PRE-MEETING DISTRIBUTION

106th Meeting, X3J3
Fort Lauderdale, Florida
November 9 - 13, 1987

Accredited Standards Committee
X3, INFORMATION PROCESSING SYSTEMS*

Jerrold L. Wagener
Amoco Production Research
P. O. Box 3385
Tulsa, OK 74102
(918) 660-3978

NOTICE: X3J3 Meeting Number 106

TIME: November 9-13, 1987, 8:15am to 6:00pm each day
Agenda on reverse side

PLACE: Royce Resort Hotel
4060 Galt Ocean Drive
Ft. Lauderdale, FL 33308
(305) 565-6611 or 1-800-23-ROYCE

HOST: David Phillimore
Gould Inc.
6901 W. Sunrise Blvd.
P.O. Box 9148
Ft. Lauderdale, FL 33310
(305) 587-2900

HOTEL: same as meeting place

REGISTRATION FEE: $70.00, charged to all attendees

Monday, November 9, 1987

8:15 Opening Business (J. Adams)
Subgroup Reports (including Fortran 77 audit of S8)
X3 Ballot Comment Responses (J. Adams)
Candidate Extensions Document (J. Adams)
X3H2 Request (M. Freeman, J. Wagener)
S9 Analysis (J. Matheny)

1:15 Subgroup Meetings
4:15 Subgroup Heads Planning and Coordination Meeting

Tuesday, November 10, 1987

8:15 Subgroup Reports
Liverpool WG5 Response Document (J. Wagener)
Responses to WG5 Resolutions R12, R17, R21 (R. Hendrickson)
REAL(*,*) Clarification for DIN - WG5 R23 (B. Smith)
Bit LOGICAL Proposal - WG5 R22 (B. Smith, J. Wagener)

1:15 Subgroup Meetings

Wednesday, November 11, 1987

8:15 Subgroup Reports
ISO Character Standards (M. Ellis, M. Metcalf)
Character Set Proposal - WG5 R19 (J. Matheny, Japanese group)

1:15 Subgroup Meetings

Thursday, November 12, 1987

8:15 Subgroup Reports
Source Form (R. Hendrickson)
Appendix C Notes, Titles (T. Hoover)
Recursive Data Structures (Pointers) - WG5 R8 (L. Schonfelder)

1:15 Subgroup Meetings
4:15 Subgroup Heads Planning and Coordination Meeting

Friday, November 13, 1987

8:15 Subgroup Reports
X3 Ballot Comment Responses (J. Adams)
Glossary (J. Reid)
Miscellaneous I/O (J. Matheny)

1:15 Fortran 77 Interpretations (A. Johnson)
Closing Business (J. Adams)

(1)

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH

Scientific Computing Division/Advanced Methods Section

P. O. Box 3000 • Boulder, Colorado • 80307

Telephone: (303) 497-1275 • FTS: 320-1275 • Telex: 45 694

MEMO TO: X3J3

FROM: Jeanne Adams, Chair

DATE: September, 1987

SUBJECT: Third Between Meetings Letter

The results of the reconsideration ballot that closed on Sept. 4 generated one more negative vote, from GUIDE, which is the IBM User's Group on COBOL. The final tally is shown in JCA-23 as 30-5-2-1, from 31-4-2-1. The draft is being forwarded to ANSI for a four-month public review. I don't know the dates for the review as of this writing. I would expect them to be somewhere late October to late February.

Included in the pre-meeting as JCA-21 are the draft responses to the X3 negative ballots. I would like you to review these responses and be prepared to comment, make suggestions and vote these responses at the next meeting. I have notified Gwendy Phillips of X3/CBEMA that these responses will be first priority at our November meeting. These responses have conditional statements about things like bit, kanji and pointers. Until the public review is concluded, final disposition of these items cannot be made.

JCA-20 is a proposal for a "Candidate Extensions Document" that I feel is important to have at this time. Two of the X3 comments requested something of this nature, and for a number of reasons stated in the proposal, I feel we should have a development document.

JCA-16 is a list of distributors for the minutes. If you have problems with the order, or you do not have permission from your management to carry out the assigned distribution assignment, please see me. I will place a finalized list in the minutes on the page following assignments to X3J3.

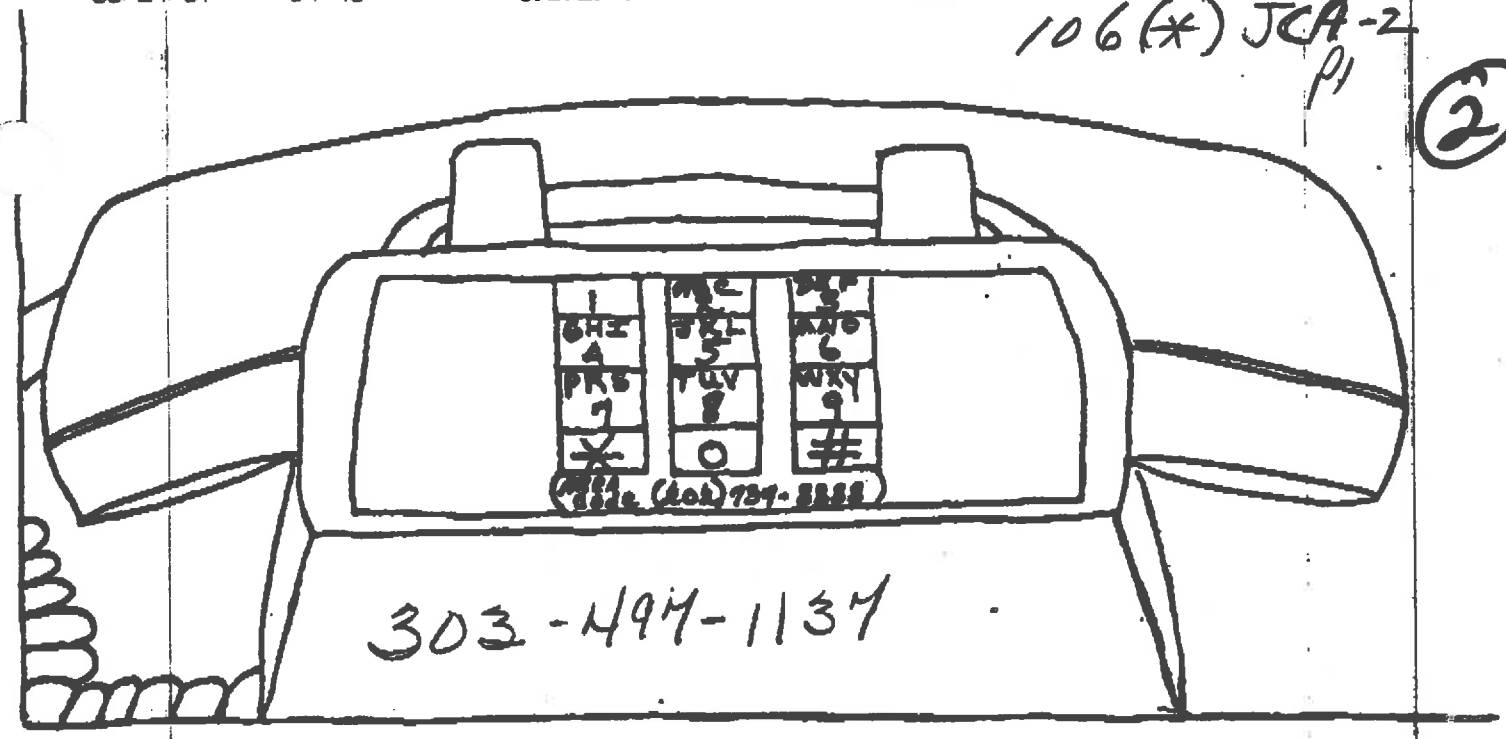
JCA-3 is a pre-release of the van Snyder comments. As indicated in his letter, he will send this again after the review has begun. Carl is initiating this as a first comment, so that our comment processing will be tested.

JCA-17 are the negative votes from X3J3 on embedded SQL. X3H2 has asked for a committee discussion on these ballots, and a statement. This item has agenda time.

I expect that the November meeting will be a very busy one.

5

106(*) JCA-2 P1 (2)



CBEMA

FROM: Gwendy Phillips

DATE: 8-24-87

SUBJECT: X3 LB 916

No. of Pages: 1 to follow

TO: Jean Adams

Please Call Upon Receipt

If Any Problems Call:

(202) 737-8888 EXT. 52

106 (*) JCA-2 p2

Accredited Standards Committee
X3, INFORMATION PROCESSING SYSTEMS*

Doc. No.: X3/87-08-091R-X3S
Date: August 24, 1987
Project:
Ref. Doc.:
Reply to:

TO: X3, SPARC, IAC, SMC
Officers, X3TC's and SPARC/SG's

Subject: PRELIMINARY FOLLOW-UP AND RECONSIDERATION of X3 Letter Ballot 916, Approval to Forvard BSR X3.9-198X, Programming Language FORTRAN, to BSR for a Four-Month Public Reviev and Comment Period; See X3/87-07-069, X3/87-07-065, X3/87-05-035

ACTION REQUESTED

The follov-up letter that was mailed out to Jeanne Adams on August 19, 1987 was incorrect. The Secretariat is nov conducting a 10-day reconsideration on X3LB 917. The Secretariat will inform Jeanne Adams of the action that is needed to be taken by the Technical Committee. We ask that you replace X3/87-08-091 with this revised transmittal and keep all attachments that were sent to you on August 19.

ACTION REQUESTED

X3 members, having voted on X3LB 916, wishing to change their votes must send a letter to us on their organization's letterhead. Be sure to reference X3 Letter Ballot 916, X3/87-07-069. If you want to change your vote to NO or ABSTAIN, please state this and explain the reasons for your position. The reconsideration period will close September 4, 1987.

Gvendy J. Phillips
Recording Secretary, X3

cc: Jeanne Adams, X3J3 Chair

8

106 (X) JCA-2
93

Accredited Standards Committee
X3, INFORMATION PROCESSING SYSTEMS*

Doc. No.: X3/87/08-091-X,S
Date: August 20, 1987
Project: 67-R
Ref. Doc.:
Reply to:

Jeanne Adams, X3J3 Chair
National Center for Atmospheric Research
SCD
P.O. Box 3000
Boulder, CO 80307

Dear Jeanne:

Subject: Follow-up to X3LB 916, Approval to Forward BSR X3.9-198x, Programming Language FORTRAN, to BSR for a Four-Month Public Review and Comment Period; See X3/87/07-069, X3/87/07-065, X3/87/05-035

This ballot closed with a preliminary tally of 31-4-2-1 (ANS). Comments which accompanied the four negatives were:

DATA GENERAL: See attached

IBM: See attached

DIGITAL: See attached

UNISYS: See attached

Comments which accompanied the two abstentions were:

OMNICOM: "No expertise in subject."

RAILINC: "Do not believe benefits are cost justified."

Two "yes" votes were accompanied with comments:

KODAK: "Per addendum and note to X3/87/07-169 of July 22, 1987."

H-P: See attached

Although there is no requirement for a 30-Day Default on an X3 Letter Ballot for Public Review, X3J3 is required to consider each negative and prepare a response prior to the X3 Letter Ballot for Final Approval. Please provide the Secretariat with documentation on this action so that there will be no delays when the time comes for the X3LB on Final Approval. This should be sent to the attention of Patti Steiner.

Gwendy J. Phillips
Recording Secretary, X3

Attachments: As stated above

9

X3 Letter Ballot

8/20
106(*) JCA-2
P. 4

Accredited Standards Committee
X3, Information Processing Systems*

Doc. No.: X3LB 916

Date: July 22, 1987
Project: 67-R
Ballot Period: 30 Days

Subject: Approval to Forward dprANS X3.9-198x, Programming Language FORTRAN, to BSR for a Four-Month Public Review and Comment Period

Return to:
X3 Secretariat/CBEMA
311 First Street, N.W. 500
Washington, D.C. 20001-2178

Ballot Closes: NOON August 20, 1987

Authorized by: X3 Procedures
Distributed by: X3 Secretariat
Ref. Document: X3/87/07-069, X3/87/05-035 X3/87/07-065

FOR ACTION

Statement: Technical Committee X3J3 has voted 29-7 to forward this document to SPARC and X3 for further processing. SPARC reviewed the dprANS for compliance at their July meeting and found that it did comply with the authorizing document.

Question: Do you approve forwarding X3.9-198x to BSR for a four-month public review and comment period?

Yes _____ *No X *Abstain _____

X3 SECRT '87 AUG -5 P4:08

See attachment

SIGNATURE W. Lee Schilke

ORGANIZATION REPRESENTED ON X3 Data General
(PLEASE PRINT)

PRINCIPAL ☒ ALTERNATE ☐ DATE 7/30/87

*Operating under the procedures of The American National Standards Institute.

NOTE: If you find that you cannot vote YES and wish to be recorded as NOT VOTING or voting NO, please state this and explain the reasons for your position in the space above or on a separate sheet.

X3 Secretariat: Computer and Business Equipment Manufacturers Association, 311 First Street, NW, Suite 500, Washington, DC 20001-2178

Tel: 202/737-8888
Fax: 202/638-4922

10

106 (*) J1A2
P. 5

Subject: X3LB 916 on approval to forward proprosed Fortran 8x standard for public review

Our response to X3LB 916 is "no" because we feel that the proprosed standard does not comply with the project proposal, resulting in a language that is not an appropriate successor to the current Fortran-77 (F77). Rather than incorporate popular vendor extensions (Hollerith, NAMELIST, INCLUDE, bit, byte, and logical data types, word and bit manipulation routines from MIL-STD 1753), the proposed standard includes many newly designed features that make Fortran 8x a very large and complex programming language. By not incorporating existing, popular vendor extensions, the proposed standard goes against the goal of promoting portability, since each vendor is likely to add its own F77 extensions to its version of Fortran 8x. The size and complexity of Fortran 8x creates several problems: 1) it will be difficult for vendors to produce efficient implementations (as experience with Ada suggests); 2) the language is less suitable for occasional use by professionals who are not computer programmers; and 3) it is likely to limit the range of computers that can effectively support the language.

We believe that a Fortran 8x standard should be based on F77, incorprate popular vendor extensions (such as listed above), avoid obsolescent and deprecated features, and remain a small and efficient language.

W Lee Schiller 7/30/87

W. Lee Schiller

Data General Corporation

11

106(*) JCA 2
p. 6

X3 Letter Ballot

Accredited Standards Committee
X3, Information Processing Systems*

Doc. No.: X3LD 916

Date: July 22, 1987
Project: 67-R
Ballot Period: 30 Days

Subject: Approval to Forward dpANS X3.9-198x, Programming Language FORTRAN, to BSR for a Four-Month Public Review and Comment Period

Return to:
X3 Secretariat/CBEMA
311 First Street, N.W. 500
Washington, D.C. 20001-2178

Ballot Closes: NOON August 20, 1987

Authorized by: X3 Procedures
Distributed by: X3 Secretariat
Ref. Document: X3/87/07 060, X3/87/05-035 X3/87/07-060

FOR ACTION

Statement: Technical Committee X3J3 has voted 29-7 to forward this document to SPARC and X3 for further processing. SPARC reviewed the dpANS for compliance at their July meeting and found that it did comply with the authorizing document.

Question: Do you approve forwarding X3.9-198x to BSR for a four-month public review and comment period?

Yes _____ *No XX _____ *Abstain _____

X3 SPARC
'87 AUG 28 P2:36

Comments attached

SIGNATURE C. L. Green

ORGANIZATION REPRESENTED ON X3 IBM CORPORATION
(PLEASE PRINT)

PRINCIPAL ☒ ALTERNATE ☐ DATE 7/29/87

*Operating under the procedures of The American National Standards Institute.

NOTE: If you find that you cannot vote YES and wish to be recorded as NOT VOTING or voting NO, please state this and explain the reasons for your position in the space above or on a separate sheet.

X3 Secretariat, Computer and Business Equipment Manufacturers Association, 311 First Street, NW, Suite 500, Washington, DC 20001-2178

Tel: 202/737-8888
Fax: 202/638-4922

12

IBM is casting a negative note on this ballot because the X3J3 committee has not met the requirements of the original project proposal (SD-3). We agree with the letter from Convex Computer Corporation (X3/87-07-025) which addresses specific points in the project proposal which have not been met in the draft proposed revision. These points are important to Fortran users.

Two of the principal problems with Fortran-8X are:

1) it does not preserve the essential characteristics of Fortran, i.e., a relatively small language permitting efficient and inexpensive implementations, and

2) it does not protect the users' investment in existing programs.

Instead, the revised Fortran should be based on Fortran-77, adding those parts of Fortran-8x which are in current implementations. The remainder of the 8x document is essentially a new language and should be repackaged and processed as a Technical Report in order to provide some implementation experience, testing and evaluation before being considered for standardization.

Further, as an international standard, any revision to Fortran needs to address the issue of multi-byte character sets which is essential for some countries.

We believe that X3J3 should readdress these issues. While a comment period would have the benefit of soliciting other views, it is also likely to result in significant delay before a revision can be produced due to the controversial nature of the Fortran-8x proposal. A revised Fortran standard based on Fortran-77 is needed now and can quickly be adopted by following the above recommendation, i.e., separation of controversial areas into a Technical Report and rapid adoption of an enhanced Fortran standard based on current usage.

106 (*) JCA-2
p. 8

X3 Letter Ballot

Accredited Standards Committee
X3, Information Processing Systems*

Doc. No.: X3LB 916

Date: July 22, 1987
Project: 67-R
Ballot Period: 30 Days

Subject: Approval to Forward dprANS X3.9-198x, Programming Language FORTRAN, to BSR for a Four-Month Public Review and Comment Period

Return to:
X3 Secretariat/CBEMA
311 First Street, N.W. 500
Washington, D.C. 20001-2178

Ballot Closes: NOON August 20, 1987

Authorized by: X3 Procedures
Distributed by: X3 Secretariat
Ref. Document: X3/87/07-069, X3/87/05-035 X3/87/07-065

FOR ACTION

Statement: Technical Committee X3J3 has voted 29-7 to forward this document to SPARC and X3 for further processing. SPARC reviewed the dprANS for compliance at their July meeting and found that it did comply with the authorizing document.

Question: Do you approve forwarding X3.9-198x to BSR for a four-month public review and comment period?

Yes ____ *No X *Abstain ____

See Attached Comments

X3 SECRETARIAT
'87 AUG 17 P1:45

SIGNATURE

ORGANIZATION REPRESENTED ON X3: Digital Equipment Corp (PLEASE PRINT)

PRINCIPAL ☒ ALTERNATE ☐ DATE 8/12/87

*Operating under the procedures of The American National Standards Institute.

NOTE: If you find that you cannot vote YES and wish to be recorded as NOT VOTING or voting NO, please state this and explain the reasons for your position in the space above or on a separate sheet.

X3 Secretariat: Computer and Business Equipment Manufacturers Association, 311 First Street, NW, Suite 500, Washington, DC 20001-2178

Tel: 202/737-8888
Fax: 202/638-4922

NO vote to X3 Letter Ballot 916 Public Comment for dprANS X3.9-198x Fortran from
Digital Equipment Corporation

Digital believes that there is a need for a revised Fortran standard. Digital further believes that X3J3/S8.104, in its current form, does not merit concideration as a dprANS and submitting it for public comment is not in the best interest of the Information Technology community. Digital has both technical and procedural reasons for this view. Digital has several technical objections concerning the viability, efficiency, and acceptability of the proposed language. If these technical issues are resolved Digital can change its vote to yes.

Digital also has comments on the procedures that X3J3 used, and believe that procedural deficiencies may have been in part responsible for the technical deficiencies in this dprANS. Digital's principal procedural objection to this dprANS is that, although the minimal official rules have been met, consensus has not been achieved in the technical committee and that this dprANS is not ready to be a replacement for the Fortran 77 standard.

1.0 TECHNICAL CONTENT

Digital has broad concerns about the acceptability of the language proposed in the dprANS to its user community. Most of the dprANS major new facilities are of little interest to Digital users, and several important improvements have been omitted. In addition, performance degradation of existing Fortran 77 capabilities is virtually certain, leading to the situation that many users will want to retain the Fortran 77 standard and product set, rather than accepting the revision.

1.1 Array Operations

It has become apparent to Digital that technology has bypassed the most important justification for this revision. The proposed syntax for whole array and partial array operations was originally conceived as the principal means of exploiting the emerging technology of array and vector processing CPUs by Fortran programmers. However, theory and practice have clearly progressed to the point where this is no longer the case. Both large (IBM) and small (Convex) vendors have offered competitive systems that are able to effectively exploit such hardware without the need for these language revisions. While this syntax will certainly be attractive to programmers developing new applications, it is not clear anymore that added convenience alone justifies such a major change.

We believe that the effect on users of Digital's systems will be detrimental rather than constructive. In the absence of a major investment in new optimization technology, *efficiency will suffer* when these new constructs are used. The added loop overhead on its scalar VAX and PDP-11 systems will usually mean that algorithms coded with the *new syntax will be slower and use more system resources* than the existing Fortran 77 formulations.

1.2 Numerical Accuracy

In its letter ballot on forwarding this document to X3, Digital addressed its concerns with the features in the dprANS designed to promote numerical accuracy of Fortran applications. In its response, X3J3 details the rationale for these features. It explains that X3J3's goals for these features have *never included achieving both true portability and true numerical accuracy*, as Digital's ballot suggests they should. Digital's users are long accustomed to the ability to crudely achieve these goals with an existing, de facto standard, set of language extensions (the "REAL n" declaration syntax). Neither Digital, nor its users, would benefit from converting from one inadequate solution to another, especially when the development cost is so great. Thus, Digital continues to recommend that these features should be removed from the dprANS.

1.3 Language Extensibility

As discussed in Digital's X3J3 letter ballot, a large number of proposed new features are intended to provide a language extensibility mechanism to the Fortran 8x language. These include MODULE/USE, derived and paramaterized data types, user defined operators, operator overloading, user defined generic functions and many others. X3J3 states in its ballot response that the primary motivation for these features is improved notational convenience when expressing algorithms used in some physics computations.

Digital's Fortran users come from a very broad spectrum of language usage, not just the areas of physics computations for which these features are intended. These users have pressing needs for efficient support of several new intrinsic data types. These include:

1. General purpose pointers, which are needed for any algorithm that must manipulate recursive data structures such as linked lists, trees, graphs, etc.
2. A bit data type, which is needed for any algorithm that must manipulate large numbers of boolean values efficiently.
3. A varying length character type, which is needed for any algorithm that must manipulate unpredictable size text strings efficiently.
4. Alternate character types for processing non-ASCII characters, which are urgently needed for data processing problems using Kanji and other large alphabets.

The programmers who need efficient access to such algorithms constitute a large proportion of Digital's Fortran user community. Digital's development resources would be better spent satisfying needs for *new intrinsic types than by improving notational convenience*. Indeed, Digital believes that the community of physics researchers would be better served with new intrinsic types for the features it needs ("physical three vectors, relativistic four vectors, rational numbers, tensors, etc.") than they would with the proposed extensibility features.

X3J3 stated in its response that the reason Digital's suggestions have not been adopted is that proposals have not "been able to command and maintain majority support." While this may be the case, the need for standardization in this area is clearly evident, as the various vendors have been widely extending the language into these areas in the past several years. Digital Fortran products allow bit manipulation and general data structures, for example. Cray has recently introduced a pointer facility to its Fortran product, and several Japanese vendors offer alternate character types. Note that X3J3's response indicates that *X3J3 acknowledges the validity of Digital's concerns, but has simply been unable to address them.*

On this basis, Digital suggests that X3J3 should remove these extensibility features from the dprANS, and work to add support for the new intrinsic types listed above.

1.4 Deprecated Features

X3J3 also addressed Digital's concerns about the "deprecated" status of some Fortran 77 features. X3J3 stated that the choice of features assigned to this category was based on "the committee's best judgment of the expected use of these features." Digital's users need a better reason than this. Digital's *users have billions of dollars invested in Fortran 77 code that makes heavy use of these features.* Conversion costs of this magnitude cannot be justified on something so ill defined. Digital recommends that X3J3 either desist in its attempt to remove these features from the Fortran language, or provide solid justification for the costs involved.

2.0 PROCEDURAL ISSUES

Digital believes that the procedural discrepancies involved, and here detailed, may have contributed to the technical problems of the current dprANS. While these problems address the procedure and process, rather than the content of the standard, the consensus process is of extreme importance in insuring that the standard is responsive to the needs of the market.

While the final X3J3 roll call vote to forward the dprANS did meet the letter of the X3 requirements for such action, it has not achieved the consensus required for a viable ANSI standard. Digital believes that this lack of consensus could justify returning the dprANS to the TC for further processing, rather than continuing the approval process of this dprANS. In addition, Digital believes that X3J3 has deviated from X3 procedures in several areas that need to be rectified before sending this dprANS to public comment.

2.1 Consensus

There are several indications that the consensus that ANSI rules require X3 to follow has not been achieved at the X3J3 level. First, while the vote to forward the dprANS did meet X3 majority requirements, the identity of the negative ballots, together with the gravity of their objections and their continued opposition in spite of the X3J3 responses to their comments, indicates that X3J3 does not have a true consensus. Negative votes not only included many of the largest vendors of Fortran processing systems, but major corporate and governmental users, and smaller vendors as well.

Second, many of the affirmative voters provided comments that clearly stated that, while they wished the public to see X3J3's work, they did not agree that this draft was worthy of forwarding as a dprANS. Thus it would appear that these voters would vote NO on this dprANS were they to be voting to forward this document as the final standard, rather than just for public comment. Such NO votes would show that even the 2/3 majority

equired by X3 would not be achieved.

Third, many of the important issues have split the X3J3 nearly in half, and the dprANS wording has see-sawed back and forth as a result. This is evident in the Appendix F text, which describes several particularly controversial features that are not currently part of the standard. It is also evident in the records of X3J3 proceedings on subjects such as deprecated features and significant blanks. This situation is even evident in X3J3's discussion of Digital's concerns about new intrinsic types. In spite of the fact that these types have been continually and vociferously requested, X3J3 admits: "The reason that these three intrinsic facilities are not in the current draft is that no proposal has been able to command and maintain majority support."

In fact, the committee has been split from the very beginning between those that wish to make Fortran over into a different, "more modern", language, and those who believe that it is most important to protect the existing investment in Fortran, and be content with well tested, incremental improvements. The current dprANS represents an uneasy compromise that achieves neither goal. For example, the discussion of the term "deprecated" shows that some members strongly favor outright removal of these features, while others favor perpetual inclusion. The current definition states, essentially, that deprecated features "might" be removed. This is a worse situation than either of the extremes since it leaves users with no guidance about whether or not they should make large investments in code conversion projects.

In light of this lack of consensus, Digital believes that proceeding to a public comment period would be largely a wasted effort, since the members would seize on the comments favorable to their own views, and disregard views they disagreed with. The best that could be hoped for would be a strong mandate for one side or the other, but the evidence of the last 10 years of negotiations shows that this is unlikely.

2.2 Procedural Issues

In addition to the lack of consensus, there have been several procedural anomalies in X3J3 that Digital believes should be rectified before X3 votes to forward this dprANS.

1. The rules concerning how X3J3 may proceed with its modifications to the dprANS during and after the public comment period were not made clear to the membership at the time of the vote to forward at Milestone 9. Many of the membership believed that a simple majority was required to effect individual changes in the dprANS after this point. Since that time, the procedural rules have been clarified and it is now apparent that a 2/3 majority is required to make each separate change. Many of the membership voted in favor of forwarding on the basis that their future proposals could still be considered under the more liberal, simple majority, rules. In fact, one of the membership (a representative from Los Alamos Scientific Laboratory), changed his vote from positive to negative on this basis (see p. 353 of the Documents section of the 104th meeting minutes). Digital recommends that the Milestone 9 vote be retaken now that these rules have been clarified.
2. X3J3 appears to have responded to all the negative comments made in the Milestone 8 vote. The official responses did receive committee votes. However, only one of Digital's proposals was allowed actual agenda time and discussion. The others, and many comments from other members, were simply rejected out of hand. X3J3's term for this rejection was "Category 5". The only consideration they received was the wording of the official X3J3 response. Digital believes that its proposals deserve a fair hearing in X3J3, even though they may be subsequently rejected.
3. Digital believes that this proposal departs from its authorizing SD-2 in several ways. The most serious is in section 4.1, where it states that "augmentations will mainly be drawn from functionality that exists in advanced implementations of existing processors." However, a large part of the new functionality described by the dprANS has not been implemented in any existing Fortran processor. In fact, the dprANS either ignores or attempts to replace much existing practice that has become de facto standard in the industry.

3.0 SUMMARY

In summary, Digital believes that the technical content of this dprANS is inadequate to meet its user's needs, and that X3 requirements for consensus have not been met. For these reasons, Digital recommends that this dprANS be sent back to X3J3 for resolution of these problems prior to further processing as a dprANS.

106 (*) JCA-2
p. 12

COMMENTS TO ACCOMPANY UNISYS CORPORATION NEGATIVE ON X3LB 916

The proposed standard FORTRAN 8x describes new features that will increase compilation cost, program size, and object code execution time. Memory management activity brought on by the run-time dynamic definition of space requirements will force operating systems contraints that do not now exist in current FORTRAN 77 programs. The new features seem to "enhance" the language beyond recognition making a "modern" FORTRAN that will require re-education and development of new programming skills for traditional FORTRAN users.

Since the new standard is needed, procedures that seek consensus on the issues should permit the J3 committee to address current practices and produce a FORTRAN 198x standard. Desirable features may also be designed and placed in a recommendation for Extended Fortran.

S.D. Fenner
UNISYS Corporation Alternate
8/18/87

X3 Letter Ballot

JUL 27 1987
106 (*) JCA-2
p. 13

Accredited Standards Committee
Information Processing Systems*

Doc. No.: X3LB 916

Date: July 22, 1987
Project: 67-R
Ballot Period: 30 Days

Subject: Approval to Forward dprANS X3.9-198x, Programming Language FORTRAN, to BSR for a Four-Month Public Review and Comment Period

Return to:
X3 Secretariat/CBEMA
311 First Street, N.W. 500
Washington, D.C. 20001-2178

Ballot Closes: NOON August 20, 1987

Authorized by: X3 Procedures
Distributed by: X3 Secretariat
Ref. Document: X3/87/07-069, X3/87/05-035 X3/87/07-065

FOR ACTION

Statement: Technical Committee X3J3 has voted 29-7 to forward this document to SPARC and X3 for further processing. SPARC reviewed the dprANS for compliance at their July meeting and found that it did comply with the authorizing document.

Question: Do you approve forwarding X3.9-198x to BSR for a four-month public review and comment period?

Yes ✓ *No _____ *Abstain _____

SIGNATURE Donald C. Loughry

ORGANIZATION REPRESENTED ON X3 Hewlett-Packard
(PLEASE PRINT)

PRINCIPAL ☒ ALTERNATE ☐ DATE Aug '87

*Operating under the procedures of The American National Standards Institute.

NOTE: If you find that you cannot vote YES and wish to be recorded as NOT VOTING or voting NO, please state this and explain the reasons for your position in the space above or on a separate sheet.

X3 Secretariat: Computer and Business Equipment Manufacturers Association, 311 First Street, NW, Suite 500, Washington, DC 20001-2178

Tel: 202/737-8888
Fax: 202/638-4922

106 (*) JCA-2
p. 14

HEWLETT PACKARD

COMPUTER LANGUAGE LAB • 19447 Pruneridge Avenue, Cupertino, CA 95014 • Telephone: (408) 725-8111

FROM: Carl Burch

DATE: August 19, 1987

TO: CBEMA

SUBJECT: X3 Ballot on dpANS Fortran

The next page contains the comments on HP's YES vote for the Fortran 8x draft standard.
Ms. Trudy Reusser will deliver the ballot itself as she is currently in Washington for an OSI meeting.
Thank you,

Carl Burch

20.

106(x) JCA-2
p.15

Hewlett-Packard Company votes YES on this ballot.

The design of Fortran 8x extends FORTRAN 77 to:

- provide array-level operations
- improve facilities for portable numerical computation
- allow user-defined data types
- provide more structured means of data and procedure encapsulation
- introduce the concept of language evolution

Hewlett-Packard believes that these goals represent a long-overdue modernization of Fortran and that the design of Fortran 8x accomplishes these goals. In particular, the software engineering enhancements proposed address the worst problem of Fortran - the difficulty of producing well-structured programs that are both more portable and reliable than their FORTRAN 77 counterparts. As applications programs grow ever larger, the primitive data types and procedure interfaces available in FORTRAN 77 have become unwieldy and unexpressive. Fortran 8x successfully addresses these problems by borrowing well-tested concepts from other languages, while avoiding their known pitfalls.

The new array operations provided are sometimes erroneously thought of as a means of efficiently utilizing vector processing hardware. In fact, the real value of array operations lies in their greater expressive power and more natural syntax as used in mathematics and the physical sciences.

The improved facilities for writing portable numerical software include several new intrinsic functions, but the primary new concept is that of generalized precision reals. Generalized precision reals allow the user to specify the basic precision needed without the current confusion between REAL and DOUBLE PRECISION as implemented by various vendors. It should be noted that this extension could only come from the standards process, not from any existing implementation (since REAL and DOUBLE PRECISION seldom vary greatly within a given vendor's line, only between vendors).

User-defined data types and the module mechanism advance the structured programming support of Fortran to make it again a credible choice for large scientific and engineering programs. That such programs exist today is a tribute more to the programmers responsible than to the language. Large FORTRAN programs frequently become very expensive to maintain due to their lack of structure. Module mechanisms and structured data types may be efficiently implemented, as clearly demonstrated by the existing implementations of Pascal and Modula-2; the MODULE extensions in HP Pascal are very similar to those in Fortran 8x.

The concept of language evolution is crucial to all language standards - it is very shortsighted to believe that anything can grow without bound. Providing a means of removing outmoded features and mistakes is necessary to the long-term life of any standard. X3's concept has been implemented by X3J3 in establishing rigorous definitions of the obsolete, obsolescent, and deprecated features categories. Because no obsolescent features were listed in FORTRAN 77, the obsolete category is empty for Fortran 8x - 100% backwards compatibility. X3J3 recognizes that the obsolescent and deprecated categories are recommendations and not orders for future committees. They have provided the possibility of graceful evolution of long-term programs over a period of many years.

Hewlett-Packard believes that these goals are both appropriate and well implemented in Fortran 8x. The draft standard should be forwarded immediately for public review.

106(*) JCA-3
P. 1

(3)

Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109

July 31, 1987

Dr. Jeanne Adams, Chair X3J3
National Center for Atmospheric Research
Scientific Computing Division
Box 3000
Boulder, CO 80307

Dear Dr. Adams:

I had been mistakenly informed that the public review period for S8 had begun, and would end in September, so I prepared a critique. Since I already have it, and don't expect much change in S8, your having my critique will expedite attention to it. I will also submit it during the formal public review period. It is my (perhaps erroneous) understanding that X3J3 must act formally on every public comment.

Although I also mention many of the additional features for which desire was expressed in letters and ballot remarks of observers and members of X3J3, my primary concern is that active handicaps, that is, features that cause difficulty by their presence or form, not be institutionalized. I continue to believe that there is one substantial active handicap about to be institutionalized into the standard that prevents the simultaneous application of modern principles of software engineering and construction of an efficient program: the referential syntax for elements of derived types. All of the other concerns expressed in the attached critique, especially regarding features that could be compatibly added, are secondary.

Our previous proposals would prevent the above described defect, and address concerns of several observers and members of X3J3 that the language is becoming too large, complex and irregular to allow comprehensive understanding or support efficient implementation. I notice in particular that Tadayoshi Kan advocates an intrinsic type for multi-byte character codes, asserting[1] that a derived type would not allow sufficiently efficient implementation.

Our earlier proposals advocated a general mechanism that would allow application of principles of modern software engineering, while allowing efficient implementation. Absent such a mechanism, X3J3 is doomed to be deluged with requests of the same kind as Mr. Kan's for independent new features.

I also include a review of a book on programming language design by David Harland, and illustrations of the module SET_INTEGER (from appendix C) cast in the syntax I advocate. Some of the ideas therein, when cast in the syntax of S8, would clarify that example.

Sincerely,

W. Van Snyder
Mail Stop 301-490

---

[1]Page 214 of 105th Pre-meeting distribution.

106(X) JCA 3
P. 2

W. Van Snyder
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

0. INTRODUCTION

The language proposed by X3J3/S8.104 is usable. But it institutionalizes many unnecessary handicaps. The ability to function in the presence of a handicap does not make it desirable, or even acceptable. Furthermore, any handicap cast into the standard is essentially eternal. It is therefore desirable to remove any identified handicaps, even if doing so might delay official promulgation of a standard. In particular, the features of S8.104 that prevent the simultaneous application of modern software engineering principles and the construction of an efficient program should be removed or revised. Promulgation should not be delayed, however, simply to add new features.

A general criticism of the proposed language is that it uses many independent, weak, special-purpose mechanisms when fewer regular, powerful, general mechanisms would accomplish the same ends, and result in clearer, more compact, more efficient programs. We have previously supplied several concrete suggestions to remedy this defect, and review them here superficially. This general criticism appears in several of the following six categories of specific objections:

1. The proposed language is too small.

2. The proposed language is too large.

3. The proposed language does not adequately support modern software engineering principles.

4. The proposed language contains too many arbitrary and unnecessary restrictions, many apparently retained only for compatibility.

5. The proposed language includes unnecessary impediments to portability.

6. There are many errors and ambiguities in the description.

These complaints overlap somewhat, so some may be addressed in several sections below.

The assertions that the language is simultaneously too small and too large may seem contradictory, but the dimensions in which the proposed language is too small and too large are not entirely dependent on one another.

1. THE PROPOSED LANGUAGE IS TOO SMALL BY NOT HAVING ENOUGH *CAPABILITIES*.

The proposed language lacks
- the ability to access bits, bit strings or parts of integers.
- the ability to create, access and destroy anonymous structured data objects.
- the ability to detect and cope with exceptional conditions in a graceful way.

- conditional expressions, a useful facility that could be provided by an intrinsic function or a distfix operator.
- provision for conditional compilation, a substantial aid in the maintenance of several related versions of a single program. This could be provided by an extension of the semantics of IF and CASE statements that would not require extension of the syntax (but see [1]).
- the ability to define subprograms having an arbitrary number of arguments. The intrinsic functions MAX and MIN have this property, but one could not define a MAX function that accepts an arbitrary number of arguments of a derived type.
- the ability to determine whether an optional argument in a computed position, e.g. the i'th optional argument, is present. The PRESENT intrinsic function can only act on the name of an optional argument, not a variable that denotes the position of an optional argument. To remedy this and the previous deficiency, one might provide an ARGUMENT(I) intrinsic that accesses the i'th argument. Then PRESENT(ARGUMENT(I)) = "the i'th argument is present," and ARGUMENT(I) = Z would store Z in the i'th argument.

