:formula it would be padded on the left with a single bit of value zero, which would thus become the rightmost bit of the ↑conditional:formula, leading to the logical predicate "false". As an operand of ←AND, OR, etc., it would become a string of bits of value zero to be combined with the bits of the other operand. Example: 1a4a1

←ALPHA = 'OA2C4E6G8I'; 1a4a2

←BETA = BYTE (ALPHA,3,5); 1a4a3

←GAMMA = BETA <> 'C4E6G'; 1a4a4

.2 In the above sequence of code, ←GAMMA becomes zero because ←BETA does indeed contain the value ←C4E6G. 1a4a5

.3 the ↑ampersand is the only operator that can apply to ↑character:formulas. It means concatenation. 1a4a6

↑character:formula ←& ↑character:formula 1a4a7

is a ↑character:formula. Its value is the concatenation of the bytes (all the bytes) of its left operand on the left with the bytes of its right operand on the right. Its size is the sum of the sizes of its operands. Example: 1a4a8

.4 A ↑character:formula can consist of concatenations. The ordinary left-to-right rule applies--the two leftmost operands are concatenated first. Then the result is concatenated with the next ↑character:formula to the right. Ordinarily it really makes no difference if concatenation is done left-to-right or right-to-left, but in cases where the resultant size might exceed system-dependent limits some system-dependent differences might arise. Example: 1a4a9

←(ALPHA & BETA) & (GAMMA & DELTA) 1a4a10

.5 Notice the ↑parentheses in the above example. A parenthesized ↑character:formula is also a ↑character:formula. The utility of the ↑parentheses is to change the order of concatenation--operations within ↑parentheses are performed before the value of the parenthesized ↑formula is used in further operations. In the above example ←ALPHA is concatenated with ←BETA, GAMMA is concatenated with ←DELTA and then these two results are concatenated together. A ↑formula of any type can be used as a ↑formula of any other type--its value is appropriately transformed. ↑Parentheses may, at times, be significant in determining the type of ↑formula. 1a4a11

.6 A ↑bit:formula may be used in a context requiring a ↑character:formula. The most obvious such context is as the first ↑actual:input:parameter to the ↑byte:string:function:call. Assignment to a ↑character:variable does not make a ↑bit:formula into a ↑character:formula. For the use of a ↑bit:formula in assigning a value to a ↑character:variable see Section 5.5.1. In concatenation of a ↑bit:formula and a ↑character:formula the ↑bit:formula is stronger--the ↑character:formula is treated as a ↑bit:formula. In the ↑byte:string:function:call, a ↑bit:formula as the first ↑actual:input:parameter is padded on the left with however many bits of value zero are needed to yield an integral number of bytes in the value. The resulting bit string is then considered a byte string and the ↑numeric:formulas are used to select the desires byte string. For example, suppose that in a system in which bytes consist of eight bits each, there is a ↑byte:string:function:call requiring ←3 bytes starting with byte ←1 (the 2nd byte) of a ↑bit:formula of ←35 bits. The following table illustrates the example and shows the resultant value of the ↑byte:string:function:call 1a4a12

4.5 ↑Numeric:Formula 1a5

↑Numeric:constant is explained in section 2.8.11. ↑Numeric:variable is explained in section 3.5. A ↑numeric:function:call is the invocation of a ↑procedure:declaration (see Section 8.4.) having an implicit output parameter of numeric type. Several of the ↑intrinsic:function:calls are ↑numeric:formulas (see Section 4.19). 1a5a

.1 A ↑bit:formula in a context requiring a ↑numeric:formula is treated as an unsigned integer value. The string of bits comprising the value of the ↑bit:formula is considered, without any change, conversion or alteration, as the magnitude of a non-negative integer value. If its size is too great for the use to which it is being put, leading bits are truncated to reduce its size to the maximum that can be used for the arithmetic, conversion, indexing, pointing or formatting. If its size is unknown at compile time it is given a system-dependent default size (if there is any possibility it could be larger) in which the rightmost bits are right justified and any extra leading bits at execution time are zeros. This default size is most likely to be the largest size of unsigned integer with which integer arithmetic may be done conveniently. If its default size is unknown, but its maximum possible size is known to be less than the default size, the maximum possible size is taken as the size of the unsigned integer in the numeric context. 1a5a1

.2 Being in a position to be assigned to a ↑numeric:variable, being an ↑actual:input:parameter corresponding to a numeric ↑formal:input:parameter, or being compared with a ↑numeric:formula, does not impose numeric assumptions on a ↑bit:formula. The contexts requiring any formula to be treated as a ↑numeric:formula are as follows: 1a5a2

a. As an operand to participate in arithmetic. 1a5a2a

b. As an operand to be converted to a numeric in accordance
with attribute quidance. 1a5a2b

c. As an ↑index:component. 1a5a2c

d. As a ↑pointer:formula. 1a5a2d

e. As an operand to be encoded for "output" in accordance with a ↑numeric:format. 1a5a2e

4.6 Arithmetic 1a6

↑Arithmetic:operators are used to specify arithmetic calculation in determining numeric values. The meanings of the ↑arithmetic:operators are as follows: 1a6a

←+ Add. 1a6a1

←- Subtract. 1a6a2

←* Multiply. 1a6a3

←/ Divide. 1a6a4

←\ Determine the residue (modulo). 1a6a5

←** Raise to the power of (exponentiation). 1a6a6

.1 The syntax equations permit long sequences of ↑plus:minus and ↑minus:signs before an operand. The effect of such a sequence can easily be determined by counting the ↑minus:signs and ignoring the ↑plus:signs. If there is an even number of ↑minus:signs, the entire sequence is equivalent to one ↑plus:sign. If there is an odd number of ↑minus:signs, the entire sequence is equivalent to one ↑minus:sign. 1a6a7

.2 The ↑minus:sign as a unary operator means to negate (take the additive inverse of) the following ↑numeric:formula. The ↑plus:sign can be used as a unary operator, but it has no effect. Multiplication must be indicated by means of an ↑asterisk; there is no operation specified by merely placing ↑formulas next to one another. Since there is no provision for vertical spacing, exponentiation must be shown by means of the double ↑asterisk. The meanings of addition, subtraction, multiplication, division and exponentiation are well known, but it is well to emphasize certain points. The result of division by a zero value is undefined. The result of exponentiation of a negative base by a non-integer exponent is undefined. 1a6a6

.3 Determination of a residue, ←x\y, means finding the value of the archetypal number to which ←x is congruent, modulo ←y. In the sence that ←x*y is called ←"x times ←y", let us refer to ←x\y as ←"x modulo ←y". For a given value of ←y, ←x\y is a sawtooth function of ←x. For positive values of ←y, ←0 <= x\y < y, if ←0 <= x < y, x\y = x; otherwise x\y = x - n*y, where ←n is an integer value (positive or negative). For negative values of ←y, let ←y = -u; then ←-u < x\y <= 0, if ←-u < x <= 0, x\y = x; otherwise ←x\y = x - n*u, where ←n is a positive or negative integer value. For ←y = 0, x\y is undefined. These relationships are illustrated in the graphs of Figure 4-1. Examples: 1a6a9

.4 The order of evaluation of a ↑numeric:formula is left to right, except that an operator of higher precedence makes use of an operand lying between it and an operator of lower precedence. Enclosing a ↑formula in ↑parentheses raises the precedence of all operators within the ↑parentheses above that of all operators outside the ↑parentheses. Within one parenthesized or unparenthesized group, exponentiation has the highest precedence of arithmetic operations;multiplication, division, and determination of residues have the next lower precedence; and addition and subtraction (or negation) have the lowest arithmetic precedence. The value of ←54/6/3 is ←3, not ←27, because of the left-to-right rule. Precedence and evaluation order are discussed in considerable detail with respect to all possible operations (including arithmetic) in Section 4.15. 1a6a10

4.7 Default Scaling 1a7

The type (integer, fixed, or floating) of a value denoted by a ↑numeric:formula, and its scaling, depend on the attributes of its constituent ↑numeric:formulas and the arithmetic involved. The left-to-right rule and the precedence rules determine the order in which the values of two operands are combined--to form a single value to be an operand in another combination--or for assignment or other uses. The resultant value has scaling and type attributes to be taken into account with respect to further processing. 1a7a

.1 Floating values in some systems have only method of representation, with a given number of bits in the significand and a given number in the exrad. Other systems may provide forms of representation with extra precision (more bits in the significand), or extra range (more bits in the exrad), or both. 1a7a1

.2 If both operands for an arithmetic operation are floating values, the operation is carried out in floating form and the result is a floating value. The precision and range for the operation and of the result are the maximums, respectively, of the precisions and ranges of the two operands. 1a7a2

.3 If one operand is a floating value and the other is fixed or an integer, the operation is carried out in floating form and the result is a floating value. The precision and range for the operation and of the result are those of the floating operand. The fixed or integer operand must, of course, be converted to floating form before the operation. 1a7a3

.4 Several following sections discuss the scaling in arithmetic with values that are not floating. We use codes consisting of one or two characters with the following meanings: 1a7a4

.5 The number of fraction bits of integers is undefined, and disregarded in the scaling formulas below. The number of integer bits of integers is the same as the size. The number of integer bits of fixed values is the size minus the number of fraction bits. (Fraction bits or integer bits, but certainly not both, can be less than zero in number.) The sizes and fraction bits of items are determined by their ↑declarations. The sizes and fraction bits of ↑constants are implicit in their values (no leading zeros are included). For certain ↑variables, notably ↑letter:control:variables, there are system-dependent sizes. Probably, the size of ↑letter:control:variables is the size the system uses for addresses. The sizes of ↑intrinsic:function:calls are stated in Section 4.19. The sizes and fraction bits of other ↑function:calls match the sizes and fraction bits of their implicit output parameters. The sizes of the values represented by ↑bit:formulas must often be computed dynamically during execution of a program. This is too great a burden to impose, however, in the general case of scaling ↑numeric:formulas. Therefore, the sizes of ↑bit:formulas used as ↑numeric:formulas are determined as stated in Section 4.5.1. 1a7a5

.6 If both operands for an arithmetic operation are integer values, the result is an integer (possible exception for exponentiation) with the following scaling: 1a7a6

a. For addition and subtraction: 1a7a6a

←IR ←= minimum (←Z, ←1 + maximum (←I1, ←I2) 1a7a6a1

b. For multiplication: 1a7a6b

←IR = minimum (←Z, ←I1 + I2) 1a7a6b1

c. For division: 1a7a6c

←IR = IN 1a7a6c1

d. For determination of residues: 1a7a6d

←IR = minimum (←IN, ←IM) 1a7a6d1

e. For exponentiation, only if the exponent is a positive ↑integer:constant 1a7a6e

←IR = minimum (←Z, ←VE * IB) 1a7a6e1

.7 For addition and subtraction of an integer value and a fixed value or of two fixed values: 1a7a7

a. ←IR ←= 1 + maximum (←I1, I2) 1a7a7a

b. ←AR = minimum (←A1, ←A2) 1a7a7b

If ←IR + AR > Z, convert both operands to floating values, carry out the operation in floating form, and keep the result as a floating value. The precision of the floating form is system dependent. 1a7a7c

.8 For multiplication of an integer value and a fixed value or of two fixed values: 1a7a8

a. ←IR = I1 + I2 1a7a8a

b. ←AR = A1 + A2 1a7a8b

c. If ←IR + AR > Z, convert to floating mode as in Section 4.7.7c. 1a7a8c

.9 For division of an integer numerator by a fixed denominator: 1a7a9

a. ←IR = IN +AD 1a7a9a

b. ←AR = 2 * ID + AD - 1 1a7a9b

c. If ←IR + AR > Z, convert to floating mode as in Section 4.7.7c. 1a7a9c

.10 For division of a fixed numerator by an integer denominator: 1a7a10

a. ←IR = IN 1a7a10a

b. ←AR = ID + AN 1a7a10b

c. If ←IR + AR > Z, convert to floating mode as in Section 4.7.7c. 1a7a10c

.11 For division of two fixed values: 1a7a11

a. ←IR = IN +AD 1a7a11a

b. ←AR = IR + AN 1a7a11b

c. If ←IR + AR > Z, convert to floating mode as in Section 4.7.7c. 1a7a11c

.12 For determination of the residue of an integer numerator by a fixed modulus: 1a7a12

a. ←IR = minimum (←IN, ←IM) 1a7a12a

b. ←AR = AM 1a7a12b

.13 For determination of the residue of a fixed numerator by a fixed or integer modulus: 1a7a13

a. ←IR = minimum (←IN, ←IM) 1a7a13a

b. ←AR = AN 1a7a13b

.14 For exponentiation by any exponent not an integer constant value, convert to floating mode as in Section 4.7.7c. 1a7a14

.15 For exponentiation of a fixed base by a positive ↑integer:constant 1a7a15

a. ←IR = VE * IB 1a7a15a

b. ←AR = VE * AB 1a7a15b

c. If ←IR + AR > Z, convert to floating mode as in Section 4.7.7c. 1a7a15c

.16 For exponentiation of an integer base by a negative integer constant value: 1a7a16

a. ←IR = 1 1a7a16a

b. ←AR = - 2 * VE * IB - 1 (Note that ←VE is negative.) 1a7a16b

c. If ←IR + AR > Z, convert to floating mode as in Section 4.7.7c. 1a7a16c

.17 For exponentiation of a fixed base by a negative integer constant value: 1a7a17

a. ←IR = 1 - VE * AB 1a7a17a

b. ←AR = - VE * (2 * IB + AB) - 1 (Note that ←VE is negative.) 1a7a17b

c. If ←IR + AR > Z, convert to floating mode as in Section 4.7.7c. 1a7a17c

4.8 Uniform Rules of Calculation 1a8

The scaling rules for ↑formulas used in indexing and pointing are the same as the rules for all ↑formulas. When the value is finally set up to be used as an address (base or increment) it is as if it were being assigned to an ↑integer:variable of the system-dependent size used for addresses. Certain arithmetic operations are carried out without explicit direction from the programmer--operations involved with such activities as calculation of addresses and the incrementing and testing of ↑control:variables. 1a8a

.1 All intrinsic numeric quantities have system-dependent sizes. All calculations carried out in response to implicit directions are scaled in accordance with the default scaling rules applied to calculations explicitly directed. System-dependent documwntation may make specific exceptions to this rule. 1a8a1

4.9 Attribute Guidance 1a9

A ↑description:attribute is a numeric ↑item:description (one beginning with ←F, ←S, or ←U or the ↑name of an item whose ↑declaration contains a numeric ↑item;description. In any case its meaning is the same whether the ↑item:description is cited directly, or indirectly through the ↑item:name. A character ↑item:description is not used with ↑attribute:association since it would provide only a fraction of the power available in the ↑byte:string:function:call. 1a9a

.1 The effect of applying ↑attribute:association to a ↑formula is to first consider the ↑formula as a ↑bit:formula and then to impose the ↑description:attribute on this string of bits, causing it to be treated as a ↑numeric:formula of the stated type, size and precision. (↑Status:constants in the ↑item:description are of no effect with regard to the type, size, and precision imposed on the ↑formula.) If the next use of this ↑numeric:formula is as a numerator (for division or residue determination), its maximum permitted size is increased from ←Z to ←Y (see Section 4.7.4). Usually ↑parentheses are required to delimit the ↑formula to which ↑attribute:association is applied, but if the ↑formula is a ↑function:call, a ↑variable without an explicit ↑pointer:formula, or a ↑constant, the enclosing ↑parentheses are not required. Examples: 1a9a1