The lack of a capability is not alone sufficient cause to delay promulgation of a standard, or to delay presenting it for public review. But a feature of the design of the language that prevents subsequent compatible addition of a capability *is* sufficient cause. Before the standard is promulgated, or presented for public review, there should be an effort by X3J3 to determine that every capability for which there has been substantial support can be added in a compatible way.

Further deficiencies will be discussed below.

2. THE PROPOSED LANGUAGE IS TOO LARGE BY HAVING TOO MANY *MECHANISMS*.

Much of the baggage of S8 is given over to explanation of whole array and array section operations. By allowing these to be abstractions defined by the standard, but replaceable by the user, all of the power and expressiveness of the language as presently defined would be allowed, and much more. A useful abstracting device in this case is an in-line subprogram having *quoted* arguments, that is, arguments that are textually substituted for their references. (They would necessarily be evaluated differently in value-receiving and value-providing contexts.) By using this device, the semantics of operations on aggregates (such as addition or assignment) could be precisely defined by the standard by subprograms that use more elementary features, and for which an implementor is free to provide a more efficient version of equivalent functionality. By defining more general modes of reference to arguments, the standard would provide all the functionality of the present ARRAY facility by a more general mechanism, which would therefore have broader utility. This capability could easily subsume the character string facility as well, thereby reducing the number of defined mechanisms (while retaining compatibility with Fortran 77).

Selection of an element of a derived type is denoted by an unique syntax. Just as is array element selection, so also selection of an element of a derived type consists of invoking a subprogram that the translator happens to know how to write (and which most translators put in-line). This may seem to be a distinction without a difference, but the requirement of a different syntax prohibits the simultaneous application of modern principles of software engineering and generation of efficient machine code. This is discussed further in

conjunction with the discussion of the failure to support modern software engineering.

It would be better to adopt a function-like syntax, in which the structure element appears in the position of the function name, and the parent type appears in the argument position[1]. A natural extension of the semantics, without change to the syntax, would then encompass access to anonymous structures by way of pointers. This is in fact what is done by the compiler in the case of static structures, as the *parent structure* clause of a structure component reference constitutes a pointer the compiler knows how to evaluate.

Some of the intrinsic functions are redundant. For example, ATAN2 could be deprecated, in favor of overloading ATAN. ALL would be the same as PRODUCT if PRODUCT could be applied to LOGICAL arrays, and the logical product were defined to be ".AND.". Similarly, ANY would be the same as SUM if SUM could be applied to LOGICAL arrays, and the logical sum were defined to be ".OR.". COUNT could also be replaced by SUM in contexts that demand the argument to be a LOGICAL array and the result an INTEGER. Alternately, ALL could be replaced by MINVAL and ANY could be replaced by MAXVAL if ".FALSE." were defined to be less than ".TRUE.". Even without changing the semantics of presently defined intrinsics, ANY and ALL are redundant, because ANY(L) = COUNT(L).ne.0, and ALL(L) = COUNT(L).eq.ESIZE(L)[2]. SETEXPONENT could be removed by a mechanism discussed in the next section.

The responses to many of the ballots that advocated removing various parts of the proposed language were essentially that none could be removed independently without compromising the whole, or that there was sufficient sentiment among users to justify inclusion. The foregoing arguments do not advocate removing any functionality of the proposed language; they instead advocate a more regular, more powerful, more efficient notation for the capabilities X3J3 has already decided are necessary.

3. THE PROPOSED LANGUAGE DOES NOT ADEQUATELY SUPPORT MODERN SOFTWARE ENGINEERING PRINCIPLES.

One of the central tenets of modern software engineering is that the use of an abstraction should not depend on its representation or mechanism. When elementary abstractions provided by the programming language are represented by different syntaxes, a conscientious programmer will encapsulate references to them in subprograms, thereby providing a uniform syntax that reflects nothing of the representation or mechanism of the encapsulated abstractions. This practice allows the representation or mechanism or each abstraction to change independently in response to changing environments or requirements, but imposes the expense of subprogram invocation to access simply represented objects, and the

---

[1]In the response to the letter ballot from Anil Lakhwara, there was a remark that "... great attention was paid to making them [extensions] as Fortran-like as possible." This is untrue for the referential syntax for derived types unless one means it is no more unusual or irregular than other new features.

[2]Implementing ANY and ALL using COUNT is inefficient. Using SUM and PRODUCT, or MAXVAL and MINVAL, could be as efficient as a direct intrinsic, because the underlying logical operations can still be short-circuited.

expense of writing and validating the subprograms. Furthermore, the acts of accessing and assigning a value aren't symmetrically represented. Clarity and efficiency therefore suffer unnecessarily.

There are two simple changes that would allow direct (efficient) access to abstractions, without compromising the principle that the client of an abstraction should not depend on its representation or mechanism.

The first is to use the same syntax for access to every elementary data structure, and to subprograms. Since access to functions and arrays already uses the same syntax, this requires no departure from compatibility with Fortran 77. But the referential syntax for elements of derived types should be the same as for functions and arrays, with the element name occupying the same syntactic position as the function or array name, and the parent structure name (or pointer expression denoting an anonymous parent structure) occupying the position of arguments or subscripts. This would allow references to structure elements to be replaced by function references or array element references. It does not alone, however, allow completely transparent replacement of the representation or mechanism of an abstraction.

In order to allow complete flexibility to replace the representation or mechanism of an abstraction transparently, one needs to define *accessor* functions, that is, subprograms that act exactly as conventional functions now do when references to them appear in value-providing contexts, but to which references may also appear in *value-receiving* contexts, causing the invocation of a separate body of instructions. By allowing complete symmetry of syntax of reference between arrays, subprograms and derived types, a conscientious programmer would no longer be required to write a myriad of trivial subprograms simply to hide syntactic differences between the representations chosen for the several data objects of a typical program. In fact, there would be no syntactic device that would allow a difference of representation to be exposed. Furthermore, an object represented by an elementary mechanism known to the translator can be accessed efficiently. If subprograms are the only mechanism to encapsulate syntactic differences between references to elementary representations, it is difficult for a translator to produce an efficient program. A language that allows definition of in-line subprograms might admit an efficient translation, but the programmer must still write and validate the subprograms. I presented an argument for the principle of uniform syntax at the November 1986 meeting at Albuquerque. See also [2].

Using an uniform referential syntax and providing accessor functions exchanges a special-purpose syntax that denotes a rather weak capability for a powerful general mechanism that supports rather than inhibits the efficient application of software engineering principles. This general-purpose mechanism, combined with an ability to materialize a subprogram in-line rather than invoke it remotely, could also subsume the functionality of ALIAS declarations, ALIAS arrays and IDENTIFY statements. By allowing the generalized form of subprogram argument described in section 2, in-line accessor subprograms could subsume the RANGE attribute and SET RANGE statement. Several intrinsic functions could be made accessors (or, transparently, elements of derived types -- the user would never know which): EXPONENT and FRACTION might be accessors or structure components whose parents are the several different representations of real numbers (SETEXPONENT would be redundant); so also might REAL and AIMAG be defined for operations on complex objects. (COMPLEX could be defined to be a

derived type the translator knows without declaration, together with functions that implement the operations now defined, without a declaration.) PACK and UNPACK could be combined into a single accessor. These changes would provide a non-trivial reduction in the size of the language definition. The functionality of the language could be increased without increasing the size by defining MERGE to be an accessor, allowing selective assignment of values as well as selective access.

David Harland has written a book [3] that makes educational reading for anyone who pretends to the difficult task of language design. An excellent review was presented in [4]. Harland also argues that a few general, powerful, regular mechanisms are to be preferred over a large number of independent weak special-purpose mechanisms.

4. THE PROPOSED LANGUAGE CONTAINS TOO MANY ARBITRARY AND UNNECESSARY RESTRICTIONS. MANY APPARENTLY RETAINED ONLY FOR COMPATIBILITY.

There may be, or may have been, reasons for the restrictions mentioned below. But most of them seem silly. If there are good reasons for them, they should be explained in the standard or the Section Notes appendix.

Many restrictions present in Fortran 77 are now irrelevant. For example, there is still a restriction that variables in BLANK COMMON cannot be assigned initial values in a BLOCK DATA subprogram. I realize that BLOCK DATA and COMMON are deprecated features, but this restriction was always silly, and many compilers relaxed the Fortran 77 standard by allowing variables in BLANK COMMON to be initialized by BLOCK DATA. Removing the restriction from the standard would not invalidate any standard-conforming Fortran-77 program, but would make the S8 standard shorter, and admit programs admitted by compilers that extended the Fortran 77 standard.

There was a restriction in Fortran 77 that a function referenced from an input or output list must not initiate input or output. This derived from the absence of recursion. Since the language proposed by S8 admits recursion, it seems silly to maintain the restriction that functions referenced from input or output lists not initiate input or output.

On page 7-3, line 2, why is *concat-op* restricted to operate only on character expressions? It could certainly be sensibly applied to any rank-one section of an array (with useful side-effect on the effective range of a variable to which the expression was assigned or bound). For arrays of rank greater than one, one must specify the dimension in which concatenation is to take place, but one could still define the effect of applying *concat-op* to arrays of rank greater than 1, for example by stipulating that concatenation takes place in the first dimension, since that is what really happens during character array concatenations.

Several new restrictions of no apparent need have been introduced in S8.

An EXIT statement may refer only to a *do-construct*. By allowing EXIT to apply to any construct certain control strategies may be more clearly realized. Consider detecting the condition $x \notin S$, where the first N elements of an array S represent the set S (see [5] for elaboration):

```
B:    DO BLOCK
          DO i = 1, N
              IF (x .eq. S(i)) EXIT BLOCK
          END DO
          !    x ∉ S here
      END BLOCK
```

The maximum length of a name or defined operator is limited to 31 characters.

A line of a free-form source statement must contain no more than 132 characters.

A statement must contain no more than 2640 characters.

The maximum rank of an array is seven.

Either the parent structure of a derived type, or all its elements, must be scalars. (If function-like syntax is used, subscripts appear in the "right" order even if both the parent and component are arrays.)

If the only reason for prohibiting an internal procedure to be an actual argument is a perceived implementation difficulty, then the prohibition should be removed. The difficulty is not intrinsic, but rather is due to ignorance of efficient mechanisms. Allowing a procedure that is internal to a recursive procedure to be an argument causes difficulty only if *displays* are used to keep track of activation records. If *up-links* are used there is not substantial difficulty, even if arbitrary multilevel nesting is allowed.

Several functions may have the same name, so long as the constellations of their argument specifications are disjoint. But no array may have the same name as another array (even if they are of different rank), nor the same name as a function (even if functions with the same name have a number of arguments different from the rank of the array, or arguments not suitable for subscripting). Since an array declaration is a terse declaration of an accessor function that the translator usually materializes in-line, the latter two restrictions seem arbitrary.

The *action-stmt* in an *if-stmt* must not be an *if-stmt*. Relaxing this constraint would allow the predicate of the consequent *if-stmt* not to be evaluated if the predicate of the primary *if-stmt* were false. This of course can be specified by using a block IF construct, but an *if-stmt* that is a consequent of another *if-stmt* is more concise.

A stop code may consist of no more than 5 digits.

The execution part of a *main-program* must not contain a RETURN statement. A RETURN statement appearing in a *main-program* could be interpreted as a STOP statement having no *stop-code*.

If the ARRAY argument of DSIZE, DUBOUND, ESIZE or EUBOUND is an assumed-size array, DIM must be present with value less than the rank of ARRAY. Is this because the upper bound of the last dimension of an assumed-size array is not passed from the actual argument to the formal argument? If this is so, how are ALL, ANY, COUNT, MINVAL, PACK and SUM defined without this restriction?

MAXVAL, PRODUCT and UNPACK avoid the problem by requiring that their ARRAY argument (MASK argument of UNPACK) be of defined shape.

5. THE PROPOSED LANGUAGE INCLUDES UNNECESSARY IMPEDIMENTS TO PORTABILITY.

The processor-defined nonzero values assigned to status variables constitute an impediment to portability. The standard should define an intrinsic function that accepts a processor-defined nonzero status value, prints the error message that would have been printed had the status clause been absent, and returns values defined by the standard that suggest whether the condition is a warning, an error that might be corrected with revised input, or a condition from which recovery is impossible (in which case the program is advised to stop gracefully). An example of the first is an end of file upon input; an example of the second is the failure to open a file, which might be corrected if the user supplied a revised file name; an example of the last is deallocation of an array not allocated. The sets of errors that are failures or correctable could be processor-dependent without serious impediment to portability.

The "particular processor-dependent unit that is preconnected for formatted sequential access" constitutes an impediment to portability. The INQUIRE statement should be extended to allow discovering the values of these unit numbers, or an intrinsic function should be provided to discover them.

6. THERE ARE MANY ERRORS AND AMBIGUITIES IN THE DESCRIPTION.

Every keyword should be in an index. In particular, PUBLIC and PRIVATE are not indexed. There may be others that are not indexed.

On page ii, line 28, the sentence "With additional operation definitions, derived data types provide an effective implementation mechanism for data abstractions." is false, and should be deleted. The ability to define operations is useful for abstraction, but given the specified referential syntax, derived data types are a mechanism for data "concretization." The only *effective* mechanism for data abstraction in the language proposed by S8.104 is a subprogram, just as was the case in Fortran 77.

On page 8-10, line 40, what does it mean to say that "the stop code, if any, is accessible?"

On page 9-5, line 40, does the statement "There may be no means of reconnecting an unnamed file once it is disconnected" mean that a standard conforming processor is prohibited from providing such a means, or that a standard conforming processor is allowed not to provide such a means?

If one executes

```
READ (10,"(a/a)") S1,S2
BACKSPACE 10
```

what is the final position of file 10? Is it between the data transmitted into S1 and S2, or before the data transmitted to S1?

On pages 13-32 and 13-34, MAX and MIN are defined to allow a variable number of

arguments. Which of the arguments are accessible by specifying a keyword? Would MAX(X,Y,A13=Z) make sense?

Is the description on Page C-9, lines 23-31, correct in the event the carriage control character for printed output is "+"? What is the distinction, if any, between printed lines and records? What is the interaction between "/" editing and the "+" carriage control character? For example, what happens if one prints using an empty I/O list and FMT='('' a''/''+b'')'? The interaction is not described in any of §§ 9.4.5, 10.6.2 or C.10.

In addition to errors and ambiguities there are several examples of poor programming practice, and many that provide no information. Only a small number are discussed below.

Page C-8, line 33, would be clearer and more efficient if written

```
Z = CMPLX(X,Y)
```

Page C-12, line 50, would be clearer and more efficient if written

```
ELEMENT = ANY(A%ELEMENT_VALUE(1:CARD(A))==X)
```

or, using the uniform syntax we advocate

```
   ELEMENT = ANY(ELEMENT_VALUE(A,1:CARD(A))==X)
or ELEMENT = ANY(ELEMENT_VALUE(A)(1:CARD(A))==X)
```

It is not clear from the definition of ANY whether one can replace all of lines 48 to 50 by the first of the above statements, but from the definition of COUNT, one could surely replace them by

```
ELEMENT = COUNT(A%ELEMENT_VALUE(1:CARD(A))==X) > 0
```

On page C-13, line 37 should be

```
DO J = ELBOUND(V), EUBOUND(V)
```

The sorting algorithm used on page C-13 line 50 through page C-14 line 4 is roundly condemned in textbooks on algorithm design. See e.g. [6].

On page C-14, lines 6-11, SUBROUTINE SET_ASSIGNMENT_COERCION is needlessly complicated, because the ELEMENT_VALUE fields of all objects of type SET have the same declared size, and the RANGE attribute is not specified.

7. CONCLUSIONS

We have outlined above what we believe to be several deficiencies in the language proposed by S8.104, or the exposition therein. In some cases, we have outlined suggested repairs. Many of our suggestions directly address the concerns of several of the members that voted not to send S8 to X3 for public review. In particular, the proposal to substitute regular, powerful, general mechanisms for irregular, weak, special-purpose mechanisms would perpetuate a syntax more like Fortran 77, allow greater functionality than allowed by S8.104, reduce the size of the description of the language, reduce the diffi-

culty to learn the language, allow a translator to produce a more efficient translation, allow a more readable program, and allow a program that is robust in the face of changing requirements without compromising efficiency or readability.

8. REFERENCES

1. W. Van Snyder, Letter to Jeanne Adams of 12 January 1987.

2. Charles M. Geschke and James G. Mitchell, *On the problem of uniform references to data structures*, **IEEE Transactions on Software Engineering, SE-1**, 2 (June 1975) pp 207-219.

3. David M. Harland, **Concurrency and Programming Languages**, Halsted Press, New York, NY (1986).

4. Benedict Heal, review 8706-0446 of *Concurrency and Programming Languages*, **Computing Reviews** (June 1987).

5. W. Van Snyder, *Multilevel EXIT and CYCLE aren't so bad*, **ACM SIGPLAN Notices 22**, 5 (May 1987) pp 20-22.

6. Robert Sedgewick, **Algorithms**, Addison-Wesley, Reading, MA (1983) pp 94-99.

We include here an alternate view of the module INTEGER_SETS presented on pages C-12 through C-14. We illustrate the transparency and efficiency that would be allowed by the notation we advocate. Notice, for example, that there is no need for a CARD function: the CARDINALITY field of a SET object can be examined but not changed outside the module, exactly as would be the case if CARDINALITY were implemented by a function. But if it became necessary to change the representation of a set, and it consequently became necessary to represent CARDINALITY as a function, the clients of this module would not be affected. Even if cast in the notation of S8, this module would be clearer, more efficient, and a better example of features of S8 than the example presented in S8.104.

```
MODULE INTEGER_SETS

PARAMETER (MAX_SET = 200)
SECTION, PARAMETER :: EMPTY = 1:0

TYPE SET                                              ! DEFINE THE SET DATA TYPE
   INTEGER, LIMITED PRIVATE :: CARDINALITY            ! READ_ONLY OUTSIDE MODULE
   INTEGER, PRIVATE :: ELEMENT_VALUE(MAX_SET)         ! INVISIBLE OUTSIDE MODULE
END TYPE SET

CONTAINS

SUBROUTINE SET_ASSIGN (A, B) ASSIGNMENT               ! FOR A = B
   TYPE (SET) A, B
   CARDINALITY(A) = CARDINALITY(B)
   ELEMENT_VALUE(A,1:CARDINALITY(A)) = ELEMENT_VALUE(B,1:CARDINALITY(B))
END SUBROUTINE SET_ASSIGN

SUBROUTINE SET_ASSIGN (A, V) ASSIGNMENT               ! FOR A = V; EACH DISTINCT
   INTEGER V(:)                                       ! ELEMENT OF V IS PUT INTO A.
   TYPE (SET) A
   INTEGER J
   CARDINALITY(A) = 0
   DO J = 1, EUBOUND(V); CALL INSERT (V(J), A); END DO ! J
END FUNCTION SET_ASSIGN

LOGICAL FUNCTION ELEMENT (X, A) OPERATOR (.IN.) INLINE
*  RETURNS "X IS IN SET A"
   INTEGER X
   TYPE (SET) A
   ELEMENT = ANY(ELEMENT_VALUE(A,1:CARDINALITY(A)) == X)
END FUNCTION ELEMENT

SUBROUTINE INSERT (X, A)                              ! MAKES X AN ELEMENT OF A
   INTEGER X
   TYPE (SET) A
   IF (.NOT. X.IN.A) THEN
        CARDINALITY(A) = CARDINALITY(A) + 1
        ELEMENT_VALUE(A,CARDINALITY(A)) = X
   END IF
END SUBROUTINE INSERT
```

```
FUNCTION UNION (A, B) OPERATOR (+)                ! UNION OF SETS A AND B
   TYPE (SET) A, B, UNION
   INTEGER J
   UNION = A                                      ! INVOKES SET_ASSIGN
   DO J = 1, CARDINALITY(B)
        CALL INSERT (ELEMENT_VALUE(B,J), UNION) ! UNION += ELEMENT_VALUE(B,J)
   END DO ! J
END FUNCTION UNION

FUNCTION DIFFERENCE (A, B) OPERATOR (-)           ! DIFFERENCE OF SETS A AND B
   TYPE (SET) A, B, DIFFERENCE
   INTEGER J, X
   DIFFERENCE = EMPTY
   DO J = 1, CARDINALITY(A)
        X = ELEMENT_VALUE(A,J)
        IF (.NOT. (X .IN. B)) CALL INSERT (X, DIFFERENCE) ! DIFFERENCE += X
   END DO ! J
END FUNCTION DIFFERENCE

FUNCTION INTERSECTION (A, B) OPERATOR (*), INLINE
*  RETURNS THE INTERSECTION OF SETS A AND B
   TYPE (SET) A, B, INTERSECTION
   INTERSECTION = A - (A - B)
END FUNCTION INTERSECTION

LOGICAL FUNCTION SUBSET (A, B) OPERATOR (.IN.)  ! RETURNS "A IS A SUBSET OF B"
   TYPE (SET) A, B                              ! OVERLOADS .IN. OPERATOR
   SUBSET = CARDINALITY(A) <= CARDINALITY(B)    ! INLINE WOULD BE REASONABLE
   IF (SUBSET) SUBSET = ALL(ELEMENT_VALUE(A,1:CARDINALITY(A)) .IN. B)
END FUNCTION SUBSET

FUNCTION VECTOR (A)                             ! TRANSFER THE VALUES OF SET A
   TYPE (SET) A                                 ! INTO A VECTOR HAVING ELMENTS
   INTEGER, ALLOCATABLE, RANGE :: VECTOR(:)     ! IN ASCENDING ORDER
   INTEGER J, K
   ALLOCATE (VECTOR (0:CARDINALITY(A)))
   VECTOR(0) = -HUGE(J)
   DO J = 1, CARDINALITY(A)
        K = J
        DO
             IF (ELEMENT_VALUE(A,J) >= VECTOR(K-1)) &
        EXIT
             VECTOR(K) = VECTOR(K-1)
             K = K - 1
        END DO
        VECTOR(K) = ELEMENT_VALUE(A,J)
   END DO ! J
   SET RANGE (1:CARDINALITY(A)) VECTOR
END FUNCTION VECTOR
```

We include here an alternate view of the module INTEGER_SETS presented on pages C-12 through C-14. We illustrate the transparency and efficiency that would be allowed by the notation we advocate. The cardinality of a set is represented by the effective bound of the RANGEd vector that contains its contents. Thus the CARDINALITY function is just another spelling for EUBOUND. Even if cast in the notation of S8, this module would be clearer, more efficient, and a better example of features of S8 than the example presented in S8.104. Although the use of RANGE and SET RANGE provide a commendable terseness and clarity in subprograms SET_ASSIGN(A, B), ELEMENT(X, A) and SUBSET(A, B), this use is prohibited by lines 24-25 on page 6-7 of S8.104.

```
MODULE INTEGER_SETS

PARAMETER (MAX_SET = 200)
SECTION, PARAMETER :: EMPTY = 1:0

TYPE SET                                          ! DEFINE THE SET DATA TYPE
   PRIVATE
   INTEGER, RANGE :: ELEMENT_VALUE(MAX_SET)       ! INVISIBLE OUTSIDE MODULE
END TYPE SET

CONTAINS

INTEGER FUNCTION CARDINALITY (A) INLINE
   TYPE (SET) A
   CARDINALITY = EUBOUND(ELEMENT_VALUE(A))
END FUNCTION CARDINALITY

SUBROUTINE SET_ASSIGN (A, B) ASSIGNMENT           ! FOR A = B
   TYPE (SET) A, B
   SET RANGE (1:CARDINALITY(B)) ELEMENT_VALUE(A)
   ELEMENT_VALUE(A) = ELEMENT_VALUE(B)
END SUBROUTINE SET_ASSIGN

SUBROUTINE SET_ASSIGN (A, V) ASSIGNMENT           ! FOR A = V; EACH DISTINCT
   INTEGER V(:)                                   ! ELEMENT OF V IS PUT INTO A.
   TYPE (SET) A
   INTEGER J
   SET RANGE (EMPTY) ELEMENT_VALUE(A)
   DO J = 1, EUBOUND(V); CALL INSERT (V(J), A); END DO ! J
END FUNCTION SET_ASSIGN

LOGICAL FUNCTION ELEMENT (X, A) OPERATOR (.IN.) INLINE
*  RETURNS "X IS IN SET A"
   INTEGER X
   TYPE (SET) A
   ELEMENT = ANY(ELEMENT_VALUE(A) == X)
END FUNCTION ELEMENT

SUBROUTINE INSERT (X, A)                          ! MAKES X AN ELEMENT OF A
   INTEGER X
   TYPE (SET) A
   IF (.NOT. X.IN.A) THEN
*       ELEMENT_VALUE(A) = ELEMENT_VALUE(A) // X ! UPDATES EFFECTIVE SHAPE
        SET RANGE (1:CARDINALITY(A)+1) ELEMENT_VALUE(A)
```

```
        ELEMENT_VALUE(A,CARDINALITY(A)) = X
   END IF
END SUBROUTINE INSERT

FUNCTION UNION (A, B) OPERATOR (+)                ! UNION OF SETS A AND B
   TYPE (SET) A, B, UNION
   INTEGER J
   UNION = A                                      ! INVOKES SET_ASSIGN
   DO J = 1, CARDINALITY(B)
        CALL INSERT (ELEMENT_VALUE(B,J), UNION) ! UNION += ELEMENT_VALUE(B,J)
   END DO ! J
END FUNCTION UNION

FUNCTION DIFFERENCE (A, B) OPERATOR (-)           ! DIFFERENCE OF SETS A AND B
   TYPE (SET) A, B, DIFFERENCE
   INTEGER J, X
   DIFFERENCE = EMPTY
   DO J = 1, CARDINALITY(A)
        X = ELEMENT_VALUE(A,J)
        IF (.NOT. (X .IN. B)) CALL INSERT (X, DIFFERENCE) ! DIFFERENCE += X
   END DO ! J
END FUNCTION DIFFERENCE

FUNCTION INTERSECTION (A, B) OPERATOR (*), INLINE
*  RETURNS THE INTERSECTION OF SETS A AND B
   TYPE (SET) A, B, INTERSECTION
   INTERSECTION = A - (A - B)
END FUNCTION INTERSECTION

LOGICAL FUNCTION SUBSET (A, B) OPERATOR (.IN.)  ! RETURNS "A IS A SUBSET OF B"
   TYPE (SET) A, B                              ! OVERLOADS .IN. OPERATOR
   SUBSET = CARDINALITY(A) <= CARDINALITY(B)    ! INLINE WOULD BE REASONABLE
   IF (SUBSET) SUBSET = ALL(ELEMENT_VALUE(A) .IN. B)
END FUNCTION SUBSET

FUNCTION VECTOR (A)                             ! TRANSFER THE VALUES OF SET A
   TYPE (SET) A                                 ! INTO A VECTOR HAVING ELMENTS
   INTEGER, ALLOCATABLE, RANGE :: VECTOR(:)     ! IN ASCENDING ORDER
   INTEGER J, K
   ALLOCATE (VECTOR (0:CARDINALITY(A)))
   VECTOR(0) = -HUGE(J)
   DO J = 1, CARDINALITY(A)
        K = J
        DO
             IF (ELEMENT_VALUE(A,J) >= VECTOR(K-1)) &
        EXIT
             VECTOR(K) = VECTOR(K-1)
             K = K - 1
        END DO
        VECTOR(K) = ELEMENT_VALUE(A,J)
   END DO ! J
   SET RANGE (1:CARDINALITY(A)) VECTOR
END FUNCTION VECTOR
```

HARLAND, DAVID M. (Linn Products Ltd., Glasgow, Scotland) 8706-0446
Concurrency and programming languages.
Halsted Press, New York, NY, 1986, 182 pp., $36.95, ISBN 0-490-20362-5. [Ellis Horwood series in computers and their applications.]

This is one of the most stimulating books on programming that I have read in a long time. I hope that the title does not put off potential readers by suggesting any narrowness of concern, as this book is as wide in its implications as it is deep in insight. It should be read by anyone interested at any level in programming notations. The main concern is a compelling argument for the central importance of abstraction in programming languages.

Most programming languages consist of a rather ad hoc collection of features favored by the designer. They usually have a large number of fixed parts, and few variable ones. If the selection happens to match a specific application, all is well; if not, programming is often a struggle to match the abstractions of the design process to the fixed set of abstractions provided in the programming language. No fixed set of features, however rich, can suit all applications. Even 4GLs are only fixed collections of predefined abstractions. Only if the programming language is extensible can it be expected to be of general applicability. It is not enough to build onto a language a few special calculi for specific areas, nor to have numerous different languages that cannot be integrated. We need a single computational model, capable of extending and modifying itself. Mathematics achieves its power not just as an aggregate of the structures we build with it, but as the way by which we define new structures.

This book is an argument for such a system, and demonstrates the immense power and expressiveness of a very simple kernel language when it is provided with very general facilities for abstraction. Harland argues that anything at all in the language must be abstractable, or it will restrict the programmer. By abstraction, he means the ability to replace any clause of a program by a name, without changing the meaning. If this is allowed uniformly, a consequence is that it is impossible for the user to tell of a particular value if it is "primitive" or constructed. Most languages provide some restricted form of abstraction ability in the shape of a procedure or function mechanism. In Pascal, one can replace a sequence of statements by a procedure name with parameters, but one cannot replace a write statement by a user-defined name, as this would entail defining a polymorphic procedure with a variable number of arguments. In an unrestricted system, we would be able to abstract over the act of application itself, and define EVAL (function, arglist). Thus, we are soon led along the route of higher-order polymorphic languages, though Harland is emphatic that purely functional, referentially transparent languages are, in general, too weak as they are unable to cope with the concept of change, as required in some systems such as real-time programming. They may be excellent subsystems, but they are not all-encompassing. Something like a store must be present, though, of course, it need not be seen in all abstractions presented to users. The insistence that anything can be abstracted implies that anything can be a value—not just integers and data structures, but functions, faults, and the act of application itself. If we are not to allow language designers to specify in advance what abstractions are permissible, we must as users be allowed to manipulate all the values that a compiler would—including bindings and types—in order to build our own abstractions.

The second part of the book is concerned with sequencing. Though this is fundamental to most imperative languages, in fact it is not usually presented in a very flexible way, as the static type conventions prevent us from building abstract general-purpose sequencers. Pascal provides a for-statement for ordinal types, and CLU allows users to define "iterators" over abstract types, but for full generality we should be able to define sequencing for all data structures. We are shown this in general, and then parametrized to specific cases like integers, lists, and stacks. Similarly, languages are usually inflexible as to the type of function call provided. If eager functions are available, lazy ones are probably impossible to define. By providing a notion of function free from any particular model of parameters, together with the ability to send in a general binding of names to unevaluated arguments, it is possible to construct any parameter mechanism we need, including default values and named parameters.

Though it is often argued that operating system shells require a different style of language in order to provide a dynamic and interactive environment for users, we are convincingly shown that, given the ability to manipulate bindings and environments, it is possible to construct all the desired features ourselves.

The final section is concerned with concurrency. The implicit concurrency of applicative languages is insufficient, as until concurrency is made explicit, it is impossible to abstract over it. To introduce as few new primitives as possible, only process creation and a test-and-set operation are provided. From these, semaphores, monitors, and conditional critical regions are all constructed. Instead of basing concurrency on shared data, we could as easily develop a polymorphic message-passing system out of the same primitives. Because processes themselves are values, we can pass them as messages, and hence describe a dynamically configurable distributed system. This convincingly demonstrates that for expressiveness, languages do not need to contain ever more features, but rather a few simple ones with powerful facilities for abstraction. The challenge now is to build the hardware to run them.

A poor index and a lack of precision in defining the language of the example programs only slightly mar an extremely exciting book.

—*Benedict Heal*, Milton Keynes, UK

GENERAL TERMS: LANGUAGES

W. Van Snyder
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109

We present a proposal to define character strings of varying length. The idea could be extended to strings of other objects of varying length by generalizing *concat-op* (//).

We do not know the best place to write these ideas into the proposed standard, so no concrete text is supplied. We believe the amount of text would be small.

- A character variable (scalar or array) is interpreted to be an array in which the zero'th dimension is the dimension along which characters are consecutive.

- If a character variable (scalar or array) has the RANGE attribute, then assignment to the variable sets the effective range of the zero'th dimension to the effective range of the value assigned to the variable. If a character variable is an array then different elements may have different ranges in the zero'th dimension.

- If A and B are character variables having the RANGE attribute, the expression A // B produces a character string having effective range(s) in the zero'th dimension equal to [1..EUBOUND(A,0)+ESIZE(B,0)].

- If A is a character variable then LENGTH(A) ≡ ESIZE(A,0).

- If A is an array character variable, then DLBOUND(A,0), DSIZE(A,0), DUBOUND(A,0), ELBOUND(A,0), ESIZE(A,0) and EUBOUND(A,0) produce values that are the same shape as A.

Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109

July 30, 1987

J. H. Matheny
c/o Dr. Jeanne Adams, Chair X3J3
National Center for Atmospheric Research
Scientific Computing Division
Box 3000
Boulder, CO 80307

Dear Mr. Matheny:

I read your memorandum 015(16)JHM-01 to X3J3, dealing with derived type I/O, with some interest. From my position of ignorance, it appears that the problem is being made more difficult than necessary, but the proposed solutions preserve the beloved Fortran tacked-on look. In several of my other communications to X3J3 I have advocated the use of powerful, regular, general mechanisms in place of weak, irregular, special-purpose mechanisms wherever possible.

One of the scraps of paper in the 105th pre-meeting distribution (I don't remember which) mentioned that object oriented concepts should be deferred. But by borrowing at least a few object-oriented ideas, the problem of defined-type I/O could be solved, along with several other problems, at a single stroke.

The first principle of object orientation that is different from other programming disciplines is that objects have capabilities that may be applied to them, rather than vice-versa. That is, the major organizational unit is an object, not a process.

The second principle is that an object may inherit capabilities from a more general object, ultimately from the standard unless otherwise specified by an intermediate object, so only new or different capabilities need be respecified.

These principles could be applied to derived types by augmenting them with capability subprograms, preferably including accessor subprograms, declared *in the body* of the type declaration. Certain capabilities of predefined names would be invoked automatically in certain circumstances. The others could be explicitly invoked.

For example, a capability EDIT_FORMAT(T)(f,x) might automatically be invoked when a datum x of a derived type T appears in an output list with corresponding format f; APPLY(T)(Z,R1,R2, ...) might be automatically invoked whenever a reference of the form Z(R1,R2, ...) appears, where Z is of derived type T. There might be several APPLY capabilities for a type, with different constellations of arguments, including the possibility that (R1,R2,...) might be absent. APPLY must necessarily be an accessor to allow Z(R1,R2,...) to appear in value-receiving contexts as well as value-providing contexts. If an intrinsic derived type SECTION_DESCRIPTOR, together with appropriate capabilities, were defined, this concept, together with the ability to materialize capabilities in-line, would allow RANGEd arrays to be specified in terms of derived types, with no loss of efficiency.

Sincerely,

Van

W. Van Snyder
Mail Stop 301-490

Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109

July 30, 1987

Carl Burch, Dick Hendrickson
c/o Dr. Jeanne Adams, Chair X3J3
National Center for Atmospheric Research
Scientific Computing Division
Box 3000
Boulder, CO 80307

Dear Mssrs. Burch and Hendrickson:

I read in 105(*)CDB-2 an account of an exchange regarding the utility of modules to implement new types efficiently. Carl disagreed with a statement by Jeanne Adams that BIT data type could be implemented by a module, on the grounds of efficiency. Dick pointed out that with **intrinsic** modules, the compiler is allowed to "cheat," and therefore able to provide an efficient translation.

It is unfortunate that the ability of the compiler to "cheat" must be restricted to translations using *intrinsic* modules. Making a module intrinsic increases the bulk of the standard almost as much as defining a new data type (although silly capabilities necessary only for "orthogonality" can be omitted from intrinsic modules but not intrinsic types). Furthermore, *every* abstraction, not just intrinsic abstractions, should be implementable as efficiently as possible. To this end, the standard should allow/define an **inline** attribute of subprograms. If this were done, there would be no need for a module to be **intrinsic** in order to allow the compiler to "cheat."

Sincerely,

Van

W. Van Snyder
Mail Stop 301-490

40

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
Scientific Computing Division/Advanced Methods Section
P. O. Box 3000 • Boulder, Colorado • 80307
Telephone: (303) 497-1275 • FTS: 320-1275

August 28, 1987

W. Van Snyder
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109

Dear Van Snyder;

I forwarded your letters, and will also place them in the pre-meeting distribution for November. I also received your comment. I am distributing this as well, and using it as a first case in processing comments, since you are sending it to CBEMA when the Public Review is announced. It will help us smooth out our comment processing. The vote in X3 was 31-4-2-1. Currently, a 10 day reconsideration period is in effect until Sept. 4, when X3 members may change their votes based on the 4 negatives that are distributed to them.

You have some very well considered suggestions which will all be considered by X3J3. I know that some members of X3J3 feel that your suggestions for changing the syntax for data structures to match those of function references and the inclusion of accessor functions are excellent ones. However, you cannot tell ahead of time how X3J3 will deal with your issues. We appreciate your careful study of the document.

After you submit your comments to CBEMA for formal public review, you will receive a formal acknowledgement letter from me.

Regards,

Jeanne Adams

Jeanne Adams, Advanced Methods Group
Chair, X3J3

cc. Carl Burch
X3J3 Distribution

106(*) JCA-4
P. 1 of 1

(4)

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
Scientific Computing Division/Advanced Methods Section
P. O. Box 3000 • Boulder, Colorado • 80307
Telephone: (303) 497-1275 • FTS: 320-1275 • Telex: 45 694

August 26, 1987

Presley Smith, Manager
Convex Computer Corporation
701 Plano Road
Richardson, Texas 75081

Dear Dr. Smith;

Your letter to the X3 Secretariat was forwarded to me by X3 for information to X3J3. It was distributed to X3J3 at the Liverpool meeting earlier this month.

X3J3 believes that the work on Fortran 8x does meet the requirements of the project proposal, as evidenced by the vote to forward Fortran 8x to X3 for further processing. We anticipate that you will forward your comments as a formal public review comment if X3 approves the public review processing. At that time, X3J3 will respond to your comments in detail.