←(AA + BB) @@ [S,R 17] 1a9a1a

←CC @@ [U, SIZE (CC)] 1a9a1b

←ALT (P1) @@ [F] 1a9a1c

←7 @@ [U,3] 1a9a1d

←(DD [I,J,K] @ PNTR) @@ [U] 1a9a1e

←EE [X,Y,Z] @ (FF @@ [U]) 1a9a1f

.2 In the first example, the rightmost 18 of the bits representing the sum of ←AA and ←BB are treated as a signed, rounded integer, 17 bits in size. Then, the bits of ←CC are treated as an unsigned, fixed value of default size with all ← CC's magnitude bits (however, many there are) after the point. Then the bits representing the value representing the currently active entrance of procedure ←P1 are treated as a floating value. Then the bits (three in this case) representing the ↑constant ←7 are treated as an unsigned, fixed value of default size with three bits after the point (padded with enough integer bits of value zero to the default size). In the next-to-last example, after the particular instance of ←DD is found it is treated as an unsigned integer of default size. In the last example, it is ←FF that is first treated as an unsigned integer of default size and then used as a pointer to find an instance of ←EE. 1a9a2

.3 ↑Evaluation:control can be applied, in exactly the same manner as ↑attribute:association, to any ↑formula. The effect is somewhat different, however. The value of the ↑formula to which ↑evaluation:control is applied is converted to the numeric configuration required by the ↑description:attribute. Examples: 1a9a3

←(AA * BB) @ [S 30,15] 1a9a3a

←BYTE(CITY[15],J)@[F] 1a9a3b

.4 ←AA and ←BB are multiplied, using the normal scaling rules, and then the value is converted to the form of a signed, fixed value of size 30 (not counting the sign) with 15 bits after the point. It is, of course, permissible for the compiler to optimize the operation and avoid, for example, converting ←AA and ←BB each to floating form and the result back from floating form. In the second example, one byte of character data is in a position calling for a numeric value. So, according to the rules, the character datum is considered first a ↑bit:formula, then an unsigned integer, and then it is converted to floating form. 1a9a4

4.10 Scaling under ↑Evaluation:Control 1a10

↑Evaluation:control, unlike ↑attribute:association, can be applied to a binary ↑arithmetic:operator as shown at the top of the box in Section 4.9 The effect is to require that the operation be performed so that the result comes out in the form prescribed by the ↑description:attribute. The precedence rules for ↑arithmetic:operators are unchanged when they are followed by ↑evaluation:control. 1a10a

.1 If the prescribed form of the result is floating, both operands are converted to floating form of the prescribed precision before the operation, and the operation is then carried out in the prescribed mode. The compiler may, of course, do the operation in a more efficient manner if, on the basis of the known attributes of the operands, no accuracy is lost thereby. 1a10a1

.2 For non-floating addition and subtraction, the maximum allowable size is ←Z. If rounding is not prescribed, both operands are rescaled with at least as many integer bits as ←IS and at least as many fraction bits as ←AS. If rounding is prescribed and ←IS + AS = Z, both operands are rounded to ←AS before the operation. If rounding is prescribed and ←IS + AS < Z, both operands are rescaled with at least ←AS + 1 fraction bits, and rounding is done after the operation. Rescaling of operands before the operation includes the conversion of floating operands to fixed form. 1a10a2

.3 For non-floating multiplication, the scaling must be done after the operation. If both operands are floating, the multiplication is done in floating form and the result is converted to the prescribed scaling. If one operand is floating, it is converted to fixed in accordance with the following formulas (operand 2 is the one converted from floating to fixed) before the multiplication: 1a10a3

a. ←I2 = IS - I1 1a10a3a

b. ←A2 = AS - A2 1a10a3b

.4 In multiplication, if ←I1 + I2 (for integers) or if ←I1 + I2 + A1 + A2 (for fixed numbers) is not greater than ←Z, it may be that the system can provide a less expensive multiplication. In any case, the prescribed size, ←IS (or ←IS + AS must usually be no greater than ←Z. Depending on the system, however, if the next use of the product is to be treated as a bit string or as the numerator in division or determining a residue, the maximum permitted size may be ←Y. 1a10a4

.5 If even one operand is floating, division must be carried out in floating form and the quotient then converted in accordance with the prescribed scaling. For fixed or integer operands, divison is carried out with the prescribed scaling. The programmer guarantees that no machine divide error will occur. 1a10a5

.6 In determining a residue, floating operands are converted to the prescribed scaling before the operation. The operation is carried out with the prescribed scaling. If a division is involved, the programmer guarantees that no machine divide error will occur. 1a10a6

.7 For non-floating exponentiation, the operation in accordance with the default rules and then rescaled as prescribed. 1a10a7

4.11 Calculating, Rounding, Packing, Storing, Retrieving 1a11

The discussions of scaling above are concerned with assumptions of what bits are worth saving in performing numeric calculations, If ←IS + AS or ←IR + AR turn out less than ←Z, there is no requirement for the compiler to see to it that extra bits are scraped off, except as specifically explained below, before an intermediate result is used in further calculation. Most algorithms are insenstive to the presence of noise bits. In the case of an algorithm that is sensitive to these bits, the programmer must be careful--perhaps using shorter statements--to insure cleaning up these bits. If a calculation produces extra bits on the left--beware--the programmer is responsible. 1a11a

.1 When a numeric value is rounded in accordance with the appearance of an ←R in an ↑item:description or in a ↑description:attribute it means that, in terms of absolute values, a ←1 is added to the leftmost noise bit (perhaps causing a carry into the rightmost signifcant bit) and then all the noise bits are replaced with bits of value zero. 1a11a1

.2 "Significant bits" are the bits included in the size of fixed and integer ↑variables and ↑formulas included in the significand of floating ↑variables and ↑formulas. "Noise bits" are any bits to the right of the rightmost significant bit, representing a value less in absolute value than a 1 bit as the rightmost significant bit. Noise bits ordinarily arise during the execution of arithmetic operations--which often produce bits of no significance according to the scaling rules or a ↑description:attribute. 1a11a2

.3 If rounding is not specified, it does not mean to take any measures to suppress noise bits. When storing a rounded or unrounded value, the compiler protects items adjacent to the stored item, in adjacent words or in adjacent bits in the same word (assuming the ↑packing:specification does not deny such care). 1a11a3

.4 When retrieving a numeric value from storage, the compiler avoids retrieving bits from adjacent items, in adjacent words or in adjacent bits in the same word. The compiler is concerned about avoiding the retrieval of noise bits or bits to the left in the same word as the retrieved item if and only if those bits are in space allocated to other items. Among items with positioning information, dense packed items are, by definition, adjacent to other data and medium packed items are alone in the word part defined by the medium packing, but adjacent word parts are occupied. For compiled packed data, the compiler knows what is adjacent. The density may be less than the programmer specifies in the ↑declaration. 1a11a4

.5 Although specific storage and retrieval methods are not specified here, the compiler avoids narrow storing followed by broad retrieval. If "garbage" is retrieved, it is only because the programmer causes a ↑variable to be used before it is set, sets the ↑variable using legitimate but excess bits developed during a calculation, or sets something else "overlaid" with the ↑variable. 1a11a5

4.12 ↑Bit:Formula 1a12

A ↑bit:formula is the representation of a string of bits, without regard to any meaning it might have as a numeric value or as a string of bytes. Thus, in a context requiring a ↑bit:formula, a ↑numeric:formula or a ↑character:formula may be used, and the bit string it represents is utilized without regard to its numeric or character meaning. 1a12a

.1 ↑Pattern:constant is explained in Section 2.8.9. ↑Entry:variable is explained in Section 3.2.1. ↑Bit:form is one of the two types of ↑form explained in Section 4.17. ↑Function:calls invoking procedures declared by the programmer cannot be ↑bit:formulas since there is no way to specify "bit" as a type for the implicit output parameter. Three of the ↑intrinsic:function:calls, however, are ↑bit:formulas. These three are the ↑shift:function:call, the ↑signed:function:call, and the ↑bit:string:function:call. 1a12a1

.2 Any ↑formula, even a ↑character:formula, represents a value consisting of a string of bits. For its use in the ↑bit:string:function:call, the bits of any ↑formula used as the first ↑actual:input:parameter are numbered, starting with zero on the left. The leftmost bit of the leftmost byte of a ↑character:formula is bit zero. The sign bit of signed (←S) values is bit zero and the leftmost magnitude bit is bit one. The leftmost magnitude bit of unsigned (←U) values is bit zero. The leftmost bit of floating values (←F) is bit zero, but it is system depepdent whether this is the sign of the significand, the sign of the exrad, a magnitude bit of the significand, or a magnitude bit of the exrad. With respect to this numbering of the bits of the first ↑actual:input:parameter, the second ↑actual:input:parameter tells which bit of the stated ↑formula is to become the first (leftmost) bit of the derived ↑bit:formula. The third ↑actual:input:parameter, if it is present, tells how many bits (following consecutively to the right) are to be included in the derived ↑bit:formula. If the third ↑actual:input:parameter is missing, just one bit is used. The ↑numeric:formulas must yield non-negative values. Only the integer parts of these values are used--the fractions are truncated. The sum of the two values must not exceed the number of bits represented by the first ↑actual:input:parameter. If the third ↑actual:input:parameter has a value of zero, then the ↑bit:string:function:call represents a bit string of zero size. Such a value as an operand in concatenation leaves the other operand unchanged. 1a12a2

.3 The ↑shift:function:call yields a ↑bit:formula derived from the first ↑actual:input:parameter by shifting it left or right in accordance with the value of the second ↑actual:input:parameter. The specifics of the shifting are as follows: 1a12a3

a. The string of bits representing the value of the cited ↑bit:formula is considered to be framed by a window whose width is the size of the ↑bit:formula. 1a12a3a

b. There are infinite strings of zero bits attached to the left and right sides of the ↑bit:formula and hidden by the window frame. 1a12a3b

c. The ↑numeric:formula is evaluated to an integer, truncated if necessary. 1a12a3c

d. The infinite string of bits consisting of those to the left behind the window frame, those within the window, and those to the right behind the window frame is shifted left or right with respect to the window by the number of bits indicated by the value of the ↑numeric:formula. The shift is to the left past the window if the ↑numeric:formula is positive. The shift is to the right past the window if the ↑numeric:formula is negative. 1a12a3d

e. The resulting ↑bit:formula is the same size as the original ↑bit:formula and has the value now appearing in the window. 1a12a3e

.4 The following table gives some sample results: 1a12a4

.5 The ↑signed:function:call is a ↑bit:formula one bit in size. Its value depends only on the type, not the value, of its ↑actual:input:parameter. The value of the ↑signed:function:call is ←1 if its ↑actual:input:parameter is floating or signed; otherwise the value is zero. The sign of a ↑numeric:formula depends on many factors, as follows: 1a12a5

a. ↑Attribute:association or ↑evaluation:control overrides all other considerations; otherwise 1a12a5a

b. A ↑bit:formula treated as a ↑numeric:formula is unsigned. 1a12a5b

c. ←SIGNED ←(↑function:call←) depends on the attributes of the implicit output parameter for an intrinsic or a programmed function. 1a12a5c

d. If a ↑formula is floating, none of the below rules relating to arithmetic apply. 1a12a5d

e. Any arithmetic operations, other than subtraction, involving unsigned operands leave the ↑formula unsigned. 1a12a5e

f. Exponentiation by an even ↑constant yields an unsigned ↑formula. 1a12a5f

g. Determining a residue with an unsigned modulus yields an unsigned ↑formula. 1a12a5g

h. In all other cases, the formula is ↑signed. 1a12a5h

4.13 ↑Comparisons and ↑Chain:Comparison 1a13

A ↑comparison is a ↑bit:formula one bit in size. A ↑comparison consists of a left operand, a ↑relational:operator, and a right operand. It has the value ←1 if the left operand stands in the relationship stated by the ↑relational:operator with respect to the right operand. Otherwise, the ↑comparison has the value zero. The ↑relational:operators, with their meanings, are given in the box above. If both operands are ↑numeric:formulas, the truth or falsity of the ↑comparison is based on the numeric value resulting from the subtraction of one operand from the other. In performing this subtraction all the rules that apply to arithmetic between ↑numeric:formulas are in force. 1a13a

.1 If one operand is a ↑bit:formula, the other operand becomes a ↑bit:formula (if it is not to begin with); i.e., the bits representing the value are merely considered as a string of ones and zeros, without further meaning. The truth or falsity of the ↑comparison then is based on subtracting one ↑bit:formula from the other--now considering each to be an unsigned integer. For the purpose of ↑comparison of ↑bit:formulas, subtractions of one unsigned integer from another can accommodate operands of any size. If one ↑bit:formula is shorter than the other, the shorter is considered to be extended or padded on the left with bits of value zero before the subtraction. There is no prescribed method for the compiler to implement the ↑comparison. As long as the results are the same, the arithmetic can be done by parts or backwards or forwards, or the bits can be compared one by one until the value of the ↑comparison is determined. 1a13a1

.2 If one operand of a ↑comparison is a ↑numeric:formula and the other is a ↑character:formula, they both become ↑bit:formulas for the purpose of the ↑comparison. 1a13a2

.3 If both operands of a ↑comparison are ↑character:formulas, the truth or falsity of the ↑comparison is determined by considering each operand to be an unsigned integer and then subtracting one from the other, as in comparing ↑bit:formulas. However, if one ↑character:formula consists of fewer bytes than the other, the shorter is padded on the right with space characters to equalize the sizes before the subtraction. 1a13a3

.4 A ↑chain:comparison is a ↑bit:formula having a size of one bit and a value of zero or 1. Each ↑chain:comparison is nearly equivalent to the logical product of two or more ↑comparisons. Consider the following logical product, where each ←R is a ↑relational:operator and each ←F is a ↑formula (see Section 4.14.3 for meaning of ←AND): 1a13a4

←F R F AND F R F AND ... F R F 1a13a4a

.5 The effect is nearly the same as the ↑chain:comparison 1a13a5

←F R F R F R ... F R F 1a13a5a

.6 It is nearly the same because in the form with the explicit ←ANDs, ←F to ←F each appear twice. If these ↑formulas contain ↑function:calls requiring an explicit execution for each explicit appearance, such ↑function:calls would be executed twice, while in the ↑chain:comparison they would be executed just once. ←F to ←F , if they are numeric, may require different scalings (or worse) in their two ↑comparisons. Nevertheless, they are each evaluated just once. If ←F is a ↑fixed:formula, it may be seen that not all its precision is needed for the subtracting in either of its two ↑comparisons. Enough precision must be saved, however, for its more precise ↑comparison. It may be that in one of its ↑comparisons it must be converted to floating--or perhaps it will be treated as a

↑bit:formula. Then all the precision called for by the scaling rules must be saved. 1a13a6

.7 A ↑chain:comparison requires some ↑formulas to be used twice in effecting ↑comparisons. The scaling or interpretation of a ↑formula needed in effecting one ↑comparison does not influence the scaling or interpretation of that same ↑formula in effecting its second ↑comparison. Consider, for example: 1a13a7

←BIT (ALPHA,I,J) < GAMMA (BETA) < EPSILON + DELTA 1a13a7a

.8 In the above ↑chain:comparison, the first of the three ↑formulas being compared is clearly a ↑bit:formula and the third is clearly a ↑numeric:formula. Let us suppose the middle ↑formula is a ↑function:call that returns the factorial of its ↑actual:input:parameter, a ↑numeric:formula. The output of ←GAMMA is treated as a ↑bit:formula for ↑comparison with the bit string from ←ALPHA and as a ↑numeric:formula for ↑comparison with the sum of ←EPSILON and ←DELTA. 1a13a8

4.14 Operations on ↑Bit:Formulas 1a14

↑Bit:formulas represent strings of bits, each of value zero or ←1. ↑Comparisons and ↑chain:comparisons are ↑bit:formulas, in these cases only one bit long, with values of zero or ←1. ↑Bit:formulas can be combined or transformed in various ways as indicated below. 1a14a

.1 When ←NOT is applied to a ↑bit:formula it produces a derived ↑bit:formula in which each ←1 in the value of the stated ↑bit:formula is replaced with zero and each zero is replaced with ←1. The derived ↑bit:formula is the same size as the stated ↑bit:formula. 1a14a1

.2 Concatenation of two ↑bit:formulas, indicated by an ↑ampersand between the two ↑bit:formulas, yields a ↑bit:formula whose size is the sum of the sizes of the two component ↑bit:formulas. The value of the resultant ↑bit:formula is the bits of the ↑bit:formula on the right appended to the right of the bits of the ↑bit:formula on the left. Examples: 1a14a2

.3 A ↑logical:operator applies to all the pairs of the bits of the two ↑formulas to which it is applied as an infix operator. The two bit strings are right justified and matched bit by bit from right to left. Whichever ↑formula is a shorter value is padded out with zero bits to match the size of the longer value. The size of this longer value is the size of the resulting ↑bit:formula. In the table below, ←p and ←q represent matched bits, each from a ↑formula to which ↑logical:operators are applied. For all values of ←p and ←q, the corresponding values are shown which result from application of the operators. (←NOT is included in the table, but it only applies to ←p and is not called a ↑logical:operator in this manual.) 1a14a3

.4 We should take particular note of the way the ↑bit:formula rules affect operations with ↑character:formulas 1a14a4

a. When two ↑character:formulas are combined using a ↑logical:operator, they each become ↑bit:formulas before the operation. If one is shorter than the other it is padded on the left with zero bits before the operation. 1a14a4a

b. When comparing a ↑character:formula with any formula not a ↑character:formula, they each become ↑bit:formulas before the operation, are right justified, and are compared as unsigned integers. 1a14a4b

c. When assigning a ↑character:formula to any ↑variable not a ↑character:variable, it first becomes a ↑bit:formula and is assigned as a bit string, right justified and truncated on the left or padded on the left with zeros if necessary. 1a14a4c

.5 A ↑bit:formula in ↑parentheses is also a ↑bit:formula. The ↑parentheses do not change the value of the enclosed ↑bit:formula, but they may be necessary to override the precedence of operations. Precedence is discussed in the next section. 1a14a5

4.15 Precedence of Operations 1a15

Precedence applies mainly in determining the values represented by ↑formulas. It also applies in assignment of values, however, and is treated in detail at this point even though assignment is discussed in later chapters. In general, operations are performed from left to right, except as overridden by precedence rules, grouping by means of ↑parentheses, and the need to determine a value before a ↑variable can be set (or reset). 1a15a

.1 Basic exceptions to the left to right rule: 1a15a1

a. The value of a ↑formula must be determined before that value can be assigned to a ↑variable. Therefore: 1a15a1a

(1) The ↑formula is evaluated first. 1a15a1a1

(2) Any ↑index needed to select the ↑variable is evaluated next. 1a15a1a2

(3) Any ↑pointer:formula needed to locate the ↑variable is evaluated next. 1a15a1a3

(4) The ↑variable is assigned its new value. 1a15a1a4

b. In an ↑assignment:statement all the ↑formulas to the right of the ↑assignment:operator are evaluated from left to right. Then all the ↑variables to the left of the ↑assignment:operator are set from left to right, the ↑index and ↑pointer:formula for each being determined just before it is set. 1a15a1b

c. In an ↑exchange:statement the ↑index and ↑pointer:formula on the left are evaluated, the ↑index and ↑pointer:formula on the right are evaluated, and then the values of the ↑variables are interchanged. 1a15a1c

d. If a binary operation is indicated immediately preceding a unary operation, the unary operation is completed first. 1a15a1d

e. Indexing and pointing can only be applied to ↑named:variables, not to ↑formulas and not to ↑functional:variables. The ↑index and the ↑pointer:formula applied to a ↑variable must both be evaluated before the ↑variable is evaluated. The ↑index precedes the ↑pointer:formula. First, the ↑index:components are evaluated from left to right. Then the ↑pointer:formula is evaluated. ↑Index:brackets may be thought of as being replaced by ↑parentheses and an indexing operator before the ↑left:parenthesis. If the ↑index:component ↑formulas and the ↑pointer:formula contain operations these operations will have higher precedence than indexing and pointing because of the ↑parentheses. 1a15a1e

.2 With due regard to all the above exceptions, consider the following list. The basic precedence of each operation is given in this list: 1a15a2

.3 ↑Parentheses may be considered to raise the precedence order of enclosed operations. The precedence order of every operation is effectively raised by 20 for every pair of ↑parentheses that encloses it. The operands of a ↑chain:comparison include the results of operations with precedence order greater than that of the ↑relational:operators forming the chain. A ←NOT before the leftmost operand of a ↑chain:comparison is applied to the result of the entire chain, not merely to the first ↑comparison of the chain. The chain is broken by operations of lower precedence, but not by the implied ←AND due to the chaining. An operand and a ↑relational:operator are part of an apparent ↑chain:comparison unless the meaning is changed by ↑parentheses. Consider, for example, three unsigned integers with the following values (in binary): 1a15a3

←B1 01 1a15a3a

←B2 10 1a15a3b

←B3 11 1a15a3c

.4 The following two ↑formulas then have the indicated values: 1a15a4

←B1 < B2 < B3 1 1a15a4a

←B1 < (B2 < B3) 0 1a15a4b

.5 The diagram and flow chart of Figure 4-2 illustrate left to right evaluation as modified by precedence. ↑Parentheses are not shown in the diagram, but precedence value for each operation is determined in accordance with the above list, as modified by the presence of ↑parentheses (or ↑index brackets). 1a15a5

.6 Figure 4-3 summarizes all conversions of data from one type to another possible in JOVIAL 73. Formulas or variables represented by ←XYZ, and of the five possible types as indicated at the top of the figure, are converted as indicated in the body of the figure under the influence of the operations and the types of the other operand (←ABC) as shown at the left. To determine the conversion applying to both operands of a given operation, first consider one and then the other as ←XYZ. Whenever an operand of bit type is converted to integer ("Int"), it is to unsigned integer. "Scale" in the figure means to consult the scaling rules for the details of arithmetic scaling. In some cases, a series of conversions (at least conceptually) is required. These are indicated by references to the following notes: 1a15a6

Note 1. In arithmetic operations with floating and character operands, the character string becomes a bit string, then an unsigned integer, then the integer is floated. 1a15a6a

Note 2. In arithmetic operations with floating and bit operands, the bit string becomes an unsigned integer, which is then floated. 1a15a6b

Note 3. In arithmetic operations a character string becomes a bit string, then an unsigned integer, then this integer is scaled appropriately. 1a15a6c

Note 4. In arithmetic operations a bit string becomes an unsigned integer which is then scaled appropriately, depending on the other operand. 1a15a6d

Note 5. In comparing two character strings, the shorter is padded on the right with blanks. Then both are converted to bit strings and then to unsigned integers for the comparison. 1a15a6e

Note 6. In comparing numeric with bit, character with bit, or numeric with character, the non-bit type is converted (or are converted) to bit type. Then both are converted to unsigned integer for comparison. 1a15a6f

Note 7. A character string used for pointing or indexing is converted first to a bit string and then to an unsigned integer. 1a15a6g

4.16 Short-Circuit Evaluation 1a16

A JOVIAL ↑program:declaration specifies a number of ↑statements to be executed in a particular order, subject to dynamic changes involvng ↑conditional:statements, ↑switch:statements, ↑go:to:statements, and ↑exit:statements. Within a ↑statement, there are ↑formulas to be evaluated in a particular order, subject to ↑conditional:formulas and precedence rules. All these requirements are for effect only. As long as the computational results are the same, the compiler is free to rearrange the order of computations--even to omit some calculations--in the interests of efficiency. Consider ↑formulas involving expressions such as: 1a16a

```
←0   *     ALPHA
```
1a16a1

←0 AND BETA 1a16a2

←1 OR GAMMA 1a16a3

.1 In the above examples, the zeros and the ←1 could be values determined at execution time or known at compilation time--it could make a difference with regard to efficiency. In any case, the value of the first example does not depend on the value of ←ALPHA. If the second and third examples are ↑conditional:formulas, their values o not depend on the values of ←BETA and ←GAMMA. These are cases wherein the compiler might choose to avoid evaluating ←ALPHA, ←BETA, and ←GAMMA. 1a16a4

.2 The omission and rearrangement of computations are aspects of optimization. Chapter 11 discusses optimization and the assumptions the compiler may make with regard to hidden interactions within a ↑program:declaration. The ↑order:directive (Section 11.7.4) puts the compiler on notice that it must not make certain assumptions. If the compiler can determine, from its analysis of the ↑program:declaration and making the assumptions it is allowed to make, that it would not impair the accuracy or effect of the compiled program, it may rearrange or delete ↑formulas or even whole ↑statements. 1a16a5

.3 There are programs that can analyze a JOVIAL ↑program:declaration, delete parts that cannot be executed, put the remainder in canonical form, and describe the transformation so the programmer can see some of his errors of logic. 1a16a6

4.17 ↑Form 1a17

The ↑form:declaration (Section 8.9) provides a structure for the convenient assembly of a list of values into a single bit value or character value. If the ↑abbreviation ←B follows the ↑form:name in the ↑form:declaration, each reference to the ↑form:name is a ↑bit:formula. If the ↑abbreviation is ←C, each reference to the ↑form:name is a ↑character:formula. 1a17a

.1 A ↑bit:form consists of the ↑form:name followed by a parenthesized list of ↑bit:formulas. Within the ↑parentheses there must be one ↑bit:formula for each ↑field:width in the corresponding ↑form:declaration. Each ↑formula is converted to its bit value and truncated from the left or padded with zero bits on the left to its respective ↑field:width in bits. The value of the ↑bit:form is then the concatenation of all these truncated or padded bit values. Examples: 1a17a1

←FORM DUAL B 16,16; 1a17a1a

←ABC = DUAL (ABCISSA, ORDINATE); 1a17a1b

←FORM OPWRD B 6,4,4,4,2,16; 1a17a1c

←OP = OPWRD (CODE,JMOD,AREG,BREG,0,ADDR+4; 1a17a1d

.2 A ↑character:form consists of the ↑form:name followed by a parenthesized list of ↑formulas. Within the ↑parentheses there must be one ↑formula for each ↑field:width in the corresponding ↑form:declaration. If the ↑formula is a ↑character:formula with a different number of bytes from that specified by the corresponding ↑field:width, it is truncated from the right or padded with blanks on the right to its respective ↑field:width. If the ↑formula is other than a ↑character:formula it is treated as a ↑bit:formula. The required size is then the corresponding ↑field:width times the number of bits per byte in the system. The ↑bit:formula is then truncated from the left or padded on the left with zero bits to match this required size. The value of the ↑character:form is then the concatenation of all these truncated or padded values. The ↑character:form is a ↑character:formula whether the parenthesized ↑formulas are ↑character:formulas, ↑bit:formulas, or a combination. 1a17a2

4.18 ↑Function:Call 1a18

↑Intrinsic:function:calls are discussed in Section 4.19. Other ↑function:calls are very similar to ↑procedure:call:statements, discussed in Section 5.11. The

↑procedure:name or ↑alternate:entrance:name must be one whose ↑declaration associates an ↑item:description with the ↑name. This association of an ↑item:description makes the procedure or alternate entrance a function, describes the implicit output parameter for the function, and establishes the ↑formula type and size for the ↑function:call. 1a18a

.1 The use of ↑actual:input:parameters in a ↑function:call is the same as their use in a ↑procedure:call:statement, with one exception. Ordinarily, if exit from a procedure is effected by a ↑go:to:statement referencing a ↑formula:input:parameter, the ↑actual:output:parameters at the active call are set before (or simultaneously with) the exit. In a similar situation with regard to a ↑function:call, there is nothing that can be done with the function value, so it is immaterial if the implied output parameter is "set" or not in conjunction with this abnormal exit. 1a18a1

.2 Normally, a ↑function:call is the invocation of the corresponding ↑procedure:declaration consisting of first, the setting of the ↑formal:input:parameters from the ↑actual:input:parameters (or establishing the correspondence for those ↑formal:input:parameters that are not ↑variables); second, execution of the procedure; and third, utilization of the value of the implied output parameter in place of the ↑function:call. 1a18a2

.3 If the procedure corresponding to the ↑procedure:name or the ↑alternate:entrance:name is declared to be pointed to, the ↑function:call must include the ↑pointer:formula to provide a location for the data space of the procedure during this invocation. 1a18a3

4.19 ↑Intrinsic:Function:Call 1a19

↑Format:function:call provides a set or list of values of various types and sizes to be assigned to a set or list of ↑variables. Details are given in Section 6.1.4. ↑Byte:string:function:call is a ↑character:formula. Details are given in Section 4.4.1. ↑Bit:string:function:call is a ↑bit:formula. Details are given in Section 4.12.2. ↑Shift:function:call is a ↑bit:formula. See Section 4.12.3

for details. ↑Signed:function:call is a ↑bit:formula one bit in size. See Section 4.12.5 for details. 1a19a

.1 The ↑alternate:entrance:function:call is an unsigned ↑integer:formula of default size. Its value is an unsigned integer that indicates the entrance of the named procedure that is active. The ↑formula is only meaningful within a ↑procedure:declaration. The reason for citing the ↑procedure:name is to be able to interrogate the status of an outer procedure from within an inner procedure. If the ↑procedure:name is omitted (the ↑parentheses are required even so), ←ALT provides the active entrance of the innermost ↑procedure:declaration within which the ↑function:call is issued. This makes it possible to interrogate the status of this innermost procedure even if its ↑name has been redeclared for some other local use within the ↑procedure:declaration. Associated with each possible value of the ↑alternate:entrance:function:call citing a particular ↑procedure:name, there is an intrinsic ↑status:constant. The correlation is illustrated in the table below: 1a19a1

.2 "First alternate", "second alternate", etc., smply refer to the lexical order of the ↑alternate:entrance:names, the order in which the ↑alternate:entrance:declarations are written within the ↑procedure:declaration. The ↑status within each ↑status:constant in the above list is just the relevant ↑name. There is no way to qualify these ↑status:constants explicitly and the only meaningful use of such a ↑status:constant is as follows: 1a19a2

←ALT (procedure:name ←) ↑relational:operator ←V(↑procedure:name

↑alternate:entrance:name 1a19a2a

.3 In the above example, the ↑procedure:name must be the same on both sides (even if the one on the left is only implied), or the ↑alternate:entrance:name must be one associated with the ↑procedure:name on the left (even if it is only implied). 1a19a3

.4 The ↑number:of:entries:function:call is an unsigned ↑integer:formula of default size. Its value, if the ↑index:range is omitted, is the product of multiplying together the extent of the cited table in all its dimensions. The extent of a table in any dimension is, for that dimension: 1a19a4

↑upper:bound ←+ 1 - ↑lower:bound 1a19a4a

.5 If a table is implicitly pointed to, if its ↑pointer:formula is a ↑function:call, and if its ↑allocation:increment indicates less than the entire table, then its extent in each dimension relating to the ↑function:call is ←1 and the ↑number:of:entries:function:call is the product of the extent of the allocation submanifold in all its dimensions. 1a19a5

.6 If the ↑index:range is present (see Section 10.4), the value of the ↑number:of:entries:function:call is the product of multiplying together the extents indicated by each ↑index:component:range present (not the ↑index:components). The extent indicated by an ↑index:component:range is: 1a19a6