I appreciate your continued interest in the Fortran Language, which will continue to serve the scientific community well. Thank you for your comments.

Sincerely yours,

Jeanne Adams

Jeanne Adams
Advanced Methods Group, NCAR
Chair, X3J3

cc. Gwendy Phillips, X3 Secretariat
cc. X3J3 Distribution

106(*) JCA-5
P. 1 of 1

5

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
Scientific Computing Division/Advanced Methods Section
P. O. Box 3000 • Boulder, Colorado • 80307
Telephone: (303) 497-1275 • FTS: 320-1275 • Telex: 45 694

August 26, 1987

Dr. John Reid
Computer Science and Systems Division
AERE Harwell, Oxfordshire
OX11 ORA
England

Dear John;

Your letter to the X3 Secretariat was forwarded to me as information to X3J3, and subsequently distributed to X3J3. We appreciate the support of IFIP WG2.5 while we attempt to reach the public comment period for Fortran 8x.

Your concerns are important, and your interest in our work and Fortran 8x is very much appreciated, especially at this time when there appears to be much public attention genererated, and not all of it favorable.

Regards,

Jeanne Adams

Jeanne Adams
Advanced Methods Group, NCAR
Chair, X3J3, Fortran Standards Committee

45

106 (X) JCA-6
P. 1 of 1

6

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
Scientific Computing Division/Advanced Methods Section
P. O. Box 3000 • Boulder, Colorado • 80307
Telephone: (303) 497-1275 • FTS: 320-1275 • Telex: 45 694

August 26, 1987

J. Turner, Director, Quality Assurance
Boeing Computer Services
PO box 24346
Seattle, Washington 98124-0346

Dear Dr. Turner;

Your letter to the X3 Secretariat was forwarded to me by X3 for information to X3J3. It was distributed to X3J3 at the Liverpool meeting earlier this month.

X3J3 believes that the work on Fortran 8x does meet the needs of the Fortran User Community, as evidenced by the vote to forward Fortran 8x to X3 for further processing. We anticipate that you will forward your comments as a formal public review comment if X3 approves the public review processing. At that time, X3J3 will respond to your comments in detail.

X3J3 is very well served by your Boeing member, Ivor Phillips. He has agreed, with your company's permission, to undertake the task of Data Base Coordinator for the public review. This is a very large contribution toward getting Fortran 8x out before the public, and carefully examining public opinion. I want to thank you and Boeing for your support.

I appreciate you continued interest in the Fortran Language, which will continue to serve the scientific community well. Thank you for your comments, as well as your support.

Sincerely yours,

Jeanne Adams

Jeanne Adams
Advanced Methods Group, NCAR
Chair, X3J3

cc. Gwendy Phillips, X3 Secretariat
cc. X3J3 Distribution

106(X) JCA-7

P. 1 of 1

(7)

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH

Scientific Computing Division/Advanced Methods Section

P. O. Box 3000 • Boulder, Colorado • 80307

Telephone: (303) 497-1275 • FTS: 320-1275 • Telex: 45 694

August 26, 1987

Prof. J. Lermouth
Salford University
Business House, University Road
Salford, M5 4pp
England

Dear Prof. Lermouth;

I received your facsimile transmission in Liverpool, and distributed it at the meeting. Miles Ellis and Mike Metcalf are researching the standards that you mentioned and promise to have them in the pre- meeting distribution for November when we will be closely examining the Kanji proposal.

We appreciate your communication about these standards, and hope that you will continue to be concerned with X3J3 work.

Sincerely yours,

Jeanne Adams
Advanced Methods Group, NCAR
Chair, X3J3

4/9

106 (*) JCA-8
P.1 of 3

P.O. Box 97 Mail 42
Beijing,
People's Republic of China

July 13, 1987

8

Jeanne C Adams
Scientific Computing Div
NCAR
P.O. Box 3000
Boulder, CO 80307

Dear Jeanne C Adams

We are very glad to know that you will discuss the problem about Kanji (or we call it as Chinese character) in X3J3 meeting number 105.

The computers are widely used as a tool of information processing in our country. It is necessary that various of character codes be processed. So Chinese character processing, sooner or later will be solved in ISO/TC 97.

In current FORTRAN 8X draft, it defines that one character in character type is represented by one character storage unit. However, one Chinses character must be represented by two or more character storage units.

We hope that a new character type --- Chinese character type be able to be added, as a new feather in FORTRAN 8X. The correspond function module can be as an optional module of FORTRAN 8X.

yours sincerely

CAS/FORTRAN
Working-group

106 (*) JCA-8
P.2 of 3

我们很高兴地得知 X3J3/105 次会议将要讨论有关汉字的问题。

计算机作为一种信息处理的工具在我国已得到了广泛的应用，需要处理各种不同的字符编码。因此，汉字处理迫切要在 ISO/TC 97 得以解决。

在当前 FORTRAN 8X 草案中，规定字符类型中一个字符由一个字符存储单元表示。然而，一个汉字却需要二个或二个以上字符存储单元。

我们希望，一种新的字符类型 —— 汉字类型能够增加，并以一种新的特点增加到 FORTRAN 8X 中。相应的功能模块则可作为 FORTRAN 8X 的任选模块。

中国标准协会 / FORTRAN 工作组

106(*) JCA-8
P. 3 of 3

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
Scientific Computing Division/Advanced Methods Section
P. O. Box 3000 • Boulder, Colorado • 80307
Telephone: (303) 497-1275 • FTS: 320-1275

August 26, 1987

CAS/FORTRAN Working Group
P. O. Box 97 Mail 42
Beijing
People's Republic of China

Dear Friends (CAS/Fortran Working Group);

Thank you very much for your letter concerning the Chinese character (sometimes called Kanji) in Fortran 8x. The X3J3 committee is studying the problem and, if possible, will try to add this facility.

There were some alternatives considered--an intrinsic module, the popular NCHARACTER implementations in Japan, and an alternate syntax, CHARACTER(KINDS=n). The latter one (KINDS=n) was the one most favored by X3J3, and a proposal to this effect will be brought to the November meeting in Ft. Lauderdale, Florida. Your letter will be placed in the pre-meeting distribution for that meeting.

I appreciated your lovely Chinese character letter, accompanying the English translation. Unfortunately I could not read the Chinese, but wished very much that I could. We welcome any of your group to our meetings. If you have any questions, feel free to contact me.

Sincerely yours,

Jeanne Adams, Advanced Methods Group
Chair, X3J3

53

106 (*) JCA-9
p. 1 of 1

9

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
Scientific Computing Division/Advanced Methods Section
P. O. Box 3000 • Boulder, Colorado • 80307
Telephone: (303) 497-1275 • FTS: 320-1275

August 26, 1987

Stanley Allen, Secretary SMC
CBEMA, X3, Information Processing Systems
311 First Street, N. W., Suite 500
Washington, D. C., 20001-2178

Dear Mr. Allen;

James Matheny has resigned as Vocabulary Representative for X3J3 to X3K5. I have appointed Kurt Hirchert to this post.. His current address is:

Kurt Hirchert
National Center for Supercomputing Applications
152 Computing Applications Bldg
605 E. Springfield Ave.
Champaign, IL 61821

Would you notify the SMC of this appointment and modify the SD-6. By this letter, I would like to thank Mr. Matheny for his contributions to this work, and state that Mr. Hirchert is a well qualified replacement.

Sincerely yours,

Jeanne Adams, Advanced Methods Group
Chair, X3J3

55

106 (*) JCA-10
P. 1 of 1

(10)

From @TUNNEL.CS.UCL.AC.UK,@CS.UCL.AC.UK:@emas-a.edinburgh.ac.uk:D.T.Muxworthy@edinburgh.ac. Wed Aug 26 03:44:08 1987 Received: from hao.UCAR.EDU (hao.ARPA) by scdpyr.UCAR.EDU (4.12/4.7) id AA18940; Wed, 26 Aug 87 03:44:05 mdt Received: by hao.UCAR.EDU (5.54/1.00.UUCP-MOD.8-11-85) id AA00939; Wed, 26 Aug 87 02:46:33 MDT Received: from tunnel.cs.ucl.ac.uk by RELAY.CS.NET id aa19253;
26 Aug 87 5:45 EDT Received: from emas-a.edinburgh.ac.uk by nss.Cs.Ucl.AC.UK via Janet with NIFTP
id aa04265; 26 Aug 87 10:34 BST Date: 26 Aug 87 10:32:42 bst From: D.T.Muxworthy%edinburgh.ac.uk@CS.UCL.AC.UK Subject: Archive To: Jeanne Adams <@CS.UCL.AC.UK,@RELAY.CS.NET:jeanne@HAO.UCAR.EDU>,
Jeanne Martin <@CS.UCL.AC.UK:martin@LLL-CRG.ARPA> Message-Id: <26 Aug 87 10:32:42 bst 140392@EMAS-A> Status: RO

Jeanne & Jeanne - The University of Manchester is setting up a "National Archive for the History of Computing" at its Centre for the History of Sciences, Technology and Medicine and recently appealed for documents. The initial collection is based on the pioneering work at Manchester in the late 1940s and early 1950s.

I have X3J3, and X3.4.3, minutes going back to 1968 (when clarification work was going on) and have been loath to discard these since they could be of use to anyone interested in the development of Fortran. However they are of little use packed in boxes in my office and I certainly don't have time currently to work on them. I have therefore offered them to the archive, and they have been accepted. What was left open was the period to be submitted. It would seem appropriate to send 1968 to 1977 or 1978, that is to distinguish Fortan 77 from Fortran 82 (as it was going to be). As these are strictly speaking internal papers I trust this will be acceptable to the committee. Ten years is far enough in the past that noone is going to be embarrassed by what appears in the minutes.

I should be interested in any comments you might have. Is there a comparable archive in the US where researchers have access to old X3.. minutes?

Best wishes, David

Accredited Standards Committee
X3, INFORMATION PROCESSING SYSTEMS*

Doc. No.: X3/87-07=130 S
Date: July 23, 1987
Project:
Ref. Doc.:
Reply to:
8457 Rushing Creek Court
Springfield, VA 22153

11

Mr. John Wood
Chair, X3K5
IBM Corporation
Communications Products Division
P. O. Box 12195, E37-656-3
Research Triangle Park, NC 27709

Dear John:

SPARC recently received copies of the X3K5 transmittals to X3J3 and X3J4 of edited copies of the Fortran and COBOL Glossaries for their review prior to being included in the ANDDIS. SPARC was concerned with certain statements in your transmittal letters since they, inadvertently I am sure, misrepresented SPARC's positions relayed in the past to the previous X3K5 Chair. Specifically:

a. In your letter to X3J4 you quoted SPARC as advising that "editing or not editing of X3 TC input was a management option that X3K5 could exercise at its option". That statement is true so far as it goes. What you failed to say was that, if X3K5 editing was done, the TC that provided the input was the final authority on the exact wording of the definition as it would appear in the ANDDIS.

b. In your letter to X3J3 you quoted SPARC as directing that the Fortran Glossary be included in the ANDDIS. This is simply not true. SPARC has never directed any TC to have their Glossary included in the ANDDIS; that decision is a TC decision.

Based on reviewing thess letters, SPARC considered the issue of ANDDIS Glossaries for Technical Committee' definitions again at its last meeting on July 7-9, 1987. SPARC reaffirmed at that time its previous position on this matter and I was asked to convey it to you in writing so that there would be no further confusion in this regard:

a. When a TC submits a Glossary for X3K5 inclusion in the ANDDIS, it is very important that X3K5 take the initiative, immediately upon receipt of the Glossary and before any "editing" is undertaken, to establish a liaison with the TC submitting the Glossary. At X3K5's request, SPARC has instructed all TC's to establish a Vocabulary Representative position to identify the individual with which that liaison should be undertaken. All

stages of X3K5 editing should be done in coordination with the Vocabulary Representative. Waiting to contact a Vocabulary Representative until after X3K5 has finished its editing is unfair to both TC's concerned and adds considerable time and effort to the process. In both cases noted above, at least 50 man hours had to be spent by the TC submitting the Glossary to determine exactly what had been done by X3K5 and whether or not it was technically acceptable.

b. In cases where agreement cannot be reached on an issue, the TC submitting the Glossary is the final authority on the exact wording (definition).

The TC Glossary as presented to X3K5 is the "standard" glossary for the subject matter area over which that TC has development authority. That Glossary will, in most cases, already be a part of an approved ANS. In some cases, the Glossary definitions are the "property" on yet another organization and are being added to the ANS according to X3 direction. In either event, X3K5 should recognize this and make sure that by "editing" a definition they are not, in fact, creating a second "standard" definition which conflicts with the first "standard" definition and that the audience for either the subject matter standard or the ANDDIS befuddled by conflicting definitions of the same subject-specific term in two different documents. That would benefit noone. SPARC recognizes that, while X3K5 editing is done with the best of intentions, minor, seemingly editorial changes can cause a significant reinterpretation of a term's meaning. This was the case in X3K5 editing of both the Fortran and COBOL Glossaries.

SPARC appreciates the significant contribution X3K5 is making to Information Systems technology by providing a valuable communications tool. SPARC fully supports these efforts and hopes that the above guidelines will aid in X3K5's future endeavors.

I hope this explanation clarifies SPARC's position on the responsibilities of X3K5 and its interaction with the TC's furnishing Glossaries for incorporation into the ANDDIS. I would be happy to discuss this with you if further clarification is necessary. In any event, I look forward to meeting you when next X3K5 presents its Annual Report and, in the interim, offer you my best wishes for a long successful tenure as the new X3K5 Chair.

Sincerely

WILLIAM C. RINEHULS
Chairman, SPARC

10 6 (*) JCA -12

12

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
Scientific Computing Division
P. O. Box 3000 • Boulder, Colorado • 80307
Telephone: (303) 497-1275 • FTS: 320-1275 • Telex: 45 694

August 18, 1987

Gabriel J. DeSalvo, Vice President
Swanson Analysis Systems, Inc.
Johnson Road
PO Box 65
Houston, Pa 15342-0065

Dear Dr. DeSalvo:

I am returning your check for $30. My organization does not distribute the proposed draft standard for Fortran. When the public review period is announced by X3, copies for distribution along with the comments from members of X3J3, the Fortran Standards Committee, will be available from:

Global Engineering Documents, Inc.
East Coast 800-248-0084
West Coast 800-854-7179
International 714-540-9870

I do not know the charge that will be made for this document, but I suspect it will be more than $30. As yet, I do not have the results of the X3 ballot on this standard and do not know when the public review process will begin.

Copies of the work of the committee are also available from CBEMA, the X3 Secretariat. These are primarily minutes of the meetings, and are available, if you join as an observer member of the Fortran Standards Committee. If you have any questions, please call me. I would be glad to help.

Sincerely Yours,

Jeanne Adams

Jeanne Adams, Advanced Methods Group
Chair, X3J3

cc. X3J3 Distribution
NCAR Accounting Department

61

X3/87/08-069-X, S

106 (*) JCA-13

13

NEW PRICE LIST (AUGUST 17, 1987)

THE FOLLOWING DOCUMENTS ARE TO BE SOLD BY THE X3 SECRETARIAT

TO ORDER: Send check (made payable to X3 Secretariat) plus self-addressed mailing label to X3 Secretariat/CBEMA, 311 First St., NW, Washington, D.C. 20001

Std. Desig.	X3 TECH. COMM.	Title	Price	PR Closes	X3 Proj. No.
CIB-23	X3J4	COBOL Information Bulletins (CIB NO. 23)	20.00	00/00/00	0021-D
FIB-1	X3J3	Fortran Information Bulletin #1 (FIB-1)	20.00		
X3.27-1987	X3B5	Magnetic Tape Labels and File Structure for Information Interchange	15.00	07/28/86	0038-M
			ANSI APPROVED - X3 SECRETARIAT WILL SELL UNTIL ANSI PRE-PUBLICATION NOTICE		
X3.134.1-198X	X3L2	8-Bit ASCII - Structure and Rules	15.00	07/22/86	0495-D
X3.134.2-198X	X3L2	7-Bit and 8-Bit ASCII Supplemental Multilingual Graphic Character Set (ASCII Multilingual Set)	15.00	07/22/86	0392-D
X3.153-1987	X3T5.5	Open Systems Interconnection - Basic Connection Oriented Session Protocol Specification	30.00	07/28/86	0333-M
			ANSI APPROVED - X3 WILL SELL UNTIL PRE-PUB NOTICE		

• • • • • • X3 TECHNICAL REPORTS • • • • • •

Std. Desig.	X3 TECH. COMM.	Title	Price	PR Closes	X3 Proj. No.
X3/TR-1-1982	X3K5	American National Dictionary for Information Processing Systems (ANDIPS)	27.00	00/00/00	0026-D
X3/TR-2-1982	X3B9	Conversion of paper Substance Weights from Ream Weights to g/m2	04.00	00/00/00	0440-M
X3/TR-3-1982	X3J7	APT Language - Expository Remarks Concerning X3.37-1980	08.50	00/00/00	0316-MT
X3/TR-4-1982	X3J7	APT Language - Postprocessor Interface Modules	08.50	00/00/00	0315-MT
X3/TR-5-1982	X3A1	Design of OCR Forms	10.00	00/00/00	0274-M
X3/TR-6-1982	X3K1	Guide for Technical Documentation of Computer Projects	10.00	00/00/00	0016-M

• •

THE FOLLOWING REAFFIRMATIONS AND/OR PROPOSED WITHDRAWALS ARE TO BE SOLD BY THE AMERICAN NATIONAL STANDARDS INSTITUTE FOR ORDERING INFORMATION: CALL 212-642-4900

Prices are subject to the following shipping and handling charges:

Value of Order
$ 5.00 to 9.99 = $ 2.00
$10.00 to 24.99 = $ 4.00
$25.00 to 49.99 = $ 5.00

Std. Desig.	X3 TECH. COMM.	Title	Price	PR Closes	X3 Proj. No.
ISO 1860-198x	X3B6	Precision Reels for Magnetic Tape Used in Interchange Instrumentation Applications	13.00	11/30/87	0390-D
ISO 7810-198x	X3B10	Identification Cards - Physical Characteristics (REF. ISO 7810)	15.00	12/14/87	0586-D
ISO 7811/1-198x	X3B10	Identification Cards - Recording Techniques - Part 1: Embossing	36.00	12/14/87	0590-D
			See also projs. 633, 634, 635, 636		
ISO 7811/2-198x	X3B10	Identification Cards - Recording Techniques - Part 2: Magnetic Stripe	20.00	12/14/87	0633-D
			See also Projects 590, 634, 635, 636		
ISO 7811/3-198x	X3B10	Identification Cards - Recording Techniques - Part 3: Location of Embossed Characters on ID Cards	13.00	12/14/87	0634-D
			See also Projects 590, 633, 635, 636		
ISO 7811/4-198x	X3B10	Identification Cards - Recording Techniques - Part 4: Location of Read-Only Magnetic Tracks - Tracks 1 & 2	13.00	12/14/87	0635-D
			See also Projects 590, 633, 634, 636		

(ANSI SALES CONTINUED)

ISO 7811/5-198x	X3B10	Identification Cards - Recording Techniques - Part 5: Location of Read-Write Magnetic Track - Track 3	15.00	12/14/87	0636-D See also Projects 590, 633, 634, 635
ISO 7812-198x	X3B10	Identification Cards - Numbering System and Registration Procedure for Issuer Identifiers	20.00	12/14/87	0595-D
ISO 7813-198x	X3B10	Identification Cards - Financial Transaction Cards	15.00	12/14/87	0609-D
X3.26-198x	X3L2	Hollerith Punched Card Code	06.00	06/20/86	0103-RF
X3.42-1975	X3T2	Representation of Numeric Values in Character Strings for Information Interchange	08.00	05/15/84	0104-RF
X3.44-198X	X3S3.5	Determination of the Performance of Data Communication Systems	10.00	05/15/86	0028-RF
X3.52-198x	X3B7	Unrecorded Single Disk Cartridge (Front Loading, 2200 BPI)	09.00	04/28/87	0224-RF BSR-9 SUBMITTED TO ANSI
X3.76-198x	X3B7	Unformatted Single-Disk Cartridge (Top Loading, 200 TPI, 4400 BPI)	09.00	04/28/87	0277-RF BSR-9 SUBMITTED TO ANSI - X3 SECRETARIAT WILL SELL UNTIL PRE-PUB NOTICE
X3.83-198x	X3L2	USA Sponsorship Procedures for ISO Registration According to ISO 2375	05.00	06/24/87	0013-RF
X3.89-198x	X3B7	Unrecorded Single-Disk, Double-Density Cartridge (Front Loading 2200 BPI, 200 TPI)	09.00	04/28/87	0275-RF BSR-9 SUBMITTED TO ANSI - X3 SECRETARIAT WILL SELL UNTIL PRE-PUB NOTICE
X3.143-198x	X3V1.8	Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML) (AKA--DIS 8879)	35.00	03/22/86	0203-D

THE FOLLOWING DOCUMENTS ARE TO BE SOLD BY GLOBAL ENGINEERING DOCUMENTS
TO ORDER: PHONE 800-854-7179 OR 714-540-9870

NOTE: Prices are higher if purchased outside of the continental U.S. The first price listed represents the cost of the document if purchased within the U.S. The second price (if applicable) represents the cost of the document & postage when purchased outside of the U.S.

CCS	X3T9.2	Common Command Set (CCS) of the Small Computer System Interface (SCSI)	25.00	N/A	-
PHIGS PASCAL	X3H3.4	Programmer's Hierarchical Interactive Graphics System (PHIGS) Pascal Binding	35.00	N/A	-
PHIGS X-PASCAL	X3H3.4	Programmer's Hierarchical Interactive Graphics System (PHIGS) Extended Pascal Binding	30.00	N/A	-
C CGI	X3H3.4	C Language Binding of the Computer Graphics Interface	35.00	00/00/00	0559-D
C PHIGS	X3H3.4	C Language Binding of the Programmers Hierarchical Interactive Graphics System (PHIGS)	35.00	00/00/00	0534-D
FORTRAN CGI	X3H3.4	Fortran Language Binding of the Computer Graphics Interface	30.00	00/00/00	0560-D
GKS-3D	X3H3.5	Three-Dimensional Extensions to GKS (Graphical Kernel System)	75.00	00/00/00	0547-I
X3.3-1976	X3S3	Synchronous Signaling Rates for Data Transmission	15.00	05/02/87	0110-R
X3.23A-198x	X3J4	Addendum to ANSI X3.23-1985, Programming Language COBOL	20.00	08/10/87	0585-D

(GLOBAL SALES CONTINUED)

X3.31-198x	X3L8	Structure for the Identification of the Counties of the U.S. for Information Interchange	10.00	09/08/87	0091-R
X3.37-1987	X3J7	Programming Language APT	55.00 ANSI APPROVED - GLOBAL WILL SELL UNTIL PRE-PUBLICATION	11/18/86	0055-M
X3.38-198x	X3L8	Identification of the States, the District of Columbia, and the Outlying Areas of the U.S. for Info. Interchange	10.00	09/08/87	0090-R
X3.47-198x	X3L8	Structure for the Identification of Named Populated Places and Related Entities of the States of the U.S.	10.00	09/08/87	0092-R
X3.74-198x	X3J1.3	PL/I General Purpose Subset	75.00	04/12/87	0296-R
X3.80-198X	X3T9.3	Interfaces Between Flexible Disk Cartridges and Their Host Controllers	20.00 26.00	12/14/87	0052-R
X3.108-198x	X3T9.5	Physical Layer Interface for Local Distributed Data Interfaces to a Non-Branching Coaxial Cable Bus	25.00	05/16/87	0337-D
X3.124.2-198x	X3H3.4	ANS for the Pascal Language Binding of the Graphical Kernel System (GKS)	35.00	05/16/87	0531-D
X3.124.3-198X	X3H3.4	ANS for Ada Language Binding of the Graphical Kernel System (GKS)	50.00	11/18/86	0529-D
X3.124.4-198x	X3H3.4	C Language Binding of the Graphical Kernel System (GKS)	35.00 59.15	10/19/87	0533-D
X3.131-198x	X3T9.2	Small Computer Systems Interface (SCSI)	75.00 97.50	03/03/86	0375-R
X3.135.1-198x	X3H2	Database Language SQL/Addendum 1 (Integrity Enhancement Feature)	25.00 32.50	01/11/88	0594-D
X3.137-198x	X3B8	3.5 Inch Flexible Disk Cartridge	25.00	01/21/87	0373-D
X3.138-198x	X3H4	Information Resource Dictionary System (IRDS)	75.00	11/25/86	0336-D
X3.144-198x	X3H3.1	Programmer's Hierarchical Interactive Graphics System (PHIGS)	75.00 97.50	09/30/87	0460-D
X3.144.1-198x	X3H3.4	ANS for the Fortran Language Binding of the Programmers Hierarchical Interactive Graphics Standard (PHIGS)	35.00	01/12/87	0532-D
X3.147-1986	X3T9	Intelligent Peripheral Interface — Logical Device Generic Command Set for Magnetic Tape	15.00 19.50	10/28/87	0505-M
X3.148-198x	X3T9.5	Fiber Distributed Data Interface (FDDI) Physical Layer	25.00	12/10/86	0379-D
X3.154-198x	X3V1.9	Keyboard Arrangement for Alphanumeric Machines	15.00	05/13/87	0424-R
X3.155-198X	X3B7	5 1/4 Inch Rigid Disk Removable Cartridge	20.00	11/18/86	0360-D
X3.156-1987	X3B7	Nominal 8 inch Rigid Media Removable Cartridge	20.00 ANSI APPROVED - GLOBAL WILL SELL UNTIL PRE-PUB NOTICE	02/24/87	0369-M
X3.157-1987	X3B5	Recorded Magnetic Tape for Information Interchange, 0.5 Inch (12.7 mm) Tape, Nine Track, 3200 CPI (126 CPMM)	15.00 ANSI APPROVED - GLOBAL WILL SELL UNTIL PRE-PUB NOTICE	02/24/87	0391-M
X3.158-1987	X3B5	Serial Recorded Magnetic Tape Cassette for Info. Interchange, 0.150 in (3.81mm) 8000 bpi (315 bpmm) Group Code Recording	20.00 ANSI APPROVED - GLOBAL WILL SELL UNTIL PRE-PUB NOTICE	03/07/87	0406-M

(GLOBAL SALES CONTINUED)

X3.159-198x	X3J11	Programming Language C	65.00	03/07/87	0381-D
X3.160-198x	X3J9	Extended Programming Language PASCAL	35.00	05/02/87	0345-D
X3.161-198x	X3H3.3	Computer Graphics Interface (CGI)	75.00	06/13/87	0346-D
X3.162-198x	X3B8	5.25 Inch High Density (130 mm) Flexible Disk Cartridge	20.00	08/09/87	0494-D
X3.163-198x	X3B7	A Contact Start/Stop Metallic Thin Film Storage Disk, 83,333 Flux Transition Per Track, 130MM Outer Dia. & 40MM Inner Dia	25.00	08/10/87	0356-D
X3.164-198x	X3B5	Unrecorded Magnetic Tape Cassette for Info. Interchange 3.81 mm (0.150 In), 252 to 394 ftpmm (6400 to 10000 ftpi)	20.00	08/10/87	0405-D
X3.165-198x	X3J12	Programming Language DIBOL	30.00	08/10/87	0507-D
X3.166-198x	X3T9	ANS for Fiber Distributed Data Interface (FDDI) Physical Layer, Medium Dependent (PMD)	80.00 99.00	12/14/87	0541-D

106 (*) JCA-14 p. 1 of 3 4

X3/87/08-089-X, S
Project 505-M

Accredited Standards Committee
X3, INFORMATION PROCESSING SYSTEMS*

NEWS RELEASE

For more information contact:
Del Shoemaker, X3T9 Chairman
202-383-5622

14

Date: August 10, 1987

X3 ANNOUNCES PUBLIC REVIEW AND COMMENT PERIOD ON REVISED TITLE, AND CHANGES TO SECTIONS 1 AND 3, OF AMERICAN NATIONAL STANDARD X3.147-1986, INTELLIGENT PERIPHERAL INTERFACE - DEVICE GENERIC COMMAND SET FOR MAGNETIC TAPE DRIVES

Washington, D. C. -- X3, the Accredited Standards Committee on Information Processing Systems, announces a two-month public review and comment period on the revised title, and changes to sections 1 and 3 of American National Standard, X3.147-1986. The public review period extends from August 28, 1987 to October 28, 1987.

Although X3.147-1986 has been approved by ANSI, final editing resulted in changes made to keep the standard consistent with its sister-standards, X3.130-1986 and X3.132-1987. These changes were deemed to require a second period of public review. X3.147-1986 facilitates the development and utilization of tape peripherals on computer systems by providing a common logical interface which permits the interconnection of intelligent tape peripherals with diverse systems. It is intended for use with the IPI Physical Interface and the IPI Device Generic Command Set for Magnetic and Optical Discs.

These changes to X3.147-1986 are available for public review and comment for a two-month period ending October 28, 1987. Copies may be obtained from GLOBAL ENGINEERING DOCUMENTS, INC. by calling 800-854-7179.

Single Copy Price: $15.00
International Orders: $19.50

# # # # #

X3/87/08-088-X, S

106(*)JCH/14 P. 2 of 3 4

Accredited Standards Committee
X3, INFORMATION PROCESSING SYSTEMS*

NEWS RELEASE

For more information contact:
Donald Deutsch, X3H2 Ch
615-371-6038

Date: August 10, 1987

X3 ANNOUNCES PUBLIC REVIEW AND COMMENT PERIOD ON DRAFT AMERICAN NATIONAL STANDARD X3.135.1-198x, DATABASE LANGUAGE SQL, ADDENDUM 1 (INTEGRITY ENHANCEMENT FEATURE)

Washington, D. C. -- X3, the Accredited Standards Committee on Information Processing Systems, announces a four-month public review and comment period on draft proposed American National Standard, X3.135.1-198x. The public review period extends from September 11, 1987 to January 11, 1988.

Database Language SQL (ANSI X3.135-1986) was approved as an American National Standard in 1986. This proposed addendum develops additional statements or clauses in Database Language SQL to specify an "integrity enhancement feature". Specifically, the integrity enhancement feature includes: referential constraints between tables, but without cascade effects, check constraints to be applied to rows of a table, and a default value for a column when a row is inserted into a table.

This draft standard is available for public review and comment for a four-month period ending January 11, 1988. Copies may be obtained from GLOBAL ENGINEERING DOCUMENTS, INC. by calling 800-854-7179.

Single Copy Price: $ 25.00
International Orders: $ 32.50

# # # # #

*Operating under the procedures of The American National Standards Institute.
X3 Secretariat: Computer and Business Equipment Manufacturers Association
311 First Street, N.W., Suite 500, Washington, DC 20001-2178

Tel: 202/737-8888
Fax: 202/638-4922

106(*) JCT-1 p. 383 4

X3/87/08-082-X, S

Accredited Standards Committee
X3, INFORMATION PROCESSING SYSTEMS*

NEWS RELEASE

For more information contact:

*** * * MEETING DATES AND LOCATIONS ANNOUNCED * * ***
FOR DATA REPRESENTATION COMMITTEE AND TASK GROUPS

DATE: AUGUST 18, 1987

X3L8, Data Representation, a Technical Committee of X3, Information Processing Systems, has announced the dates and location for the next meeting of its task groups and the full plenary session. The task groups will meet October 5 and 6, 1987, and the full plenary session will meet the following day on October 7. The meetings will be held at the Computer & Business Equipment Manufacturers Association (CBEMA) in Washington, D.C.

Two of the task groups of Committee X3L8, are involved in innovative data management activities for the purpose of producing national and international guidelines and standards regarding data classification, data attribution, and mnemonic representation of data. The task groups are also producing a draft of terminology with associated definitions which are pertinent to their work and to the national and international information processing community.

The task group on data classification and attribution (X3L8.6) is currently developing guidelines for a data element classification methodology and an attribute list for identifying and describing data elements relative to their administrative, relational and representational properties. The task group on mnemonic codes (X3L8.7) is developing an algorithm to generate mnemonic representation of data item sets associated with data elements. The activities of these two groups will provide useful tools for data management and data administration.

The full X3L8 Committee, which is composed of members from both the private and public sectors, monitors the activities and progress of these and other special interest work groups and continues to develop and approve national (ANSI) standards for the representation of selected widely used data elements.

- MORE -

*Operating under the procedures of The American National Standards Institute.
X3 Secretariat: Computer and Business Equipment Manufacturers Association
311 First Street, N.W., Suite 500, Washington, DC 20001-2178
Tel: 202/737-8888
Fax: 202/638-4922

Interested persons from government, academia, and private industry are welcome to initially attend the X3L8 Committee meeting as an observer to determine interest in membership on one or more of the task groups of X3L8.

Interested persons may contact the committee Chairman, Mr. William H. Kenworthey, Jr. at (202) 694-3361 for additional information.

106 (*) SCA-15
P. 1 of 1

FORMAL DESCRIPTION TECHNIQUES
U.S. POSITION

1. Introduction

In view of the widespread implementation of TC97 standards, and of the necessity for developing conformance tests to insure interoperability and portability, it is imperative for TC97 to use advanced tools which will aid in these efforts. One important tool is the use of Formal Description Techniques (FDTs). FDTs also can provide important benefits in the development of TC97 standards because of the nature and complexity of this work. In fact, TC97 has already made a major start in investing in these tools, with results that have been very positive and rewarding.

The U.S. believes that these techniques should be utilized within proper guidelines in order to assure that the maximum benefits are obtained. Accordingly, the following points are recommended for consideration by the Special Working Group on Formal Description Techniques.

2. Definitions

The following definitions are needed in order to clarify the distinction between formal descriptions and natural language descriptions:

2.1 A *Formal Description Technique* (FDT) is a descriptive tool that can be used in the development and specification of an international standard (or parts thereof). It may be based upon mathematical or other notation and is intended to convey information in a complete, unambiguous and rigorous manner.

2.2 *Natural Language* includes one of the languages used by ISO to publish standards, and may be supplemented with mathematical and other accepted notation, figures, etc.

3. Objectives for Use of FDTs

In general, FDTs are used to satisfy the following objectives:

3.1 unambiguous, clear and concise specifications;

3.2 a foundation for analyzing specifications for correctness, efficiency, etc.;

3.3 a basis for determining completeness of specifications;

3.4 a basis for verifications of specifications against the requirement of the standard;

3.5 a basis for determining tests for conformance of implementations to specifications;

3.6 a basis for determining consistency of specifications between standards;

3.7 a basis for implementation support.

4. Criteria for Selection of FDTs

The following criteria should be used in the selection of an FDT:

4.1 The technique should be known; i.e., there should be experience in its use.

4.2 The technique should be appropriate to the application.

4.3 A resource base should be available for using that technique.

4.4 The technique must be documented either within the standard or externally so that a user can read and understand the standard.

5. Procedures of Use of FDTs

The following procedures should govern the use of FDTs:

5.1 At this time, the development of standards should be based on conventional natural language approaches, leading to standards where the natural language text is the definitive standard. The U.S. strongly endorses the use of FDTs in the development of International Standards to supplement natural language descriptions. Furthermore, at some time in the future, formal descriptions should become a normative part of the definitive version of the standard.

5.2 The New Work Item should spell out which FDT(s) might be used in order to assure knowledge of the technique and its usefulness to liaison organizations (i.e., the set of possible FDTs should be stated but it should not be necessary to choose one at NWI time).

5.3 If more than one formal description version of the standard is to be produced, this intention should be stated in the NWI.

(5)

16

106 (*) JCA-16
October 1987
p. 1 of 1

Assignments for Distributions

Preparation and Pre-meeting Distribution Continuing Assignment beginning in 1980	Dick Hendrickson, Cray Research
Preparation and Distribution of Minutes Continuing Assignment as Secretary, 1980-1986	Jeanne Martin, LLL
Preparation of Minutes Assignment as Secretary, beginning 1987	John Reid, Harwell

Distribution of Minutes:

May 1987	Jeanne Martin, LLL
August 1987	Jeanne Martin, LLL
November 1987	Ivor Phillips, Boeing
February 1988	Bowe, Unisys
May 1988	Johnson, Prime
August 1988	Marusak, Los Alamos
November 1988	Lakhwara, Peritus
February 1989	Moss, Slac
May 1989	Phillimore, Gould
August 1989	Ragan, CDC
November 1989	Smith, Argonne
February 1990	Weaver, IBM
May 1990	Harris, DEC
August 1990	Allison, Harris
November 1990	Thompson, Concurrent
February 1991	Swift, Alliant
May 1991	Martin, Grumman
August 1991	Marshall, EG&G
November 1991	Hoover, Data General
February 1992	Gridley, Masscomp
May 1992	Burch, HP

Note: Distribution of Minutes is among the members.

X3 Subgroup Letter Ballot

(30)

106 (*) JCA-17 (24)

Accredited Standards Committee
X3, Information Processing Systems*

Doc. No. X3J3-87-104/JCA-15

(17)

Date: May 1, 1987
Project:
Ballot Period: May 10, - June 20, 1987

Subject: Database Language Embedded SQL

X3J3 is coordinating liaison to X3H2

Ballot Closes: NOON

FOR ACTION

Ref. Document: X3H2-87-32
X3H2-87-79

Statement:

X3H2 has requested that X3J3 ballot on the above referenced document as coordinating liaision to X3H2.

Question:

Do you approve X3H2-87-79 (draft proposed) Database Language Embedded SQL, March 1987, for submission to X3 for further processing as an American National Standard?

X Yes

___ No, with reasons

Mail to: Jeanne Adams
NCAR/SCD
PO Box 3000
Boulder, CO 80307-3000

NAME Valerie G. Bowe
(PLEASE PRINT)

SIGNATURE Valerie G. Bowe

DATE: 6/17/87

*Operating under the procedures of The American National Standards Institute.

NOTE: If you find that you cannot vote YES and wish to be recorded as NOT VOTING or voting NO, please state this and explain the reasons for your position in the space above or on a separate sheet.

X3 Secretariat: Computer and Business Equipment Manufacturers Association, 311 First Street, NW, Suite 500, Washington, DC 20001-2178

Tel: 202/737-8888
Fax: 202/638-4922

Unisys votes yes on the X3H2 document, Database Language Embedded SQL, with one reservation.

We are concerned by the statement on page 24 of the document that "Blanks are significant in <embedded SQL statement>s" of <embedded SQL FORTRAN program>s. This is a major inconsistency within the FORTRAN language. A FORTRAN programmer, who uses the insignificant blanks feature of FORTRAN, would find it more that a little troublesome to remember that blanks are now sometimes significant. Also, FORTRAN implementors will now have the burden of writing processors which must deal with both insignificant and significant blanks.