↑high:point ↑low:point 1a19a6a

←+ 1 - 1a19a6b

↑upper:bound ↑lower:bound 1a19a6c

.7 In the above ↑formula for extent, ↑upper:bound is used only if ↑high:point is missing and ↑lower:bound is used only if ↑low:point is missing. Examples: 1a19a7

←NENT (TAB [: , , , :]) 1a19a7a

←NENT (TAB [: J, K :]) 1a19a7b

.8 The value of the ↑function:call in the first example is the extent of ←TAB in the first dimension, times its extent in the fourth dimension. The value of the ↑function:call in the second example is (J + 1 - ↑lower:bound of first dimension) times (↑upper:bound of second dimension ←+ 1 - K). 1a19a8

.9 The ↑location:function:call is an unsigned ↑integer:formula of default size. Its value is the sum of possibly three elements: 1a19a9

a. The value of the ↑pointer:formula or ↑pointer:variable pointing to the structure (procedure instruction space, table, data block) containing the named entity. If the structure is not pointed to, this is simply the compiler-assigned location of the structure. 1a19a9a

b. The relative position of the named entity in its structure--item in entry, table in data block, ↑statement in procedure, etc. 1a19a9b

c. Relative positioning due to the ↑index if present. In a table allocated space by submanifolds, the value of the ↑index can, of course, influence the value of the primary pointer. 1a19a9c

.10 In any citation of a table or item in a pointed-to structure, the pointer, whether implicit or explicit, points to the beginning of the structure. This is not generally the value of the ↑location:function:call. The ↑location:function:call is not the inverse of pointing. In general, ←XX[YY] @ LOC(XX[YY]) is a different ↑variable from ←XX[YY]. 1a19a10

.11 The ↑absolute:function:call is a ↑numeric:formula of the same size, precision, and type as its ↑parameter, except that if the ↑parameter is not floating the function is unsigned. The value of the function is the absolute value of its ↑parameter. 1a19a11

.12 The ↑words:per:entry:function:call is a signed

↑integer:constant of the size required to represent the ↑constant value. For a serial or parallel table it is the number of words in an entry of the cited table. For a tight table, it is the negative of the number of entries in a word of the cited table. 1a19a12

.13 The ↑exrad:function:call is a signed ↑integer:formula with a system-dependent size. Its size is related to the size of exrads provided by the system for floating values, but not necessarily the same. If the system uses a radix other than 2, the required relationship between the ↑exrad:function:call and the ↑significand:function:call necessitates a few extra bits. If the ↑actual:input:parameter of the ↑exrad:function:call is floating, it yields the exrad of that floating value. If its ↑parameter is not floating, the ↑exrad:function:call returns as its value the size of its ↑parameter (not including the sign) minus the number of bits after the point. Remember that the number of bits after the point can be negative--so the ↑exrad:function:call can return a value greater than the size of its ↑parameter. 1a19a13

.14 The ↑significand:function:call is a ↑fixed:formula; unsigned if the ↑parameter is unsigned, signed otherwise. If the ↑parameter is floating, the size and precision of the ↑formula is system dependent and its value is the significand of the floating ↑parameter. If the ↑parameter is not floating, the size and precision of the ↑formula are both the size of the ↑parameter, and its value is the value represented by the string of bits constituting the ↑parameter with the binary point just to the left of the leftmost magnitude bit. If ←NF is any ↑numeric:formula, then: 1a19a14

```
←NF = SIG (NF) * 2 ** XRAD (NF)
```
1a19a14a

.15 The ↑signum:function:call is a signed ↑integer:formula one bit (besides the sign bit) in size. The value of the ↑signum:function:call is zero if its ↑parameter is zero, ←+1 if its ↑parameter is greater than zero, and ←-1 if its ↑parameter is less than zero. 1a19a15

.16 The ↑size:function:call is an unsigned

↑integer:formula of default size. If its ↑parameter is a ↑character:formula, the value of the function is the number of bytes in the ↑formula. If the ↑parameter is floating, the value of the function is the number of bits in the significand plus the number of bits in the exrad, exclusive of both signs. This is not the numbers declared for these parts, but the system-dependent sizes provided to accommodate the declared sizes. If the ↑parameter is a ↑bit:formula, ↑integer:formula, or ↑fixed:formula, the value of the ↑size:function:call is the number of bits in the ↑parameter, not including the sign if there is one. If the ↑parameter is a ↑data:block:name, the value of the ↑size:function:call is the number of words in the cited data block. 1a19a16

.17 The ↑type:function:call is an unsigned ↑integer:formula three bits in size. Its values are related to the type of its ↑parameter in accordance with the table below. There is also an intrinsic ↑status:list associated with the ↑type:function:call having the ↑status:constants also listed in the following table: 1a19a17

.18 The last column above indicates a ↑qualified:status:constant that can be used where the unqualified ↑status:constant is not permitted (everywhere not in a ↑comparison with the ↑type:function:call). 1a19a18

.19 The ↑fraction:part:function:call is a ↑numeric:formula of the same size and type as its ↑parameter. Its value is the fractional part of its ↑parameter, of the same sign as its ↑parameter and with a value greater than ←-1 and less than ←1. If ←NF is any ↑numeric:formula, then ←ABS (FRAC(NF)) = FRAC(ABS(NF)). 1a19a19

.20 The ↑integer:part:function:call is a ↑numeric:formula of the same size and type as its ↑parameter. Its value is the integer part of its ↑parameter. If ←NF is any ↑numeric:formula, then ←XF = INT(NF)+FRAC(NF). 1a19a20

.21 The ↑instruction:size:function:call is an unsigned ↑integer:formula of default size. Its value is the number of words in the load module for the cited procedure. This may be required for dynamic procedure loading (see Section 8.6.11). 1a19a21

.22 The ↑data:size:function:call is an unsigned ↑integer:formula of default size. Its value is the number of words in the private or pointed-to data space of the cited procedure, if the ↑procedure:heading contains a ↑data:allocation:specifier or an ↑environmental:specifier making the unnamed data space (and any named data space not individually excepted) strictly private. This information is needed for requesting data space for a pointed-to procedure (see Sections 8.6.6 and 8.6.9). 1a19a22

4.20 Use and Qualification of ↑Status:Constants 1a20

Each ↑status:constant is given a constant integer value by means of its position in a ↑status:list (see Section 7.17). Wherever the ↑status:constant is subsequently used (except in another ↑status:list) it represents that constant integer value. The meaning of a ↑status:constant may be ambiguous, however, because it can appear in more than one ↑status:list--and be defined by each such appearance. The ambiguity is resolved by context. A ↑status:constant may be used, and represents its constant integer value, only in the contexts described in Sections 4.20.1 through 4.20.3. 1a20a

.1 A ↑status:constant may be used to represent its value as the presetting ↑constant of or in the ↑constant:list of an ↑item:declaration (or ↑ordinary:table:heading or ↑specified:table:heading) containing an ↑item:description that contains or cites the ↑status:list in which the ↑status:constant is given its value. Examples: 1a20a1

```
←ITEM  WEATHER U 2 V(RAINY),V(FAIR),V(SUNNY)=V(SUNNY);
```
1a20a1a

```
←STATUS ALPHABET V(A),V(B),V(C),...V(Y),V(Z);
```
1a20a1b

```
←TABLE  ....
```
1a20a1c

```
←ITEM  LETTER S 5 ALPHABET [0,4]=2(V(A),V(B)),V(Z);
```
1a20a1d

.2 A ↑status:constant may be used as the entire ↑numeric:formula providing the value to be assigned to an ↑integer:variable by means of a ↑simple:assignment:statement if the ↑item:description for that ↑integer:variable contains or cites the ↑status:list in which the ↑status:constant is given its value. A ↑status:constant may be used as the entire ↑actual:input:parameter corresponding to a ↑formal:input:parameter whose ↑item:description contains or cites the ↑status:list in which the ↑status:constant is given its value. A ↑status:constant may be used in the following context: 1a20a2

↑integer:variable 1a20a2a

↑status:constant ↑relational:operator 1a20a2b

↑function:call 1a20a2c

.3 In the above context, the ↑item:description associated with the ↑variable or the implied output parameter of the ↑function:call must contain or cite the ↑status:list in which the ↑status:constant is given its value. 1a20a3

.4 In other contexts (and even in the contexts described above) a ↑qualified:status:constant may be used. A ↑qualified:status:constant may be considered to consist of two parts--the ↑name preceding the ↑status, and the ↑status:constant that remains when that ↑name and its following ↑colon are deleted. The meaning of the ↑qualified:status:constant is the same as the meaning of its corresponding ↑status:constant derived from the ↑status:list (contained in or cited in the ↑item:description) associated with its corresponding ↑name. Example: 1a20a4

←STATUS USDA V(PRIME),V(CHOICE),V(GOOD),V(COMMERCIAL; 1a20a4a

←ITEM SWIFT U 5 USDA; 1a20a4b

```
←STEW = ONION + CARROT + V(SWIFT:CHOICE);
```
1a20a4c

JOVIAL Manual--Chapter 4

(J30194) 8-MAR-74 05:36; Title: Author(s): Duane L. Stone/DLS;
Distribution: /RJC; Sub-Collections: RADC; Clerk: DLS;
Origin: <STONE>C4.NLS;1, 8-MAR-74 05:31 DLS ;

Contains font markers and structure

Chapter 1 1

INTRODUCTION 1a

1.1 Purpose of the Manual 1a1

The purpose of this manual is to describe the 1973 version of the JOVIAL Computer Programming Language, and to establish standard language specifications upon which the acquisition of compilers for the language can be based. The JOVIAL 73 (abbreviated J73) language is to be considered a replacement for the previous standard, JOVIAL (J3), defined by AIR FORCE MANUAL AFM 100-24, dated 1967 June 15, with amendments thereto. 1a1a

1.2 Scope and Changes 1a2

This manual contains the complete set of JOVIAL (J73) language features. The scope of these language features is designed to provide both effective support of today's processing requirements and evolutionary growth as future system requirements dictate. Implementation of the full J73 language is not intended at this time. A basic set of 3 language features is being identified for standard implementation by all compiler systems. Methods of extending the basic set of language features has not yet been determined. Existing J3 programs may not be completely converted to J73 language because of machine dependencies and resultant changes in language features. Conversion requirements and aids should be considered in conjunction with compiler acquisition for each replacement system. Using activities are requested to submit recommended changes, additions, and deletions to the manual in sufficient detail to permit both a technical and economic evaluation. AFR 300-10 prescribes both policy and procedures for using standard computer programming languages (i.e., COBOL< FORTRAN< JOVIAL) and for specifying computer programming language compilers. 1a2a

1.3 Overview and Objectives of the Language 1a3

JOVIAL 73 has developed out of nineteen years of study and experience with regard to appropriate programming languages for command and control applications. JOVIAL has also been found to be well suited to the programming of many other applications including general scientific and engineering problems involving numeric computation and logically complex problems involving symbolic data. Because of its wide applicability and the optional control it provides over the details of storage allocation. JOVIAL is especially suitable for problems requiring an optimum balance between data storage and program execution time. The earliest versions of JOVIAL borrowed heavily from ALGOL 58. This latest version incorporates features permitting the design and utilization of the most sophisticated data structures, yet at the same time simplifies the manipulation of elementary forms--the sort of manipulation that typically involves over 95% of computation time (Knuth, D.E., "Software, Practice and Experience", Vol. 1, pp. 105-133, 1971, John Wiley & Sons, Ltd.). 1a3a

.1 The prime motivation for the development of JOVIAL is the desire to have a common, powerful, easily understandable, and mechanically translatable programming language, suitable for wide-range applications. Such a language must be relatively machine independent, with a power of expression in logical operations and symbol manipulation as well as numerical computation. A JOVIAL ↑program:declaration describes a particular solution to a data processing problem, meant to be incorporated by translation into a machine language program. The two main elements of this description are: 1a3a1

a. A set of ↑data:declarations, describing the data to be processed. 1a3a1a

b. A set of ↑statements, describing the algorithms or processing rules. These two sets of descriptions are, to a great extent, mutually independent, so that changes in one do not necessarily entail changes in the other. Further, the pertinent characteristics of an element of data need be declared only once and do not have to be repetitiously included with each reference to the data. 1a3a1b

.2 One of the further requisites of a programming language intended for large-scale data processing systems is that it include the capability of designating and manipulating system data, as contained in a communication pool (compool). A compool serves as a central source of data description, communication changes in data design by supplying the compiler (or assembler) with the current data description parameters, thus allowing automatic modification of references to changed data in the machine language program. Though highly desireable for any data processing system, a compool is a vital necessity for large-scale systems where problems of data design coordination between programmers are apt to be otherwise unsolvable. 1a3a2

.3 JOVIAL is a readable and concise programming language, using self-explanatory English words and the familiar notations of algebra and logic. In addition, JOVIAL has no format restrictions and, with the ability to intermix ↑comments among the ↑symbols of a program and to define notational additions to the language, the only limit to expressiveness is the ingenuity of the programmer. A JOVIAL program may thus serve largely as its own documentation, facilitating easy maintenance and revision by programmers other than the original author. 1a3a3

.4 The convenient subordination of detail without loss of detail afforded by JOVIAL also contributes to readability and expedites the task of uniting programs. One simple JOVIAL ↑statement can result in the generation of scores of machine instructions which might normally take hours to code in a machine-oriented language. This reduction in source program size proportionally reduces the opportunity for purely typographical errors which are much more obvious when they do occur, due to JOVIAL's readability. Since many coding errors based on the idiosyncrasies of computer operations are eliminated, experience has shown that JOVIAL programs may be written and tested, even by neophyte programmers, in less time than previously required with machine-oriented programming languages. 1a3a4

.5 Computer users are often faced with the necessity of producing large numbers of computer programs in short periods of time. A readable language such as JOVIAL alleviates the heavy burden this places on the existing programming staff, by permitting an augmentation with relatively inexperienced programmers. 1a3a5

.6 JOVIAL simplifies and expedites the related problems of training personnel in the design of data processing systems and the development of computer programs for such systems. Although JOVIAL was designed primarily as a tool for professional programmers, its readability makes it easy for nonprogrammers to learn and use. It also helps to broaden the base of JOVIAL users beyond those engaged in actual programming. 1a3a6

.7 The objectives of standardizing JOVIAL are as follows: 1a3a7

a. To attain a greater degree of inter-system compatibility. 1a3a7a

b. To provide a clear guidance to the computer manufacturing community in the production of computer-based systems. 1a3a7b

c. To use existing programs and ease the transition when upgrading to new computers. 1a3a7c

d. To improve the producti&ity of programmers. 1a3a7d

e. To establish a base for language improvement. 1a3a7e

f. To establish a training requirement on which to base a comprehensive skill resource development program. 1a3a7f

1.4 The Descriptive Metalanguage for JOVIAL 1a4

One purpose of this manual is to specify a language. The purpose of the language is to specify algorithmic processes for the solution of computational problems. We must carefully distinguish between the elements of the JOVIAL language and other objects, including the objects a JOVIAL ↑program:declaration discusses. ←A, ←B, ←C, ←B+C, and ←A=B+C are five structures in the JOVIAL language. There are, however, an infinite number of structures in the JOVIAL language. In order to speak about them all we need to classify them. We give names to the classes of JOVIAL structures and we distinguish them from all other objects by writing them in italics. The classification scheme and the names of classes used in this manual are arbitrary. JOVIAL 73 can be validly described using other classification schemata and/or class names. 1a4a

.1 Every class of structures in the JOVIAL language that we discuss in this document is named by a word in italics or by a phrase in italics with colons (in italics) between the words of the phrase. We do not distinguish between a class and a general element of the class. We use plurals in italics when we mean several elements of the class. Italics are used for no other purpose except also to number the syntax equations in Appendix A. Thus, ↑letter is a class (having 26 members) of elements of JOVIAL. A ↑letter is also a member of that class. ↑Name is a class (having infinitely many members) of elements of JOVIAL. A ↑name is also a member of that class. We use the phrase "metalinguistic term" to mean one of these italicized words or phrases. Every metalinguistic term (except ↑system:dependent:character) is defined in terms of other metalinguistic terms and the 59 elements of the JOVIAL alphabet. By substitution, every metalinguistic term is ultimately defined in terms of the 59 elements of the JOVIAL alphabet (and ↑system:dependent:character). 1a4a1

.2 The definition of a metalinguistic term is called a "syntax equation" or a "metalinguistic equation". Several notational devices are needed in constructing syntax equations. The syntax equations occur throughout the document and are all gathered together in Appendix A in alphabetical order. In fact, Appendix A may be considered the syntactic specification of JOVIAL 73. In Appendix A, each heavily black-bordered box (except one) contains the definition of a single metalinguistic term. Each syntax equation is preceded, in its box, with a sequential number in italics, followed by a colon, followed by a list of the numbers of the syntax equations in which this metalinguistic term is part of the definition. 1a4a2

.3 Following the metalinguistic term being defined is the definitional operator: 1a4a3

::= 1a4a4

Following the definitional operator is the definition, consisting of elements of the JOVIAL alphabet (the ↑signs of JOVIAL), metalinguistic terms, and metalinguistic symbols indicating choice, repetition, and continuation. Many definitions contain optional elements or mandatory choices. Braces ordinarily denote a choice. One line must be selected from among the lines within the braces in order to satisfy the definition. If there is only one line within the braces, it must be chosen--the braces then only indicate the extent of application of a repetition operator. 1a4a5

Brackets denote an option or an option and a choice. The line within the brackets may be included or omitted. If there is more than one line within brackets, zero or one of the lines within may be used to satisfy the definition. ↑Brackets are elements of the JOVIAL alphabet, all of the same size. Brackets are distinguished from ↑brackets by being considerably larger (and of various sizes). Arrows are used to indicate continuation of a line. If a line is too long for the page (or the space available within braces or brackets) an arrow is plaed at the right of the first part of the line and is repeated at the left of the continuation line. In one or two places vertical arrows are used for similar purposes where a column (a stack of lines within braces) is too long for the page. There are two repetition symbols. means that the preceding element of the definition may be repeated an arbitrary number of times. means also that the preceding element may be repeated, but that ↑commas must be inserted between occurrences of the repeated element. If the repetition symbol follows a metalinguistic term, it is that one metalinguistic term that may be repeated. If the repetition symbol follows a right bracket or a right brace, it is the entire structure within the brackets or braces that may be repeated. A bracketed structure followed by a repetition symbol means "use this structure zero or more times, choosing any one of the lines herein, independently, for each occurrence." A braced structure followed by a repetition symbol means the same except that "zero or more times" becomes "one or more times." 1a4a6

.4 There is no terminator symbol for a syntactic equation. One ends where another begins or where there is nothing left in the box. In a few of the boxes there are some anomalies. Syntactic equation 144 defines ↑mark. Opposite each ↑mark is a metalinguistic term. This association serves to define each of these metalinguistic terms, as the ↑mark to its left. Opposite ↑space is only space. That's the definition of ↑space, the ↑mark indicated by not marking the paper. Syntactic equation 172 defines ↑pattern:digit. It also gives tabular information involved with the significance of ↑pattern:digits. Syntactic equation 190 defines ↑relational:operator and gives a phrase for each ↑relational:operator indicating its meaning. Box 234 defines ↑system:dependent:character by means of a prose discussion. Syntactic equations 247 and 248 are in one box. Each is a definition of ↑variable in terms of different collections of covering sets. And equations 94 and 95, for ↑format:list, are in one box. 1a4a7

.5 Leading and trailing spaces in the definition of a metalinguistic term are of no significance. Spaces between the ↑symbols of a definition may or may not be significant; the body of this manual clarifies the issues. Certainly, if there is no space between elements of the definition, then no ↑space is permitted in the corresponding positions in a ↑program:declaration. For example, ←BEGIN must not be rendered as ←B ←E ←G ←I ←N or as ←BE ←GIN. 1a4a8

.6 The syntax equations are not completely correct. There are actually limitations on the seeming generality of the syntax equations. The limitations that must be observed to maintain syntactic integrity are stated in the text. In addition, the text tells what the programmer can do with the syntax and explains the meanings of all JOVIAL constructs. 1a4a9

1.5 JOVIAL ↑Characters, Examples 1a5

Anything in a syntax equation that is not in italics is composed of JOVIAL ↑signs, the actual alphabet used to write a ↑program:declaration. These ↑signs (and ↑system:dependent:characters) are used also in examples illustrating what may be written in substitution for a metalinguistic term. Examples and metalinguistic terms are never hyphenated for the sake of composing the type in this document. A metalinguistic term never continues from one line to the next in a syntax equation. In text, however, a multiword metalinguistic term may start on one line and continue on the next. In this situation, the italicized colon at the end of one line is repeated at the beginning of the next line. ↑Colon happens to be one of the JOVIAL ↑signs. The JOVIAL ↑colon is not in italics and is always separated by at least one space from any italicized word. The metalinguistic colon is closely pressed on both sides by words in italics. 1a5a

.1 Metalinguistic terms (the words and phrases in italics) represent structures that can be understood and translated by a JOVIAL compiler, or at least they represent elements of such structures. A ↑program:declaration can be understood by a compiler and translated into computer instructions. ↑Simple:statements and ↑table:declarations are elements of ↑program:declarations. The translated version of a ↑program:declaration and the structures it manipulates, however, are an entirely different class of objects. The collection of computer instructions is known as a "program." The word is not in italics because the thing it represents does not exist in JOVIAL. JOVIAL can contain ↑program:declarations; it cannot contain programs. In a similar manner, a ↑table:declaration, upon being processed by a compiler, gives rise to a structure, known as a "table", to be manipulated by a program. 1a5a1

.2 ↑Program:declaration and ↑table:declaration are distinguished from program and table both by the use of different type fonts and the use of the word "↑declaration." With many terms, the distinction is only made by means of type fonts because the use of extra words would make the explanations awkward. For example, a ↑variable is part of a ↑program:declaration, whereas a variable is a value that can be set, used and changed by a program at different times. 1a5a2

1.6 Notational Symbols, System-Dependent Values 1a6

In various parts of this manual, various numeric values that may change from time to time or that are system dependent are represented by letters or character combinations after the manner of algebraic notation. The meanings of these notational symbols are given where they are used. They have no pervasive meaning and are to be considered valid only in the local context where they are used. 1a6a

.1 Knowledge of many of the system-dependent values is vital to a sufficient understanding of the environment to enable the programmer to construct valid and useful ↑program:declarations. Such information is not available at this writing and is not appropriate to this manual. This information must be made available in other documentation. 1a6a1

1.7 One-Dimensional Nature of a Program 1a7

Regardless of the forms used for coding, the input medium, or the arrangement of the coding on that medium, the language definition considers a JOVIAL ↑program:declaration to be a continuous stream of JOVIAL ↑signs. 1a7a

1.8 Syntax and Semantics--Illegal, Undefined, Ungrammatical 1a8

This manual gives complete specifications for writing legitimate JOVIAL ↑program:declarations, except for the necessary system-dependent values and compiler capacities, explains in detail how the particular compiler deviates from these specifications, and lists and explains all error messages that the compiler may generate. 1a8a

.1 For a ↑program:declaration to be legitimate, it must be meaningfully structured in accordance with the specifications in this manual. If the ↑program:declaration or any part of it fails to meet this requirements, it is of small concern whether it is called illegal, undefined, or ungrammatical. 1a8a1

.2 It often happens that compilers do not reject certain illegal or undefined structures, but compile them instead, giving results that the programmer considers appropriate. It is recommended that programmers avoid exploiting these quirks, since there is no guarantee that a new version of the compiler will exhibit the same eccentricities. Using such discovered idiosyncrasies leads to extra work in reprogramming when transferring the work to another computer or when an updated compiler replaces the old one. 1a8a2

.3 As part of the structure of a JOVIAL ↑program:declaration, nothing is permitted by unstated implication. If it is not prescribed by this manual (or other documentation in the case of system-dependent features), it is not legitimate JOVIAL code. In the matter of exceptions to prescribed forms, nothing is prohibited by innuendo. All exceptions are explicitly stated. 1a8a3

.4 The document is to be taken as a unit. All sections, all figures, the list of syntax equations, and the index-glossary are interrelated. 1a8a4

(J30195) 8-MAR-74 05:43; Title: Author(s): Duane L. Stone/DLS;
Distribution: /RJC; Sub-Collections: RADC; Obsoletes Document(s):
30164; Clerk: DLS;
Origin: <STONE>C1.NLS;1, 8-MAR-74 04:52 DLS ;

Contains font markers and structure

Chapter 2 1

ELEMENTS 1a

2.1 Introduction 1a1

A ↑program:declaration written in JOVIAL consists, basically, of ↑statements and ↑declarations. The ↑statements specify the computations to be performed with arbitrarily named data. ↑Simple:statements can be grouped together into ↑compound:statements in order to help in specifying the order of computations. Among the ↑declarations are ↑data:declarations and ↑processing:declarations. The ↑data:declarations name and describe the data on which the program is to operate, including inputs, intermediate results, and final results. The ↑processing:declarations generally contain ↑statements and other ↑declarations. They specify computations, but they differ from ↑statements in that the computations must be performed only when the particular ↑processing:declaration is specifically invoked by ↑name. In addition to ↑statements and ↑declarations, there are ↑directives which serve various purposes. They designate externally defined ↑names the compiler is expected to recognize, they control selective compilation of various ↑statements and ↑declarations, and they provide information the compiler needs in order to optimize the object code. The ↑statements, ↑declarations, and ↑directives are composed of ↑symbols, which are the words of the JOVIAL language. These ↑symbols are, in turn, composed of the ↑signs that constitute the JOVIAL alphabet. 1a1a

.1 The general order in which the elements of a ↑program:declaration are introduced in the preceding paragraph represents the general order in which one looks up definitions when trying to clear up a question. The definitions in this manual are introduced, however, in the opposite order. Such arrangements lead to complaints that one must "read the book backwards." This comment arises from the process of looking up a form in the table of contents, turning then to the late chapter where it is defined in terms of earlier defined forms. These, more elementary, forms are then found, via the table of contents, in an earlier chapter. And so forth. Nevertheless, the document is arranged for the use of a reader rather than for reference. Difficult as this may be for reference use, the opposite arrangement is much more difficult for a reader. 1a1a1

.2 An index-glossary is included which facilitates reference. The index-glossary answers many questions directly. In other cases, it references syntax equations and sections by number. 1a1a2

2.2 Spaces and ↑Spaces 1a2

It is important to distinguish between a ↑space, an element of JOVIAL, and a space, an element of our descriptive language. JOVIAL is written using ↑symbols, the words of the language. The ↑symbols are composed of ↑signs, the elements of the JOVIAL alphabet. In general, ↑symbols do not contain ↑spaces. The exceptions are pointed out in Section 2.5.2, with respect to ↑comment, and in Section 2.8.2, with respect to ↑character:constraints. In general, ↑symbols are separated by ↑spaces. Again the exceptions are noted in Section 2.10; however, these exceptions are permissive; i.e., it is always correct to put ↑spaces between ↑symbols. 1a2a

.1 The following example is wrong: 1a2a1

←PLXMPY (1. 375, -. 75, 5 ., 7.3 : REAL, IMAG) ; 1a2a1a

.2 The following examples are right: 1a2a2

a. ←BEGIN 1, 3, +5, - 7 END 1a2a2a

b. ←SL:PLXMPY(1.375,-.75,5.,7.3:REAL,IMAG); 1a2a2b

c. ←SL : PLXMPY (1.375 , - .75 , 5. , 7.3 : REAL , IMAG) ; 1a2a2c

.3 In defining and explaining ↑signs and ↑symbols, any spaces included in the metalanguage formulas are not meant to be included in the definition. The phrase "string of" implies that there are to be no ↑spaces between the elements strung together. Similarly, phrases such as "followed by", "enclosed in", and "separated by", imply that there are to be no ↑spaces between the elements concerned. This is the situation (except where explicitly stated to be different) in this chapter, Chapter 2. In Chapter 3 and beyond, the opposite view is maintained with respect to these phrases. 1a2a3

2.3 ↑Signs, Elements of the JOVIAL Alphabet 1a3

(equ) 1a3a

.1 ↑Sign means a ↑letter, a ↑numeral or a ↑mark. ↑Letter means one of the 26 letters of the English alphabet, written in the form of a roman capital. ↑Numeral means one of the ten arabic numerals: ←0,←1,←2,←3,←4,←5,←6,←7,←8 or ←9. (The slash through the zero is only for the purpose of distinguishing it from the ↑letter O in definitions and examples of JOVIAL.) ↑Sign, ↑letter, and ↑numeral are defined more formally by means of the syntax equations in the boxes at the head of this section. ↑Mark is most easily defined by the formal means of the syntax equation in the box above. The box above also contains a metalinguistic term associated with each ↑mark; this serves to define these terms. 1a3a1

2.4 ↑Symbols, The Words of JOVIAL 1a4

(equ) 1a4a

.1 The ↑symbols or words of the JOVIAL language are composed of strings of ↑signs, in some cases a single ↑sign. Most ↑symbols do not contain ↑spaces. In fact, ↑spaces serve to separate ↑symbols from one another. 1a4a1

2.5 ↑PRIMITIVE, ↑Ideogram, ↑Directive:Key, ↑Comment 1a5

(equ) 1a5a

.1 ↑Primitives may be considered the key words of the JOVIAL language. They are generally used to give the primary meaning of a ↑statement or ↑declaration, although some are used for second purposes. ↑Ideograms are generally used as ↑arithmetic:operators, as ↑relational:operators, and for purposes such as grouping, separating, and terminating. ↑Directive:keys are used to state the primary meanings of ↑directives. ↑Comments can be used to annotate a ↑program:declaration; explaining to readers (and often the original programmer) what is going on. 1a5a1

.2 Notice that a ↑comment is delimited by ↑quotation:marks. Therefore, ↑spaces are permitted within a ↑comment, but a ↑quotation:mark is not permitted within a ↑comment. Also, a ↑semicolon is not permitted within a ↑comment. The reason for this is to permit some recovery in case a delimiting ↑quotation:mark is left off a ↑comment. If the ↑comment were not then terminated by the next ↑semicolon, the entire remainder of the ↑program:declaration would be turned inside out; the ↑comments being interchanged with the ↑statements and

↑declarations. Even with this rule, failure to terminate a ↑comment can lead to disaster. If an ←END is swallowed up, the entire program structure can be disarrayed. 1a5a2

.3 The ↑system:dependent:characters that can be included in ↑comments (and other structures) are simply those ↑characters, other than JOVIAL ↑signs, that the particular system and compiler can read and write. 1a5a3

.4 Notice that ↑primitives, ↑ideograms, and ↑directive:keys do not contain ↑spaces. ↑Spaces are significant in a ↑program:declaration; usually in that they separate ↑symbols. ↑Comments, on the other hand, may contain ↑spaces. This permits easier reading and writing of the commentary. The ↑quotation:marks delimiting the ↑comment provide the necessary grouping so that the ↑spaces do not cause trouble. 1a5a4