X3 Subgroup Letter Ballot

(30) (22)

Accredited Standards Committee
X3, Information Processing Systems*

Doc. No. X3J3-87-104/JCA-15

Date: May 1, 1987
Project:
Ballot Period: May 10, - June 20, 1987

Subject: Database Language Embedded SQL

X3J3 is coordinating liaison to X3H2

-

Ballot Closes: NOON

FOR ACTION

Ref. Document: X3H2-87-32
X3H2-87-79

Statement:

X3H2 has requested that X3J3 ballot on the above referenced document as coordinating liaision to X3H2.

Question:

Do you approve X3H2-87-79 (draft proposed) Database Language Embedded SQL, March 1987, for submission to X3 for further processing as an American National Standard?

_____ Yes

X No, with reasons (see attachment)

Mail to: Jeanne Adams
NCAR/SCD
PO Box 3000
Boulder, CO 80307-3000

NAME Kurt W. Hirchert
(PLEASE PRINT)

SIGNATURE Kurt W. Hirchert

DATE: 87/6/18

*Operating under the procedures of The American National Standards Institute.

NOTE: If you find that you cannot vote YES and wish to be recorded as NOT VOTING or voting NO, please state this and explain the reasons for your position in the space above or on a separate sheet.

X3 Secretariat: Computer and Business Equipment Manufacturers Association, 311 First Street, NW, Suite 500, Washington, DC 20001-2178

Tel: 202/737-8888
Fax: 202/638-4922

For the most part, my review of this document was limited to those sections pertaining to Fortran, as I do not have the expertise to adequately review the sections pertaining to the other languages.

My negative vote stems from the fact that the description in this document is insufficiently precise to serve as a specification for a product. In particular, I found the following "holes":

- It is unclear whether an embedded SQL statement is to be allowed as the object of a logical IF statement. If the translation of the SQL statement would be multiple Fortran statements, it would also be necessary to convert the logical IF into a block IF. Also recognizing an embedded SQL statement, in which blanks are significant, as the terminal part of a Fortran statement, in which blanks are not significant, may be too much of a burden on the translation processor.

- I believe it is necessary to specify that an embedded statement not be the terminal statement of a DO loop. If the translation of an embedded SQL statement would be multiple statements, there is no placement of the label on the translation that would be correct both for transfers of control and termination of the loop.

- Fortran does not have a concept of declaring a variable. Instead, names are specified to have particular attributes, such as type. The attribute of being a (scalar) variable is deduced from the absence of specifications or usage contrary to that hypothesis. Thus, it is necessary to specify what specifications or usage of a name are inconsistent with its being specified in an embedded SQL declaration. For example, the document should probably prohibit appearance in other type specifications, appearance as an array declarator (in a DIMENSION or COMMON statement), appearance in an EXTERNAL statement, appearance in a PARAMETER statement, definition as a statement function, and reference as a function. On the other hand, it could permit appearance in a COMMON statement, appearance in a SAVE statement (including the SAVE of a common block in which it appears), appearance in a DATA statement, and appearance in the dummy argument list of the host program unit.

- The document uses the term "compilation unit". This term is not defined here or in the Fortran standard and is ambiguous in current usage. Do you mean the minimum unit that can be separately compiled (i.e., a program unit) or that part of a program which is compiled at one time (possibly several program units)? This question has relevance in determining the effective range of an embedded exception declaration.

- The statements concerning which embedded SQL statements can have statement numbers make no sense. Note that in the Fortran standard all statements can have statement numbers. However, the statement numbers on some statements may not be referenced.

In addition to the above questions, I have a number of concerns that would not be sufficient to cause me to vote negatively on this document, but which I would like to see addressed:

- I object to the procedures that lead to X3J3 members being forced to vote as individuals on this document. At not time were we ever asked whether we, as individuals, were willing to accept this burden. Moreover, the question we have been asked is unreasonable. Few, if any, of us have the expertise to vote on this

document as a whole. The best we can be expected to do is review its implications on Fortran.

- The editorial approach in this document is poorly suited for its intended audience. Most users of embedded SQL and implementors of the translators for embedded SQL are likely to be interested only in a single host language. This intermingling of rules for 6 different languages only serves to confuse things. Ideally, the rules for the different languages should be described in separate documents or at least in separate sections of this document. As an aid to those few people who are interested in multiple host languages, identical language should be used to describe concepts and rules common to all of the languages and a mechanism such as revision bars should be used to distinguish text specific to a particular lanugage from text common to all languages.

  At the very least, syntax which is different in the different host language should be provided separately for each language in question rather than providing a description of the union of all such syntaxes!

- I have misgivings about the use of the word EXEC as the initial keyword on all embedded SQL statements. This is preempting a "useful" keyword and may eventually interfere with extension of the host language standards or extension of this standard to a host language already using the keyword (or reserved word) EXEC. In addition, EXEC is a misleading keyword when applied to declarations. I suggest the SQL EXEC would be preferable to EXEC SQL for executable statements and the SQL BEGIN DECLARE SECTION would be preferable to EXEC SQL BEGIN DECLARE SECTION for declarations. Similarly, SQL WHENEVER may be preferable to EXEC SQL WHENEVER.

- Although it may be implicit in your description of equivalent modules and host language programs, I would appreciate an explicit statement that embedded SQL declare sections may appear only where the equivalent host language declarations may appear and that executable embedded SQL statements may appear only where executable host language statements may appear.

- Failure to use appropriate host language terminology and syntax can be misleading. For example, the syntax for a Fortran statement label (not statement number) is similar to, but not identical to, the syntax for an unsigned integer constant, so that an embedded exception declaration really ought to contain the former rather than the latter.

- In clause 8, case d, item 3, reference is made to a data type CHARACTER(L). Note that Fortran has only CHARACTER*(L) or CHARACTER*_l_ (where _l_ is an unsigned integer constant).

- The statement that any valid Fortran name is a <FORTRAN host identifier> could be misinterpreted. Note that since blanks are not significant in Fortran, a name such as N CASES is valid in Fortran. I assume that a name with embedded blanks would not be valid in SQL.

- It is unfortunate that this standard does not support Fortran arrays or the Fortran types LOGICAL and COMPLEX.

- The allowed options on embedded exception declarations are inadequate. In Fortran, I can imagine wishing to use an assigned GOTO or computed GOTO. In

Ada, I would strongly desire the ability to use the language's exception handling mechanisms. It might be desirable to add a third form that would bracket arbitrary host language statements as means of providing a general mechanism.

- Although the current Fortran standard allows only 6 character variable names, there is no immediate ambiguity, but if longer variables names are permitted (as in the proposed revision of the Fortran standard and many implementations of the existing standard) the possibility exists of ordinary Fortran statements beginning with EXEC SQL. (Since we were not provided with the syntax of SQL statements, I can't discount the possibility that there may be statements that would be both valid embedded SQL statements and valid Fortran statements.) It seems incredibly short-sighted not to address this issue.

- I hope that every C reviewer has pointed out the error in saying that SQL REAL should correspond to C double. Since you say elsewhere that SQL REAL corresponds to C float, it is unlikely that anyone would be mislead, but an error as serious as this really ought to be fixed before this document is released to the public.

•

X3 Subgroup Letter Ballot

(32) 22

Accredited Standards Committee
X3, Information Processing Systems*

Doc. No. X3J3-87-104/JCA-15

Date: May 1, 1987
Project:
Ballot Period: May 10, - June 20, 1987

Subject: Database Language Embedded SQL

X3J3 is coordinating liaison to X3H2

Ballot Closes: NOON

FOR ACTION

Ref. Document: X3H2-87-32
X3H2-87-79

Statement:

X3H2 has requested that X3J3 ballot on the above referenced document as coordinating liaision to X3H2.

Question:

Do you approve X3H2-87-79 (draft proposed) Database Language Embedded SQL, March 1987, for submission to X3 for further processing as an American National Standard?

_____ Yes

X No, with reasons

1) END-EXEC is not consistent with FORTRAN keywords (hyphen not allowed).

2) Blanks are not significant in FORTRAN but are significant in embedded SQL. This will confuse FORTRAN users.

Mail to: Jeanne Adams
NCAR/SCD
PO Box 3000
Boulder, CO 80307-3000

NAME Richard Ragan
(PLEASE PRINT)

SIGNATURE Richard R. Ragan

DATE: 6/12/87

*Operating under the procedures of The American National Standards Institute.

NOTE: If you find that you cannot vote YES and wish to be recorded as NOT VOTING or voting NO, please state this and explain the reasons for your position in the space above or on a separate sheet.

X3 Secretariat: Computer and Business Equipment Manufacturers Association, 311 First Street, NW, Suite 500, Washington, DC 20001-2178

Tel: 202/737-8888
Fax: 202/638-4922

X3 Subgroup Letter Ballot

(22)

Accredited Standards Committee
X3, Information Processing Systems*

Doc. No. X3J3-87-104/JCA-15

Date: May 1, 1987
Project:
Ballot Period: May 10, - June 20, 1987

Subject: Database Language Embedded SQL

X3J3 is coordinating liaison to X3H2

-
Ballot Closes: NOON

FOR ACTION

Ref. Document: X3H2-87-32
X3H2-87-79

Statement:

X3H2 has requested that X3J3 ballot on the above referenced document as coordinating liaision to X3H2.

Question:

Do you approve X3H2-87-79 (draft proposed) Database Language Embedded SQL, March 1987, for submission to X3 for further processing as an American National Standard?

_____Yes

_____No, with reasons

✓ NOT VOTING, see ~~#~~ ~~title page~~ reverse side

Mail to: Jeanne Adams
NCAR/SCD
PO Box 3000
Boulder, CO 80307-3000

NAME Brian T. Smith
(PLEASE PRINT)

SIGNATURE Brian T. Smith

DATE: 5/13/87

*Operating under the procedures of The American National Standards Institute.

NOTE: If you find that you cannot vote YES and wish to be recorded as NOT VOTING or voting NO, please state this and explain the reasons for your position in the space above or on a separate sheet.

X3 Secretariat: Computer and Business Equipment Manufacturers Association, 311 First Street, NW, Suite 500, Washington, DC 20001-2178

Tel: 202/737-8888
Fax: 202/638-4922

Comment: I am ~~completely~~ unqualified to comment on this draft standard X3H2-87-79, Database Language Embedded SQL, March ~~8~~ 1987, and do not have the time to become sufficiently knowledgeable of the Database SQL [reference 6]. My vote of 'Yes' or 'No' would be ~~completely~~ misleading, and therefore I refuse to vote.

I have however ~~two~~ two comments to make:

1. Page 16, line 3, change 'whebn' to 'when'.

2. Page 24, points 3 and 4 are contradictory. A Fortran variable name is allowed to be X Y where there is a blank between X and Y (as blanks are insignificant in X 3.9-1978) yet according to point 3, blanks are significant in SQL so X Y is not the variable XY. How do you resolve this contradiction?

Bonnie T Smith
5/14/87

X3 Subgroup Letter Ballot

(22)

Accredited Standards Committee
X3, Information Processing Systems*

Doc. No. X3J3-87-104/JCA-15

Date: May 1, 1987
Project:
Ballot Period: May 10, - June 20, 1987

Subject: Database Language Embedded SQL

X3J3 is coordinating liaison to X3H2

Ballot Closes: NOON

Ref. Document: X3H2-87-32
X3H2-87-79

FOR ACTION

Statement:

X3H2 has requested that X3J3 ballot on the above referenced document as coordinating liaision to X3H2.

Question:

Do you approve X3H2-87-79 (draft proposed) Database Language Embedded SQL, March 1987, for submission to X3 for further processing as an American National Standard?

_____ Yes

_X_ No, with reasons attached

Mail to: Jeanne Adams
NCAR/SCD
PO Box 3000
Boulder, CO 80307-3000

NAME John Reid
(PLEASE PRINT)

SIGNATURE John K. Reid

DATE: 11 June 1987.

*Operating under the procedures of The American National Standards Institute.

NOTE: If you find that you cannot vote YES and wish to be recorded as NOT VOTING or voting NO, please state this and explain the reasons for your position in the space above or on a separate sheet.

X3 Secretariat: Computer and Business Equipment Manufacturers Association, 311 First Street, NW, Suite 500, Washington, DC 20001-2178

Tel: 202/737-8888
Fax: 202/638-4922

American National Standards are developed by the voluntary participation of all parties and with the intention and expectation that the standards will be suitable for wide application. Since their use is likewise voluntary, an affirmative vote does not commit an organization or a group represented on the committee to the use of the American National Standard under consideration.

Reasons for No vote

John Reid

An embedded SQL FORTRAN program consists of a mixture of Fortran statements and embedded SQL statements, which I take to include <embedded SQL begin declare> and <embedded SQL end declare>. It is essential that the SQL processor can distinguish between Fortran statements and embedded SQL statements. This is achieved by starting each embedded SQL statement with EXEC SQL. This is adequate for Fortran 77 because no keyword commences like this and identifiers may have at most 6 characters.

The reason for my NO vote is that I cannot be sure that this will be adequate for Fortran 8x. The current draft maintains the interpretation of blanks as insignificant and allows identifiers of up to 31 characters. The simplest fix is to state in rule 1 on page 24 that if a statement can be interpreted as either a Fortran statement or an embedded SQL statement, it is interpreted as an embedded SQL statement.

I also have the following comments:

1. In the last sentence of page 4, you need to state that each <embedded SQL begin declare> and each <embedded SQL end declare> is removed.
2. On page 15, line 4, replace 'this document' by 'Database Language SQL'.
3. On page 15, line 11, replace 'program' by 'program unit'.
4. On page 24, line 6, replace 'Syntax Rules' by 'Syntax Rules of this Section'.
5. On page 24, syntax rule 2, replace each 'statement number' by 'statement label', which is the correct Fortran term.
6. Page 24, syntax rule 2: The BNF does not appear to allow for an <embedded SQL statement> to have a label.
7. On page 25, line 1, replace 'a host variable' by 'one or more host variables'.

106 (*) JCA -18
p. 1 of 4

Accredited Standards Committee
X3, INFORMATION PROCESSING SYSTEMS*

Doc. No.: X3/87-08-078-X,S,M,I
Date: August 17, 1987
Project:
Ref. Doc.:
Reply to:

(18)

Revision # 1

CHANGE PAGES SD-10

TO: Members, X3, IAC, SPARC, SMC
Officers of X3/TC's, SC's and SPARC/SG's

SUBJECT: Transmittal of Revision 1 to X3/SD-10, X3 Subgroup Annual Report Format

The attached are change pages to the SD-10, recommended by SPARC; X3/86-2012.

All changes have been indicated with a change bar on the right hand side where applicable and are marked "(Revision 1)". Please replace the cover sheet and the nev inserts that are included.

REMOVE	INSERT
September 1985 Cover	June 1987 Cover
September 1985 Inside Cover	June 1987 Inside Cover
September 1985 Page 5	Page 5

*Operating under the procedures of The American National Standards Institute.
X3 Secretariat: Computer and Business Equipment Manufacturers Association
311 First Street, N.W., Suite 500, Washington, DC 20001-2178
Tel: 202/737-8888
Fax: 202/638-4922

87

X3/SD-10
JUNE 1987

ACCREDITED STANDARDS COMMITTEE*
X3-INFORMATION PROCESSING SYSTEMS

X3 SUBGROUP ANNUAL REPORT FORMAT

*Operating under the procedures of the American National Standards Institute

Secretariat:
Computer & Business Equipment Manufacturers Association

INSERT
LOGO

88

X3 Standing Documents

This document is one of a series, developed by X3 and the X3 Secretariat, which provides a "data base" of information on Accredited Standards Committee X3 - Information Processing Systems. Each document is updated periodically on an individual basis.

The series is intended to serve several purposes:

- To describe X3 and its program to inquirers
- To inform committee members of the organization and operation of X3
- To provide a system of orderly administration incorporating the procedures required by ANSI together with supplements approved by the X3 Secretariat, for the guidance of X3 officers, members, subgroups and the Secretariat staff.

The series of Standing Documents consists of the following:

X3/SD-0	Informational Brochure - September 1985
X3/SD-1	Master Plan - May 1987
X3/SD-2	Organization & Procedures - October 1985
X3/SD-3	Project Proposal Guide - May 1987
X3/SD-4	Projects Manual - Updated Quarterly
X3/SD-5	Standards Criteria - September 1984
X3/SD-6	Membership and Officers - Updated Quarterly
X3/SD-7	Meeting Schedule and Calendar - Updated Quarterly
X3/SD-9	Policy and Guidelines - (to be issued)
X3/SD-10	X3 Subgroup Annual Report Format -June 1987

SD-10, X3 SUBGROUP ANNUAL REPORT FORMAT

X3 subgroups are required to produce an annual report to X3 through SPARC using this format. The schedule for this report is contained in X3/SD-6, Membership and Officers.

The annual reports address each project, domestic and international, within the subgroup's Program of Work. Each project, as it is approved, has a specific target date. This report highlights the status of the project, as well as committee activities, project plans and future trends in its area of standardization.

Corrections and suggestions for improvement will be welcomed, and should be addressed to:

X3 Secretariat/CBEMA
311 First Street, NW
Suite 500
Washington, DC 20001-2178

4. **SECTION IV Anticipated Projects**

The committee should report on its future standardization plans and briefly describe probable project proposals. It is in this section that a representation of the relationship of the committee's projects would be helpful. For example, a representation such as that used in Appendix II of the SD-1 Master Plan would be appropriate.

5. **SECTION V Future Trends Section in This Technical Area**

The committee should prepare a brief analysis of future technical trends, of standardization, both domestic and international, and how this will affect its work in the future. The X3 members have expressed a particular interest in Appendix III of the X3 Master Plan entitled Future Trends. Therefore, in the interest of providing improved information to X3 and to those in the industry who are interested in what X3 is doing, and also to give positive recognition to the work of your committee, we urge you to share your thoughts on the trends you see in your area.

This Future Trends section is intended to solicit your committee's thoughts on trends in the technology with which you are working. This Future Trends section is NOT intended to deal with future plans on projects which should be described in Section IV above.

The Future Trends described here need not be limited to the range of your current projects. If you see a trend or convergence of interrelationship in the work of your committee and others in the future, this could highlight a gap were X3 should anticipate a possible need for standards via an appropriate Study Project in your committee or via a joint Study Group.

This section will be reproduce in the next addition of the Master Plan. Please view it in that context to be sure it accordingly reflects the committee's thinking.

Accredited Standards Committee
X3, INFORMATION PROCESSING SYSTEMS*

Doc. No.: X3K5/87-18
Date: August 31, 1987
Project: 0026-D
Ref. Doc.: X3/87-07-130 S
Reply to: John R. Wood
IBM Corporation
E48/656-3
P.O. Box 12195
RTP, NC 27709
(919) 254-0182

106(*) JCA-19

19

Mr. William C. Rinehuls
Chairman, SPARC
8457 Rushing Creek Court
Springfield, VA 22153

Dear Bill,

I have received a letter from Vicki Eckels, X3J4 Vocabulary Representative, indicating that the COBOL committee has decided that the COBOL glossary previously submitted to X3K5 should not be published in the next edition of the American National Standard Dictionary for Information Systems (ANSDIS). Consequently, I propose that the new edition of ANSDIS, which is in the final stages of preparation, be published as soon as possible in the form reviewed and approved by the TCs and by X3, that is, without language-related appendices. Also, as recommended by Ms. Eckels, I propose that a disclaimer, such as the following, be included in the frontmatter of ANSDIS:

> This dictionary defines generic terms that apply to more than one programming language. For definitions specific to a particular programming language, refer to the relevant ANSI Standard.

Vicki characterized the J4 position as follows: that the COBOL glossary, when removed from the context of the COBOL standard and its explanatory text, serves no purpose. I understand this position. Most of the glossaries contained in Technical Standards are written for the primary purpose of supporting those standards. They are not intended to stand-alone nor are they necessarily written for the general information processing community. It is for this reason that K5 believes it must rework such glossaries to some extent before they can be published in ANSDIS in a manner compatible with other material and in a form suitable for the intended audience. Only in this way can K5 exercise its responsibility to maintain the integrity and cohesiveness of ANSDIS.

K5 fully agrees that TCs who submit terminologies for inclusion in ANSDIS have the final word on what the definitions will be. That is why the edited versions were submitted for J3/J4 review. However, the process of developing suitable entries for ANSDIS requires the time and attention of both K5 and the submitting TC. Of course, this work can be delegated to the VR and accomplished off-line, or in conjunction with K5 meetings when the VR is able to attend.

X3K5 will continue to work with other Technical Committees to include their terminologies in future editions of ANSDIS. With regard to languages, however, my personal view is that K5 should not publish an incomplete set. To publish FORTRAN and BASIC, for example, while not publishing COBOL, would create an unbalanced document. I therefore recommend that language glossaries remain exclusively in the language standards of which they are a part. If X3J4 should change its position, we could later publish an ANDIS supplement of language terminology in which all major languages would be represented. This, of course, would require the agreement of all language committees.

Finally, I would like to clarify two issues you raised in your letter:

1. That certain statements in my transmittal letters "misrepresented SPARC's positions relayed in the past to the previous X3K5 Chair." Your letter to the Chairs of X3K5 and the X3H and X3J committees, dated June 16, 1986, instructed K5 to include definitions from other TCs "... exactly as submitted (no editorial license implied) into ANDIPS ...". However, in November, during the presentation of her annual report, the previous Chair of K5 was verbally instructed by the presiding officer of SPARC to include any language glossaries submitted by TCs "with the management option of editing them". Both the J3 and J4 committees have submitted glossaries. The previous Chair has acted in good faith to implement the SPARC policy as she and the other members of K5 understand it. (See her letter to Ms. Catherine Kachurik, dated March 25, 1987.) I believe that much of the confusion on this matter stems from the fact that the SPARC discussion was inadequately documented in the minutes of the November 1986 SPARC meeting. I therefore recommend that any future SPARC recommendations or instructions to Technical Committees be made in the form of motions, that they be voted upon, and that they be fully documented in SPARC minutes.

2. That "... waiting to contact a Vocabulary Representative until after X3K5 has finished its editing is unfair to both TCs concerned". K5 is working with the VRs of other TCs and has in fact been doing so for some time. K5 distributes agendas to Vocabulary Representatives in order to alert them to any planned activity on terminology pertaining to their areas of responsibility. Ideally, VRs should participate

with K5 when their terminologies are on the agenda, but often this is not possible. For example, the X3J4 VR had planned to participate in the July meeting of K5 but was unable to attend earlier. K5 will frequently interact with other TCs by mail; in fact, I expect this to be the rule rather than the exception. Accordingly, the K5 edits of the COBOL and FORTRAN submissions were transmitted by mail for review. In view of the fact that many VRs do not have the time or the support of their organizations to attend K5 meetings, I consider this to be a normal part of the VR process. That process will be included as part of the 1987 X3K5 Annual Report in November. Since I will be in Europe at that time, the K5 report will be presented by Mr. Helmut E. Thiess, Vice Chairman of X3K5, who attended Miss Walkowicz's presentation of the 1986 X3K5 Annual Report to SPARC.

Sincerely,

John R. Wood
Chairman, ASC X3K5

cc: Jeanne Adams, Chair ASC X3J3
Margaret Butler, SPARC/X3K5 liaison
Vicki Eckels, ASC X3J4 Vocabulary Representative (COBOL)
Catherine A. Kachurick, Director, X3 Secretariat, CBEMA
James Matheny, ASC X3J3-Vocabulary Representative (FORTRAN)
Don Schricker, Chair ASC X3J4
Helmut E. Thiess, Vice Chairman, ASC X3K5
Josephine L. Walkowicz, Member and Past Chair, ASC X3K5

Accredited Standards Committee
X3, INFORMATION PROCESSING SYSTEMS*

Doc. No.: X3J4/X-976

Date: 7 August 1987
Project:
Ref. Doc.:
Reply to:

Vicki Eckels
Texas Instruments
P.O. Box 2909
MS 2078
Austin, TX 78769
Tel: (512) 250-6512

Mr. John R. Wood
Chairman, ASC X3K5
IBM Corporation
E37/656-3
P.O. Box 12195
Research Triangle Park, NC 27709

Dear Mr. Wood:

The edited copy of the COBOL Glossary which you sent to me in May was forwarded to X3J4 members for consideration at our July meeting which took place two weeks ago. After discussing the matter at some length, the committee passed a motion against including the COBOL Glossary as an Appendix in the next edition of ANSDIS. While our committee compliments X3K5 on the work they have done with the COBOL Glossary, X3J4 believes that it serves no purpose, in either edited or verbatim form, when taken out of the context of ANSI X3.23-1985.

X3J4 respectfully requests X3K5's compliance with this decision. Furthermore, ANSDIS should include a disclaimer which states that the definitions given do not necessarily apply to COBOL.

In light of this development, I will not be attending the X3K5 meeting at the end of September.

Sincerely,

Vicki Eckels

Vicki Eckels
X3J4 Member
Vocabulary Representative to X3K5

cc: Mr. William C. Rinehuls, Chairman, SPARC
Mr. Don Schricker, Chairman, X3J4

UNITED STATES DEPARTMENT OF COMMERCE
National Bureau of Standards
Gaithersburg, Maryland 20899

March 25, 1987

Ms. Catherine A. Kachurik
Director, X3 Secretariat
CBEMA
311 First St., N.W., Suite 500
Washington, DC 20001

Dear Cathie:

Earlier this month I completed the edit and revision of the American National Standard Dictionary for Information Processing Systems (ANSDIPS) and forwarded the preliminary draft to the new Chairman of X35 for further processing as a dpANS. This represents a "grand finale" (of sorts) to my responsibilities as Chair of X3K5, though I will continue my association with the Committee for a while, as I wind down and wrap up a few X3K5 affairs.

Among the latter are recent changes in ISO vocabulary standardization policy. As recorded in our most recent Annual Report, new ISO policy calls for publication of working terminologies as technical reports, in order to enable workers in emerging technologies to speak a common language. These working terminologies would be incorporated into "standard" dictionaries only after they have reached a point of stability. This parallels X3K5 procedures for national vocabulary development, as described in our SD-3.

In the Annual Report, X3K5 recommended that X3 implementation of the new ISO policy parallel that of TC 97. As things stand now, X3 Technical Committees may submit for inclusion in ANSDIPS not working terminologies, but definitions taken verbatim from their standards. In a letter dated June 16, 1986, from the Chairman of SPARC, X3K5 was directed to include this input verbatim in ANSDIPS. I called the SPARC Chair to verify this requirement and to caution him about the inconsistencies that would result in ANSDIPS. He considered this matter and informed me that he would have no objections to the inclusion of this type of X3 Technical Committee input as Appendices to ANSDIPS.

The SPARC Chairman was absent from the meeting on the day (November 13, 1986) I presented X3K5's Annual Report. None of the SPARC members present had heard of the Chairman's letter cited in the above paragraph, nor could a copy be located at CBEMA. Discussion of the issue at the SPARC meeting concluded with a general impression that X3K5's Annual Report contained an unsubstantiated statement to the effect that all input from X3 Technical Committees must be included verbatim in ANSDIPS. In addition, the presiding officer of the SPARC meeting decided that editing or not editing of X3 TC input was a management option that X3K5 could exercise at its discretion. This might have settled the issue, were it not for the fact that none of the discussion of this matter was included in the record of the SPARC meeting of November 1986.

The new input to the revised ANSDIPS has been submitted for review by all X3 Technical Committees. During this review period the J3 and J4 Committees submitted lists of terms for inclusion in ANSDIPS <u>without</u> any editing by X3K5. Since I am no longer involved in Committee management, I will defer to X3K5's new management the choice of whichever management option serves the Committee's best interests.

With many thanks for both your public and private expressions of interest and personal support.

Sincerely,

Josephine L. Walkowicz

Josephine L. Walkowicz
Past Chair, ASC X3K5

cc: Chair, SPARC
Chair, ASC X3K5
Dr. Margot Butler, SPARC Liaison

Accredited Standards Committee
X3, INFORMATION PROCESSING SYSTEMS*

Doc. No.: X3K5/86-12

Date: 1986 June 20
Project:
Ref. Doc.:
Reply to: JLWalkowicz
NBS, ICST
Room A-265, Technology
Gaithersburg, MD 20899

ACTION REQUESTED

Memo to: X3K5 Members

Subject: Transmittal of (1) X3/86-1179-S, Definitions for the ANDIPS SPARC Chair to X3H, X3J, and X3K5 Chairs

(2) 100(13) JKR-2, X3J3 S8 Glossary

The above documents will be discussed under agenda item 7.1 of the July 16-18, 1986 meeting of X3K5. Please note that under policy set forth in reference (1) above, no review of the X3J3 Glossary will be necessary.

Josephine L. Walkowicz
Chair, X3K5

*Operating under the procedures of The American National Standards Institute.
X3 Secretariat: Computer and Business Equipment Manufacturers Association
311 First Street, N.W., Suite 500, Washington, DC 20001-2178

Tel: 202/737-8888
Fax: 202/638-4922

Accredited Standards Committee
X3, INFORMATION PROCESSING SYSTEMS*

Doc. No.: X3/86-1179-S

Date: June 16, 1986
Project:
Ref. Doc.:
Reply to:
8457 Rushing Creek Court
Springfield, VA 22153

TO: Chairs, X3H and X3J Committees
Chair, X3K5

SUBJECT: Definitions for the ANDIPS

Recently SPARC has been receiving an increasing number of complaints on the handling, or lack of handling, of language definitions for inclusion in the ANDIPS. Most of the problems raised to SPARC appear to be based on a lack of understanding by both X3K5 and the language committee of the current management policy in this regard. Since this policy has been in effect for many years and reaffirmed by SPARC as each problem arises, SPARC has directed that I provide it to you again. A lot of new people have entered the standards- making arena since this policy was last published:

> The H and J Committees may, on an "as required" basis, forward certain or all of their language definitions to X3K5 for inclusion in the ANDIPS.
>
> X3K5 will include these definitions exactly as submitted (no editorial license implied) into the ANDIPS with a notation that that particular definition of a term applies to a specific language.

For example:

> "data item
> (generic definition developed by X3K5) followed by
> (language definition supplied by X3J4) with the notation [COBOL]
> (language definition supplied by X3J3) with the notation [FORTRAN]
> (language definition supplied by X3H3) with the notation [GKS]
> etc."

SPARC recognizes that this policy will cause the ANDIPS to be different from the International dictionary since TC97 has resolved the language definition issue differently, but consistently, in the International arena.

Internationally, ISO TC97 approved the following resolution at its May 1986 meeting:

> "TC97 authorizes its SC's, where necessary, to generate two kinds of terminology which need dissemination to all P and L members of TC97 and ISO.

6/19/86

106 (*) JCA-20
P. 1 of 1
Nov. 1987 (20)

MEMO TO: X3J3

FROM: Jeanne Adams

DATE: September 1, 1987

SUBJECT: Candidate Extensions Document

Proposal: Create a standing document called "Candidate Extensions Document". The initial content of this document will be what is contained in Appendix F after the public review is concluded. Other robust, complete and adequately developed proposals of sufficient importance may be added to this document. Additions to the document would not be automatic based on committee votes on technical issues. Specific formal actions by the committee to enter any technical feature into this document would be required. Approval of this proposal constitutes a vote to place features remaining in Appendix F in this standing document.

Rationale:

1. Comments from WG5, from the X3J3 final ballots, and from the X3 negative ballots have requested development of this document.
2. It is important that the technical material that has been developed in the past several years not be "lost" in the minutes. New features like pointers, bit, and Kanji, if not included in the standard, should have a place of special recognition and not placed in the "bit bucket". This document is a good idea.
3. Since a special vote would be required to place a feature in this document, it would contain only well developed candidate features that have been approved by X3J3.
4. This document is to be viewed as a document of development for Fortran. It is not the place for worn out old features nor is it a catch-all for "removal" of features from the draft standard.

D R A F T

(To be reviewed and voted by X3J3)

RESPONSE TO DIGITAL NEGATIVE X3 BALLOT

TECHNICAL CONTENT

Array Operations

The array operations provide a way of expressing array computations without masking the inherent parallelism in such computations. This has two significant advantages:

1. more efficient code from simpler compilers on architectures providing parallel capabilities, and
2. more concise, mathematically oriented notation for application programmers.

While advances have been made in "vectorizing" sequential do-loops, such compilers are complex and are not generally able to extract all of the parallelism masked by the sequential code. Moreover, the notational properties of the Fortran 8x array operations offer significant software engineering advantages, such as shorter and clearer code that is more reliable and easier to maintain. While do-loops may still be used as an array computation notation by those who prefer to do so, X3J3 believes the significant advantages of the 8x array operations should be available to Fortran programmers.

Numerical Accuracy

The robustness and portability of numerical algorithms is of paramount importance in scientific software. The specified precision facilities of Fortran 8x will significantly improve Fortran (including current extensions) in both of these respects. Facilities have been carefully designed to provide these improvements with minimal impact on compilation and execution efficiency. Accordingly, X3J3 believes that the specified precision features are an important part of Fortran 8x.

Language Extensibility

Any language with good procedure and data structuring facilities is inherently extensible. As Fortran 8x has both, it accordingly is somewhat extensible. Both Fortran 66 and 77 have good procedural abstraction facilities, making well-designed procedure libraries an effective extensibility tool. With the addition of data structures and improved means to share data and procedures, Fortran 8x will have even better extensibility tools.

However, the Fortran 8x design has not been primarily motivated by the need for such extensibility tools. The data structures in Fortran 8x were motivated by user requirements for a simple, useful data aggregation mechanism. The MODULE/USE facilities were motivated by the need for a simple, practical way to share data, procedures, and procedure interfaces among an application's various subprograms. With these features, it will be possible to build better libraries than was possible in FORTRAN 77, even though extensibility was not the primary motivation for their inclusion. If anything, the inclusion of these features will allow the Fortran language itself to remain smaller in the future than it otherwise might be.

It is not the case that these features are either large and complex or included in the language at the expense of additional intrinsic data types. The addition of new data types will continue to be considered; the justification for their inclusion will be based on the needs of the user community.

The situation regarding the intrinsic data types you requested follows:

1. Generalized Pointers. If, at the conclusion of the public review, there is sufficient user demand for this feature, X3J3 will address this need. At this time, there is support from the ISO/WG5 for this feature, and at least three X3J3 members mentioned this in their ballots. However, as yet an adequate consensus for a pointer facility has not materialized and, a pointer facility is one that affects program efficiency more than some of the other facilities. As execution efficiency is of paramount importance in Fortran 8x, the pointer issue must be considered very carefully.

2. Bit Data Type. There is support from ISO/WG5 for this feature as well as your request. Others may request this feature during public review. X3J3 will be responsive to this demand, should the public review indicate considerable public support. This issue is being reviewed currently.

3. Varying Character. This facility can be provided quite conveniently and efficiently with an intrinsic module, for those users who require this facility.

4. Multi-byte Characters. X3J3, in cooperation with the Japanese Standards Group on Fortran, is studying multi-byte character implementations. At the Liverpool meeting, straw votes indicated that a proposal should be prepared for including a key word modifying CHARACTER type to include two byte characters. The facility proposed will allow extension to larger than two byte character sets if needed in the future. The inclusion of multi-byte characters is currently supported in an ISO/WG5 resolution, as well as the Chinese Fortran Standards Group. X3J3 will be responsive to the indicated support for this feature. However, the specific proposal is still under debate within X3J3.

Deprecated Features

Fortran 8x is based on FORTRAN 77; all of FORTRAN 77 is contained within

Fortran 8x. No feature is in the "deleted" category. The obsolescent features, in small font, are very limited and truly obsolete. The deprecated features are only candidates for the obsolescent category. The decision for continuing to examine these features for obsolescence and redundancy is with the next X3J3 committee, and user demand will certainly dictate their disposition. In fact, it is the principle of removing obsolete features that is the main goal of the architecture at issue with Fortran 8x. It provides a means for keeping the language current, and at the same time, placing limits on its size in future revisions. The Fortran 8x revision is very conservative, by having no deletions and a very small set of obsolescent features.

CONSENSUS

Every effort has been and will continue to be made to achieve consensus in a large and diverse technical committee. It is not surprising that there remain a few members who are yet dissatisified with the resulting draft. A compromise position was developed after the initial committee ballot. Compromises always involve give and take where everyone must lose some and win some. Examining a draft standard on a feature by feature basis does not provide the best solution for the language as a whole. For that reason, all features were examined together to provide the best compromise solution for the language as a whole. X3J3 will carefully examine any new features demanded in the public review to determine if the overall affect is detrimental to the language. At the same time, X3J3 will attempt to maximize the support for the inclusion or exclusion of any feature in gaining a best possible consensus.

PROCEDURAL ISSUES

X3J3 denies any procedural discrepancies at any time during the development of Fortran 8x, including processing ballot comments. X3J3 has followed the procedures described in the SD-2 carefully, and followed the guidance provided by SPARC, who determined that the draft standard is in compliance with the scope and program of work.

106 (*) JCA-21

(21)

D R A F T

(To be reviewed and voted by X3J3)

RESPONSE TO IBM NEGATIVE X3 BALLOT

STATED PRINCIPAL PROBLEMS

1. Fortran 8x is based on FORTRAN 77; all of FORTRAN 77 is contained within Fortran 8x. An audit, that began at the August 1987 meeting, is underway to make sure that every principle, feature, and rule, however small, is indeed contained within the Fortran 8x draft standard. No feature is in the "deleted" category. The obsolescent features, in small font, are very limited and truly obsolete. The deprecated features are only candidates for the obsolescent category. The decision for continuing to examine these features for obsolescence and redundancy is with the next X3J3 committee, and user demand will certainly dictate their disposition. The principle of removing obsolete features provides a means for keeping the language current, and at the same time, limiting its size in future revisions. The Fortran 8x revision is very conservative, by having no deletions and a very small set of obsolescent features.

2. The Fortran 8x draft does protect the user investment in software, since there are no deletions, and a very small set of obsolescent features. All software presently in existence will run on Fortran 8x compilers. Any obsolescent feature that continues to be needed by users will not be able to be removed in Fortran 2000, by popular demand. There has been little complaint concerning the specific features marked as obsolescent.

An Extensions Document

Some vendors, like yourself, are requesting that a standing document be provided for some of the new features that X3J3 has developed, but not yet included in the draft standard. Appendix F, since it will not appear in the final standard, is an excellent candidate for the initial standing document on candidate extensions. A proposal is underway creating a standing document, called "Fortran Candidate Extensions Document". Initially it will contain Appendix F and other features under and other features under discussion where the proposal is robust and complete, but not yet a candidate for inclusion in Fortran 8x.