2.6 ↑Abbreviation, ↑Letter:Control:Variable, ↑Name 1a6

(equ) 1a6a

.1 ↑Abbreviations are specific ↑letters having specific meanings in specific contexts, usually ↑data:declarations. The specific uses are documented later on without, usually, calling the ↑letter an ↑abbreviation. 1a6a1

.2 The ↑letter:control:variable is a special ↑variable having meaning only within a ↑loop:statement and passing out of existence when the ↑loop:statement is not being executed. It is explained more fully in connection with explanation of the ↑loop:statement. 1a6a2

.3 Regardless of the syntax in the box above, a ↑name must not be the same as any ↑primitive. Notice that a ↑name must include at least two ↑signs. The use of the ↑dollar:sign is system dependent. That is, it provides a means whereby a ↑name can be designated to have some special meaning in relation to the system in which the compiler is embedded. Such special meanings are outside the scope of this manual, however, and ↑names containing ↑dollar:signs are considered the same as other ↑names herein. ↑Names do not contain ↑spaces. An embedded ↑space would change a ↑name into two ↑names or other ↑symbols. 1a6a3

2.7 ↑Number, ↑Constant, ↑Status 1a7

(equ) 1a7a

.1 The above definitions are obviously not complete, in that several kinds of ↑constants mentioned in the box are not yet defined. This discussion is mainly concerned with the use of ↑spaces together with ↑numbers, ↑constants, and ↑statuses as ↑symbols. 1a7a1

.2 A ↑number is a string of ↑numerals, without ↑spaces. In some places, a ↑number can stand alone as a ↑constant. In other places, particularly ↑data:declarations, it stands alone as a ↑symbol but is not considered a ↑constant. In yet other places, a ↑number is part of another ↑symbol. A case in point is the ↑character:constant, defined above. The optional ↑count in a ↑character:constant is a ↑number. (In several places, ↑numbers or other constructs are given new names reminiscent of their uses in those places.) 1a7a2

.3 A ↑character:constant is a ↑symbol. If it begins with a ↑count, there must be no ↑spaces between the ↑count and the first ↑prime. Between the ↑primes, the string of ↑characters may include ↑spaces, but these ↑spaces are significant. They represent part of the value represented by the ↑character:constant. (There are restrictions on the ↑characters permitted in a ↑character:constant, discussed in Section 2.8.2). In a ↑status:constant and a ↑qualified:status:constant, the ↑left:parenthesis, the ↑name, the ↑colon, the ↑status, and the ↑right:parenthesis are all ↑symbols. ↑Spaces are permitted between these elements, but not within the ↑name or the ↑status. ↑Space is not pemitted between ←V and the ↑left:parenthesis. All other ↑constants are ↑symbols, not containing ↑spaces. 1a7a3

2.8 ↑Constants and Values 1a8

(equ) 1a8a

.1 ↑Character:constants are the means of representing character values to be manipulated by a program. (↑Character:variables and ↑character:formulas are indirect means.) The ↑characters acceptable as character values are whatever the system will accept from among those given in the body of Figure 2-1. At least the 59 JOVIAL ↑signs must be accepted. Comparison of Figure 2-1 with Section 2 of USAS X3.4-1968, "USA Standard Code for Information Interchange", shows the graphic characters in identical positions in the two tables. Figure 2-1 includes eight additional columns presently under consideration by standardization bodies. The positions of the ↑characters in the table are the only correspondence. This manual does not require that internal representation be in accordance with USAS X3.4-1968. If, however, JOVIAL ↑program:declarations generate messages for transmission to other systems or process messages received from other systems, these messages are required by other directives to conform to USAS X3.4-1968 in their external representation. 1a8a1

.2 All of the character values indicated in the body of Figure 2-1 can be represented in ↑character:constants (except for system-dependent limitations). Artifices are required, however, to represent some of the values. Any ↑spaces within the delimiting ↑primes, except within a three-↑character code, represent characters of value "space". ↑Primes, ↑semicolons, and ↑dollar:signs have special meanings. Therefore, in order to represent a single occurrence of one of these ↑signs, two of them are used in succession. If a succession of these ↑signs is desired as part of the value represented by a ↑character:constant, the entire string is doubled. In summary: 1a8a2

←2n ↑primes are used to represent ←n ↑primes. 1a8a2a

←2n ↑semicolons are used to represent ←n ↑semicolons. 1a8a2b

<2n ↑dollar:signs are used to represent ←n ↑dollar:signs. 1a8a2c

.3 The reason for doubling the ↑primes inside a ↑character:constant is that single ↑prime terminates the ↑constant. The reason for doubling ↑semicolons inside a ↑character:constant is the same. Although it is illegal, a single ↑semicolon terminates a ↑character:constant; and for the same reason it terminates a ↑comment, to avoid turning the whole ↑program:declaration inside out if the correct terminator is omitted. The reason for doubling ↑dollar:signs is that a single ↑dollar:sign introduces the codes described in the next two paragraphs. 1a8a3

.4 Any ↑character represented in the body of Figure 2-1, if it is acceptable at all by the system as a character value, may be represented by a three ↑character code beginning with a ↑dollar:sign. The second ↑character is a column code from the figure; i.e., any ↑numeral or one of the ↑letters from ←A through ←F. The third ↑character is any ↑character from the body of the figure that can be recognized by the compiler. The character specified by such a code is the one at the intersection of the column designated by the column code and the row in which the third ↑character is found. For example, the percent mark can be represented by any of several three ↑character codes, including these two: 1a8a4

←$25 1a8a4a

←$2U 1a8a4b

.5 Within a ↑character:constant, there is a recognition mode for ↑letters. Initially, the mode is "general", in which all ↑characters, including uppercase and lowercase ↑letters, and the three-↑character codes are recognized as described above. The mode can be changed to "lowercase", however, by including the two-↑character mode code consisting of ↑dollar:sign followed by uppercase or lowercase ←L. All ↑letters following such a mode code in a ↑character:constant, regardless of the case used, are considered to be in lowercase. The two-↑character mode consisting of ↑dollar:sign followed by uppercase or lowercase ←U sets the "uppercase" mode, in which all ↑letters are considered uppercase. The three-↑character codes pevail, without changing the mode, regardless of the mode. Hence, the appropriate case can be specified for one ↑letter in a stream of ↑letters. For example, here are four ↑character:constants with the value "De Gaulle": 1a8a5

←'De Gaulle' 1a8a5a

←'D$6E G$6A$7U$6L$6L$6E' 1a8a5b

←'D$LE $4GAULLE' 1a8a5c

←'udle$u g$laulle' (none of these are ones) 1a8a5d

.6 If the ↑count is present in a ↑character:constant, there must be no ↑spaces between the ↑count and the first ↑prime, and the ↑count gives the number of concatenated repetitions of the character values represented within the ↑primes. Examples: 1a8a6

←2'TOM' is equivalent to ←'TOMTOM' 1a8a6a

←10'*' is equivalent to ←'**********' 1a8a6b

←3' ' is equivalent to ←'   ' 1a8a6c

.7 Notice that it is indeed the values that are repeated, not the ↑characters making up the ↑constant before evaluation. Thus, ←2'T$LOM' is equivalent to ←'TomTom'; it is not equivalent to ←'Tomtom'. 1a8a7

.8 The system may impose a limit on the number of characters in strings representable by ↑character:constants, ↑character:variables, or ↑character:formulas. The size of a ↑character:constant is the number of characters represented in the value; not the number of ↑characters between the ↑primes. 1a8a8

.9 ↑Pattern:constants directly represent values consisting of strings of bits. (Various ↑variables and ↑formulas also represent bit values.) The ↑numeral to the left of the ←B in the ↑pattern:constant is the "order" of the ↑constant and controls the possible ↑pattern:digits and affects their meanings. These relationships are displayed in the box above wherein ↑pattern:digit is defined. The right column contains the possible orders. The ↑pattern:digits are displayed in the center in braces. The permissible ↑pattern:digits are only those on the line with or above the selected order. For example, if the pattern is of order ←4, only ←F and the 15 ↑pattern:digits above ←F are permitted as part of this particular ↑pattern:constant. The meaning of each ↑pattern:digit is given in the column on the left, but these are also affected by the order. If the order is ←n, then the ←n rightmost bits of each pattern represent the meanings of the corresponding ↑pattern:digits. The optional ↑count gives the number of concatenated repetitions of the ↑pattern:digits enclosed in ↑primes. No ↑spaces are permitted anywhere within this structure. 1a8a9

.10 The meaning of a ↑pattern:constant is the string of bits resulting from the concatenation of the strings of bits (as modified by the order) represented by each ↑pattern:digit. The size of the ↑pattern:constant is the number of bits in the string and may be obtained by multiplying the order times the ↑count (assumed to be ←1 if not specified) times the number of ↑characters inside the ↑primes. In the following examples, a ↑pattern:constant on the left is shown with the bit string it represents on the right: 1a8a10

←4B'7CF03'   01111100111100000011   1a8a10a

←3B'3120' 011001010000 1a8a10b

←1B6'10' 101010101010 1a8a10c

←5B2'R' 1101111011 1a8a10d

.11 ↑Numeric:constants represent numeric values. (There are also ↑numeric:variables and ↑numeric:formulas.) ↑Numeric:constants, as well as ↑numeric:variables and ↑numeric:formulas, are described in terms of their three possible modes of representation; as integer values, fixed values, and floating values. The compiler may represent constants in modes other than those indicated by the ↑program:declaration; as long as the overall effect of the ↑program:declaration is not compromised. (This principle applies in general; i.e., the compiler can do things differently as long as the result is the same.) Suppose, for example, an ↑integer:constant is used in a context that requires it to be converted to a floating value. It is far more efficient for that conversion to be done once, at compile time, instead of each time the code executed 1a8a11

.12 An integer value is a numeric value represented as a whole number without a fractional part, but treated as if it had a fractional part with value zero to infinite precision. In this manual, precision means the number of bits to the right of the point in binary representations of numeric values. A ↑number used as an ↑integer:constant represents an unsigned integer value. The size of an ↑integer:constant is the number of bits needed to represent the value; from the leading one bit to the units position, inclusive (value zero has size 1). No ↑spaces are permitted in an ↑integer:constant. The system may impose a limit on sizes of integer values. 1a8a12

.13 Floating values ←v are represented within the computer by three parts, the significand ←s, the radix ←r, and the exrad ←e, having the following relationships (with regard to the absolute value): 1a8a13

←v = s x r 1a8a13a

←s = 0 or ←m s m x r 1a8a13b

.14 The radix ←r and the minimum value ←m are fixed in any system. Therefore, only the significand and the exrad are saved as representations of a floating value. For a negative value (not a ↑constant), a minus sign is also saved with the significand. Regardless of the system values of ←r and ←m, we assume that ←r = 2 and ←m is one-half. The language permits inquiry into the values of significands and exrads based on radix and minimum of these values. Therefore, with respect to value, internal representation of floating values exhibits (so far as the programmer can see from results) the relationships: 1a8a14

←v = s x 2 1a8a14a

←s = 0 or ←1/2 s 1 1a8a14b

.15 ↑Floating:constants are written with the assumption that, externally, ←r = 10, and there is no ←m. Thus, the value of a ↑floating:constant is given as: 1a8a15

←v = s x 10 1a8a15a

.16 A ↑floating:constant must not contain any ↑spaces. In the syntactic equation for a ↑floating:constant, the ↑number (or ↑numbers) and the ↑decimal:point (if present) give the value of the external significand. The ↑scale (with or without its ↑plus:sign or ↑minus:sign) following ←E gives an exrad (exponent of the radix) to be used as a power of ten multiplier. If the exrad is zero, it and the ←E can be omitted. To be a ↑floating:constant, the ↑symbol must contain a ↑decimal:point, or a ↑scale as exrad, or both. It must not contain an ←A; that would make it a ↑fixed:constant. 1a8a16

.17 A ↑floating:constant can contain information relating to the precision of its internal representation. The ↑scale following ←M gives the minimum number of magnitude bits in the significand of the internal representation. In most systems, there are one or two or, at most, a very few modes of representation of floating values. If the ↑scale following ←M is greater than the maximum number of magnitude bits in any of the system-dependent modes of representing floating values, the ↑floating:constant is in error. Otherwise, the compiler chooses the mode with the smallest number of magnitude bits in the significand at least as large as the ↑scale following ←M. If there is a choice of exrad size also, the compiler chooses one that can encompass the value of the ↑floating:constant. These sizes are based on the numbers of bits in the actual representations, not on what may be a fictional assumption that the radix is 2. If the ←M and its following ↑scale are omitted, the compiler chooses its normal mode of floating representation or one that can contain the value. 1a8a17

.18 A fixed value is an approximate numeric value. Within the computer, it is represented as a string of bits with an assummed binary point within or to the left or right of the string. The number of bits in the string, not counting a sign bit if there is one, is the size of the fixed value. The number of bits after the point (positive or negative, larger or smaller than the size) is the precision of the fixed value. 1a8a18

.19 A ↑fixed:constant is seen, in the syntactic equation above, to be an ↑integer:constant or a ↑floating:constant (without an ←M and its ↑scale) followed by the ↑letter ←A and a ↑scale. The ←A and its ↑scale are essential to make the form a ↑fixed:constant. ↑Spaces are not allowed anywhere within a ↑fixed:constant. All that precedes the ←A determines the value of the ↑fixed:constant. All that precedes the ←A determines the value of the ↑fixed:constant (which may then be truncated on the right). The ↑scale after the ←A tells how many bits there are after the point. (If the ↑scale is negative, the bits don't even come as far to the right as the point). The size of the ↑constant is the number of bits from the leftmost one-bit to the number after the point as specified by the ↑scale after ←A, inclusive. Here are some ↑fixed:constants, their values, their sizes, and their precisions: 1a8a19

.20 There must be no ↑spaces within a ↑fixed:constant. The system may impose a size limitation on fixed values. 1a8a20

.21 ↑Integer:constants, ↑floating:constants, and ↑fixed:constants cannot have embedded ↑spaces and cannot have negative values. Both of these characteristics are changed for ↑status:constants and ↑qualified:status:constants. In ↑status:constants and ↑qualified:status:constants, there must be no ↑spaces within the ↑status, within the qualifying ↑name, or between the ←V and the ↑left:parenthesis. There may be ↑spaces elsewhere within such ↑constants. 1a8a21

.22 ↑Status:constants and qualified:status:constants represent constant integer values. How they become associated with these values and how they may be used are explained elsewhere. In distinction to ↑integer:constants, which can only stand for zero and positive integer values, ↑status:constants and qualified:status:constants can also stand for unvarying negative integer values. 1a8a22

2.9 Computer Representation of ↑Constants and ↑Variables 1a9

JOVIAL is designed to be compatible with binary computers, machines in which numeric and other values are represented as strings of binary digits, ones and zeros. The bits (binary digits) of a computer are organized in a hierarchical structure. A compiler may impose a different structure on the computer, but for reasons of efficiency it usually adopts a structure identical to or at least compatible with the structure of the machine. The structure discussed in this section is the system structure; i.e., the structure presented to the programmer by the combination of a particular computer and a particular JOVIAL compiler that produces object code for that computer. 1a9a

.1 JOVIAL ↑program:declarations are not completely independent of the system. The extent of dependence, however, is related to the use of certain language features. Dependence is increased by the use of features, such as ↑pattern:constants and ←BIT, that relate to bit representation or those, such as ←LOC, that relate to system structure. The value of a ↑pattern:constant is completely independent of the system, but its use implies knowledge of the representation of other data. It is that knowledge, built into te ↑program:declaration, that is system dependent. 1a9a1

.2 Even if such deliberate system dependence is avoided, the programmer must still have knowledge of structure and representation in his system so that he may know the limitations on precision, how his tables must be structured, and how to avoid gross inefficiencies. For example, in processing long strings of character data, it is often much faster to examine and manipulate them in word-size, instead of byte-size, hunks. 1a9a2

.3 A "byte" is a group of bits often used to represent one character of data. The number of bits in a byte is system dependent. Although JOVIAL permits some leeway in positioning bytes, there are usually preferred positions. When referring to these preferred positions, we often use the term "byte boundary". 1a9a3

.4 A "word" is a system-dependent grouping of bits convenient for describing data allocation. Entries and tables are allocated in terms of words. Data are overlaid in terms of words. The maximum sizes of numeric values may, but need not, be related to words. Word boundaries usually correspond to some of the byte boundaries. 1a9a4

.5 The "basic addressable unit" is the group of bits corresponding to each machine location. In many machines, the basic addressable unit is the word. In others, it is the byte. If it is the word, each value of the location counter refers to a unique word. If the basic addressable unit is the byte, each location value refers to a unique byte. In these latter circumstances, it often happens that adresses are somewhat restricted. For instance, it may be permitted to refer to a string of characters starting in any byte, or to double-precision floating values starting only in bytes with locations divisible by 8. 1a9a5

.6 Integer and fixed values are represented in binary as strings of bits. The number of bits used to represent the magnitude of a value is known as its size and is (in most cases) under the control of the programmer. The position of the binary point is understood and takes up no space. For signed values, the sign bit is an additional bit not counted in the size of the value. For purposes of the use of ←BIT, the sign bit is considered to lie just to be left of the most significant bit accounted for by the size of the value. The maximum permissible size of an integer or fixed value is system dependent. The maximum size of a signed integer or fixed value is one less than this system-dependent size and the places where unsigned values of maximum size may be used are restricted; i.e., they must not be used in conjunction with any ↑arithmetic:operators, nor with the four nonsymmetric ↑relational:operators ←<, ←>, ←<=, ←>=, and when used with the symmetric ↑relational:operators (←= and ←<>) the other operand must not be signed. 1a9a6

.7 The compiler determines the sizes of ↑constants. The programmer usually supplies the sizes of ↑variables. The size does not include the sign bit for signed data. For unpacked or medium packed data, there may be more bits in the space allocated for an item than are specified by the programmer. Whether or how these extra bits are used is system dependent, but in any case they are known as "filler bits". The sign bit, if there is one, and any filler bits are to the left of the magnitude bits. It depends on the system whether the sign bit is to the left or right of the filler bits. 1a9a7

.8 The meanings of bit values ←0 and ←1 are not stipulated, but in most implementations ←0 stands for ←0 and ←1 for ←1 in positive values. For negative values, there is considerable variation. All the following are known and acceptable representations of ←-12 in an unpacked, signed, integer item declared to be four bits long: 1a9a8

```
←11111111111111111111111111111111111111111111111110011     1a9a8a

←                1000000000000000000000000000000000001100     1a9a8b

←                                                 10100     1a9a8c
```

.9 Floating values are represented by two numbers, both signed. The significand contains the significant digits of the value and the exrad is the exponent of the understood radix. Each system has a standard mode of representing floating values, known as "single precision", with a specified number of bits in the significand and a specified number in the exrad. Many systems have one or a few additional modes in which there are more bits in the significand, the exrad, or both. If there is more than one mode, the programmer can usually choose the mode for each floating value. In the absence of an indication of such choice, the compiler will usually choose single precision. The radix is an implicit constant having a system-dependent value. 1a9a9

.10 Character values are represented by strings of bytes, each byte consisting of a string of bits. The number of bits in a byte is system dependent. The number of bytes used to represent a character value is under control of the programmer, but there is a system-dependent maximum. 1a9a10

.11 A character item that fits in one word is always stored in one word, by the compiler. By use of a ↑specified:table:declaration, the programmer may override this rule. If it is not densely packed, a character item always starts at a byte boundary. If it crosses a word boundary, a character item always starts at a byte boundary. The programmer must not attempt to override this rule. 1a9a11

.12 An entry variable whose relevent ↑table:declaration does not describe it as being of some other type is a bit variable. It is merely the string of bits, of a size corresponding to the number of words in an entry, representing the entry. 1a9a12

2.10 ↑Spaces, ↑Comments 1a10

The syntactic structures of all ↑symbols have now been explained, as well as the places where ↑spaces are permitted or prohibited within them. All further structures that go to make up a ↑program:declaration are composed of strings of ↑symbols. It is always permitted to place one or more ↑spaces between ↑symbols. It is sometimes required to put at least one ↑space between ↑symbols. The criterion is to avoid ambiguity. Comments can often replace required ↑spaces. 1a10a

.1 ↑Spaces are required in many situations to enable the compiler to detect the end of one ↑symbol and the beginning of the next. Generally, at least one space is required between two ↑symbols of any class except ↑ideograms, but including the ↑quotation:mark. The rule is exhibited in detail in the following table. The rows are labelled with the ending ↑signs of the left ↑symbol of a pair of ↑symbols. The columns are labelled with the beginning ↑signs of the right ↑symbol of a pair. "SR" at the intersection of row and column indicates that at least one ↑space is required between the pair of ↑symbols: 1a10a1

.2 A ↑comment may occur between ↑symbols. However, it must not occur within a ↑definition nor within any ↑constant, such as a ↑status:constant or a ↑character:constant. A ↑comment may be used instead of the required ↑space between ↑symbols unless use of the ↑comment would cause the occurrence of two ↑quotation:marks in succession. In fact, only the use of a ↑comment can bring about the situation indicated by the lower right corner of the table above. Introduction of a ↑comment between ↑symbols where a ↑space is permitted but not required may then require a ↑space to prevent the ↑comment from interfering with another ↑symbol. 1a10a2

.3 A ↑comment must not be used where the next structure required or permitted by the syntax is a ↑definition. That is, a ↑comment must not follow the ↑define:name or a ↑right:parenthesis in a ↑define:declaration. And a ↑comment must not follow a ↑left:parenthesis or a ↑comma in a ↑definition:invocation. A ↑comment, as defined above, must not occur in a ↑definition delimited by ↑quotation:marks. 1a10a3

JOVIAL Manual--Chapter 2

(J30196) 8-MAR-74 05:52; Title: Author(s): Duane L. Stone/DLS;
Distribution: /RJC; Sub-Collections: RADC; Obsoletes Document(s):
30166; Clerk: DLS;
Origin: <STONE>C2.NLS;1, 8-MAR-74 05:03 DLS ;

Contains font markers and stucture

Chapter 3 1

↑VARIABLES 1a

3.1 Concept of ↑Variables 1a1

A JOVIAL ↑program:declaration consists of a string of ↑statements and ↑declarations that specify rules for performing computations with sets of data. The basic elements of data are items. Items are named to distinguish one from another. Sometimes, a ↑name applies to a group of items, requiring indexing to tell one member of the group from another. Several named groups may be subsumed under another group, which is known as a table and which is itself named. Tables and items may in turn be collected in another group called a data block which, again, is named. Space may be allocated these data structures either statically at compile time or dynamically at execution time. 1a1a

.1 The value of items and other data can be changed in various ways. A data element whose value can be changed by means of an ↑assignment:statement is known as a variable. Items, then, are variables. Table entries can function as variables, as can parts of items under the influence of the ↑primitives ←BIT and ←BYTE. 1a1a1

.2 A ↑variable is the designation, within a ↑program:declaration, of a variable to be manipulated within the computer. The two syntax equations for ↑variable (above) indicate, first, the type of data involved, and second, the grammatical form of the ↑variable related to the kind of data structure in which the variable exists. 1a1a2

3.2 ↑Named:Variable 1a2

A ↑named:variable is a reference to a variable by means of a ↑name associated with the variable through a ↑data:declaration. A ↑simple:variable is a reference (for the purpose of using or changing its value) to a variable declared to be a simple variable; one not declared as a constituent of a table. No ↑index is involved in a ↑simple:variable because the reference is to a variable that is one of a kind, not part of a matched set. Use of the ↑pointer:formula is explained in Section 7.8 1a2a

.1 A ↑table:variable is a reference to a variable declared to be part of a table. A table consists of a collection of entries and there is an occurrence of each table item in each entry. An ↑entry:variable is a reference to the entire entry as a single variable. An ↑indexed:variable (a ↑table:variable or ↑entry:variable) generally includes an ↑index to select the particular occurrence of the variable being referenced. 1a2a1

.2 An ↑index is correlated with a ↑dimension:list. Every ↑table:declaration contains a ↑dimension:list which prescribes the number of dimensions of the table and the extent of the table in each of these dimensions in terms of its ↑lower:bound and its ↑upper:bound. (Some of the detailed specifications can be omitted; the defaults are explained elsewhere.) Each ↑index:component must evaluate to an integer value (↑numeric:formulas are explained in Sec 5) not less than the ↑lower:bound and not greater than the ↑upper:bound in the corresponding position of the relevant ↑dimension:list. The relevant ↑dimension:list is, of course, the one in the ↑table:declaration bearing the ↑table:name beginning the ↑entry:variable or in the ↑table:declaration containing the ↑item:declaration bearing the ↑item:name starting the ↑table:variable. The rightmost ↑index:component selects the element, of the row selected by the ↑index:component second from the right, from the plane selected by the index:component third from the right, etc. 1a2a2

.3 If the ↑index is omitted from an ↑indexed:variable, whether or not the empty ↑brackets remain, the meaning is the same as if the complete ↑index were present and each ↑index:component were equal to its corresponding ↑lower:bound. In fact, a legitimate form of ↑indexed:variable is to omit one or more ↑index:components, marking their positions of necessary with ↑commas. The meaning of such a form is the same as if each missing ↑index:component were present with a value equal to its corresponding ↑lower:bound. The following example shows an ↑ordinary:table:declaration and three ↑entry:variables, all with exactly the same meaning: 1a2a3

←TABLE ALPHA [3:7, 9, 100:157, 0:50]; NULL; 1a2a3a

←ALPHA [3, 3, 100,0] 1a2a3b

←ALPHA [, 3,, 0] 1a2a3c

←ALPHA [,3] 1a2a3d

3.3 ↑Letter:Control:Variable, ↑Functional:Variable 1a3

A ↑letter:control:variable is a reference to a variable designated within a ↑loop:statement to aid in control of execution of the ↑controlled:statement and to have meaning only within the ↑loop:statement. It is explained in Section 5.8 in conjunction ↑loop:statements. 1a3a

.1 ↑Format:variable is a special form that enables a list of values to be converted to character type and assembled into a character value. The details are given in Section 6.1.7 1a3a1

.2 The above construct selects a string, of the characters denoted by the ↑named:character:variable, to be considered as the variable to be given a new value. The ↑named:character:variable can be any ↑simple:variable or ↑indexed:variable of character type. The bytes of the ↑named:character:variable are considered to be numbered, starting with zero at the left. The ↑numeric:formula following the first ↑comma is evaluated as an integer and used to select the byte of the ↑named:character:variable to be considered the leftmost byte of the ↑functional:variable. If there is no second ↑comma and no second ↑numeric:formula, the leftmost byte of the ↑functional:variable is its only byte. Otherwise, the second ↑numeric:formula is evaluated and tells how many bytes there are including the leftmost byte, in the ↑functional:variable. 1a3a2

.3 The ↑named:variable in the above metalinguistic formula can be of any type. The construct selects a string of bits, from the bits denoted by the ↑named:variable, and treats that string of bits as a bit variable. The bits of the ↑named:variable are considered to be numbered, starting with zero at the left. The ↑numeric:formula following the first ↑comma selects the bit to be considered the first bit of the derived variable. The ↑numeric:formula following the second ↑comma (if there is one) determines the number of bits in the derived string (one bit if there is no such ↑numeric:formula). In signed variables, the sign bit is bit zero and the leftmost magnitude bit is bit one. In unsigned numeric variables, the leftmost magnitude bit is bit zero. In entries, the leftmost bit of the first word is bit zero. In character variables, the number of bits per byte is system dependent. In floating variables, the sign bits of the significand and exrad are included in the bit count, but the arrangement of bits is system dependent. 1a3a3

3.4 ↑Format:Variable, ↑Bit:Variable, ↑Character:Variable 1a4

↑Format variable is explained in Section 6.1.7. 1a4a

.1 The construct using ←BIT is explained in Section 3.3.3. A ↑bit:variable denotes a string of bits without consideration of any numeric or other meaning associated with those bits. Almost all ↑named:variables carry an implication of some data type other than "bit". However, an ↑entry:variable, if the ↑table:name is not declared so as to imply some specific data type, denotes only the string of bits constituting the entry. 1a4a1

.2 The construct using ←BYTE is explained in Section 3.3.2. The ↑named:character:variable is a ↑named:variable using a ↑name declared to denote a variable (an item or an entry) of character type. 1a4a2

3.5 Numeric:Variable 1a5

Any ↑numeric:variable can be used as a ↑pointer:variable. The details of the use of ↑pointer:variables are given in Chapter 7 in conjunction with discussion of controlled allocation. ↑Letter:control:variable is explained fully in connection with ↑loop:statements. Without being explicitly declared, it becomes an ↑integer:variable through its usage. All ↑names that can be used as ↑named:variables are declared as explained in Chapter 7. Some ↑entry:variables may use ↑names not associated with any data type. All other ↑named:variables use ↑names that are associated with ↑item:descriptions. These ↑item:descriptions give the data type among other things (see Section 7.16 for details). One data type is "character" as mentioned above in Section 3.4.2. Another data type is "floating". ↑Floating:variables use ↑names declared to be of floating type. The other descriptive terms in ↑item:descriptions denote "signed" and "unsigned", but we are interested here in other attributes. Signed and unsigned data are also associated with one or two ↑numbers. The first ↑number declares the size of the datum, the number of bits in its magnitude. If this is the only ↑number in its ↑item:description, the datum is an integer value and the ↑named:variable denoting it is an ↑integer:variable. The second ↑number in the ↑item:description for a signed or unsigned value declares the precision of the value, the number of bits in its magnitude after the point. If this second ↑number is present, even if its value is zero, the datum is a fixed value and the ↑named:variable denoting it is a ↑fixed:variable. 1a5a

(J30197) 8-MAR-74 05:58; Title: Author(s): Duane L. Stone/DLS; Distribution: /RJC; Sub-Collections: RADC; Obsoletes Document(s): 30182; Clerk: DLS;
Origin: <STONE>C3.NLS;1, 8-MAR-74 05:17 DLS ;

GREETING AND TEST

HELLO DLAE. WOULD YOU LET ME KNOW IF YOU GET THIS MESSAGE? I'M TRYING TO SEE HOW THE NIC JOURMAL SYSTEM LIKES ME. MY IDENT IS DON AND MY NET ADRESS IS CANTOR AT MULTICS. COMPUTING IS MY GAME, OR MORE OR LESS. HOW ARE YOU GETTING ALONG AT CCA? I HEAR YOU ARE CHAIRPERSON OF THESTERRING COMMITEE, OR SOMETHING. MAYBE YOU ARE TO BUSY TO READ THIS NOTE, MUCH LESS ANSWER IT. OUR NLM THING IS STILL ALIVE. MICHAEL STILL WON'T SLEEP THROUGH THE NIGHT, BUT WE LOVE HIM ANYWAY. 1

NCC TIP Hardware Work

On Tuesday, March 12, the NCC TIP will be taken down from about 1800 to about 1900 (EDT) for hardware work. We hope this does not cause great inconvenience to our users. 1

NCC TIP Hardware Work

(J30199) 8-MAR-74 07:01; Title: Author(s): Alex A. McKenzie/AAM; Distribution: /BBN-NET BBN-TENEX; Sub-Collections: NIC BBN-NET BBN-TENEX; Clerk: AAM;

Project ADMIN - ROC USAF 17-73 - Administrative Management Information System

This copy is as close as I can get it. Some small liberties have been taken in format. NOTE: This is several pages long.