Multi-byte Characters

X3J3, in cooperation with the Japanese Standards Group on Fortran, is studying multi-byte or very large character set implementations. At the Liverpool meeting, straw votes indicated that a proposal should be prepared for including a key word

modifying CHARACTER type to provide for additional character sets. The facility proposed will allow extension to arbitrary character sets, if needed in the future. The inclusion of multi-byte characters is currently supported in an ISO/WG5 resolution, as well as the Chinese Fortran Standards Group. X3J3 will be responsive to the indicated support for this feature. However, the specific proposal is still under debate within X3J3.

A Timely Fortran Standard

The contents of Fortran 8x has the support of the required number of members of X3J3. We hope that the public review will indicate that there is general support for the features that it contains. A new Fortran standard will result in the most timely manner by this route.

D R A F T

(To be reviewed and voted by X3J3)

Response to Data General

The proposed standard has been approved by SPARC to be in compliance with the project proposal.

Comments on "popular vendor extensions"

Hollerith. Hollerith was replaced by type CHARACTER in FORTRAN 77. The policy adopted by X3J3 was the inclusion of all of FORTRAN 77 in Fortran 8x. Since Hollerith was not in FORTRAN 77, it was not included in Fortran 8x.

NAMELIST. NAMELIST has been included in Fortran 8x. See the draft proposed standard, S8.104.

INCLUDE. The functionality provided by the INCLUDE facility is available in a more powerful extension to FORTRAN 77, called MODULE/USE.

BIT. BIT data type was included, but was placed in Appendix F after the first X3J3 ballot to reduce the size of the language. Appendix F serves as a repository for extensions considered favorably by X3J3 in the past. Current resolutions from WG5, the international group for Fortran, has brought this facility back to the floor. Consideration is underway based on this ISO WG5 resolution. Should public review favor addition of this facility, X3J3 will be responsive to these demands.

Byte. The byte capability has been incorporated in other mechanisms in the language such as the CHARACTER data type. A further complication arises since a byte is not the same on all systems.

In addition, X3J3, based on WG5 resolutions at their August meeting, is considering a general form of byte oriented CHARACTER data type that will include KANJI and the Chinese characters. This facility has the generality of extension to other multibyte facilities as well.

MIL-STD 1753. This standard is well known to X3J3 and has been used in considering the above topics.

New Features

The new features of Fortran 8x are long overdue. Structures and array processing facilities are already included in various Fortran extensions. Fortran 8x is a major extension of FORTRAN 77, but it is neither immense nor radical. The extensions constitute approximately one-third of Fortran 8x, and great attention has been paid to making them as Fortran-like as possible. About half of these extensions are due to the array processing facilities and the numerical computation enhancements. Both these features are greatly desired by the scientific community. The committee feels that compile-time efficiency of FORTRAN 77 codes will be minimally affected when compiled on Fortran 8x. In addition, the array features will decrease the number of Fortran statements required; some reduce many lines of FORTRAN 77 to one line of Fortran 8x. Ease of use and additional functionality will be worth the added compilation time required for Fortran 8x statements.

Language Architecture.

The architecture for Fortran will give users at least two standard revision cycles for removing a feature. Older languages, like Fortran, that have had a long and successful history, should not be left to expand beyond a reasonable number of features; nor should they be prevented from adding new modern features that make a language too large. The method suggested (deleted, obsolescent and deprecated categories) allows Fortran to grow and change and remain within the bounds of size amd complexity. Note that all of FORTRAN 77 is a subset of Fortran 8x. There are no "deleted" features; that is no FORTRAN 77 features has been removed from the draft Fortran 8x, as Hollerith was from FORTRAN 77. The architecture has been designed to allow Fortran to grow and be responsive to technological change without becoming too large. It is an important design feature of the draft standard.

DRAFT

(To be reviewed and voted by X3J3)

RESPONSE TO UNISYS NEGATIVE X3 BALLOT

Efficiency

Efficiency was an important consideration for each new feature in the draft for Fortran 8x. Performance impact on FORTRAN 77 features was considered as well. During the discussions of these new features, a workable implementation model was presented that would allow reasonable implementations. X3J3 believes that FORTRAN 77 programs will be minimally affected. As Fortran 8x compilers appear, improvements will gradually be designed that contain better ways to deal with the new features. Each new feature was judged to be very much worth any added cost in implementations, training and user support required.

Costs

There are many justifications for any added costs resulting from compiling Fortran 8x programs.

1. Extensions to FORTRAN 77 are already incurring added costs for vendors. User demands dictate these extensions.

2. Fortran 8x provides guidelines channeling these investments into areas that will benefit users the most.

3. Generally, over the past ten years, implementation costs have been declining due to availability of faster processors, new program development tools, and improved compiling techniques.

4. Implementations of Fortran 8x may be spread over time releasing progressively more robust and efficient implementations.

Memory Management

Stack storage may, if the vendor desires, only be used for temporaries, automatic variables, and recursive procedures. These costs would then only be incurred by programs that use these new features. Heap storage could be similarly addressed. Storage managment costs for all users are only the overhead costs by the system in setting up an efficient storage management system. These concepts, stack and heap storage, are usually already available in operating systems in today's market.

Training

Users are demanding these new features. Nearly all of the features are available in other high level languages, and so are not really new to users. The conservative user may use only FORTRAN 77 concepts, while the demanding user anxious to try these familiar concepts in Fortran will be easy to train. In addition, the new features need only be learned as they are needed in programming applications.

An Extensions Document

Some vendors, like yourself, are requesting that a standing document be provided for some of the new features that X3J3 has developed, but not yet included in the draft standard. Appendix F, since it will not appear in the final standard, is an excellent candidate for the initial standing document on extensions. A proposal is underway to create a standing document, called "Candidate Extensions Document" to contain Appendix F and other features under discussion where the proposal is robust and complete, but not yet a candidate for inclusion in Fortran 8x.

106 (*) JCA-21

D R A F T

(To be reviewed and voted by X3J3)

RESPONSE TO HP AFFIRMATIVE X3 BALLOT

X3J3 appreciates the support you have provided in the development of the current draft Fortran 8x standard. Your words of encouragement are greatly appreciated.

106 (*) JCA-22

Accredited Standards Committee
X3, INFORMATION PROCESSING SYSTEMS*

Doc. No.: X3/87-09-041-X,S,M,T
Date: September 9, 1987
Project:
Ref. Doc.:
Reply to:

22

TO: X3 Technical Committee Officers -- for review/action
cc: Members, X3, SPARC, SMC -- for information

SUBJECT: ANSI Notification -- Automatic Withdrawal of Standards Ten Years or Older

Effective by end of year, '87, ANSI will implement procedures to automatically withdraw any standard which is ten years or older. That is, if we have failed to revise or reaffirm by that date, action will be taken.

In the last few years, we have gotten our act together and few X3 standards fall into that category. However, there are a few. A listing of those will be issued soon.

We are sending you this advance notice so that you can schedule this on your fall agenda should you be responsible for any standard in that category.

When we send the notice out, we will instruct you further. Those who are dealing with standards close to that timeframe should start your review process as soon as possible so that you don't fall into the same trap. I understand that the five-year review rule and this type of situation adds to your administrative overhead, and we certainly all feel a lot more warm and fuzzy about current, exciting new projects than those tired old standards we dealt with in the past. However, to keep the integrity of X3 standards at the highest level possible, we'll all need to do our part, even if it is booorrriiinnnggg!

Catherine A. Kachurik
Administrative Secretary, X3

106 (*) JCA-23

Accredited Standards Committee
X3, INFORMATION PROCESSING SYSTEMS*

Doc. No.: X3/87-09-040-X,S

Date: September 11, 1987
Project:
Ref. Doc.:
Reply to:

23

Ms. Jeanne Adams
National Center for Atmospheric Res.
GCD
PO Box 3000
Boulder, CO 80307

Dear Jeanne:

Subject: FINAL FOLLOW-UP to X3 Letter Ballot 916, Approval to Forward BSR X3.9-198x, Programming Language FORTRAN, to BSR for a Four-Month Public Review and Comment Period

You will recall that this ballot originally closed with a tally of 31-4-2-1; see X3/87/08-091 and 08-091R. During the 10-day reconsideration period we received the following comment:

GUIDE: "In support of the comments made by Digital Equipment in its negative vote on Letter Ballot 916, GUIDE International, Inc. wishes to change its vote on that ballot to NO."

The final tally, therefore, shall be recorded as 30-5-2-1. The X3 Secretariat will forward X3.9-198x and the X3J3 ballot results to ANSI for a four-month public review and comment period. We are not including X3 ballot results in the public review submittal as we have not received requests to do so.

Patricia A. Stemen for

Gwendy J. Phillips
Recording Secretary, X3

29

To: X3J3

From: Walt Brainerd

Subj: Derived type definitions

Consider the example in 106(*) BPW-1 to set up a data type representing zip codes with the operation of finding the state in which the zip code is located. With no modules or derived types, it might look like this:

```
   . . .
INTEGER ZIP
CHARACTER (LEN = 2), ARRAY (0 : 99999) :: STATE
   . . .
ZIP = 87122
PRINT *, STATE (ZIP)
```

However, if one attempts to do the desirable thing of creating a derived data type in a module, there is trouble because the user of the module cannot access the integer ZIP % CODE directly if it is private, or must know that it is a structure component and use the funny percent sign if it is public. This can be avoided by making STATE a function, but it seems much more natural to me to simply make ZIP_TYPE be INTEGER, **not** a structure whose only component is an integer.

I feel that it is very important to ensure that the Fortran 88 syntax permit extensions to allow derived types that are not structures and to allow anonymous structures (i.e., allow declaration of structures without using derived types), even if we do not permit either one in Fortran 88. It has been suggested that a possible extension to allow derived types other than structures would look like the following (using the example above).

```
TYPE ZIP_TYPE : INTEGER
```

This means that a type not consisting of a structure is recognized because the type definition is contained on one logical line, whereas a type definition that is a structure appears on lines following the line containing TYPE ... . All others would have a colon and be on the same line. It seems to me to be the wrong way around to have a structure be, in effect, the default, i.e., when there is no other indication of a type declaration on the first line. Also, even though it solves the problem of allowing a reasonable extension for derived types other than structures, it does not appear to allow for an anonymous structure declaration.

To solve both of these problems, I think there should be syntax for a structure declaration and this syntax should be incorporated into the derived type definition, making the syntax for a derived type be uniform, whether defining a structure or not. A proposed syntax for a structure declaration is:

```
STRUCTURE
   [ component-def-stmt ] ...
END STRUCTURE :: name-list
```

and the syntax for a derived type might be:

```
TYPE type-name [ ( type-param-name-list ) ]: type-declaration
```

where type-declaration would include a structure declaration as well as a type declaration statement.

If the semantics of such a type definition are that an appearance of the type name simply stands for its definition, the zip code example can be done.

```
MODULE ZIP_CODE
   TYPE ZIP_TYPE : PRIVATE, INTEGER :: CODE
   CHARACTER (LEN = 2), ARRAY (0 : 99999) :: STATE
END MODULE ZIP_CODE
   . . .
USE ZIP_CODE
TYPE (ZIP_TYPE) ZIP
   . . .
ZIP = 87122
PRINT *, STATE (ZIP)
```

Proposal 1: Change the syntax (R416-418) for a derived type to:

```
derived-type-def is [ access-spec ] TYPE type-name X
                    X [ ( type-param-name-list ) ] : X
                    X type-declaration
type-declaration is structure-declaration
structure-declaration is STRUCTURE
                            [PRIVATE]
                            component-def-stmt
                            [ component-def-stmt ]
                         end-structure-stmt
end-structure-stmt is END STRUCTURE
```

End of proposal.

Proposal 1 would allow extensions of both kinds mentioned above, but would not include them in Fortran 88. The only real change is requiring STRUCTURE in the first line of the type definition. This is a very modest change to accomplish the important goals of permitting two reasonable extensions.

The following proposals indicate what could be done if proposal 1 is adopted. It is not necessary that they become a part of Fortran 88; it is important only that something like them be possible. They are not complete proposals, because the semantics are not described.

Proposal 2: Change the syntax (R416-417) of a derived type definition to the following:

derived-type-def is [access-spec] TYPE type-name X
X [(type-param-name-list)] : X
X type-declaration
type-declaration is structure-declaration
or type-declaration-stmt

End of proposal.

This proposal extends type definitions to allow types other than structures. It would permit the array implementation of zip codes shown above.

Proposal 3: Move the syntax rules for structure-declaration (proposal 1) and type-declaration (proposal 2) to the beginning of Section 5. Change line 9, page 2-2 from type-declaration-stmt to type-declaration.

End of proposal.

This permits anonymous structures (i.e., it permits variables to be declared to be structures without the use of a derived data type). This will be useful to some and perhaps unused by others, but it fills an obvious hole and encourages everyone to implement this extension using the same syntax.

25

To: X3J3

From: Walt Brainerd

Subj: Strings

As you well know, proposals to add bit strings to Fortran 88 always have made me uncomfortable. We now have additional requests to add other string types, such as Kanji and Chinese characters. I have an idea about how these things might be added without nearly as much baggage added to the language that many of us complain is already too large.

The problem is that we have been thinking of character strings, bit strings, and Kanji strings as three data types. We do not think of arrays of integers or arrays of logicals as data types; we think of integer and logical as data types and have a **data aggregation mechanism** that allows construction of arrays of almost any data type. I think we should do the same with strings. I propose that there be a **data aggregation mechanism** that allows construction of strings of various data types.

That a mechanism different from arrays is needed is evidenced by the fact that X3J3 (rightly so, I believe) has never adopted the view that a character string is the same as an array of characters; some of the operations and properties are different. Also, if a string aggregation mechanism is adopted, we can provide dynamic strings, a feature many have been requesting for a long time.

The following is a rough outline of a proposed string facility. If this appeals to X3J3, I would be happy to work on a full-blown proposal.

1. A new kind of data object, called a **string**, can be formed from elements of various data types. For Fortran 88, these data types should be character (i.e, characters of length 1), logical, bit (if added to Fortran 88), and Kanji (if added to Fortran 88). Extensions to allow integer, real, complex, structures, arrays, etc. should be fairly straightforward, but should not be added to Fortran 88 at this time.

2. A string is declared by using the keyword STRING, much like an array is declared. Examples are:

```
CHARACTER, STRING :: CS1, CS2
LOGICAL, STRING, DATA :: L1 = L"FFTF"
```

3. Fortran 77 character objects of length other than one go into the obsolete, deprecated, junk, or whatever category and all of our new stuff to try to make length declarations consistent can be dropped.

4. Character positions are indexed by the integers 1, 2, ... The length of any string varies dynamically. It is defined to be the largest position currently defined. A reference to the whole string is a reference to positions 1 through the length. Any string position referenced must be defined.

5. A possibility to aid implementation and perhaps increase efficiency would be to put a declared maximum length on each string in parentheses following the keyword STRING.

6. A string constructor consists of a sequence of constants of the appropriate data type enclosed in quotes (???single or double???) and preceded by L for logical, B for bit, K for Kanji, etc. If there are extensions to permit strings of integers and others, the items in the sequence must be delimited (???by commas???). (Incidentally, our syntax for constructing aggregates is a mess. We use "" for strings, [ ] for arrays, and the type name for structures. The last case is a particularly insidious mixing of the concepts of derived type and structure; it provides no way to construct an anonymous structure. These should all be similar.)

7. Substringing is done just as it is now with character strings. Also an individual element of a string is referenced by s(i).

8. Concatenation (//) is a string operation. Also, many of the character intrinsic functions extend naturally to any type of string.

9. Any unary or binary operations defined on the type of the string elements are extended to apply element-by-element to the string. This would apply to some intrinsic functions that return something of the right type. For this purpose, concatenation is a string operation, not an operation defined on single characters (but of course, is defined for strings of length one). Alternative 1: the strings must be the same length for a binary operation. Alternative 2: define a "pad value" for each type and each operation; an example would be that a short logical string would be considered to be extended with "T" values if an and operation is being performed.

10. The following coercions/conversions are needed:

    element to string of length 1
    logical to character
    logical string to integer???
    bit to character
    bit string to integer

11. A new format descriptor is needed for any new data type (e.g., B or K). Strings would be read and printed just as they are now for characters; the supplied length is the one defined above, rather than the declared length as it is for Fortran 77 character strings.

12. Arguments match only if both are strings of the same base data type.

If something like this is adopted, a new bit data type (if it is still deemed desirable) consists of just two values B"0" and B"1". The operators are +, -, *, and ++ (for mod 2 addition). The string mechanism takes care of all the rest. Of course, the bit data type is isomorphic to the logical data type, but it may be desirable to have the additional notation, especially when thinking of bit strings as integers. Note that bit stands for **binary digit** and that logical operators, such as **and** that are to be applied to truth values, must not be adopted for use with numeric operands, such as bits.

106(*) WSB-3

(26)

To: X3J3

From: Walt Brainerd

Subj: Miscellaneous

A few things have happened recently that may be of interest to some of the members of the committee. It seemed easier to write this than to try to talk to many of you individually.

1. I have resigned my position at UNM. I can no longer be accused of being a pointy-headed academic computer scientist. I am now just a tough minded business person (see item 4).

2. I will be a series editor for one of the largest publishing companies. The series will be based around Fortran and scientific computing. If you are thinking about writing a book, I would appreciate the opportunity to talk to you about it and tell you what some of the possibilities are. Right now, I am looking for books on graphics, scientific data base management, and numerical methods, as well as any other ideas you might have.

3. I think I will be involved editing a newsletter aimed at the Fortran programmer. I hope it will have things in it that are actually useful, like news of Fortran-related products, benchmarks, and maybe even some programs. I expect to have a little money to pay for contributions to this newsletter, so would welcome contributions here, too. If it is done subtly, you should be able to promote your expertise or that of your organization by making a contribution.

4. This is a little off the subject, but I could use your help with something else. Another person and I have developed an enhanced version of device-independent troff, the text formatting program that comes with Unix and has been used to typeset the Fortran standard in recent years. We have done this under license from AT&T and can sell our results. What I need are contacts in your organization that either acquire software for documentation (particularly if Unix is used) and the person who acquires software from outside parties (particularly for Unix systems). If you get our version, you will be able to write loops in troff!

5. Thank you for your support.

(27)

To: X3J3

FROM: Lloyd Campbell

Subject: Edits for S8.104 June 1987

The following edits are "corrections" for what seem to be minor "errors":

1. 2-1/24, delete extra "or".
2. 4-4/29, letter "O" should be a zero as in lines 27 and 32.
3. 4-8/10, add "a" after "called".
4. 5-13/32, delete ", derived type,". (see line 6)
5. 5-16/3, "ot" should be "to".
6. 7-19/17, add "real" after "double precision".
7. 8-6/14 and line 16, add "a" after "or".
8. 9-8/17, the comma should be a period after "connection".
9. 12-1/20, add "a" after "by".
10. 12-3/4, after "interface block" add "(except the ENTRY statement)".
11. 12-8/1, delete "52". (old line number)
12. 12-13/13, delete extra "the".
13. 13-18/37, add "real" after "double precision". Also in lines 39 and 41.
14. 13-23/37, delete line or add "real" after "Double precision".
15. 14-3/14, before "has a scope" add "or an INQUIRE statement".
    14-3/15, after "DATA" add "or INQUIRE". (see 14-1/18,19)
16. B-1/16, add "a" before "CONTINUE".
17. B-5/25, add space between "length" and "(".
18. C-8/45, "List directed" should have a hyphen.
19. C-20/36, "(N, N, N)" should be"(N)". (see C-21/47)
20. F-14/25, "R616" should be "R615".
21. F-23/26, "enablestmt" should be "enable-stmt".
22. F-25/9, "list" should be "lists". (see line 12)
23. H-1/34, add "a" before "CYCLE".
24. H-2/8, add "a" after "is". (see 2-7/19)
25. H-2/36, after "12.5.2.3," add "12.5.2.5,".
    H-2/37, change "or" to ", a" and add ", or an ENTRY" after "SUBROUTINE".
    Also add "a" before "statement function".
26. H-3/8, add "a" after "by". (see 9 above)
27. H-5/6, "R620, R621" should be "R619, R620".

The following edits are suggested to improve the readability, clarity, and/or consistency of the document:

28. Whole document; Single digit section numbers should have a period after the digit in headings, but other section numbers should not have an ending period. (ANSI Style Manual) Current S8 is very inconsistent and should be fixed.

29. 2-1/27 and 28, add "an" before "end-".

30. 2-1/29, add "a" before "subroutine".

31. 2-7/30, change "operations" to "uses". (disagrees with 4-2/5,6)

32. 2-8/5, change "structured" to "derived-type". (use term previously defined)

33. 2-8/33, delete comma after "elements".

34. 2-9/28, change "defines" to "specifies". (seems a more appropriate term and agrees with line 30)

35. 3-1/30, maybe angled quote should be straight vertically to give a better picture of the graphic desired. Also, the quote is vertical in the examples. Also see page 4-5, line 28.

36. 3-1/42, delete "of" after "delimiting". (or else it should go before it)

37. 3-2/39,40,41, it would be better if underscore did not go partially under adjacent characters.

38. 3-4/30, change "for" to "of".

39. 3-5/10, change "A" to "An" before "&". (see line 15)

40. 4-1/8, add "lexical" before "tokens".

41. 4-5/9, 11(twice), 12; add "literal" before "constant". (to agree with BNF)

42. 7-9/1 and 7, add "type" before "parameter".

43. 7-11/12, add comma after "B".

44. 8-1/18, change "fixed form" to "fixed source form".

45. 8-6/41, add "a" before "loop-control".

46. 8-9/29 and 8-10/3, add "The" before "label".

47. 8-10/40, add comma after "code". (see 8-11/4)

48. 9-5/7, add "the" before "scalar-".

49. 9-8/11, insert sentence "If an existing file does not have an endfile record, APPEND positions the file at its terminal point." (This case was omitted by a recent change.)

50. 9-9/3,4; replace first two sentences by "The <u>scalar-char-expr</u> has a limited list of character values." and delete the last sentence of the paragraph in line 6. (paragraph is too general)

51. 9-9/14, add "named" before "file". (see 9-5/39,40,41)

52. 9-12/44, change "an" to "a list-directed", delete "input" in line 45, and delete the sentence that starts on line 45 and the one on line 47. (9-10/41 allows NULLS= only in list-directed input statement.)

53. 9-13/4, after "values" add "for list-directed input".

54. 9-13/17, add "a" before "scalar".

55. 9-14/9, change "can" to "may".

56. 9-17/23, change "to or from" to "from or to".

57. 9-18/28, change "but" to "and". (see 5-9/34)

58. 9-19/37 and 39, add "the" before "file".

59. 9-20/4, add "the" before "scalar-". (see line 2)

60. 9-20/18, add "the" before "file".

61. 9-21/4, change "processor-defined" to "processor-dependent". (see 9-21/40)

62. 9-21/5, add "the" before "scalar-". Also in lines 9,11,16, 23,27,33,37,22.

63. 9-21/20, after "positioning" add "before its endfile record or". (see 49 above)

64. 10-1/8, replace ", or" with ". Instead of a format specifier,". (avoid implication that namelist is some form of format specifier.)

65. 10-1/13, add period after "expression".

66. 10-11/23, add comma after "blanks" and in line 25 add ", or" after "blanks".

67. 11-1/4, add "external" before "subprograms". (to agree with line 3)

68. 11-1/43, change "procedures internal to" to "internal procedures in". (avoid implication that statement functions follow CONTAINS)

69. 11-1/44, add "must" before "follow". (is user requirement)

70. 12-6/23 &24, add "an" before "array". ("a" isn't correct)

71. 12-6/35 & 36, add "if" before second "the". (readability!)

72. 12-6/44, delete "of". (should "effects of" the in line 43 be deleted?)

73. 12-8/23, add "if" after "and" at end of line.

74. 13-2/36, replace "that" with "the scope".

75. 13-4/38, change "ones" to "arrays".

76. 13-4/39, after "one" add "array".

77. 13-5/24, replace "when passing actual arguments using the keyword form" with "for keywords when using the keyword form for actual arguments".

78. 13-~~3~~/29, indent "of an ..." (indent second line of description as most others are.) Also at line 32, 13-7/13, 13-8/45)

79. 13-22/13, delete "see". Also 13-26/24.

80. 13-26/24, add "the" before "range".

81. 14-1/24, add "named" before "constructs".

82. 14-2/3, change "that" to "the". (as in line 5)

83. 14-3/35, add "totally" before "associated". (isn't true for partial association)

84. 14-4/2, delete second "Section", before "12.4".

85. 14-4/2, change "12.4" to "12.4 through 12.4.4".

86. 14-4/9, change "called" to "referenced". (include functions)

87. 14-4/24, after "renaming" add "and no conflict with another local name".

88. 14-4/29, after "6.2.6" add "through 6.2.7".

89. 14-5/41, change "and" to "or". (to agree with committee intention?)

90. 14-5/41, before "range" add "exponent". (avoid confusion with ranged arrays)

91. 14-9/22, delete "a" before "part".

92. A-5/22,23, delete sentence "A formal grammer ... document."

93. B-2/35, replace "complex" with "complicated".

94. B-3/36, replace "BLOCK DATA" in title with "Block Data" and replace other BLOCK DATA in lines 36 and 37 with "block data".

95. B-3/42 & 48, replace "MODULE" with "module".

96. B-5/2, delete "region".

97. C-3/12, replace "..." with " etc."

98. C-7/6, put title (lines 7 & 8) after "Table C.1" in line 6.

99. C-8/38, delete comma after "single".

100. C-17/22, change "for" to "of". (as used in rest of sentence)

101. C-24/29, delete line 29. (empty section)

102. F-2/10, replace "Bit" with "Bit assignment is defined in F.1.1.5.12 and bit". (Bit assignment was recently added)

103. F-20/15, change "synchronous" to "exceptional". (see line 18)

104. H-2/45, add "a" before "DO, "an" before "ENABLE", "an" before "IF", and add "a" before "WHERE".

105. H-5/23, delete hyphen in "substring-range". (see H-1/27)

106. 13-46/20 & 21, replace "that" with "the value".

107. 13-14/14, change "the value" to "a value". (as in line 26)

108. 13-20/11, delete "of".

109. 13-28/4 & 9, replace "value returned" with "result".

110. 13-34/2, add "the" before "value".

111. 13-32/35 & 43, add "the" before "maximum".

112. 13-35/10 & 18, add "the" before "minimum".

113. 13-37/33-34, replace "is otherwise .FALSE." with "otherwise has the value .FALSE.".

114. 13-39/18, change "processor-determined" to "processor-dependent".

115. 13-38/35, replace "and b" with "and the value b".

116. 13-38/16, add "the" before "true".

117. 13-46/29, add "a" before "real part".

118. 13-47/24, add "a" before "value".

119. replace "with value" with "with the value" at 13-31/37,39, & 41; also at 13-37/13,19, & 24 and at 13-46/29.

120. replace "with value" with "with a value" at 13-13/22; 13-16/15 & 40; 13-19/30; 13-21/15 & 31; 13-22/38; 13-23/22; 13-25/8,10, &32; 13-33/15; 13-35/33; 13-38/3; 13-44/8; F-18/31; and F-19/14. Also 13-21/16 & 13-25/30.

121. replace "has value" with "has a value" at 13-11/8 & 21; 13-12/13; 13-13/~~22-3~~ 30; 13-14/6; 13-15/35; 13-16/6,21, &23; 13-17/17; 13-19/35 & 39; 13-21/19,38, & 43; 13-23/1 & 5; 13-25/37 & 42; 13-26/14 & 23; 13-28/35; 13-30/31 & 41; 13-33/23,26, & 29; 13-35/41 & 44; 13-36/31; 13-38/11,14 & 18; 13-42/25 & 34; 13-43/26 & 39; 13-44/~~25~~16,19 & 23; 13-45/16 & 24; 13-25/13; F-11/25 & 37; F-12/15; F-17/16; F-18/10 & 20(2nd); and F-19/1 & 26; 13-36/2.

122. replace "has value" with "has the value" at 13-12/29 & 32; 13-13/28 & 29; 13-14/23 & 24; 13-15/28; 13-16/22; 13-20/17,20,~~22(split)~~,23 ~~24~~;13-22/1. 13-24/21; 13-26/25,34 & 35(twice); 13-27/5; 13-28/39; 13-31/26,27,29,30, 32 & 33; 13-32/16; 13-33/24 & 28; 13-34/28; 13-35/42; 13-36/1; 13-37/22; 13-38/12,16 & 35; 13-40/11 & 35; 13-41/1 & 37; 13-42/6; 13-43/4; 13-44/17 & 21; 13-45/34; 13-46/28,30 & 39; 13-47/9 & 23; 13-48/10; F-17/21-22(split).

28

TO: X3J3
FROM: DICK HENDRICKSON
SUBJECT: BIT DATA TYPE PROPOSAL

This is an attempt at unifying various BIT proposals. There are several basic uses for BITs: as a form of packed logical, as a form of control words for external equipment, and as a form of integers produced by photomultipliers, etc. It is my belief that there is a great deal of interest in and pressure for bits, both within X3J3 and in the general public. I don't believe that the present BIT in appendix F meets the needs very well. I also don't think that merely providing a packed logical is an adequate solution for processes whose fundamental nature is numerical. This proposal attempts to provide a BIT data type that is natural both for logical and numerical use. It is somewhat weak on using BITs as control words. It allows a BIT variable to be more than one bit wide and allows both logical and mathematical operations on BITs. It provides for BIT I/O. Many of the details of the BIT operations are left as processor dependent. For example, there is no requirement that BITs be stored in any particular way, one BIT per byte or word would be acceptable to the standard (but probably not to many customers). The maximum number of bits per variable is processor dependent and there is no requirement that a bit variable can span multiple words, etc. The intent is to provide a completely portable implementation of "one-bit packed logical" and allow users to write transportable code that will handle real world bit data. I said transportable, rather than portable, since some tuning of parameters for different word sizes and I/O conventions may be needed. We should recognize that some (many?) bit operations aren't really intended to be portable. I think this proposal allows a fighting chance at doing "portable" things in a portable way and expressing nonportable things in a readable manner without escaping to assembly code.

This is not a formal proposal with line numbers, but I believe that I have covered all of the important areas. If there is agreement "in principle" then a formal proposal can be produced. Throughout the proposal I have included personal notes intended to justify or explain various choices. I would like to take straw votes on most of the options described in the notes. I have also included draft notes for Appendix C so that we can explain our intent to implementers and users.

I haven't explicitly covered array processing. I don't believe anything needs to be done. Arrays of bits can be used with the same generality as arrays of anything else. We might have to special case some intrinsic functions.

PROPOSAL

Add a bit data type to Fortran 8X as follows.

DECLARATION

The form of the BIT type specification is:

BIT [([WIDTH =] integer-constant)]

If WIDTH is not specified it is if it were present with a value of 1.

CONSTRAINT: Type BIT variables cannot be used in any context that implies storage association. Not in COMMON and EQUIVALENCE nor passed as actual arguments where the width changes. If used in an IDENTIFY statement, both the alias and parent must have the same width. If one entry of a function returns a BIT value, all must return a BIT value of the same width. If a bit variable is written with an unformatted output statement it can only be subsequently read into a bit variable of the same width. Add your favorite restriction here.

NOTE: We could give a little on this one and allow BITs in COMMON. A restriction that if one variable is a BIT all must be seems reasonable (it works so well for character). I wouldn't go so far as to say they MUST be packed and redefinable the way characters are. I think we should merely say the storage association is processor dependent. END NOTE

APP. C: The intent of the bit data type is to allow for an efficient implementation of bit manipulation and allow for efficient, practical, transportable manipulation of values that are not produced by Fortran, e. g. output from photomultiplier tubes, yet not restrict processors to particular storage management schemes. There is no requirement, for example, that bits be "packed". An implementation could store one bit per byte or one bit per word. There is also no requirement that several BIT variables be packed into one word. However, it is that intent that a processor map bits "naturally" onto the hardware. A processor should, possible as an option, allow for "packed" structures which could be shared with non Fortran programs or devices. Similarly, if a processor were, for example, to adopt the variant structure of F.1 3 the bits should map naturally. For example, a structure such as:

```
TYPE  FLOATING_POINT_REP
   LOGICAL    FLAG
   SELECT CASE (FLAG)
     CASE (.TRUE.) ;      REAL  X
     CASE (.FALSE.);      BIT(1)     SIGN
                         BIT (15)   EXPONENT
                         BIT (48)  MANTISSA
    END SELECT
END  TYPE
```

should map the BIT variables onto the floating point variable in a way that would allow a code to "take apart" the floating point representation of the number. Such usage would be non-standard since it would imply that X and the BIT variables would be defined at the same time. Vendors would probably extend this by not requiring a select variable to be part of the structure.

The intent is to allow reasonably transportable coding techniques. It is expected that some uses of bits will require some tuning of variable widths, array dimensions, I/O strategy, etc. as codes are moved to machines with different word sizes, etc. END APP. C

CONSTRAINT: There is a processor dependent maximum value for integer-constant.

APP. C: The intent is to allow for BIT data up to one word wide. There is no requirement for a processor to provide multiple word bits. Processors which are not word oriented can adopt any convenient limit. END APP. C

NOTE: I don't propose (and would even oppose) allowing generic functions by allowing some form of BIT (WIDTH = *) declaration. I don't believe the expected uses of BITs are parallel to real precision. People who fiddle with bits have particular bit structures in mind, whereas people who write numeric routines have to contend with a wide variety of possible floating point precisions. I believe we can allow derived type prototypes with an unspecified width, but I believe we should restrict widths much the same way we restrict PRECISION and EXPONENT_RANGE. Also, the term "integer-constant" includes constants defined in a parameter statement (which could be USEd).

A consequence of this is that it isn't possible to write reasonable general purpose modules to deal with bits. It would be nice, for example, to write a Boolean algebra module which overloads the + and * to provide the normal boolean operations. However,

if we don't allow width=* the module writer will have to provide 64**2 versions of each routine to allow for all reasonable sizes of operands. (Of course, if we allow width=* then the compiler will have to provide 64**2 instances of each routine to allow for all reasonable sizes of operands.) Given the nature of most machines I don't believe that we can relax the rules and say that bit actual arguments match bit dummy arguments regardless of width.

Throughout this proposal I'll use things like BIT6 to mean a bit variable with WIDTH = 6 and BITn to mean a BIT variable with an arbitrary width. END NOTE

BIT CONSTANTS

The general forms of bit constants are:

```
B'bbbb'     or B"bbbb"
O'oooo'     or O"oooo"
Z'zzzz'     or Z"zzzz"
```

Where the constants are interpreted in the Binary, Octal or hexadecimal number system respectively. Any form of constant may be used in place of any other form.

APP. C: The form of the bit constants strongly suggests, but does not define, the mapping of bit data onto the real hardware. END APP. C

INQUIRY FUNCTION

Add an inquiry function.

WIDTH (BIT)

returns the declared width of the bit variable or the actual width of the bit expression.

WIDTH (NON-BIT)

We have two choices for this function, both seem equally good to me. First, restrict "NON-BIT" to be a positive integer and return the width of a bit variable needed to hold the result of BIT(NON-BIT). This seems useful for checking that integer to bit conversions will work. Second, return the size of the data TYPE, not value. This seems useful for people who want to fiddle with equivalences of bits and non-bits and do storage mapping.

CONVERSION FUNCTIONS

> NOTE: The INT function will be extended to accept BIT arguments and BIT and LOGICAL functions will be added. This will define the mapping between BIT and INTEGER or LOGICAL types. END NOTE

Add two functions:

BIT (INTEGER [,MOLD/WIDTH])
BIT (LOGICAL)
LOGICAL (BIT)

and extend the INT function to accept a BIT argument

INT(BIT).

Definitions.

LOGICAL (BIT)
is equivalent to BIT .NE. 0.

> NOTE: We could restrict this to only WIDTH = 1 arguments, there seems to be no advantage to doing so and it is somewhat irregular when compared to the rest of the proposal. END NOTE

INT (BIT)

returns a positive integer value which is an interpretation of the BIT expression according to the model for integers in 13.6.1 with r=2.

CONSTRAINT: There is a processor dependent maximum width for the BIT expression.

> APP. C: The processor dependent width should be one that allows for a reasonable set of conversions. In general this is one less than the number of bits in a word and would be "q" for the model in 13.6.1. The INT function does not return a negative result. There is no concept of "sign bits" in the BIT data type. Similarly there is no implication that the underlying integer and bit representation must be one's complement, two's complement, or even base 2. The INT function merely provides an interpretation of a BIT value as an integer, not a definition of the representations. END APP. C

A BIT variable can be thought of as a series of bits:

BwidthBwidth-1...B2B1

Bwidth is the left hand bit, b1 is the right hand bit.
These bits correspond with the Wk terms in the model in 13.6.1 as follows:

b1	w1
b2	w2
...	...
bwidth-1	wwidth-1
bwidth	wwidth

If bj is 1 then the wj is 1 in the interpretation of the value.
If width is greater than "q" the conversion is not defined.

BIT(INTEGER [,MOLD/WIDTH])

returns a bit value which is the bit interpretation of the integer value according to the model in 13.6.1 with r=2. That is if Wj is 1 then bj is also 1. INTEGER must be positive or zero.

The MOLD or WIDTH parameters control the width of the result. If neither are present it is as if width were present with the processor dependent maximum value from the BIT declarative. If either is specified the other must not be specified. If WIDTH is specified it is the width of the result. If MOLD is specified the width of mold is the width of the result.

If the INTEGER value will not fit in "width" bits the right hand bits are used and the left hand bits are truncated.

> NOTE: We could forbid the latter case, but that would require a special case for assignment. It also seems inconsistent with some character operations which automatically truncate without warning. END NOTE.

EXAMPLES:

```
INT(B'100')  = 4
INT(B'101')  = 5
LOGICAL(B'1')       = .TRUE.
LOGICAL(B'0')       = .FALSE.
LOGICAL(B'101')     = .TRUE.
BIT(4)= B'000...000100'
BIT(4,3)      = B'100'
BIT(4,WIDTH=6)    = B'000100'
BIT(4,MOLD=O'77')       = B'000100'
BIT(4,2)      = B'00'
```

NOTE: I'm not happy with the MOLD/WIDTH parameter. As I've defined it I believe it is ambiguous (or at least requires very careful definition, which is as bad). I also don't like the idea of optional, but mutually exclusive arguments. They seem to provide a convenient functionality, but I'd be happy to drop MOLD. Mold is consistent with the other conversion functions. However, BIT(i,width=64) seems somewhat easier to read than does BIT(i,mold=O'1777777777777777777'). Conversely, bit(i,mold=jj) seems easier to read than bit(i,width=width(jj)), but bit(i,width(jj)) doesn't seem so bad. END NOTE

BIT (LOGICAL)

returns B'1' if the LOGICAL expression is TRUE, returns B'0' otherwise.