Project ADMIN - ROC USAF 17-73 - Administrative Management Information System

COVER LETTER 1

Required Operational Capability for Administration Management Information System (Project Admin) ROC Number: USAF 17-73 1a

Preparing Office: Systems Management and Programming Group (HQ USAF/DAX) 1a1

(Project Officer: Mr. Frank Allen, GS-13, Ext 70427) 1a2

28 December 1973 1a3

I. DEFICIENCIES/NEEDS 2

Administration management at all echelons of the Air Force is severely hampered by the outmoded and largely manual system for processing and preparing documentary communications media. Continued reliance on ad hoc, after-the-fact, corrective management has resulted in slow, inefficient, uneconomic, and all too often ineffective administration management information systems. 2a

The need for a systematic program that will provide efficient procedures and equipment for creating, reproducing, distributing, transmitting, storing, retrieving and disposing of documentation is further underscored by the amount of time spent by large numbers of Air Force personnel in the information processing and transfer functions, and by the great quantities of textual documentation involved. There are, for example, approximately 120,000 military and civil service manpower authorizations performing the administrative task of creating (typing) documents. 2b

II. REQUIRED OPERATIONAL CAPABILITY 3

An Administration Management Information System, which provides an enhanced capability for the preparation, timely transmission, and recall (cyclic or on demand) of documentary communications within the Air Force and which takes full advantage of the technological developments in automatic data processing (ADP) and communications, is required. The system must be designed so that equipments obtained and procedures developed can be phased into Air Force organizations without detrimental interruptions to the organizations' primary operational horizontal compatibility at all echelons of the Air Force as well as meeting foreseeable interface standards with other DOD and Federal agencies. 3a

W. K. Richardson, Colonel, USAF 3b

Deputy Director of Administration 3c

2 Attachments: 3d

1. ROC USAF 17-73 Sec III-VIII with attachments 3d1

2. Distribution List 3d2

III. DETERMINATION OF DEFICIENCIES/NEEDS AND THE REQUIRED OPERATIONAL CAPABILITY. 4

1. Over the past few years, the administrative workload within Air Force organizations has witnessed a dramatic growth both in magnitude and complexity. The duplicative and wasteful efforts accompanying the preparation, transmisssion, and storage of documentary communications, the untimely delays and errors in transmission, the unnecessary loss of operational personnel to support functions and the resultant reductions in mission effectiveness are no longer tolerable. It is not only desirable to initiate a program to eliminate these deficiencies, it is essential. 4a

The following illustrations exemplify the magnitude and complexity of the administrative workload and indicate the scope of the effort required to resolve the deficiencies which occur in every office, regardless of functional assignment or responsibility. 4b

a. Over 500 million pieces of correspondence and 100 million copies of messages are processed annually through administrative communications channels. An average of 30,000 pieces of mail is generated daily by Air Staff members alone at an estimated cost of $200,000 each day. 4b1

b. 700,000 cubic feet of records are being retired annually by the over 52,000 offices of record. One cubic foot represents 2000 8x10-1/2 inch pages weighing nine pounds. Thus, the Air Force is retiring 3,000 tons of paper or one billion pages each year. 4b2

c. Between 40 and 50 tons of publications and blank forms are received and shipped daily at the Publications Distribution Center. 4b3

d. Over 120,000 manpower positions (military and civil service) create written documents by some typing effort, as based on the requirement of having the typing skill as part of their position description. Attachments 1 and 2 are listings of these authorizations by Air Force Specialty code, title, typing skill requirements, and number. 4b4

e. An estimated 12,000 typewriters are purchased annually as replacement items at a cost of $5 million to provide typists with basically the same production capability that has been available for years. 4b5

f. Over one million printed pages form the Departmental portion of the administrative data file of regulations, pamphlets, and manuals. This is only a small part of the data base to which an Air Force manager must have access in order to efficiently carry out his mission. He must also have access to the Major Command, base, legal, financial, and technical publications. 4b6

2. A number of studies and analyses have been performed which relate to this ROC. 4c

a. During the Mission Analysis of Base Communications (BCMA), large potential savings were identified by providing a "fully responsive, integrated, information transfer system" to the Air Base. Details of the methodology and results are presented in Section IV and Appendix 6 of the needs Panel Report of the Base Communications Mission Analysis. The concept included all forms of communications: face-to-face conversation, closed circuit television and mail, as well as the classic telephone and electrical message systems. The basic actions which people take to cause information to flow - data entry, address, signature entry, retrieve or store, etc. - are common to all modes of information transfer in all places and at all levels and thus form a baseline from which information transfer needs can be derived. 4c1

The way in which information is transferred today is heavily influenced by the communications systems which have been made available. For example, many, if not most, of the methods and procedures employed for the creation, transfer, storage, retrieval and delivery of information from writer to reader were developed with the available information transfer systems as a governing factor. Remembering that the basic actions are similar regardless of where they occur, it becomes obvious that any improvement in the transfer of information at the basic level could foretell impprovements in the entire spectrum of information transfer functions. 4c2

The information flows identified in Appendix 6 of the Needs Panel Report were examined with a fully responsive information transfer system in view and new flows were developed which assumed the availability of such a system. These flows, showing the minimal time and actions required, are presented in Appendix 12 of the Report. A comparison of the current and minimal flows revealed startling differences in the number of actions and the time required to take those actions. Savings ranged from 24 to 96 percent in the flows that were examined. 4c3

b. A study by AFSC/ESD determined that in a mixed (manual/automatic) typing center, when the typing load is as low as 20 per cent of the total effort and four drafts are normally required before final typing, efficiencies and savings could accrue to the organization by replacing manual stations with automatic typing stations instead of adding additional manual stations. With a group of two or more typists and typing workload as low as 20 percent of the total effort, the time and money saved in set-up and retyping corrected drafts in an automatic typing mode --magnetic tape cartridge, magnetic card or on-line computerized text editing -- was less than the cost of hiring additional typists. 4c4

Although, not all correspondence requires four drafts prior to final draft, the nuumber four is believed an acceptable average for paperwork going outside of the originating organization. Additional benefits which accrue are the time saved in preparing identical correspondence to multiple addressees, the ease of producing a final clean copy and the ease of correcting mistakes. 4c5

c. The AFSC/ESD East Coast Study Facility has used the IBM Magnetic Tape Selectric Typewriter, the IBM Magnetic Card Selectric Typewriter in conjunction with the Bowne Time Share "Word/One" text editor, resident on an IBM 360 computer, and the Redactron Tape Cartridge system. These systems were used for the high volume reports required by the Weather 85 and Base Communications Mission Analysees (BCMA). (The report of the Needs Panel of the BCMA alone was in excess of 1200 pages. Experience on these systems has shown a marked savings over the time and personnel required in an equivalent manual environment to produce such reports. 4c6

d. The AFSC/ESD Directorate of Information Systems Technology has also used an on-line text editor for production of high volume, high priority reports and correspondence -- e.g. Engineering Plans, Program Management Plans, Procurement Packages, Command letters, multiple address correspondence, personnel and manning statistics, and technical reports. Although only four terminals are in use, the six typists are able to share the terminals through phasing of the workload and control of priorities. 4c7

e. The Department of Defense study "Mini-Cats", Miniaturization of Supply Catalogs, was conducted in July 1971. The report emphasized economies in printing and use of supply catalogs in microform rather than paper books. Conversion to microfiche started in January 1973. The impact to the Air Force is: 4c8

(1) Annual Printing expense: 4c8a

Before $300,000 4c8a1

After $180,000 4c8a2

Annual Mail Costs: 4c8b

Before $ 96,000 4c8b1

After $ 5,000 4c8b2

(3) Impact on User: 4c8c

Before 4c8c1

Shelf Space 10 feet 4c8c1a

Book size 50,000 pages 4c8c1b

Weight 200 pounds 4c8c1c

After 4c8c2

Shelf Space 2 inches 4c8c2a

Book size 200 microfiche 4c8c2b

Weight 2 pounds 4c8c2c

The same savings, plus increased user efficiency, is available to the Air Force publication area with a total administrative system that generates the data via automatic typing stations, transmits the data electronically for publishing review and editing, and then transmits the data electronically to an electronic microfiche composer for generation of microfiche for reproduction and distribution. 4c8c3

f. An Air Force Indicia Policy Study Group report, completed for the Director of Administration in July 1972, analyzed the impact of the new United States Postal Service on the annual Air Force budget requirements for mail. The mail costs are rising from the FY 1972 12 million dollar annual cost to a figure in excess of 36 million -- an increase at a minimum rate of 200%. A comparison of paper (225 pages to the pound) to microfiche (270 pages to 1/7 ounce) dictates a change. A further comparison to completely electronic distribution (zero mail cost) underscores the thrust of Project Admin. 4c9

IV. SOLUTIONS 5

1. There are many mechanical aids and techniques which can be phased into the Air Force inventory to alleviate major portions of the cited administrative problems. The solution envisioned comprises a mix of automatic typewriters/remote terminals for data generation, rapid digital or micro-image transmission for distribution, and digital, micro-image, and video for storage and retrieval. This solution can best be satisfied by: 5a

Giving the clerk-typist the capability of preparing various types of documentary communications with the minimum of human effort. 5a1

Providing a transmission system which can distribute these documentary communications from the originator to the user with the minimum of human intervention. 5a2

Furnishing the user with a storage and retrieval capability which can recall pertinent documents on demand with the minimum of technical knowledge of sophisticated or complicated procedures. 5a3

2. A progression toward the desired capability is proposed as follows: 5b

A detailed analysis and evaluation of Administration functions must be performed to establish the functional requirements baseline. 5b1

An engineering development plan should be prepared for a prototype Administration Management Information System, describing system objectives, the prototype system, cost factors, resource requirements, schedules, program management, and other necessary events and milestones leading to prototype implementation, test and evaluation. 5b2

Following prototype test and evaluation a program management plan should be prepared detailing the desired approach based on cost, potential benefit and technical feasibility. 5b3

V. CLASS V MODIFICATIONS Not applicable 6

VI. QUANTITIES INVOLVED 7

One prototype system is envisioned at this time. A combination of off-the-shelf equipments to provide flexibility in handling processing tasks, provide for the accomplishment of the functions described in III above, and provide for future expansion is required. Only broad estimates may be given on quantities of equipment involved until the prototype test and evaluation is completed. 7a

VII. HARMONIZATION 8

Harmonization with other agencies/systems will be determined concurrent with prototype implementation. However, coordination will be effected between MAJCOM Headquarters, USAF, and participating lower echelons in order to exploit standardization opportunities in the areas of hardware, software, procedures, skills, man/machine interface and training. 8a

The National Archives and Records Service (NARS) of the General Services Administration (GSA) indicates that to its knowledge no other federal agency is contemplating or has undertaken a project of this scope. The successful design of the Air Force system would furnish the guidelines for expansion among other DOD and federal agencies. The information transfer procedures and equipment would, of necessity, have to be designed or acquired with the capability to interface with other existing and proposed systems. 8b

VIII. SPECIAL COMMENTS 9

The Directorate of Administration has a Systems Management and Programming Group which has the responsibility under AFR 4-1 for improvements in administration management and the basic background in the needs of the Administration function. It does not have the research facilities or the technical knowledge to cover the entire spectrum of tasks required to design, develop and procure a system. However, the Group can be used in support of system implementation. 9a

The Mission Analysis of Base Communications developed analysis tools which can be directly applied to this task. Additionally, concepts developed by that group presumed automated tools having the capabilities described herein would be required in the 1985 time frame. The concepts in Sections IV, VI and the BCMA have provided for that requirement. 9b

Study of the Support of Air Force Automatic examining the computer capability to support the total Air Force wide needs, including the administrative needs. (sic) 9c

Initial Operational Capability for a pilot system using currently available technology can be achieved within 18 months. A pilot operation would be useful to measure actual savings and develop new methods and techniques of accomplishing Air Force administrative functions. Selection of location(s) for the pilot system will be dependent upon the initial effort prescribed in Section IV. 9d

Although potential savings -- which can ultimately result in manpower savings -- are known from past experiences and recent studies, most notably the BCMA, the definitive manpower savings can only be determined through a detailed analysis of a prototype system. Therefore, the type, quantity, and the placement of the terminal equipments have not been determined. The most noticeable savings will be in finished product production time; transmission, decision and response time; and printing, editing, publication and distribution time. 9e

Project ADMIN - ROC USAF 17-73 - Administrative Management Information System

(J30200) 8-MAR-74 11:19; Title: Author(s): Edmund J. Kennedy/EJK; Distribution: /RADC; Sub-Collections: RADC; Clerk: EJK;

USING idents

What happened to the NETBAGRIPES and NETCCMMENTS idents??? 1

USING idents

(J30201) 8-MAR-74 17:07; Title: Author(s): David H. Crocker/DHC; Distribution: /JAKE BUGS MDK; Sub-Collections: NIC BUGS; Clerk: DHC;

Tenex RJS to CCN
cc: FIELDS at BBN, BURCHFIEL at BBN, HEARN at UTAH-10, BOYNTON - -
- -
(I'm not sure who this letter is specifically intended for. All of you may find it relevant/interesting).

A major piece of Network software -- the RRJwTenex RJS to CCN program -- turns out to be unsupported. It is not currentlypossible to get bugs fixed in either the Harslem/Fagan Bliss version or the Hicks Fail version. 1

We can't even locate the source to the Bliss copy 2

Dave.
------- 3

Tenex RJS to CCN
cc: FIELDS at BBN, BURCHFIEL at BBN, HEARN at UTAH-10, BOYNTON - -
- -
(I'm not sure who this letter is specifically intended for. All of you may find it relevant/interesting).

(J30202) 8-MAR-74 17:30; Title: Author(s): David H. Crocker/DHC; Distribution: /DHC; Sub-Collections: NIC; Clerk: DH;

Documentation
cc: lou, rossiter;bin(1200) at UCLA-CCN
- - - -
Lynn -- I want to cerify what documents are currently in the mill and what you should do as you complete them.

(there is no priority implied in this list. I'm putting them down as i think of them): 1

Three or four NUTS notes on Tenex. 1a

NUTS Notes on 1) document printing, 2) NUSEXDOC, 3) CCNJOB, 4) RJS (NMCRJS) 1b

LTSET 1c

Spencer's write-ups 1d

Tables of contents for MA and NMC Notebooks. 1e

Table of contents for NUTS Notebook will need updating 1f

Various Notes to secretaries (your copies are marked, so you can tell from them). 1g

I would like you to take the attitude that they are completely responsibility (in terms of the care you take in proffing them) and then leave me messages (thru sndmsg to dcrocker at isi) when you feel a document is ready. Leave a 'clear text' draft in DOC.LAR;# (where # differentiates the different documents). 'Clear text'', you will recall, refers to running the document through Output Device Printer and Sendprint, as per the Document Printing document. 2

I'll be checking in Wed though Friday and then the following friday (and maybe Monday). 3

Rots o' ruck. 4

P.s., Lou -- I just remembered that Monday the 25th is a holiday. I'll see you the 26th, then) 5

Dave. 6

DHC 9-MAR-74 12:47 30204

Documentation

cc: Lou, rossiter;bin(1200) at UCLA-CCN

- - - -

Lynn -- I want to cerify what documents are currently in the mill and what you should do as you complete them.

(J30204) 9-MAR-74 12:47; Title: Author(s): David H. Crocker/DHC; Distribution: /LYNN; Sub-Collections: NIC; Clerk: DH;