NOTE: We have several options here, none of them obviously (to me) good. One would be to use a MOLD/WIDTH parameter to specify an output width. If we do that then we must decide whether to return B'1111...1111' or B'0000...0001' for the result. Both have their uses, depending on whether the result will be used in a numeric or masking computation. Personally, 0 and 1 seem more natural. There is always the IFTHEN function to provide the other result:

IFTHEN (LOGICAL, b'1111...1111',b'0000...0000')

END NOTE

BIT MANIPULATION FUNCTIONS

SHIFTL (BITn,COUNT)
SHIFTR (BITn,COUNT)
SHIFTLC (BITn,COUNT)

all perform shifting operations and all return a BIT result of width n.

CONSTRAINT: COUNT must be a non-negative integer and must be less than a processor dependent maximum, it may be at least as large as n.

The functions do "natural hardware" shifts. SHIFTL and SHIFTR shift left and right respectively, the shifts are end off with zero fill. SHIFTLC is a left circular shift.

The functions are defined as:

SHIFTx(BITn,COUNT) returns BITn if COUNT is 0.

If COUNT is 1 the Kth bit of the result is:

SHIFTL $B_k = B_{k-1}$ $2 \le k \le n$
$B_1 = 0$

SHIFTR $B_{k-1} = B_k$ $1 \le k \le n-1$
$B_n = 0$

SHIFTLC $B_k = B_{k-1}$ $2 \le k \le n$
$B_1 = B_n$

If COUNT is greater than 1 the result is SHIFTx(SHIFTx(BITn,COUNT-1),1)

NOTE: We could also have a single shift function SHIFT(BITn,COUNT) which would be a left shift if count > 0 and a right shift if count < 0. This is consistent with Mil. Spec. 1753, etc., but I personally prefer the shift left and shift right. It maps directly onto hardware and I don't believe that the shift directions are actually run time variables.

This proposal doesn't propose a substring like notation to get at some of the bits in a bit variable, rather, it relies on shifting and masking. There are 2 reasons for this. First, I believe substringing is awkward to use. It is ambiguous with bit arrays; we have to define whether ARRAY(i:j) is a subsection or a substring. We did this for characters, but lets not introduce another ambiguity. Second, I don't believe there is a natural way to substring. People who have a basically numeric interpretation of a BIT variable probably would like to number bits from right to left, starting with 0. People with a hardware control view probably like to number from left to right, with either 0 or 1 as the natural first bit number. I believe shifting, masking, and the use of BIT constants (and parameters) provide all that is needed without introducing a confusing notation.

I also don't propose a set of "GETBITS" and "PUTBITS" intrinsic functions for basically the same reasons. In addition, these functions can be trivially coded as one line arithmetic statement functions using SHIFT and .AND. functions. The general case of extracting the Ith through Jth bits is hard to code. However, I believe most real users are interested in extracting specific bits, such as bit 7, and this is very easy to do. END NOTE

ASSSIGNMENT

A bit assignment statement is of the form:

BITn = BITm

where BITm is any expression which returns a BIT result. If n=m the assignment is natural. If n > m the assignment takes place as if BITm were padded ON THE LEFT with zeros. If n < m the assignment takes place as if BITm were truncated ON THE LEFT and only the rightmost n bits are stored.

NOTE: These rules roughly parallel the character assignment rules, except the underlying interpretation is numeric. I'm not real happy with the "Little = Big" rule. I could probably support a restriction that n must be greater than or equal to m, but characters allow truncation so why shouldn't bits? END NOTE

I/O

Unformatted I/O is processor dependent. If a bit variable of a given width is written, it can only be read into a variable of the same width. The recl needed to read records with one or more bit variables is processor dependent.

NOTE: We could go further and require the same I/O list, not merely the same widths for corresponding items. This would allow "efficient", "packed" I/O of arrays without the need for lots of control structure in the record. END NOTE

APP. C: The intent of bit unformatted I/O is to allow processors the option of connecting a unit to a physical device, such as a photomultiplier tube, and performing efficient I/O on bit variables and to read files that were not prepared by a Fortran processor. There is, however, no requirement that I/O records be "packed" in a way that maps onto any particular physical I/O device. The actual devices and files and the necessary BIT variable declarations are processor dependent. END APP. C

Formatted I/O uses the B, O and Z edit descriptors to transmit BIT variables in either Binary, Octal or hexadecimal format. The edit descriptors work just like the Iw.m descriptor.

NOTE: We need to decide what forms of data are acceptable. Do we require, forbid or optional the B' ', etc., in the external forms? Personally I don't particularly care. We should be consistent with character and logical. Similarly list directed and namelist I/O need a regular set of conventions. END NOTE

The L and I edit descriptors can also be used for bit variables since there is an implicit type conversion (see below).

> NOTE: We could give this one up if necessary, but it seems so natural. There is implicit type conversion for many of the other edit descriptors. END NOTE

OPERATORS AND EXPRESSIONS

> NOTE: This proposal does not introduce any new operators except for .XOR. . It merely extends the current logical, relational and numeric operators to work on BIT variables in a natural way. I believe the BAND/BOR/BNOT operators of App. F. have little utility and are very confusing. People who want different precedences for bit and logical operations will naturally disagree with me on this point. END NOTE

EXTENSION AND CHARACTER OPERATORS

The extension and character operators are not extended to accept BIT operands.

LOGICAL OPERATORS

AND, OR, NOT, EQV and NEQV are extended to accept BIT operands. an .XOR. operator is created and extended to accept logical operands. It is the same as the NEQV operator. An expression of the form:

BITn .lop. BITm

is evaluated by performing lop on each pair of bits. For example, the result of

B'1100' .AND. B'1010'

is B'1000'.

> NOTE: We'll need to do a complete table for ch. 7, giving all of the values for all of the operators. It doesn't seem to be worth the effort for this proposal. END NOTE

The result is of type BIT and width MAX(n,m). If n .NE. m it is as if the shorter operand were extended TO THE LEFT with zero bits.

RELATIONAL OPERATORS

The relational operators are extended to accept BIT operands. An expression of the form

BITn .relop. BITm

is evaluated as if it were written:

INT (BITn) .relop. INT (BITm)

The result is of type logical.

NUMERIC OPERATORS

The numeric operators are extended to accept BIT operands. An expression of the form:

[BITn] .num op. BITm

is evaluated as if it were written:

[INT (BITn)] .num op. INT (BITm)

THE RESULT IS OF TYPE INTEGER!!! (EMPHASIS ADDED)

NOTE: We could take some heat over this one. The other choice would be to treat the result as type BIT. I believe this would be a serious mistake. I believe that people who perform numeric operations on things expect to get numbers as a result. If they want a BIT result let them use the BIT conversion function and specify the result width.

These rules have a well defined effect on parsing. An expression of the form:

BITn .lop. BITm .rel op. BITj .lop. BITk

will be evaluated as if it were written

BITn .lop. (INT(BITm) .rel op. INT(BITj)) .lop. BITk

The programmer might have wanted

(BITn .lop. BITm) .rel op. (BITj .lop. BITk)

The Appendix F BAND, etc. operators got around this by introducing a different precedence level. I think this is unnecessary. Parenthesis can force whatever precedence is

needed. BAND doesn't seem any more natural to me than does (). And it is confusing to have operators with similar names and different precedences.

The surprising case might be something like:

BITm .lop. BITn .relop. BITj

which would parse like

BITm .lop. (INT(BITn) .relop. INT(BITj))

The actual intent of the programmer is probably ambiguous. He might want the .lop. to take place first. I think App. F would have him write this case as:

BITm .Blop. BITn .relop. BITj

whereas I would have him write it as

(BITm .lop. BITn) .relop. BITj

I personally think the latter is more in line with common practice (It's been my experience that all logical expressions that were written by a physicist or engineer (not a computer scientist) and that have both an .AND. and an .OR. use parenthesis to force the precedence, even if they are redundant). END NOTE

IMPLICIT TYPE CONVERSION

NOTE: I am proposing that BIT data can be used anyplace that numeric or LOGICAL data could be used. There would be an implicit use of the BIT, INT or LOGICAL function. I think we can be just as general with bits as we are for real and integer. END NOTE

Unless a specific meaning is defined for the BIT usage, whenever a BIT expression appears in a place that demands a LOGICAL value it is as if the LOGICAL function were applied to the BIT expression. This allows the L format descriptor to be used for formatted I/O. It allows BITs to be used in WHEREs, etc., in a natural way. It does NOT allow type BIT and LOGICAL to be interchanged when used as arguments to non intrinsic procedures, even if an explicit interface is present.

Unless a specific meaning is defined for the BIT usage, whenever a BIT expression is used in a place where a numeric value is required it is as if the INT function were applied to the BIT expression. This means that an I edit descriptor can be used for BIT I/O. It does not allow BIT and INTEGER to be interchanged when used as arguments to non intrinsic procedures,

even if an explicit interface is present.

NOTE: The intent of the latter two paragraphs is to allow reasonable type conversions. We will have to look at each of the intrinsic functions and make sure that bits are appropriate. Functions like MAX and MIN are natural, SIGN isn't. I don't think there are many in the latter category. We could also choose to allow or forbid bit expressions as DO loop expressions, subscripts, data statement repeat counts, etc. on a case by case basis. This will be a good chance for the "regular language" people to interface with the "simple, useful language" people. END NOTE

The following define the interpretation of mixed Bit, numeric and Logical expressions. (I denotes a numeric variable, B a Bit variable and L a Logical variable.) (We will eventually have to define "B op I" and "I op B", for now I'll just do one case, not both.)

```
B = I  B = BIT(I,MOLD=B)
I = B  I = INT(B)
B = L  B = BIT(L,MOLD=B???)
L = B  L = LOGICAL (B)

B  .lop.  I   ILLEGAL
B  .lop.  L   LOGICAL(B) .lop.  L

B  .relop.  I INT (B)  .relop.  I
B  .relop.  L       ILLEGAL

B  num op  I      INT (B)  num op  I
B  num op  L      ILLEGAL
```

NOTE: I believe this allows BITs to be used as simple replacements for LOGICAL variables without requiring use of type conversion functions. I think it also allows BIT variables to be used naturally as unsigned integers. I don't think there are parsing problems. An expression like:

```
B .and. B .ge. I + B
```

would evaluate like:

```
LOGICAL (B) .and. ( INT(B) .ge. ( I + INT(B) ) )
```

People could use parenthesis to force an interpretation like:

```
INT ( B .and. B) .ge. (I + INT(B))
```

We might have to be careful in how we define expressions with multiple operators. Something like:

Bn .and. L .and. Bm .and. L

is INTERPRETED as

((LOGICAL(Bn) .and. L) .and. LOGICAL(Bm)) .and. L

we need to say that something like:

((Bn .and. Bm) .and. L) .and. L

is not an equivalent interpretation (whereas, it would be if the Bx variables were of type logical).

We also need to worry about expressions like:

B'1' + 1 + 1.0

a processor is allowed to evaluate this as

(B'1' + 1.0) + 1

and we need to disallow this evaluation. I think the rules for interpretation of expressions do this, since the interpretation would be:

INT(B'1') + 1 + 1.0

But we need to read chapter 7 carefully. END NOTE

29

TO: X3J3
FROM: DICK HENDRICKSON
SUBJECT: REWRITE OF SOURCE FORM SECTIONS

This is a rewrite of sections 3.2.5 and 3.3, which describe source form. I have attempted to describe what is common between the 2 source forms and yet separate the descriptions.

Proposal: Replace lines 10 thru 41 on page 3-4 and all of page 3-5 with the following.

3.2.5. Statement Labels. Statement labels provide means of refering to individual statements. Any statement not forming part of another statement may be labeled.

R324 *label* **is** *digit* [*digit* [*digit* [*digit* [*digit*]]]]

Constraint: At least one digit in label must be nonzero.

If a statement is labeled the statement must contain a non blank character. The same statement label must not be given to more than one statement in a scoping unit. Blanks and leading zeros are not significant in distinguishing between statement labels. Blanks may appear anywhere within a label. For example:

```
99999
10
 1 0
010
```

are all statement labels. The last three are equivalent.

3.3. Source Form. A Fortran program unit is an ordered sequence of one or more Fortran statements. A Fortran statement is an ordered sequence of one or more complete or partial lines. A line is an ordered sequence of zero or more characters. A line is also called a record. Lines following a program unit END statement are not part of that program unit.

Any syntax rule ending in "*-stmt*" identifies a Fortran statement.

A **character context** means characters within (between the delimiters for) a character literal constant or within a character string edit descriptor.

Except in a character context blanks are insignificant and may be used freely throughtout the program.

Any character representable in the processor may occur in a character context or in a comment.

There are two **source forms:** free and fixed. Free form and fixed form must not be mixed in the same program unit. The means for specifing the source form of a program unit are processor dependent.

3.3.1. **Free Source Form.** In free form, each source record may contain from zero to a maximum of 132 characters. There are no restrictions on where a statement may appear within a line.

3.3.1.1. **Free Form Commentary.** The character "!" initiates a **comment** except when it appears within a character context. The comment extends to the end of the source line. Lines containing only blanks or containing no characters are also comments. Comments may appear anywhere in a program unit, or precede the first statement of a program unit, and have no effect on the interpretation of the program unit.

3.3.1.2. **Free Form Statement Separation.** The character ";" separates statements, or partial statements, on a single source line except when it appears in a character context or in a comment. Statements containing only blanks or containing no characters are ignored.

3.3.1.3. **Free Form Statement Continuation.** The character "&" is used to indicate that the current statement is continued on the next line. Comment lines cannot be continued, an "&" in a comment has no effect. When used for continuation the "&" is not part of the statement.

3.3.1.3.1. **Noncharacter Context Continuation.** If an "&" is the last nonblank character on a line or the last nonblank character before an "!" the statement is continued on the next line. If the first nonblank character on the next line is an "&" the statement continues at the next character position following the "&", otherwise it continues with the first character of the next line.

3.3.1.3.2. **Character Context Continuation.** If a character context is to be continued the "&" must be the last nonblank character on the line and an "&" must be the first nonblank character on the next line and the statement continues with the next character following the "&". The "&" signifing continuation cannot be followed by commentary.

3.3.1.4. **Free Form Statements.** A label may optionally precede any statement. Note that no Fortran statement begins with a digit. A statement must not contain more than 2640 characters, including blanks and the statement label but not including commentary, blank lines, or the "&" used to indicate continuation.

3.3.2. **Fixed Source Form.** In fixed source form each line must contain exactly 72 characters and there are restrictions on where a statement may appear within a line.

3.3.2.1. **Fixed Form Commentary.** The character "!" initiates a **comment** except when it appears within a character context or in character position 6. The comment extends to the end of the line. Lines

begining with a "C" or "*" in character position 1 and lines containing only blanks are also comments. Comments may appear anywhere within a program unit, or precede the first statement of the program unit, and have no effect on the interpretation of the program unit.

3.3.2.2. Fixed Form Statement Separation. The character ";" separates statements, or partial statements, on a single source line except when it appears in a character context or in a comment. Statements containing only blanks or containing no characters are ignored.

3.3.2.3 Fixed Form Statement Continuation. Except within commentary character position 6 is used to indicate continuation. If character position 6 contains a blank or a 0, this line is the initial line of a new statement which begins in character position 7. If character position 6 contains some character other than blank or zero, including "!" or ";", character positions 7-72 of this line constitute a continuation of the preceding line. Comment lines cannot be continued. Comment lines can occur within a continued statement.

3.3.2.4 Fixed Form Statements. A label, if present, must occur in character positions 1 thru 5 of the first line of a statement. Statements which begin after a ";" separator cannot be labeled. Character positions 1 thru 5 of any continuation lines must be blank. A statement must not have more than 19 continuation lines. The program unit END statement must not be continued and no other statement in the program unit may have an initial line that appears to be a program unit END statement.

30

TO: X3J3
FROM: DICK HENDRICKSON
SUBJECT: EDITORIAL CHANGES

The following changes to /S8.104 are proposed as a result of the audit of FORTRAN 77 chapters 1,2 and 3.

1) Page 1-2, line 20, add: "in the same scoping unit" after "procedure". This is needed to match F77, page 1-2, line 36.

2) Page 1-3, after line 13 add a new (4) and renumber.

"(4) Blanks are used to improve readability, but unless otherwise noted have no significance."

This is lines 51 and 52 from F77, page 1-3.

3) Page 1-3, line 20. Delete "informal"

4) Page 2-8, line 13. After "object" add: "which has a type and ". Needed to match F77, page 2-3, line 8.

5) Copy the constraint on page 2-1, line 26 thru 29 to after line 18 on page 12-9 and to after line 26 on page 12-10.

31

TO: X3J3
FROM: DICK HENDRICKSON
SUBJECT: RESPONSE TO LIVERPOOL RESOLUTION 17

Liverpool resolution 17 states:

> That WG5 requests that X3J3 investigate the possibility that a standard conforming processor should be capable of detecting and reporting violation of its processor dependent limits on program size and complexity.

Subgroup 14/18 has discussed this resolution and proposes:

Add after "program" on line 4, page 1-2:

> "and contains the capability to detect and report any violation of limits the processor imposes on the numbered syntax rules and their associated constraints"

This seems to be in the spirit of the other conforming restrictions and doesn't require detection of run-time errors at compile time.

ISO/TC 97/SC 22/WG 12 N133 Rev 1

4 September 1987

DRAFT FOR ITEM 97.22.15.01

GUIDELINES FOR THE PREPARATION OF CONFORMITY CLAUSES IN PROGRAMMING LANGUAGE STANDARDS

150

TABLE OF CONTENTS

0. INTRODUCTION

Conformity clauses are included within the language standard to aid the user of the standard in assessing conformity of processors and programs for adherence to the language standard. If conformity requirements are imprecise, testing for compliance can be difficult, and potentially impossible for large portions of the language standard. Therefore, these guidelines seek to encourage the inclusion of conformity clauses in programming language standards, and recommend that the language standard precisely identify the criteria that must be met in order that a valid claim may be made that a processor or program conforms to the language standard.

1. SCOPE

Recognising the dissimilarity of various language standards, the objective of this document is to provide guidelines for the preparation of conformity clauses for processors and conformity clauses for programs in language standards, together with an annex containing a checklist to aid this preparation. It was not considered practical to provide model statements that would be suitable for inclusion in all language standards. Therefore, examples have been given to illustrate the type of issues that should be addressed and it is anticipated that these will be adapted, where appropriate, for inclusion in a particular language standard.

It should be borne in mind when reading this document that not all concepts will be applicable to all languages. For example, not all language standards contain subsets or permit extensions, and elements that are fully specified by one language standard may be dependent on the processor in another.

2. DEFINITIONS

For the purpose of this document the following definitions apply:

2.1 CONFIGURATION : host and target computers, any operating system(s) and software used to operate a language processor.

2.2 CONFORMING PROCESSOR : a processor which processes programs that are in the language defined by the language standard, and which obeys all the conformity clauses for processors in the language standard.

2.3 CONFORMING PROGRAM : a program which is written in the language defined by the language standard and which obeys all the conformity clauses for programs in the language standard.

2.4 CONFORMITY CLAUSE : a statement that is not part of the language definition but specifies requirements for compliance with the language standard.

2.5 DEPRECATED LANGUAGE ELEMENT : an element in the language standard which will be deleted from the next revision of the language standard.

2.6 ERROR : an incorrect program construct or incorrect program functioning, as defined by the language standard.

2.7 EXTENSION : a facility in the implemented language that is not given in the language standard but that does not cause any ambiguity or contradiction when added to the language standard (although, in some languages, it may serve to lift a restriction).

2.8 IMPLEMENTATION DEFINED : dependent on the processor, but required by the language standard to be defined and documented by the implementer.

2.9 PROCESSOR : a compiler, translator or interpreter, working in combination with a configuration.

2.10 SUBSET : a subset S of programming language L is a programming language such that every program in S
- is also a program in L and
- has the same meaning in S as it has in L.

3. THE GUIDELINES

The technical terms and meanings used in describing conformity clauses should be the same as those defined for describing the technical specifications in the language standard. When terms are used that are not defined in the language standard the terms and definitions used in ISO 2382 - Data Processing - Vocabulary, should be used.

If the language standard does not fully define a feature of the language then the effect of attempting to use such a language feature may be unpredictable. Therefore, these guidelines recommend that, wherever possible, the standard should identify these areas and require an implementor to document the action to be taken by a processor. (Annex A contains a list of such possible features.)

3.1 REQUIREMENTS OF A CONFORMING PROCESSOR

The language standard should specify the rules for a conforming processor, possibly including one or more of the following examples :

> "A conforming processor is one that correctly translates and executes conforming programs."
>
> "A conforming processor is one that rejects all program units that contain errors whose detection is required by the standard."
>
> "A conforming processor is one that contains no variation except where the language standard permits, and specifies all such permitted variations in the manner prescribed by the language standard."
>
> "A conforming processor is one that does not allow inclusion of substitute or additional language elements in order to accomplish a feature of the language as specified in the language standard."

3.1.1 Documentation

The technical specifications of the language standard may require a conforming processor to document its handling of certain features of the language. The language standard should also require that a conforming processor includes the following in its accompanying documentation:

- a list of all definitions or values for the implementation defined features in the language standard;

- a list of all the features of the language standard which are dependent on the processor and not implemented by this processor due to non-support of a particular facility;

- a list of all the features of the language implemented by this processor which are extensions to the standard language;

- a statement of conformity, giving the complete reference of the language standard with which conformity is claimed, and, if appropriate, the subset of the language supported by this processor.

There should also be a requirement of a conforming processor about the claims made in its documentation, e.g. :

> "A conforming processor shall meet the technical specification in its accompanying documentation when related to the requirements of this language standard."

3.1.2 Processor Dependencies

The language standard should specify the criteria for determining conformity with regard to facilities which depend on the processor. Those cases in the language standard which pertain to specific facilities which depend on the processor should be identified, where known, in the language standard.

Separate conformity criteria should be specified where processor facilities which depend on the processor are available, as well as the case where the processor does not have a facility for supporting particular features of the language which depend on the processor, e.g. :

> "A conforming processor shall identify in its accompanying documentation the features of the language standard, supported by this processor, which depend on the processor."
>
> "Language elements that pertain to specific facilities that are dependent on the processor and for which support is not claimed need not be implemented. The absence of such facilities and pertaining language elements from an implementation must be documented."

Where the language standard imposes no limits at all upon an implementation defined value, the writer of portable programs must make certain assumptions about the values that are likely to be supported by all language processors, e.g., a minimum high value or a maximum low value. Similarly, the ability to test for conformity with the language standard may depend upon the tester making such assumptions when defining his tests. For the purpose of determining processor conformity, consideration should be given to the specification of reasonable limits for implementation defined elements. These may take the form of recommendations, with a requirement that where they are not met this should be recorded in the documentation accompanying the processor.

3.1.3 Errors

The language standard should specify how each error, or type of error, is to be treated by a conforming processor, such as:

- there shall be a statement in an accompanying document that the error is not reported;

- the processor shall report the error during preparation of the program for execution;

- the processor shall report the error during execution of the program, and continue execution;

- the processor shall report the error during the execution of the program, and terminate execution.

The language standard should, where appropriate and practical, describe the recovery action to be taken by the processor on detection of an error.

3.1.4 Extensions to the Language

The language standard should address the question of extensions. If they are to be permitted the language standard should require that such extensions be clearly described within the documentation accompanying the processor. In order that extensions do not restrict program portability the language standard may make some requirement about their implementation, e.g. :

> "A conforming processor shall offer a facility to detect or 'flag' the use of an extension which is statically determinable solely from inspection of a program statement, without execution."

> "A conforming processor shall offer a facility to reject the use of an extension in a program."

Extensions to the language could imply extra reserved words, and the language standard may impose a restriction on the form of these words, e.g. :

> "Any words that are defined as reserved for a particular processor and are in addition to those defined as reserved in this standard shall be outside the range of identifiers permitted in a program, thus ensuring that a conforming program will still be translated by a conforming processor in the manner prescribed by the language standard."

If conformity may be claimed with a subset of the language, the language standard should address the question of extensions to the subset, e.g. :

> "Extensions to the subset shall not conflict with the requirements of the full language standard."

3.1.5 Subsets of the Language

Where subsets of the language are permitted within the language standard, the language standard should specify the rules for conformity to a subset, e.g. :

> "A conforming processor shall fully support all the language elements of the subset with which conformity is claimed."

The language standard should specify whether or not a conforming processor is permitted to support language elements of a higher subset, and may require that every processor conforming to a subset of the language provide a facility to detect or "flag" the use of language elements outside that subset, i.e. treat them as extensions.

3.1.6 Deprecated Language Elements

Where appropriate, the language standard should address the use of deprecated language elements, e.g. :

> "A conforming processor shall provide a warning mechanism which may be invoked to indicate the use of a deprecated element, and all such deprecated language elements shall be identified in the documentation accompanying the processor."

3.2 REQUIREMENTS OF A CONFORMING PROGRAM

The language standard should specify the rules for a conforming program, e.g. :

> "A conforming program shall not use any forms or relationships that are prohibited by the language standard."
>
> "A conforming program shall not depend on extensions included in a language processor."
>
> "A conforming program shall not use any extensions included in a language processor."
>
> "A program claiming conformity to the subset of a language shall not use any facility which is outside that subset."

The language standard should warn that use of deprecated language elements in a program could cause that program to be non-conforming with future revisions of the language standard.

ANNEX A

(This Annex is part of the Guidelines.)

CHECKLIST OF POTENTIAL LANGUAGE FEATURES THAT MAY DEPEND ON PROCESSOR FACILITIES

The following list describes features that may be dependent on the processor. The list is not to be considered either as exhaustive or as prescriptive.

Where an item in the list has been take directly from a particular language standard that language is indicated by the item, other items are generalisations drawn from one or more language standards.

A.1	the size or complexity of a program and its data that will exceed the capability of the processor of a particular configuration.	Ada FORTRAN
A.2	the results of attempting to process or run a program where that program or its data will exceed the capability of the processor of a particular configuration.	
A.3	the maximum or minimum allowable number of statements in a program	
A.4	the maximum or minimum allowable number of operands in a program	
A.5	the maximum or minimum length in characters of a line of source text	
A.6	the maximum or minimum number of files permitted in a particular program run	
A.7	the maximum number of files which can be opened simultaneously	
A.8	the representation of external identifiers and titles	
A.9	the character set	PL/I
A.10	the collating sequence	COBOL & PL/I
A.11	graphic representations	PL/I
A.12	symbol names	PL/I
A.13	the limit on the number of unique statement labels	

A.14	the maximum or minimum length of identifiers	
A.15	the maximum or minimum length of character strings	
A.16	the default values for uninitialised variables	
A.17	the maximum or minimum value of integers	
A.18	the internal precision of real numbers	
A.19	the maximum or minimum value of real numbers	
A.20	the method of rounding or truncating numeric results	
A.21	the results of numeric conversions	PL/I
A.22	the representation of fixed or floating point numbers where it is necessary to explain the results of operations upon them	Ada
A.23	the maximum or minimum length of a record	PL/I
A.24	the maximum or minimum length of keys in keyed files	PL/I
A.25	the maximum allowable number of dimensions in an array and the maximum allowable number of array elements	
A.26	the maximum depth of nesting	
A.27	the maximum levels of recursion	
A.28	the order of evaluation of variables or expressions or operands or parameters where an implementor may vary the order or impose his own order	
A.29	the way in which non-valid programs are rejected and how this is reported	
A.30	the reporting of errors or exceptions where more than one error condition or exception could arise	COBOL
A.31	the system action following certain error conditions or exceptions.	COBOL

32

TO: X3J3
FROM: DICK HENDRICKSON
SUBJECT: QUESTIONS ABOUT S8

Some general questions about the contents of S8 that I couldn't figure out the answers to.

1) What does SAVE or DATA mean in a generic module procedure?

```
MODULE  SAVE IT
. . .
SUBROUTINE X(Y)
REAL (*,*) :: Y
SAVE Z
DATA Q /1.0/
. . .
END
END MODULE
```

Since there will probably be different instances of subroutine X for each precision supported are there different instances of Z and Q or is there a static allocation which is shared by the different instances? Does it matter if Z or Q are tied to the precision of Y (it would seem to make sharing difficult)?

2) Same question, but for an internal procedure which is contained inside of a recursive procedure. Does each new instance of the host create a new instance of the internal procedure? If so, does it create new instances of saved variables or does the internal procedure do some sort of static allocation and share the variables?

3) Do modules work with arrays? Suppose we wish to define some sort of operator, do we need to define functions for each possible combination of array and scalar operands, including 7 sets of array dimensions? I think the elemental function stuff covers this case. But, what about assignment? There is no elemental subroutine call. To overload A = B don't I have to provide 15 different routines?

4) Is there a better way to write generic functions than:

```
FUNCTION F(X)
REAL (*,*) :: X
REAL (EFFECTIVE_PRECISION(X),EFFECTIVE_EXPONENT_  &
                        RANGE(X) ):: F
```

This seems awful verbose for a common use.

166(#) M44-6 (33)

To: X3J3
From: Dick Hendrickson
Subject: Computer Recreation

Here's a little FORTRAN program, is it standard conforming?

```
      COMMON M
      DO 10 I = IX(1),IX(2),IX(3)
10    CONTINUE
      PRINT *, M
      END

      FUNCTION IX(K)
      COMMON M
      M = K
      IX = K
      PRINT *, K
      END
```

Suppose in line 2 the "IX(2),IX(3)" were replaced by "IX(1),IX(1)"?

In either case, what should be printed?

34

To: X3J3 106(*) TAH-1
From: Tracy Hoover
Date: 17 September 1987
Subject: Appendix C

Background

At the Liverpool meeting, Jeanne Adams submitted a proposal to the editorial committee (105(*) JCA-28) to put subsection headings in sections C.10, C.11, and C.12 of Appendix C. The editorial committee determined that if any changes of this nature were to be made, they should be made consistently for the whole of Appendix C. I have spent some time looking at Appendix C and I have come up with some subsection heading placement and accompanying titles, as well as some other options we might want to consider.

Considerations

C.13 is the only section in Appendix C that has subsection headings. C.13 has an additional advantage in that the text is well structured around subsections and thus the whole of C.13 is more coherent and readable. It is conceivable that we could insert subsection headings into the text as it exists. However, an alternate plan may be to ask subgroups to modify their respective section notes, structuring them around appropriate subsection headings in the manner of C.13.

I don't think alot of rewriting would be necessary, but I do think more effort and thought should go into this Appendix C edit that merely sticking in subsection headings at what seem like appropriate spots. I also think the results would be very much worth the effort.

I have attached my suggestions for the placement of subsection headings and titles, which may be of some use whichever option we select.

Page C-1, line 4 After "C.1 Section 1 Notes." add <NL> and "C.1.1 Conformance."

line 14 Add "C.1.2 Obsolescence."

line 16 After "C.2 Section 2 Notes." add <NL> and "C.2.1 Use of Keywords."

line 18 After "C.3 Section 3 Notes." add <NL> and "C.3.1 Collating Sequence."

line 21 Add "C.3.2 Comment Lines."

line 24 Add "C.3.3 Statement Labels."

line 27 Add "C.3.4 Source Form."

Page C-2, line 1 After "C.4 Section 4 Notes." add <NL> and "C.4.1 Zero."

line 3 Add "C.4.2 Intrinsic and Derived Data Types."

line 42 Add "C.4.3 Precision and Exponent Range Parameters."

line 50 Add "C.4.4 Resolution and Storage of Derived Types."

Page C-3, line11 After "C.5 Section 5 Notes." add <NL> and "C.5.1 Type Declaration Statements."

line 22 Add "C.5.1.1 RANGE Attribute."

line 36 Add "C.5.2 Substring Array References."

line 39 After "C.6 Section 6 Notes." add <NL> and "C.6.1 Substrings."

line 42 Add "C.6.2 Structure Components."

Page C-4, line 28 After "C.7 Section 7 Notes." add <NL> and "C.7.1 Character Assignment."

line 31 Add "C.7.2 Default Real and Double Precision."

line 39 Add "C.7.3 Function References."

Page C-5, line 1 After "C.9 Section 9 Notes." add <NL> and "C.9.1 The Concept of Records in Fortran."

line 3 Add "C.9.1.1 Endfile Records."

line 8 Add "C.9.2 File I/O."

line 10 Add "C.9.2.1 File Connection."

line 17 Add "C.9.2.2 File Existence."

line 37 Add "C.9.2.3 File Names."

line 43 Add "C.9.3 OPEN Statement."

Page C-6, line 10 Add "C.9.3.1 Preconnection Properties."

line 37 Add "C.9.3.2 Processor Dependent OPEN Properties."

line 50 Add "C.9.4 CLOSE Statement."

Page C-7, before line 6 Add "C.9.5 INQUIRE Statement."

line 64 Add "C.9.6 Right of Access."

line 68 Add "C.9.7 Access Methods."

Page C-8, line 9 Add "C.9.8 Unformatted I/O."

line 16 Add "C.9.9 VALUES= Specifier."

line 37 Add "C.9.10 Scalar Objects of Derived Type."

line 45 Add "C.9.11 List -Directed I/O."

Page C-9, line 1 Add "C.9.12 I/O Restrictions."

line 3 After "C.10 Section 10 Notes." add <NL> and "C.10.1 Apostrophe Edit Descriptor."

line 15 Add "C.10.2 T Edit Descriptor."

line 17 Add "C.10.3 Length of Formatted Records."

line 19 Add "C.10.4 Formatted Input."

line 23 Add "C.10.5 Formatted Output."

line 35 Add "C.10.6 List-Directed Editing."

Page C-10, line 16 After "C.11 Section 11 Notes." add <NL> and "C.11.1 Main Program and Block Data Program Unit."

line 25 Add "C.11.2 Dependent Compilation."

Page C-11, line 14 Add "C.11.2.1 Modules and USE under Dependent Compilation."

Page C-12, line 7 Add "C.11.2.1.1 Module Attributes."

line 15 Add "C.11.2.1.2 Other Uses of Module."

line 27 Add "C.11.3 Transfer Function SET."

Page C-14, line 23 After "C.12 Section 12 Notes." add <NL> and "C.12.1 External Procedures."

line 42 Add "C.12.2 Procedures Defined by Means other than Fortran."

Page C-15, line 16 Add "C.12.3 Procedure Interfaces."

line 36 Add "C.12.4 Argument Association."

Page C-16, line 27 Add "C.12.5 Argument Intent Specification."

line 46 Add "C.12.6 Dummy Argument Restrictions."

line 51 Add "C.12.7 Internal Procedure Restrictions."

4

(35)

To: X3J3
From: David Muxworthy
Subject: Fortran 77 Audit - Sections 4 and 5.
Date: August 20, 1987

The following omissions came up during an audit by Larry Rollison and myself at meeting 105 to ensure that all of ANSI X3.9-1978 was included in S8.104. Since we had no access to a machine-readable form of S8, or to extracts, a statement "not in S8" should be interpreted as "we could not find it in S8". This note covers p 4-1 to 5-4 of F77.

1. Text in F77 not in S8.

p 4-1 l 53-54. "A generic function name does not have a predetermined type" Subgroup believes this to be an error in F77, since clearly REAL for example has a predetermined type. What F77 should have said is something like, "A generic function name does not have a type that is determined by the implicit or explicit typing rules". Given that, a corresponding statement was not found in S8.

p 4-2 l 15-16. "A symbolic name that identifies a main program, subroutine, or common block has no data type." This is not explicitly stated in S8 but can be deduced since types are given by declaration, it is objects which are typed and these are names not objects.

p 4-2 l 22-23. "The value of a signed zero is the same as the value of an unsigned zero". This is not in S8. However ".. a negative zero must not be considered different from a positive zero.. " is at S8 p C-2 l 1-2.

p 4-2 l 29-30. "Within an executable program, all constants that have the same form have the same value." This is not in S8.

p 4-3 l 23-25, l 42-25, p 4-4 l 20-23. ".. constant may be written with more digits than a processor will use to approximate the value of the constant". This is not in S8.

p 4-5 l 7-8. This may be over-pedantic, but F77 explicitly relates the token .TRUE. to the value "true" etc, S8 does not.

p 4-5 l 28-31. The use of the BNF term "character" at R414 in S8 is incorrect since this is defined to be a character in the Fortran character set only (R301); a new term is recommended. Also in S8 there is some ambiguity as to whether a character constant includes its delimiters or not. Subgroup suggests that the sentence "Note that the delimiting apostrophes or quotation marks are not part of the datum represented by the constant" be adopted from F77.

p 5-2 l 6-10. This is a restriction on source statement ordering to avoid the possibility of

```
DIMENSION A(X)
INTEGER X
```

It is not in S8; this probably does not matter.

169

p 5-3 l 4-5. The statement that properties of a (Fortran 77) array are essentially local to a program unit is not in S8.

p 5-3 l 26-27. "The size of a dimension whose upper bound is an asterisk is not specified" is not in S8. It could be added at S8 p 2-8 l 31-40.

p 5-3 l 29-32. This paragraph explicitly states that rank and size of arrays associated via common, equivalence and argument association may be different. It is not in S8 but may not be necessary. If it is in, maybe it should be deprecated.

2. Other related points.

S8 p 5-7. If R515 and R516 (lower and upper bounds) used R721 (specification expression), a constraint could be removed on S8 p 5-7.

F77 p 4-1 l 19-20. "Each intrinsic type is different and may have a different internal representation" is not explicitly in S8 but was thought unnecessary.

If it is intended to have a "Help for old friends" section the following changes in terms were noted:

double precision type	double precision real type
program unit (wrt scope)	scoping unit
arithmetic constant	numeric type constant
unsigned constant	*-literal-constant
signed constant	<no equivalent term>
optionally signed constant	signed-*-literal-constant

(36)

To: X3J3
From: David Muxworthy
Subject: Miscellaneous edits
Date: September 14, 1987

The following minor edits are offered for consideration, with a view to clarifying or correcting the document.

p 4-3 l 11. After "real numbers" add "(13.6)". The range of possible integer values is given in 4.3.1.1; there is no reference in section 4 to the range of possible real values.

p 5-8 l 2. After "dimension." add "The values of lower-bound and upper-bound may each be positive, negative or zero."

{p 5-8 l 45, p 5-9 ll 5, 7, 12. The term "dummy array" is used in this section without being defined; ditto "dummy argument" in section 12 (p 12-1 ll 26, 33, 34 etc). This needs access to the machine-readable version for resolution.}

p 8-8 ll 45-46, p 8-9 l 13. In both cases replace ", and L is not defined" by "; L is not defined by these statements". These sentences are essentially a copy of Fortran 77 p 11-9 ll 11-13. The Fortran 77 text is clear and unambiguous, the S8 text is not.

p C-4 l 20. Delete "and use of it is deprecated"

The word "mold" used in various places in S8 is, so far as I am aware, the only keyword in Fortran which has a different spelling in U.S. and Commonwealth English. The Concise Oxford dictionary has only the spelling "mould". It would be nice to avoid this potential hazard to good spellers. Roget's Thesaurus gives as near synonyms: cast, cut, fabric, fashion, figure, form, frame, last, matrix, stamp, type. Is there any sentiment for using "form" instead of "mold"?

37

To: X3J3
From: David Muxworthy
Subject: Further obsolescent feature
Date: September 14, 1987

Appendix B (page B-13) of Fortran 77 made an attempt at defining some features as deprecated. It would be consistent with the proposed architecture to define them as obsolescent in 8X.

Proposal:

p B-1 l 22+. Add "(9) Use of certain specific intrinsic function names - see B.2.5."

p B-2 l 42+. Add "B.2.5. Specific intrinsic function names. The functions IFIX, IDINT, FLOAT and SNGL were redundant and were retained in ANSI X3.9-1978 to support programs conforming to ANSI X3.9-1966; their use was not recommended in X3.9-1978. Similarly notice was given in appendix B of X3.9-1978 that the functions AMAX0, MAX1, AMIN0 and MIN1 may be deleted in a future revision of the standard. The functionality of these specific intrinsic functions was provided by generic functions in X3.9-1978. Accordingly use of the intrinsic function names AMAX0, AMIN0, FLOAT, IDINT, IFIX, MAX1, MIN1 and SNGL is deemed to be obsolescent in this standard."

Make corresponding adjustments to fonts in the body of the text.

(38)

To: X3J3
From: David Muxworthy
Subject: Intrinsic function notes
Date: September 14, 1987

The section entitled "Section 13 Notes" is in fact an essay on array features and the use of intrinsic procedures. The Fortran user looking for amplification of functions in section 13 will not find them in the notes in a manner analogous to notes for other sections.

Proposal: Move the current section 13 Notes to Appendix I. Replace the section 13 Notes with the following (the first two paragraphs are culled from X3.9-1978 page B-13):

C.13 Section 13 Notes

This standard provides that a standard-conforming processor may supply intrinsic procedures in addition to those defined in sections 13.10 to 13.12. Because of this, care must be taken when a program is used on more than one processor because a function name not in section 13.11 or 13.12 may be classified as an external function name on one processor and as an intrinsic function name on another processor in the absence of a declaration for that name in an EXTERNAL or INTRINSIC statement.

To guard against this possibility, it is suggested that any external functions referenced in a program should appear in an EXTERNAL statement in every scoping unit in which a reference to that function appears. If a scoping unit references a processor-supplied intrinsic function that does not appear in section 13.12, the name of the function should appear in an INTRINSIC statement in the scoping unit.

There is no requirement that the sequence of pseudorandom numbers returned by the intrinsic procedure RANDOM relating to a specific random seed be the same from one processor to another. However a particular random seed on a given processor must always generate the same sequence of pseudorandom numbers.

Note that any reference to the TRANSFER function which makes use of the physical representation of storage is likely to render a program non-portable between processors.

Subject: Analysis of /S9 09/29/87 (39) 106(*)JHM-01

To: X3J3 From: J. H. Matheny

This paper is an analysis of the letters to X3J3 during the period from the 93rd meeting, FIB, (February 1985) to the 105th (August 1987) as collected into X3J3 Standing Document /S9. The responders are referring to the FIB, many versions of /S8, fora, and other publications of the details of the draft Fortran 8X Standard.

All ideas appear in the analysis in a header line. This includes the /S9 log number, an idea number within a document, a judgment of the idea, the principal author of the letter, and a highly abbreviated indication of the author's organization if given. Following this is a concise description of the idea.

The document is sorted by category (judgment) and then by document number and idea number. Within category, there has been an attempt to combine like entries. Where the comment seems to be not quite the same, some individual comments are appended to the paragraph and are noted by item number. The categories are:

Major Proposal. Apparently valid, but would make a significant change to the language and to the Standard document.

Minor Proposal. Apparently valid, and only a minor change to the language and to the document.

Reject. In my judgment, the idea should be rejected. This may be because it violates "the spirit of Fortran", because it would be incompatible with Fortran 77, because we have considered the issue at length and either rejected it or provided the functionality in another way, because I can't understand the point, and so on.

Too large. In the stated, or unstated, ways, the draft Fortran is too rich to be properly Fortran.

Deprecation. Features of Fortran 77 are obsolete, obsolescent, or deprecated and should not be, or conversely should be.

No problem. Either the suggestion has been resolved In the current /S8, or the the capability is already there in some other manner, or the item objected to has been deleted from the draft Standard.

No recommendation. This might be a question, or a general comment of satisfaction or dissatisfaction. Here are also listed editorial problems, and other unclassified items.

Missing letter numbers refer to X3J3 responses to the correspondent, survey results [reported elsewhere, ISO resolutions, and other letters that did not seem appropriate in this context. Words in brackets "[]" are my comments, and not the gist of the suggestion.

Repeating the obvious caveat, this is one individual's evaluation of a mass of commentary. I trust that the paper will be useful, but, of course, any proposals resulting from the commentary should refer back to the commentor's letter.

Major Proposal	1-3	Bierman, K.H.	FEA
Major Proposal	2-2	Gay, D.M.	Bell Labs
Major Proposal	9-2	Hartl, Heinrich	FRD
Major Proposal	12-3	Deakin, J.	Australia
Major Proposal	146-9	Tincknell, M.L.	Lawrence BL
Major Proposal	147-9a	Fullerton, J.	Logopedics
Major Proposal	150-2	Payne, Fred R.	U Texas
Major Proposal	168-1	Gibbs, G. et al	CRC UK

Wants alphabetic statement labels.

Major Proposal	1-7	Bierman, K.H.	FEA
Major Proposal	59-5	LWG	DOE
Major Proposal	65-2	Willis, A.B.	Sweden
Major Proposal	182-5	Henderson, L.W.	OSU
Major Proposal	192-4	Mathews, P.D.Jr	Draper Labs

#1-7. Wants string operations. [Varying character]
#59-5. Varying length character facility is required unless the data abstraction facility is sufficient to specify the appearance of a derived data type.
#182-5. Wants dynamic strings and functions that can return dynamic strings.
#192-4. Demands varying length character type. [Specifies varying in fixed allocation.] "I reject all arguments that derived types are acceptable for STRING data. Derived types might work, but this feature is essential to the language: any risk is too great."

Major Proposal	9-6	Hartl, Heinrich	FRD
Major Proposal	163-3	Van Snyder, W.	JPL
Major Proposal	172-3	Cox, Francis E.	Swifte, UK
Major Proposal	188-3	Van Snyder, W.	JPL

Permit EXIT from any enclosing named block.
#163-3. Exit should exit from anywhere. The RETURN can thus be deprecated.

Major Proposal	9-11	Hartl, Heinrich	FRD

Add optional assertions.

Major Proposal	9-12	Hartl, Heinrich	FRD
Major Proposal	64-2	Broughton, C.	Canada
Major Proposal	114-1	Russell, J. J.	SLAC
Major Proposal	146-5	Tincknell, M.L.	Lawrence BL
Major Proposal	147-12	Fullerton, J.	Logopedics
Major Proposal	155-2	Buckley, A.	Canada CFWG
Major Proposal	157-1	Messina, P.C.	DOE LWG
Major Proposal	179-1	Reid, J.K.	IFIP
Major Proposal	173-15	Cox, Francis E.	Swifte, UK
Major Proposal	175-7	Bierman, K.H.	Bierman & A

Need pointers.
#9-12. Provide for data trees and linked lists. [Pointers?]
#173-15. Pointers are desirable. Consider:
Should be strongly typed
Should point to the heap exclusively [ALLOCATABLE]
Is allocation of a derived type with variants to be allowed?
If so, lists the problems Pascal found. [Not in /S8-104]
#175-7. "Some C compiler writers noted that reason that they do not perform many of the typical FORTRAN optimizations is the presence of pointers in C."

Major Proposal	12-5	Deakin, J.	Australia

Keyed access.

Major Proposal	17-2	Loach, K.W.	SUNY
Major Proposal	59-8	LWG	DOE
Major Proposal	168-7	Gibbs, G. et al	CRC UK
Major Proposal	174-3	Jones, Russell K.	Hughes

Stream I/O.
#59-8. A stream I/O capability would permit the user to "get" or "put" a character to the external medium. The module writer could then process derived type I/O conveniently.
#168-7. Wants a subroutine to output a single character to a device. [Stream I/O?].
#174-3. Wants an OPEN specifier, or statement GET/PUT to read/write one character at a time, or STREAM I/O.

Major Proposal	1-8	Bierman, K.H.	FEA
Major Proposal	59-1	LWG	DOE
Major Proposal	59-3	LWG	DOE
Major Proposal	64-2a	Broughton, C.	Canada
Major Proposal	78-5	Ware, John	Haverly S
Major Proposal	171-1	Whitfield, Doug	I R & D

Requires (or wants) a bit type of acceptable form.
#1-8 Wants bit operations. [character-like bit]
#59-3. The description of a bit facility must be such that implementations are sufficiently efficient
#171-1. Distressed at the dropping of bit type. The two facilities that have been in /S8 are unsatisfactory. Prefers some form of packed logicals. A module is unsatisfactory, as it could not support packed masks.

Major Proposal	59-2	LWG	DOE
Major Proposal	189-2	Page, Clive	U Leicester

A bit facility should handle formatted I/O -- Hex or octal.

Major Proposal	59-4	LWG	DOE

The bit facility must handle DATA and PARAMETER.

Major Proposal	59-6	LWG	DOE
Major Proposal	81-3	Hoekstra, J.P.	Macatawa

#59-6. The derived type/MODULE mechanism should handle formatted I/O.
#81-3. "Derived types are great! ... I hope that you plan to support them in READ and WRITE statements."

Major Proposal 59-7 LWG DOE
Facilities to specify the appearance of output data for list-directed and namelist I/O are needed.

Major Proposal 64-3 Broughton, C. Canada
Wants value-range CASE selectors, like .NEGATIVE., .ZERO.

Major Proposal 64-4 Broughton, C. Canada
Wants an intrinsic function with two character arguments that returns .LESS., .EQUAL., .GREATER. and permit these pseudo-parameters in CASE selectors.

Major Proposal 64-5 Broughton, C. Canada
Wants an intrinsic function with real arguments that returns a

Major Proposal 64-6 Broughton, C. Canada
Wants an intrinsic subroutine DIVIDE that returns a quotient and remainder.

Major Proposal 66-5 Hamaker, D.W.
Wants to extend the CASE statement to include "computed case" like now, and to express choices on distinct conditions.

Major Proposal 70-26 Ragosta. A.E. DOA Ames
Provide language (associating a unit number with dynamic storage) to provide a file that is "RAM-disk". In the OPEN statement, STATUS=INTERNAL.

Major Proposal 70-33 Ragosta. A.E. DOA Ames
Major Proposal 180-4 Jones, Russell K. Hughes
#70-33. "There is no reason RECURSIVE is needed (ie, all functions should be recursive)."
#180-4. The RECURSIVE keyword is not necessary. [Sure, if all subprograms are compiled as recursive.]

Major Proposal 70-35 Ragosta. A.E. DOA Ames
Wants the OPERATOR phrase to be instead a new type of subprogram. Claims it would be clearer. [His example is a text-substitution MACRO.]

Major Proposal 70-38 Ragosta. A.E. DOA Ames
"PRESENT - It would be nice if there was an indexed manner to reference an unknown number of arguments." [Shades of alternate RETURN.]

Major Proposal 72-1 Van Snyder, W. JPL

Major Proposal 158-3 Van Snyder, W. JPL
Major Proposal 194-1 Petry, R.K.
Provide a category of function that can appear on the left hand side of an assignment operator "=", an ABSTRACTION. This can have the same name as the equivalent function. [I believe that this is the current rage in the computer science community -- object-oriented programming. Prior crazes put into Standard Fortran include abstract data type a few years ago, and structured programming, partly put into Fortran 77 between the draft for public review and comment and the final document.

Major Proposal 91-3 Cartledge, C.J. U Salford
"The problem of precision also exists for INTEGER." Vital for image and signal processing. Equivalent mechanism as that for REAL.

Major Proposal 91-4 Cartledge, C.J. U Salford
The GKS Standard "defines value ranges as being reserved .. or future standardization."

Major Proposal 103-1 Storey, M.W.and NERC
Wants to specify, in the OPEN statement, a minimum record length (extend RECL to all files) and a minimum file length in records. [We have extended RECL.]

Major Proposal 103-2 Storey, M.W.and NERC
Wants a way to identify the * connection to the default I/O. It can thus be passed as a subprogram argument.

Major Proposal 103-3 Storey, M.W.and NERC
Wants, in the OPEN statement, a way to get at this [the default unit for PRINT, READ] when creating a NEW file.

Major Proposal 17-1 Loach, K.W. SUNY
Major Proposal 103-5 Storey, M.W.and NERC
Major Proposal 168-4 Gibbs, G. et al CRC UK
Major Proposal 189-4 Page, Clive U Leicester
#17-1, Use the minus carriage return to suppress CR/LF.
#103-5. Wants some mechanism to suppress the appended CR/LF on output.
Wants the $ edit descriptor to prohibit cursor movement.

Major Proposal 103-7 Storey, M.W.and NERC
Major Proposal 172-9 Cox, Francis E. Swifte, UK
Major Proposal 173-12 Cox, Francis E. Swifte, UK
Major Proposal 174-1 Jones, Russell K. Hughes
#103-7. Wants a way to determine the length of formatted input records.
Add the VAX 'Q' edit descriptor to return the number of characters remaining to be processed in the current input buffer.

Major Proposal 103-11 Storey, M.W.and NERC
Major Proposal 114-2 Russell, J. J. SLAC
Wants language to declare external routine names internally -- a data type that would let an external name be chosen in a program.

Major Proposal 113-1 Anderson, J.B. U Aberdeen
Wants to introduce new "standard" procedures with a syntax like READ. References STATFOR.

Major Proposal 131-5 Van Snyder, W. JPL
Suggests that the I/O DO in an *iolist* is a form of array slice operation. Suggests allowing only one form of array slice operation to simplify the language.

Major Proposal 131-6 Van Snyder, W. JPL
Suggests that character operations are a form of array slice, and that that suffices. Deprecate *LEN and substring.

Major Proposal 131-7 Van Snyder, W. JPL
Major Proposal 158-1 Van Snyder, W. JPL
Major Proposal 178-1 Van Snyder, W. JPL
Suggests that a structure component reference should be the same as an array or function reference.

Major Proposal 146-4 Tincknell, M.L. Lawrence BL
Wants an UNSIGNED INTEGER type.

Major Proposal 146-7 Tincknell, M.L. Lawrence BL
Wants an edit descriptor scheme that will convert output as specified (I, E, F, G) and then return the value with a minimum number of leading blanks for use in some text processor.

Major Proposal 146-8 Tincknell, M.L. Lawrence BL
Wants to restore Hollerith and then deprecate it.

Major Proposal 147-9 Fullerton, J. Logopedics
There is no facility to define or control concurrency.

Major Proposal 147-12 Fullerton, J. Logopedics
Why are all the operators delineated by periods? Suggests true, false. Deprecate the dots. [We've added punctuation for the relationals.] [Significant blanks are now not in the Standard.]

Major Proposal 147-18 Fullerton, J. Logopedics
Wants automatic continuation.

Major Proposal 147-19 Fullerton, J. Logopedics
Wants Disjkstra's guarded commands.

Major Proposal 148-3 Wrzesinski, D.T. Deere
Wants an inherent passed parameter routine.

Major Proposal 148-5 Wrzesinski, D.T. Deere
Wants a simple routine to allocate and free data for I/O. [?]

Major Proposal 148-6 Wrzesinski, D.T. Deere
Wants I/O routines for full screen interface to standard [sic] devices (3270, VT100, etc.)

Major Proposal 148-8 Wrzesinski, D.T. Deere
Wants unrestricted fast I/O routines for sequential and direct access processing without formatting. [Binary I/O]

Major Proposal 150-3 Payne, Fred R. U Texas
Wants arrays with up to 13 dimensions.

Major Proposal 158-1 Van Snyder, W. JPL
Use the same syntax for reference and assignment of all objects. This will minimize changes caused by design decisions.

Major Proposal 159-1 Van Snyder, W. JPL
Re-examine the syntax of ASSIGNMENT [in derived type]. [Object-oriented facilities handle the problem better.]

Major Proposal 159-2 Van Snyder, W. JPL
To preserve syntactic uniformity, it is necessary to allow subprograms to have arguments that are ranges.

Major Proposal 163-1 Van Snyder, W. JPL
This is a detailed proposal to "Regularize the Syntax of S8 Fortran". It provides suggested text for X3J3 /S8-99. Proposed are:

Regularization of referential text,
IN-LINE subprograms,
ACCESSORS [a form of subprogram that may be written to be on the left of the assignment operator],
other items.

Major Proposal 163-4 Van Snyder, W. JPL
The ALLOCATE should apply to structures.

Major Proposal 163-5 Van Snyder, W. JPL
A BLOCK - END BLOCK construct should be introduced.

Major Proposal 166-3 Bej, Mark D.
Extend EXIT to IF and CASE blocks.

Major Proposal 167-9 Chan, Donald I R & D
Provide inquiry functions for non-REAL types. These should include INTEGER and non-negative INTEGER.

Major Proposal 167-12 Chan, Donald I R & D
RANGE lists are an ad hoc feature. Why not call-lists etc.

Major Proposal 168-2 Gibbs, G. et al CRC UK
Wants indexed sequential files.

Major Proposal 168-6 Gibbs, G. et al CRC UK
Wants a function to read the keyboard.

Major Proposal 168-8 Gibbs, G. et al CRC UK
Wants a function to return the command that started the program. Thus parameters can be passed to the program.

Major Proposal 171-2 Whitfield, Doug I R & D
Urges that the array intrinsic functions DIAGONAL, PROJECT, REPLICATE, RANK, FIRSTLOC, and LASTLOC be reinstated. Valuable functionality at little cost to the implementor.

Major Proposal 171-3 Whitfield, Doug I R & D
Urges reinstatement of FORALL. We are seeing the expected proliferation of approaches to specifying parallel execution.

Major Proposal 171-4 Whitfield, Doug I R & D
"Array-constructors. Why on earth not?"

Major Proposal 171-5 Whitfield, Doug I R & D
Keep dynamic aliasing. "We suspect that the restricted mapping generated by the IDENTIFY statement will lead to more efficient execution than a more general pointer facility."

Major Proposal 171-6 Whitfield, Doug I R & D
Keep vector-valued subscripts. They complicate the language and its implementation, but "its utility would seem to outweigh such considerations."

Major Proposal 171-7 Whitfield, Doug I R & D
Restore condition handling. "it is a major fault of FORTRAN 77 that no such mechanism exists."

Major Proposal 171-8 Whitfield, Doug I R & D
Zero-sized arrays -- "have no well defined meaning or purpose." Define and use them, or drop them.

Major Proposal 172-2 Cox, Francis E. Swifte, UK
Major Proposal 173-3 Cox, Francis E. Swifte, UK
Add to the substring syntax a semi-colon. Right of the ; is the substring length. [Shades of PL/I.]

Major Proposal 172-6 Cox, Francis E. Swifte, UK
Major Proposal 173-6 Cox, Francis E. Swifte, UK
Adjust the syntax of the repeat count DO as follows:

loop-control = (*scalar-int-expr*) TIMES
This simplifies lexical analysis.

Major Proposal 172-8 Cox, Francis E. Swifte, UK
Major Proposal 173-8 Cox, Francis E. Swifte, UK
Enable construct. "Branches out of enable blocks are difficult to implement..." Extend EXIT to allow it to terminate the current or any enclosing block. Then prohibit outward branches.

Major Proposal 173-1 Cox, Francis E. Swifte, UK
Keep bit type out of the Standard -- "a complete waste of time beyond storage economy." [This family of proposals are post-"compromise".]

Major Proposal 173-1 Cox, Francis E. Swifte, UK
Conditional handling -- "the propagation scheme defined allows exceptions (conditions) to be propagated out of static name scope, and which then must be caught by the catch-all HANDLE (*) clause since they are all un-namable." ANSI-X3J9/IEEE-F770 (Pascal) avoids this sloppy method ...

Major Proposal 173-1 Cox, Francis E. Swifte, UK
It is unclear whether the intrinsic conditions are initially enabled.

Major Proposal 173-1 Cox, Francis E. Swifte, UK
There is no definition of the scope of a condition.

Major Proposal 174-2 Jones, Russell K. Hughes
Major Proposal 189-3 Page, Clive U Leicester
Add to the OPEN statement the specifier CARRIAGECONTROL = FORTRAN or ASCII. FORTRAN is the default. If ASCII is specified, position one of the record output has no special significance,

Major Proposal 175-2 Bierman, K.H. Bierman & A
"Not having arrays of structures, user-defined operators, and operator overloading severely limit the utility of the new draft. Please reconsider. [Wrote about the "compromise".]

Major Proposal 179-2 Reid, J.K. IFIP
The passing of internal procedures should be permitted.

Major Proposal 180-1 Jones, Russell K. Hughes
Change the continuation mechanism of free form source to indicate continuation by the presence of a non-numeric, non-alphabetic, non-exclam in the first non-blank position. The user could not continue at an operator, or in a character constant.

Major Proposal 182-2 Henderson, L.W. OSU
Wants an error handling mechanism.

Major Proposal 188-2 Van Snyder, W. JPL
"Should the SET RANGE and IDENTIFY statements allow a status to be returned in the event a run-time error is committed?"

Major Proposal 188-6 Van Snyder, W. JPL
The INQUIRE, or an intrinsic, should allow the program to learn the unit number(s) preconnected for the short form READ and PRINT.

Major Proposal 189-1 Page, Clive U Leicester
Wants logical operations on groups of bits. Suggests the word operations .IAND., .IOR. etc or .AND., .OR. -- the IRTF Standard. Claims that this is as portable as integer arithmetic.

Minor Proposal 1-1 Bierman, K.H. FEA
Minor Proposal 62-1 Janes, Rob Cummins
Doesn't want IMPLICIT NONE.

Minor Proposal 1-2 Bierman, K.H. FEA
Minor Proposal 52-14 Rej, Mark D.
Minor Proposal 57-3 Morris, A.H.Jr NSWC
Minor Proposal 132-2 Bierman, K.H. FEA Inc
Minor Proposal 147-10 Fullerton, J. Logopedics
Wants "." (or something else) as the qualification punctuation "%".

Minor Proposal 2-5 Gay, D.M. Bell Labs
Minor Proposal 5-1 Higbie, L. Gould
Remove multi-statement lines and the semi-colon. Readability.

Minor Proposal 9-1 Hartl, Heinrich FRD
Doesn't like the label following a colon in a source line.

Minor Proposal 9-9 Hartl, Heinrich FRD
Provide a function for bounds checking, .ISIN.

Minor Proposal 12-4 Deakin, J. Australia
Wants language to automatically name the I/O units that are preconnected for PRINT and READ. Permit automatic assignment of units 5 and 6. [Means the units preconnected to READ and PRINT.]

Minor Proposal 37-4 Nagle, F.W. METEFOR
Wants a SWAP operator -- LHS => RHS, RHS => LHS. The preprocessor syntax uses ==, which Fortran 8X reserves for .EQ.

Minor Proposal 37-5 Nagle, F.W. METEFOR
Respondent suggests an operator, the .BUT. that is synonymous with the .AND. yet is sometimes more meaningful to the user.

Minor Proposal 52-18 Rej, Mark D.
Use % to indicate commentary instead of !. [Assumes that the qualifier symbol is a dot.]

Minor Proposal 52-19 Rej, Mark D.
Require that all statements terminate with a semi-colon. Provide a "semi-colon statement" [compiler option] to turn this off.

Minor Proposal 52-24 Rej, Mark D.
Change END SELECT to END SELECT CASE.

Minor Proposal 52-25 Rej, Mark D.
Add a .XOR. relational operator.

Minor Proposal 52-29 Rej, Mark D.
Add a pattern-checking facility like MUMPS. In an If construct.

if the *char-logical-expr* contains the .PAT. relational operator, the left operand is tested based, on the right operand, for such things as valid numeric, character, and so on.

Minor Proposal 70-5 Ragosta. A.E. DOA Ames
Add the HT (horizontal tab) to the Fortran character set.

Minor Proposal 70-19 Ragosta. A.E. DOA Ames
"The OPTIONAL [attribute] should(could) be made a stand-alone specification statement."

Minor Proposal 70-25 Ragosta. A.E. DOA Ames
CASE DEFAULT should be CASE ELSE.

Minor Proposal 70-29 Ragosta. A.E. DOA Ames
The *position-spec-list* for BACKSPACE should include a RECORDS = num, the records to be backspaced over.

Minor Proposal 70-40 Ragosta. A.E. DOA Ames
Advocates several added string intrinsic functions.

Minor Proposal 70-41 Ragosta. A.E. DOA Ames
Advocates an added array intrinsic function, ZERO (array).

Minor Proposal 105-4 Delves, L.M. U Liverpool
Let BLOCK DATA be an alternative, deprecated spelling of MODULE.

Minor Proposal 127-3 McDonald, J.D. U Illinois
"...there is no reason to prohibit EQUIVALENCE-ing CHARACTER*1 with INTEGER, ..." where a machine architecture allows it, like most modern machines. [sic.]

Minor Proposal 131-2 Van Snyder, W. JPL
Wants to add a SAME to he *close-status-spec* to KEEP depending upon the status of the file when opened with UNKNOWN.

Minor Proposal 147-2 Fullerton, J. Logopedics
Doesn't like the word MODULE, with reasons. Prefers PACKAGE.

Minor Proposal 147-13 Fullerton, J. Logopedics
Be consistent re significant blanks. ENDIF, not END IF.
[Working from the FIB and a DECUS newsletter.]

Minor Proposal 163-6 Van Snyder, W. JPL
The standard should provide for a canonical format for the compiler listing.

Minor Proposal 163-9 Van Snyder, W. JPL
Minor Proposal 188-5 Van Snyder, W. JPL
Provide a subroutine that outputs the message that would have been written when IOSTAT is non-zero. The function should define

a standard value depending on whether the I/O statement would have worked with a correct value, or wouldn't have, and a way to say what kind of statement created the IOSTAT value.

Minor Proposal 169-7 Bielecki, Jan Poland
Wants to change the keyword ASIS in the POSITION= specifier of OPEN to LEAVE.

Minor Proposal 172-11 Cox, Francis E. Swifte, UK
Minor Proposal 173-14 Cox, Francis E. Swifte, UK
For the numeric type conversion intrinsics, change MOLD to MODEL. MOLD can be spelled MOULD.

Minor Proposal 188-1 Van Snyder, W. JPL
Section 9.7, prohibition of recursion, should be deleted, or rewritten.

Minor Proposal 188-10 Van Snyder, W. JPL
Provide an intrinsic function IF(P,Q,R) that returns Q if P is

Minor Proposal 190-1 Hill, J. D. CRC UK
Wants more and stricter words describing RANDOM and RANDOMSEED.

Reject 1-9 Bierman, K.H. FEA
Reject 52-3 Rej, Mark D.
Reject 175-3 Bierman, K.H. Bierman & A
Wants the MACRO facility.
#52-3. Wants a macro facility. Suggests something like the Burrough DEFINE.
#175-3. Include the existing MACRO package.

Reject 1-10 Bierman, K.H. FEA
Wants a standard procedure library for the various IEEE exception conditions/parameters.

Reject 2-1 Gay, D.M. Bell Labs
Wants to branch freely amidst CASE and IF blocks

Reject 2-6 Gay, D.M. Bell Labs
Wants # rather than ! as the comment delimiter. ! frequently means "not".

Reject 3-2 Sobel, B.A.
Wants integer rational arithmetic.

Reject. 5-2 Higbie, L. Gould
Remove IDENTIFY, damage to optimization.

Reject 9-3 Hartl, Heinrich FRD
Disallow passing odise-sections as an actual argument.

Reject 9-4 Hartl, Heinrich FRD
Remove CASE, the block IF does it.

Reject 9-5 Hartl, Heinrich FRD
Enforce naming of block constructs whenever the construct extends over more than one page.

Reject 9-10 Hartl, Heinrich FRD
Modify IMPLICIT to disallow implicit conversions. Deprecate them. Ada disallows such conversions.

Reject 10-2 Einarsson, Bo Sweden
Simplify USE, disallow any COMMON block name.

Reject 12-2 Deakin, J. Australia
Wish list consisting of DO WHILE, REPEAT UNTIL, small integers, and standard compiler directives.

Reject 12-6 Deakin, J. Australia
"Statement functions expanded before compiling to enable use on LHS of expressions."

Reject 37-1 Nagle, F.W. METEFOR
Wants the *byte notation for real, integer etc.

Reject 37-2 Nagle, F.W. METEFOR
The cited preprocessor provides vector and array operations. So does /S8.

Reject 37-3 Nagle, F.W. METEFOR
The cited preprocessor provides a PACKED integer, with user-specified bit fields.

Reject 46-2 Berns, G.W. SAIC
Lists vulnerability of Fortran, and wishes to remove them -- identify them as obsolete. These include:

1. Eliminate implicit typing.
2. Identify [in the Standard] variables referenced and not set.
3. Mandate [in the Standard] the removal of clutter. Unused code, unused COMMON blocks.
4. Require [in the Standard] checking of inconsistent COMMON blocks. [MODULE/USE can, but doesn't require, minimize the problem, as could INCLUDE.]
5. Check for inconsistent referencing of subprograms. [Interface blocks make this possible, but the user is not required to use them.]

Reject 50-1 Soulie, Edgar AFNOR
Wants a syntactic distinction between an array element reference and a function reference.

Reject 52-1 Rej, Mark D.
Suggests abbreviations for the attributes of some declarations.

Reject 52-4 Rej, Mark D.
Wants an array attribute to specify storage association -- row-wise of column-wise.

Reject 52-5 Rej, Mark D.
Doesn't like, or understand, the IDENTIFY *alias-bound-spec*.

Reject 52-6 Rej, Mark D.
Surprised to find that no increments [array processing] were allowed [?].

Reject 52-10 Rej, Mark D.
Array argument transmission, pass both dimensionality and extents.

Reject 52-11 Rej, Mark D.
Remove underscores from keywords.

Reject 52-13 Rej, Mark D.
Change the keyword PRECISION to PRECISION10. This makes it consistent with the attribute.

Reject 52-15 Rej, Mark D.
Make the template name optional in defining an identifier to be of a derived type.

Reject 52-21 Rej, Mark D.
Use colons instead of commas in the DO.

Reject 52-22 Rej, Mark D.
Add several sets of arguments for the index-variable in a DO, like ALGOL

Reject 52-23 Rej, Mark D.
In the DO syntax, allow both an increment and an independent control variable. Also throw in WHILE and UNTIL.

Reject 52-26 Rej, Mark D.
Add the operators .IMP. and .PAT. for implies and pattern checking.

Reject 52-28 Rej, Mark D.
Require complete evaluation of a logical expression.

Reject 52-30 Rej, Mark D.
For OPEN, add specifiers MEDIUM= and OTHER or SYSTEM. The user could then "specify what kind of storage is to be used", and specify other system-specific attributes.

Reject 52-37 Rej, Mark D.
Find a way to EXIT from an I/O DO.

Reject 52-39 Rej, Mark D.
Add an EA. edit descriptor that would be 1P for E ranges, and would conserve space for I, F, and G.

Reject 52-40 Rej, Mark D.
Wants to allow lower character bounds less than one. [?]

Reject 52-41 Rej, Mark D.
Allow "assumed" character bounds. [?]

Reject 52-42 Rej, Mark D.
Provide a shorthand notation for incrementing a variable.

Reject 52-43 Rej, Mark D.
Allow multiple assignment.

Reject 52-47 Rej, Mark D.

Extend string manipulation per Boroughs Algol.

Reject 52-48 Rej, Mark D.
Add an epilogue facility so that the user can "clean-up" upon trauma.

Reject 52-49 Rej, Mark D.
Provide a facility to let a program "sleep" for a user-specified period of time.

Reject 57-12 Norris, A.W.Jr NSWC
Wants an intrinsic function LOC(A,B) that is true or false as the storage address of the first element of array A is the same as B.

Reject 57-13 Norris, A.W.Jr NSWC
Do not permit expression re-ordering for the division operator.

Reject 63-1 Bell, Robert CSIRO
Tidy up the E and G edit descriptors. [But this would invalidate Fortran 77.]

Reject 66-1 Hamaker, D.W.
Wants LOOP...REPEAT rather than the new forms of DO.

Reject 66-2 Hamaker, D.W.
Wants WHILE for readability though EXIT is sufficient.

Reject 70-13 Ragosta. A.E. DOA Ames
"Order of evaluation should be specified." identical calculations could vary on different machines.

Reject 70-15 Ragosta. A.E. DOA Ames
Complains that the elements in a common block may be undefined upon the execution of a RETURN. [This is Fortran 66.]

Reject 70-16 Ragosta. A.E. DOA Ames
Objects to the new spelling of character length. "No need to change..."

Reject 70-24 Ragosta. A.E. DOA Ames
"The CYCLE statement is meaningless. It is clearly identical to GOTO label." ...

Reject 81-5 Hoekstra, J.P. Macatawa
Another possible edit descriptor ... embedded comma. CN8.2 would output 9,999.99 etc.

Reject 103-8 Storey, M.W.and NERC
Wants a way to cause an EOF on a terminal. [This is a property of the operating system.]

Reject 103-9 Storey, M.W.and NERC
Wants free formatting in the midst of a format, eg. F*.*, ...

Reject 103-10 Storey, M.W.and NERC
Wants dynamic structures in I/O, a means to browse a database.

Reject 105-1 Delves, L.M. U Liverpool
Reject 194-1 Petry, R.K.
Permit the CALL of a function. Throw away the returned value.

Reject 105-2 Delves, L.M. U Liverpool
Make the keyword CALL optional.

Reject 105-3 Delves, L.M. U Liverpool
Eliminate the need for a MAIN program. Let the OS know where to start. At least one parameterless procedure must be present in an executable.

Reject 127-5 McDonald, J.D. U Illinois
Reject 143-2 Stubblefield, P.
Reject 147-23 Fullerton, J. Logopedics
Reject 168-10 Gibbs, G. et al CRC UK
Wants INCLUDE
#127-4. MODULE/USE too restrictive.
#143-2. Wants INCLUDE if MODULE is removed (which he recommends).
#147-23. Advocates a library mechanism -- INCLUDE. Cites a DECUS paper, Languages and Tools.

Reject 127-4 McDonald, J.D. U Illinois
Wants to add to the language a way to call one of a group of functions or subroutines from a list depending on some variable. [like computed GOTO.]

Reject 131-3 Van Snyder, W. JPL
The writer observes that portability would be substantially enhanced were "processor dependent" minimized, especially in the I/O areas. [This is, of course, very true. X3J3 has rejected several proposals in this area because (1) most "processor dependent"'s pave the way for some vendor's extensions, and (2) because of a concern that such changes would invalidate some Standard-conforming programs. At Fortran 77 time, all of this vagueness got in because that was all that the committee would accept. We thought that we could be specific for the newly-described features (not in Fortran 66) but were shot down during public review and comment.]

Reject 131-4 Van Snyder, W. JPL
The use of processor dependent units for *open-recl-spec* for unformatted I/O has to be one of the worst mistakes in Fortran 77. [I was sure that this meant "word", which the standard refuses to use, and so implemented it. Other, more recent,

Fortran 77's have chosen the 8-bit byte as a measure of record length, and this is becoming a de facto standard.] [Note that the contents of the unformatted record can have no bearing on the issue. The IOLENGTH form of INQUIRE was developed in Fortran 8X to resolve this issue in a crude, though portable, way.]

Reject 132-2 Bierman, K.H. FEA Inc
"...,the current state of affairs regarding mixed mode arithmetic is unacceptable." Wants dominant mode.

Reject 132-3 Bierman, K.H. FEA Inc
Suggests adoption of an existing Macroprocessor [pre-processor] such as M4, the C standard or what have you.

Reject 146-3 Tincknell, M.L. Lawrence BL
Wants DO WHILE and DO UNTIL. Likes C FOR. Wants END DO instead of REPEAT.

Reject 146-6 Tincknell, M.L. Lawrence BL
Wants the edit descriptors E, F, G to behave as though 1P had been invoked.

Reject 147-15 Fullerton, J. Logopedics
Wants Ada convention -- quote to define strings, apostrophe for single character constants.

Reject 148-2 Wrzesinski, D.T. Deere
Wants interrupt handling. [In standardese, what is this? Attentions. Very processor-dependent.]

Reject 148-4 Wrzesinski, D.T. Deere
Wants a function/subroutine to load other modules in a library. [What's a library?]

Reject. 152-1 Wilder, Stewart Wyatt
Wants to allow EXTERNAL in BLOCK DATA. Fortran 77 disallows this.

Reject 154-2 Curtis, Alan Harwell
Wants SAVE to apply to arguments in ENTRY. [Save what? Address, value?]

Reject 162-3 Monson, M.C. Deere
Wants the ability to dynamically allocate a data set [file] from within a FORTRAN program. [Why need one allocate a file? What does this mean -- buffers?]

Reject 162-4 Monson, M.C. Deere
Wants the ability to specify execution time parameters to a FORTRAN program. [This seems not to be compiler options. Does he want to read a file? Maybe preconnected to a terminal?]

Reject 162-6 Monson, M.C. Deere
Wants the ability to dynamically load a FORTRAN subroutine. Beyond the scope of Fortran. Looks like a very exotic linker, or maybe segmentation.

Reject 163-2 Van Snyder, W. JPL
The triplet notation for subscript ranges should be allowed in a DO statement. This implies that neither the upper or lower bounds can be omitted.

Reject 163-7 Van Snyder, W. JPL
Reject 168-9 Gibbs, G. et al CRC UK
Reject 180-6 Jones, Russell K. Hughes
Looping should include UNTIL and WHILE.

Reject 166-1 Bej, Mark D.
Change the implicit default to IMPLICIT NONE. Would provide safety. [But would be incompatible with Fortran 77.]

Reject 167-1 Chan, Donald I R &D
Eliminate the maximum of 7 for array rank. Such a limit is archaic. But, without a limit, some processors might decide on 4, or even 3. This would de-standardize Fortran 77, and would also be horrid.

Reject 167-2 Chan, Donald I R &D
Eliminate the 1320 character source line. [It has been changed to 2640.] [And without a limit, some processors might specify less than 264o inhibiting portability.]

Reject 167-5 Chan, Donald I R &D
Eliminate the need for heap memory management by restricting the life of an allocatable array to a subprogram.

Reject 167-6 Chan, Donald I R &D
Eliminate SET RANGE. It is not needed with the more powerful array features of Fortran 8X, and is hard to read. The same functionality is provided by array sectioning

Reject 167-7 Chan, Donald I R &D
Eliminate IDENTIFY. It provides index remapping, but there is a better way of providing this power. IDENTIFY is more powerful than array sectioning, but it uses the insecure mechanism of aliasing. Why not a more generalized array sectioning primitive. [A page from my /S9 seems to be missing here.]

Reject 167-11 Chan, Donald I R &D
The logical operators .EQV. and .NEQV. serve no function that .EQ. and .NE. do not already perform. [But they are in Fortran 77 and resolve an ambiguity.]

Reject. 169-5 Bielecki, Jan Poland
For the unit specifier, wants a different keyword for internal files, eg. FILE=. [UNIT= is in Fortran 77.]

Reject 172-5 Cox, Francis E. Swifte, UK
Reject 173-5 Cox, Francis E. Swifte, UK
Re-introduce [sic] the DO WHILE by:
loop-control= WHILE (*logical expr*)

Reject. 173-16 Cox, Francis E. Swifte, UK
Completely support the suggestions of Gerald M. Berns -- safety by removing implicit typing etc. [But we have to be compatible with Fortran 77.]

Reject 180-5 Jones, Russell K. Hughes
"There is no fundamental reason why the INITIAL attribute {DATA] must imply that a variable so declared is saved,..." "..., it is possible and desirable to be able to initialize local data objects at run-time for each instance of a subprogram."

Reject 182-3 Henderson, L.W. OSU
Permit the character $ in variable names. [Would be objectionable to ISO.]

Reject 182-4 Henderson, L.W. OSU
A pass descriptor mechanism for all data types. [We haven't felt it appropriate do standardize in this machine-dependent area. A descriptor is implied for some array and precision usage.]

Reject 182-6 Henderson, L.W. OSU
Wants case sensitive names.

Reject 184-1 Shyrock, Phil GTE South
Wants a self-indexing variable like the C ++x for x=x+1.

Reject 188-4 Van Snyder, W. JPL
Suggests using, in the Standard, CASE and IF, with a constant expression, as a compiler directive. This would, among other things, require the acceptance of duplicate statement numbers. [Better would be to restore the compiler directives that we had once.]

Reject 188-7 Van Snyder, W. JPL
Add a "standard messages" file [and unit]. [We discussed this at great length some years ago.]

Reject 192-6 Mathews, P.D.Jr Draper Labs
"The proposed standard document should use marks in or next to the text to clearly indicate additions and changes to FORTRAN 77."

Reject 193-1 Mackrell, D.K. UK
Wants C and D as defaults [abbreviations] for COMPLEX AND DOUBLE PRECISION.

Reject 193-2 Mackrell, D.K. UK
Wants a BYTE data type, like DEC. Apparently its only merit is that you can equivalence both a real and a character item to it.

Reject 193-4 Mackrell, D.K. UK
Wants language to RETURN down a list of previous subprogram references.

Too Large 57-4 Norris, A.W.Jr NSWC
Argues that internal procedures are redundant.

Too Large 57-6 Norris, A.W.Jr NSWC
Keyword arguments would generate more problems than the facility is worth.

Too Large 57-7 Norris, A.W.Jr NSWC
"...this new Format [source form] must be rejected."
Compatibility with the old form.

Too Large 60-1 Synge, J.M. UWashington
Expensive compilation a deterrent to its use. But likes modules and derived type.

Too Large 70-1 Ragosta. A.E. DOA Ames
"...far to much change. We need to improve FORTRAN, but we don;t need another Ada.

Too Large 70-2 Ragosta. A.E. DOA Ames
"...add greatly to the complexity while adding relatively little to the functional capability of the language (eg, user-specified precision)." Microcomputers, discourage small firms, delay new compilers, degrade efficiency.

Too Large 70-8 Ragosta. A.E. DOA Ames
User-specified precision ... "with a benefit to only a few users." Remove it, keep double precision, and add quad precision.

Too Large 70-10 Ragosta. A.E. DOA Ames
"I find little reason why array sections should be necessary. Instead, an intrinsic function to convert subscripts for a section..."

Too Large 70-18 Ragosta. A.E. DOA Ames
"We don't need a USE statement in FORTRAN."

Too Large 78-1 Ware, John Haverly S.
Concerned with the loss of the close contact with the physical environment due to minimizing storage association. Concerned about the efficiency of implementations -- cites internal subprograms.

Too Large 78-4 Ware, John Haverly S.
Doesn't like the CASE statement or the block DO.

Too Large 79-1 Hendricks, R
No compiler can be written that will produce efficient code. Only programmers will be able to understand it.

Too Large 80-1 Klammer, P.F. AMAX
"I am deeply disappointed by the negative responses to FORTRAN 8X which were expressed by the vendors, and which are probably representative of their customers. In a nutshell, the FORTRAN committee is being asked to preserve nearly all of old FORTRAN in any new FORTRAN. What a shame." ...

Too Large 83-1 Pyle, I.C. U York
"I fear that 8X compilers will be large, expensive and unwieldy, and the upwards compatibility from Fortran 77 will cause unexpected problems to users who are not conscious of their manufacturers' extensions to 77 that will be different in 8X." Advocates leaving Fortran 77 alone, and adding "textual extensions (in a supplementary, declarative, language) to give new information about structure and semantics."

Too Large 94-1 Supowit, A.J OSU
"...firm belief in the philosophy of not having change solely for the sake of change."

Too Large 94-3 Supowit, A.J OSU
"Many of the new features are useful but only to a limited number of users. They provide 'elegance' to the language. The cost of this elegance is paid for in execution time, compile time and learning time."

Too Large 127-1 McDonald, J.D. U Illinois
"... it would be so big that ... too expensive to implement on --... small ones [computers]. Less generally usable.

Too Large 136-1 Morse, Will PHP Petr
Fortran has two user communities, the heavy scientific users, and the large number of users that use it only occasionally to solve relevant problems. X3J3 has been addressing the first class. Needed are two languages -- Fortran 77 extended to meet the needs of the occasional user, and a new language, not called Fortran, for the first class.

Too Large 143-1 Stubblefield, P.
As presented [the FIB], "... the proposed standard appears too large, too complex, and too great a departure from FORTRAN 77." Presents a detailed analysis of features. In general, rejects features that do not provide, in his view, significant functionality. Included in this rejection are modules, interface blocks, operator functions and assignment subroutines, recursion, free source form, event handling, entity-oriented data typing. Objects to the deprecation of many items in the then deprecated list.

Too Large 144-1 Kuras, John C. Boeing
X3J3 "is trying to make Fortran everything to everyone."

Too Large 147-1 Fullerton, J. Logopedics
Array operations are a major candidate for exclusion from the core and provided as a package. Rather see a smaller language that is used than the good stuff (many additions suggested) of Fortran 8X. [Working from the FIB and a DECUS newsletter].

Too Large 148-1 Wrzesinski, D.T. Deere
The world doesn't need another Edsel.

Too Large 150-1 Payne, Fred R. U Texas
"Professional scientists, engineers and applied mathematicians who SOLVE problems need only essentially the current standard (FORTRAN 77)."

Too Large 155-1 Buckley, A. Canada CFWG
Still too large. Suggest deleting vector valued subscripts.

Too Large 156-1 Tran, Q. AFNOR
Wants to cut Fortran 8X radically, and to use the current /S8 as the base for a "modern" Fortran that can remove redundant and obsolete features. Presents a wish list of features to be in the truncated Fortran 8X.

Too Large 162-1 Monson, M.C. Deere
"...opposed to the replacement of FORTRAN by what is essentially a new FORTRAN language.

Too Large 175-10 Bierman, K.H. Bierman & A
Provides several pages defining "Too Big". His definitions don't match those used in developing the "compromise".

Too Large 192-1 Mathews, P.D.Jr Draper Labs
Too much of the language [his metrics -- unspecified] new (1/3), and learning all of this is too difficult to be cost-effective.

Deprecation	1-11	Beirman, K.H.	FEA

Likes the pre-compromise deprecation.

Deprecation	1-6	Bierman, K.H.	FEA
Deprecation	52-20	Rej, Mark D.	

Wants to keep non-integer DO control variables.

Deprecation	3-1	Sobel, B.A.	
Deprecation	18-1	Shea, D.J.	NCAR
Deprecation	22-1	Barr, K.D.	Barr Engr
Deprecation	24-1	Curtis, Ian	
Deprecation	25-1	McVey, G.R.	
Deprecation	27-1	Schrader, S.	
Deprecation	29-1	Millett, R. P.	Brigham Y
Deprecation	31-1	Feuchter, C.A.	
Deprecation	33-1	Kreider, J.F.	
Deprecation	35-1	Price, J.C.	USDA
Deprecation	41-1	Che, Mitchell	
Deprecation	43-1	Darling, L.D.	Singer
Deprecation	53-1	Loveless et al	SAIC
Deprecation	57-95	Norris, A.M.Jr	NSWC
Deprecation	81-7	Hoekstra, J.P.	Macatawa
Deprecation	94-2	Supowit, A.J	OSU
Deprecation	118-1	Lynch, H.L.	SLAC
Deprecation	127-2	McDonald, J.D.	U Illinois
Deprecation	145-2	Downward, J.G.	KMS Fusion
Deprecation	146-1	Tincknell, M.L.	Lawrence BL
Deprecation	154-1	Curtis, Alan	Harwell
Deprecation	162-2	Monson, M.C.	Deere
Deprecation	192-2	Mathews, P.D.Jr	Draper Labs

Never deprecate anything.

Deprecation	5-4	Higbie, L.	Gould

Deprecate more.

Deprecation	46-1	Berns, G.W.	SAIC

Make obsolete [get rid of now] implicit typing and spontaneous definitions.

Deprecation	52-36	Rej, Mark D.	

Delete the PRINT statement -- redundant.

Deprecation	57-1	Morris, A.M. Jr	NSWC

Don't deprecate DOUBLE PRECISION. [We no longer do.]

Deprecation	57-10	Norris, A.M.Jr	NSWC

Deprecate EQUIVALENCE given the new precision specifications.

Deprecation	62-2	Janes, Rob	Cummins

Don't deprecate DIMENSION.

Deprecation 67-2 Umscheid, Lou GFDL
Don't deprecate EQUIVALENCE

Deprecation 70-13 Ragosta. A.E. DOA Ames
Also 17, 28, 30, 31, 39, 42, 43
Deprecated features that should not be deprecated [pre-"compromise"] are:
- CHARACTER *len
- Named COMMON
- BLOCK DATA
- DATA STATEMENTS
- DIMENSION
- DOUBLE PRECISION
- 'C' and '*' comment indicator
- 'X' edit descriptor
- Storage association and EQUIVALENCE

Features that should be deprecated are:
- H edit descriptor
- All type-dependent intrinsics accept type conversion
  [Apparently means specific names, not generic.]
- Lexical comparison intrinsics
- Carriage control characters, use FF, LF formatting

Deprecation 134-1 Grayson, P. NNC UK
X3J3 "should make a concerted effort to convince the public that deprecation is a harmless device, and in so doing make it a less contentious issue. Deprecation (and Obsolescing) is different from removal.

Deprecation 144-2 Kuras, John C. Boeing
"..., the committee's proposal to 'deprecate' and eventually eliminate 70% of the old FORTRAN II constructs shocks me!"

Deprecated 147-3 Fullerton, J. Logopedics
"To allow the perpetuation of such horribly outdated language features as these any longer that absolutely essential is, in my belief, a criminal act because of the body of knowledge about problems associated with their use. [Working from the FIB and a DECUS newsletter.]

Deprecated 147-20 Fullerton, J. Logopedics
Deprecate the format statement.

Deprecated 147-22 Fullerton, J. Logopedics
Deprecate the IMPLICIT statement. [and default implicit]

Deprecation 163-10 Van Snyder, W. JPL
Deprecation 188-9 Van Snyder, W. JPL
Deprecate ATAN2. Allow ATAN to have two arguments.

Deprecation 167-13 Chan, Donald I R & D

"All Fortran 8X processors should entirely prohibit ugly old features..." and should be provided with a source translator.

Deprecation 167-14 Chan, Donald I R & D
We should get rid of numeric labels, statement functions, and execute-at-least-once DO groups [sic].

Deprecation 172-7 Cox, Francis E. Swifte, UK
Decrement the use of any non-local variable as an iterative DO-loop variable. These are EQUIVALENCE, COMMON, dummy arguments, or FUNCTION results. For optimization and safety.

Deprecation 172-10 Cox, Francis E. Swifte, UK
Deprecation 173-13 Cox, Francis E. Swifte, UK
Deprecate the H edit descriptor.

Deprecation 175-5 Bierman, K.H. Bierman & A
"The original notion of DEPRECATED should be reinstated. [i.e. pre-"compromise."] The features listed there are EVIL.

Deprecation 175-8 Bierman, K.H. Bierman & A
It is quite easy to build translators [Fortran 77 to Fortran 8X] so that Fortran 8X can be pure. "This seems a small price to pay for getting rid of .GE. and friends, and simplifying future parsers."

Deprecation 180-7 Jones, Russell K. Hughes
Remove a list of features [Now the obsolescent list] now.

Deprecation 192-3 Mathews, P.D.Jr Draper Labs
In particular, don't deprecate storage association. Argues that we should remove derived type and modules, and then would have storage association. [What about arrays and user-specified precision?]

Deprecation 193-5 Mackrell, D.K. UK
Don't make obsolescent arithmetic IF. One line of code has to be replaced with 5, or 8, or 10.

Deprecation 193-6 Mackrell, D.K. UK
Don't make obsolescent Do termination sharing one CONTINUE. "I use up to 8..."

No Problem 1-4 Bierman, K.H FEA
No Problem 9-7 Hartl, Heinrich FRD
Fix old event handling.

No Problem 1-5 Bierman, K.H. FEA
No Problem 52-22a Rej, Mark D.
No Problem 169-4 Bielecki, Jan Poland
Prefers END DO to REPEAT.

No Problem 2-3 Gay, D,M. Bell Labs
Does not want a subset.

No Problem 2-7 Gay, D.M. Bell Labs
Wants [likes] punctuation for the relational operators.

No Problem 5-3 Higbie, L. Gould
No Problem 9-8 Hartl, Heinrich FRD
Keep the computed GO TO. Permits optimization.

No Problem 6-1 Grimes, R.G. Boeing
Permit list-directed internal files.

No Problem 10-1 Einarsson, Bo Sweden
Avoid the use of square brackets in array constructors. [We not have escape characters.]

No Problem 11-1 Oshel, D.C.
Keep Go To. Referenced Dvorak.

No Problem 12-1 Deakin, J. Australia
Wish list of features in /S8, including CASE, relaxing the 72 [sic] position limitation of source, user-defined data types, in-line comments, continuation on the line to be continued, and internal subprograms.

No Problem 39-1 Meissner, L.P. USF
Defines the INDEX function to be more meaningful if the argument is null. [This was done in /S8.]

No Problem 45-1 Nagle, F.W. METEFOR
Needs vector and array operations. The METEFOR preprocessor can't handle necessary temps, but Fortran 8X can.

No Problem 52-2 Rej, Mark D.
Wants a derived data type. [Rej was commenting on the FIB]

No Problem 52-7 Rej, Mark D.
No Problem 169-2 Bielecki, Jan Poland
Angle brackets in IDENTIFY "unfortunate." They are removed in /S8-104.

No Problem 52-8 Rej, Mark D.
Change SHAPE to RESHAPE.

No Problem 52-9 Rej, Mark D.
Combine FORALL and WHERE.

No Problem 52-12 Rej, Mark D.
Change REAL_CHAR to EXPCHAR.

No Problem 52-18 Rej, Mark D.
Specify a minimum number of positions [free form source] that a processor must accept. [We did.]

No Problem 52-27 Rej, Mark D.
Add punctuation for the relational operators.

No Problem 52-31 Rej, Mark D.
In the OPEN ACTION= specifier, suggests READWRITE rather than BOTH. We have now READ/WRITE.

No Problem 52-34 Rej, Mark D.
Wants identifiers as unit names. Fortran has always had this.

No Problem 52-44 Rej, Mark D.
Flag non-Standard constructs. We require the capability.

No Problem 52-45 Rej, Mark D.
Wants intrinsics for date and time. We have them

No Problem 52-50 Rej, Mark D.
Provide language to let the user select a "piece" of a string. [Fortran 77, as well as /S8, has a substring notation.]

No Problem 54-1 Fortmann, T. E BBN Labs
Wants end of line comments.

No Problem 57-2 Morris, A.M.Jr NSWC
Based on the FIB, was concerned about the argument transmission of arrays, and the mixing of old and New arrays.

No Problem 59-9 LWG DOE
A RANGE facility is needed. We now have it.

No Problem 66-3 Hamaker, D.W.
Wants multi-level EXIT in DO. We now have it.

No Problem 66-4 Hamaker, D.W.
Wants the label on REPEAT [END DO] to check the label of its matching LOOP [DO].

No Problem 70-6 Ragosta, A.E. DOA Ames

Wants a limit of 2000 non-blank characters per statement. [Source form] [The limit is now 2640 characters, including blanks.]

No Problem 70-7 Ragosta, A.E. DOA Ames
Wants the limit for fixed cards to be 50. The limit is now 40.

No Problem 70-11 Ragosta, A.E. DOA Ames
Should specify the effect of a lower bound greater than a higher bound. [array specification] [We do, it is null.]

No Problem 70-12 Ragosta, A.E. DOA Ames
Wants punctuation for the relationals. We have it now.

No Problem 70-14 Ragosta, A.E. DOA Ames
Doesn't like REAL_CHAR. Now it is EXPONENT_LETTER.

No Problem 70-20 Ragosta, A.E. DOA Ames
Doesn't like variant types [structure]. [It is removed from /S8.]

No Problem 70-21 Ragosta, A.E. DOA Ames
Doesn't like FORALL. It is removed from the Standard.

No Problem 70-22 Ragosta. A.E. DOA Ames
Questions the list of dis-allowed statements for the termination of an old DO (other than CONTINUE and END DO). We allow his list.

No Problem 70-23 Ragosta, A.E. DOA Ames
Is concerned about conflicts between the old DO and the new. The two forms are combined. Suggests a syntax, including UNTIL.

No Problem 70-27,32 Ragosta, A.E. DOA Ames
Doesn't like name-directed. It is now the "famous" [infamous] NAMELIST.

No Problem 78-2 Ware, John Haverly S
The new INITIAL is un-FORTRAN like. It's gone -- DATA.

No Problem 84 Wichman, B.A. NPL UK
Various transgressions of the Standard must be detected always. Such a statement is now in Fortran 8X, but not "always".

No Problem 103-6 Storey, M.W.and NERC
Wants a PROMPT=. It is now there.

No Problem 120-1 Buckley, A.G. Canada
Recommends adding a direction argument to the character functions INDEX, SCAN, VERIFY. Add LEN_TRIM. The proposals were approved by X3J3.

No Problem 120-2 Buckley, A.G. Canada
Permit the DATA attribute on an aliased entity. I believe that this is now in Fortran 8X.

No Problem 122-1 Messina, P.C. DOE LWG
Add RANGE. We did.

No Problem 131-1 Van Snyder, W. JPL
No Problem 168-3 Gibbs, G. et al CRC UK
Wants PAD=YES to be the default. It now is.

No Problem 138-1 Harris, E.B. Martin Mar.
Fortran 8X needs to address the readability and self-documenting problem. In particular, names of more than six characters. We have done so. Names may be 31 characters, with a non-leading underscore. We believe that we have addressed the basic issue in many other ways, too.

No Problem 145-1 Downward, J.G. KMS Fusion
No significant blanks. Majority of code would not recompile. Cites DECUS, which had an incorrect description, and the example given was GOTO and GO TO.

No Problem 146-2 Tincknell, M.L. Lawrence BL
Likes free-form source, but wants unrestricted blanks and tabs where a single blank can be (we had this), and other things re blanks

No Problem 147-5 Fullerton, J. Logopedics
Need a facility to define enumerated type. Derived type should do it.

No Problem 147-7 Fullerton, J. Logopedics
No good reason for including bit data type. LOGICAL, with proper implementation, should work.

No Problem 147-8 Fullerton, J. Logopedics
I also favor the entity declarations. [Working from the FIB and a DECUS newsletter.] We have them, in a different form.

No Problem 147-11 Fullerton, J. Logopedics
Why stick to the 48 character set? We allow 7-bit ASCII.

No Problem 147-14 Fullerton, J. Logopedics
Doesn't like the wording on dual case. [Working from the FIB and a DECUS newsletter.] Current wording is better.

No Problem 147-16 Fullerton, J. Logopedics
Wants underscores to be significant. They are.

No Problem 147-17 Fullerton, J. Logopedics
Blanks should be insignificant. Right now (9/8/87) they are.

No Problem 147-21 Fullerton, J. Logopedics
Conditional handling stinks -- like PL/I. [Working from the FIB and a DECUS newsletter.] It is gone.

No Problem 148-7 Wrzesinski, D.T. Deere
No Problem 162-5 Monson, M.C. Deere
The ability to return to the operating system with a completion code only, not a message. STOP should do it, with a non-character stop code.

No Problem 150-4 Payne, Fred R. U Texas
Wants variable names with up to 13 characters. We allow 31.

No Problem 159-3 Van Snyder, .W. JPL
Concerned about the variant component of a structure. We have disallowed it in Fortran 8X.

No Problem 163-8 Van Snyder, W. JPL
The intrinsic conditions that can be named in a HANDLE do not provide sufficient resolution. The event handling text is not now in the Standard -- it is in Appendix F.

No Problem 166-2 Bej, Mark D.
Provides syntax combining name-oriented and entity-oriented declarations. X3J3 has done this in a different way.

No Problem 167-3 Chan, Donald I R & D
Allow lower case alphabetics. We do.

No Problem 167-4 Chan, Donald I R & D
Provide a looping construct for non-enumerative iterations. We do.

No Problem 167-10 Chan, Donald I R & D
Eliminate BIT -- it is merely LOGICAL with permitted packing. Bit is not now in the standard, and current proposals are going this way.

No Problem 168-5 Gibbs, G. et al CRC UK
Wants an intrinsic to find the state of an OPTIONAL argument. OPTIONAL arguments are no longer in the Standard.

No Problem 169-3 Bielecki, Jan Poland
Wants to change the syntax of WHERE so that the example is correct. I believe that, with the current syntax, the example is correct and does what was desired.

No Problem 169-5 Bielecki, Jan Poland

Wants DO FOREVER. Just DO means this.

No Problem 172-4 Cox, Francis E. Swifte, UK
Major Proposal 172-1 Cox, Francis E. Swifte, UK
Major Proposal 173-2 Cox, Francis E. Swifte, UK
On derived types, withdraw the intrinsic operators for (in)equality checking (.EQ.,.NE.). Replace them by user-written operators overloading .EQ., .NE.. The rationale is to avoid hidden overheads. Ada has this problem. This has been done.

No Problem 173-4 Cox, Francis E. Swifte, UK
For CASE, prohibit the form (:) to mean DEFAULT. We have.

No Problem 174-4 Jones, Russell K. Hughes
Permit list-directed access to internal files. This is in /S8-104.

No Problem. 175-1 Bierman, K.H. Bierman & A
No Subsets. "If it isn't in the smallest sanctioned subset -- it isn't really there."

No Problem 179-3 Reid, J.K. IFIP
Operator overloading should be retained. [It has been.]

No Problem 182-1 Henderson, L.W. OSU
The writer presented a wish list for a new Fortran. This list included the following items now in Fortran 8X: long names, structured loops, structured data, list-directed for internal files, intrinsic functions for date, time, CPU, etc., omitted arguments and a check for arguments present, 132 column source lines, lower case source code, IMPLICIT NONE, in-line comments, re-entrant subprograms.

No Problem 184-1 Shyrock, Phil GTE South
"Would like to see a REPEAT UNTIL construct." We have it in a different form.

No Problem 192-4 Mathews, P.D.Jr Draper Labs
"FORTRAN 8X does not have matrix operations. ... should be handled by a standard subroutine call interface ..." Fortran 8X has matrix intrinsic functions.

No Problem 193-3 Mackrell, D.K. UK
Wants the equivalent of BASIC VAL -- convert to or from a string to the binary internal representation. READ/WRITE of internal files does this nicely.

No Recommendation 4-1 Randolf, M.W. B & V
"Horrible language", so don't improve it.

No Recommendation 8-1 Lindsay, J.H. Canada
Wants documentation on deleted things [per 1984] like varying length character, bit, macros, and loop constructs.

No Recommendation 48-1 Humar, J.L. Canada
Supports the X3J3 continuing attempt to publicize its work. Sent a copy of his forum notes.

No Recommendation 52-16 Rej, Mark D.
"Modules -- I love them!"

No Recommendation 52-17 Rej, Mark D.
On procedure calls, are the values of expressions assumed to have the IN attribute?

No Recommendation 52-32 Rej, Mark D.
Asks about carriage control on a "disk" file. "Some processors require it."

No Recommendation 52-33 Rej, Mark D.
Asks if the assumed ENDFILE is required in Standard Fortran. It is not in Fortran 77, but is in Fortran 8X.

No Recommendation 52-38 Rej, Mark D.
Clarify how values that are too large for their format editing specification are to be printed. 'Tis clear.

No Recommendation 52-46 Rej, Mark D.
Asks about dual case -- "without regard to case".

No Recommendation 52-51 Rej, Mark D.
"I do sincerely applaud the Committee's great efforts to date [January 1985] to expand Fortran and to standardize the computational aspects of the language."

No Recommendation 57-5 Norris, A.W.Jr NSWC
Recursion is definitely needed.

No Recommendation 59-8 Norris, A.W,Jr NSWC
The new facilities in Section 2.0-2.11 [of the FIB] should be quite useful.

No Recommendation 65-1 Willis, A.B Sweden
We need a standard operating system interface to achieve true portability.

No Recommendation 60-3 Synge, J.M. UWashington
Can arguments passing the dynamic aliasing of portions of an

array address routines written in other languages?

No Recommendation 64-1 Broughton, C. Canada
"i'd like to congratulate ... capture the spirit of the previous versions of the language while extending it into the domains of latter-day concerns and concepts."

No Recommendation 67-1 Umscheid, Lou GFDL
Strongly endorse the array processing features, and, in particular, the ability to have an array-valued internal function.

No Recommendation 70-3 Ragosta. A.E. DOA Ames
"The terminology needs clarification throughout." [The date on the letter is March 1985.] "A scheme should be developed to mark the differences from FORTRAN 77 in the text." Many comments in the letter are questions and mis-understandings referring to the old line numbers, and cannot be addressed here.

No Recommendation 70-9 Ragosta. A.E. DOA Ames
"I don't think all operations should be done as functions. ..."

No Recommendation 70-34 Ragosta. A.E. DOA Ames
"I fail to see the reason for the RESULT phrase."

No Recommendation 70-36 Ragosta. A.E. DOA Ames
"The whole idea behind the ASSIGNMENT phrase escapes me." "... ('=') SHOULD DEFINITELY allow assignment to two like entities regardless of form."

No Recommendation 70-37 Ragosta. A.E. DOA Ames
INTENT, OUT is meaningless. "How can you enforce this and if you can't, why should the user be forced to specify it." [sic]

No Recommendation 70-44 Ragosta. A.E. DOA Ames
"The statement 'having been superseded by superior language features' is a matter of opinion and completely ignores the work that will be required to convert source codes and create new compilers so that YOUR artistic whims can be pandered."

No Recommendation 77-1 Marsh, R.J. CEGB UK
Don't change anything. Still using Fortran 66 and bothered by the GKS interface that is Fortran 77.

No Recommendation 78-3 Ware, John Haverly S
"The related topics of data structures, derived data types and modules have some good features." Concerned about efficiency.

No Recommendation 78-5 Ware, John Haverly S
"In general, loosening the input format is fine." [Didn't like multi-statement line.] "New features that I do like:

Standardizing the collating sequence ..., dynamic data storage, recursive subroutine calls, expanded inquiry functions, bit data type, and condition enable."

No Recommendation 81-1 Hoekstra, J.P. Macatawa
Concerned about efficiency. "it is reasonable, however, to assume that the use of any new feature may degrade response time." [I don't think that this is a reasonable assumption. However, some implementors may cause it to be true, like with Fortran 77.]

No Recommendation 81-1 Hoekstra, J.P. Macatawa
Compatibility is a major concern. "... it does not seem that one could recompile a FORTRAN-77 program and take advantage of one or two FORTRAN-8X features without a lot of work." [Now, in most cases, you can.]

No Recommendation 81-1 Hoekstra, J.P. Macatawa
"...your example of the use of the ENTRY statement was the first time I have ever seen a reasonable use of ENTRY."

No Recommendation 85-1,119 Zimmer, R. ISO
Long discussions about terminology re ISO 2382, Section 15.

No Recommendation 81-1 Cartledge, C.J. U Salford
The "formal set of principles" should be in the Standard.

No Recommendation 91-2 Cartledge, C.J. U Salford
"..., I am not convinced that in real world computing it [precision specifications] gives much more power than standardizing the REAL*4 etc.

No Recommendation 92-1,109 Aharonian, G. S T & O
"In a world where Fortran exists only through IBM and DEC compilers, it is obvious that future more powerful Fortrans are needed. Fortunately, the world in this sense is much bigger, and the need for future Fortrans is less obvious." Advocates withdrawing Fortran 8X in favor of Ada and C.

No Recommendation 103-4 Storey, M.W. and NERC
Asks what happens when one opens a file with the default ACTION=READ/WRITE and the file is read only to this user. Does the job fail, does the open fail, or does the job continue till a write is attempted, or maybe until a write buffer is flushed? [I believe that the OPEN should fail with an appropriate IOSTAT.]

No Recommendation 110-1 Hybl, Albert U Maryland
Omit the exclusions rule 1.3.2(4). This means define things a lot better, particularly in the I/O sections, and would invalidate currently Standard-conforming processors and programs.

No Recommendation 118-2 Lynch, H.L. SLAC
Let Fortran, "this barbaric language", die. Work on inter-language communication. Thus Ada, C, and perhaps a clean new Fortran, can do the good things we are doing.

No Recommendation 123-1 Wichman, Brian NPL UK
Presented three Ada packages, and asked if they could be formulated adequately in Fortran 8X. Jerry Wagener's response, #124, showed that they can.

No Recommendation. 129-1 Kruyt, Erik W. U Leiden
Presents a description of FORCHECK, a Fortran 77 verifier and programming aid.

No Recommendation 132-1 Bierman, K.H. FEA Inc
Referenced panel at ACM'85. "The enormous effort put in by your committee is reflected by the quality of the proposal, all hail".

No Recommendation 140-1 Pollicini, A. Italy
Extensive refutation of the IBM paper and presentation at the August 1985 SHARE meeting in New Orleans.

No Recommendation 144-3 Kuras, John C. Boeing
"I'm sure that the committee would find all of the support that it needs from the industrial users of FORTRAN if it were to attempt to curb the mainframe manufacturers from adding extensions to their "ANSI" compilers."

No recommendation 147-4 Fullerton, J. Logopedics
Why so many keywords? Look at PASCAL, Ada, or even C. They accomplish the same thing without half the keywords. This is like PL/I.

No Recommendation 157-2 Messina, P/C. DOE LWG
Process letter ballot [spring, 1986] quickly and get out for public review and comment.

No Recommendation 161-1 Van Snyder, W. JPL
This is a paper on "A Syntactic Principle to Support Abstraction"

No Recommendation 167-8 Chan, Donald I R & D
The inquiry functions are inelegant. HUGE doesn't operate on its argument in a normal way.

No Recommendation 169-1 Bielecki, Jan Poland
Many editorial changes suggested, questions asked.

No Recommendation 170-1 Whitfield, Doug I R & D
Several editorial problems with /S8, April 1986.

No Recommendation 175-1 Bierman, K.H. Bierman & A

"We need a new standard, and we need it soon. If we delay long enough FORTRAN might just die -- of course this might actually be the intent of some."

No Recommendation 175-4 Bierman, K.H. Bierman & A
Recursion means no call by address which means lose of run-time efficiency. Does the draft specify how argument passing is to be accomplished? Will it be compatible with calls to other languages?

No Recommendation 175-6 Bierman, K.H. Bierman & A
"It is sometimes necessary to read the entire file to parse a single statement. We should fix the language so that modern parsers can be used."

No Recommendation 180-2 Jones, Russell K. Hughes
The writer mis-understood and assumed that the character constant delimiters related to the list-directed and namelist DELIM specification, and gagged.

No Recommendation 180-3 Jones, Russell K. Hughes
The writer mis-understood and assumed that only one derived type was permitted in a program, and objected.

No Recommendation 188-4a Van Snyder, W. JPL
"The processor-defined non-zero values assigned th the status variable [IOSTAT] constitutes an impediment to portability."

No Recommendation 188-8 Van Snyder, W. JPL
"Can one restrict the visibility of a subprogram to a selected set of other subprograms?"

40

106(*)JTM-1

TO: X3J3

FROM: Jeanne T. Martin, Convenor ISO/IEC JTC1 SC22/WG5

SUBJECT: SC22 Resolutions / Other News from Washington

Attached is a draft copy of 42 resolutions adopted at the SC22 meeting, September 8-11 in Washington, D.C.

All except WP-2 were adopted unanimously. WP-2 is the resolution to register S8.104 as the Fortran DPS (draft proposed standard) with the ISO Central Secretariat. The Japanese abstained. The Chinese voted yes because of the existence of resolutions WP-34 and WP-40, particularly WP-40 relating to character sets.

The Europeans feel that if programming languages are going to accommodate Kanji, they can at the very least accommodate non-English Latin alphabets as well. Several documents were distributed regarding the standardization of character sets. I have attached some; others I will bring to Ft.Lauderdale for reference. Permission was granted to distribute the SEAS White Paper to standards committees although it is copyrighted.

For those who haven't heard, the final X3 vote on going public with S8 was 30-yes, 5-no, 2-abstain, 1-not voting. The nos were Data General, DEC, GUIDE, IBM, and UNISYS.

The public comment document will be available from Global Engineering for $50 in the US, $65 outside. This is a reduced size format - 2 pages of S8 to a page of the document. The full size price would have been $110.

The US 4-month public comment period could begin as early as October. ANSI determines the dates, not CBEMA. The international voting/comment period is 3 months and will be concurrent with the US public comment period.

217

DRAFT RESOLUTIONS TO BE ADOPTED AT THE
SECOND PLENARY MEETING OF ISO/IEC JTC1/SC22
WASHINGTON, D.C., U.S.A.
1987-09-08/11

Resolution WP-1: COBOL, 1989:1985, Addendum.

ISO/IEC JTC1/SC22 requests that ANSI provide working drafts of an addendum to ISO 1989:1985 to resolve identified ambiguities and make corrections for errors.

Resolution WP-2: Draft Proposal - Fortran

ISO/IEC JTC1/SC22 instructs its Secretariat to forward document ISO/TC97/SC22 N361, Final Working Draft Fortran Revision, to ISO Central Secretariat for DP registration and to circulate it for a three month letter ballot within SC22.

Resolution WP-3: National Standards

ISO/IEC JTC1/SC22 instructs its secretariat to solicit information from SC22 Member Bodies about existing and planned national standards related to SC22's area of work which have not been submitted for international standardization. The SC22 secretariat is further requested to compile and maintain this list of national standards projects and to circulate it to SC22 Member Bodies once a year.

Resolution WP-4: Electronic Mail

ISO/IEC JTC1/SC22 thanks Gerhard Schmitt for his work in implementing recommendation 2 of document N277 by developing electronic mail links with a number of members of the SC22 Advisory Group and working groups of SC22.

ISO/IEC JTC1/SC22 invites Gerhard Schmitt to continue this work, and encourages all delegates to the second SC22 plenary who have access to electronic mail to supply their electronic mail address.

Resolution WP-5: Binding Techniques for Languages: WG11

ISO/IEC JTC1/SC22 thanks Mr. Don Nelson for his work as Interim Convenor in revitalizing WG11, and appoints Mr. Nelson as its convenor.

ISO/IEC JTC1/SC22 requests WG 11 to:

(a) continue liaison with appropriate working groups in SC21 and SC24 (formerly SC21/WG2)

(b) develop where necessary and circulate to SC22 working drafts of existing work items, at the earliest possible time in order to encourage broad participation in WG11, and

(c) prepare and submit to the SC22 Secretariat a list of language bindings that will require SC22 individual language WG attention in the coming 18 months.

Resolution WP-6: Language Bindings

ISO/IEC JTC1/SC22 directs its convenors of SC22 WGs affected by the list of language bindings called for in Resolution WP-5X requiring SC22 WG attention to:

(a) consider this list as a priority item and either include language binding work in their respective work programs, or respond to SC22 Secretariat saying why this work would not be appropriate.

(b) appoint liaison representatives to the appropriate SC21 and SC24 WGs.

Resolution WP-7: Subdivision of Project 97.22.10 - Programming Language Ada

ISO/IEC JTC1/SC22 authorizes the subdivision of Project 97.22.10 as follows:

1.	97.22.10.01	Ada Language Standardization
2.	97.22.10.02	~~Addressing~~ Issues and Questions on the Ada Standard (N353)
3.	97.22.10.03	Uniformity of Ada Implementations (N354)

Resolution WP-8: NWI-Ada/SQL Language Interface

ISO/IEC JTC1/SC22 instructs its Secretariat to forward document N355, NWI proposal - Ada/SQL Language Interface, to the ISO/IEC JTC1 Secretariat for processing and, ~~its~~ if approved, subsequent assignment to ISO/IEC JTC1/SC22.

Resolution WP-9: Publication of TR 9547, Test Methods

ISO/IEC JTC1/SC22 instructs its Secretariat to forward document ISO/TC97/SC22 N ... (SC22/WG12 N 121 Rev. 2): Test Methods For Programming Language Processors: Guidelines For Their Development And Acceptability, to the ISO Central Secretariat for publication as Technical Report (type 3), assuming the DTR's approval by TC97.

Resolution WP-10: Draft Technical Report on Conformity Clauses

ISO/IEC JTC1/SC22 instructs its Secretariat to circulate document ISO/TC97/SC22 N ... (SC22/WG12 N 133 Rev. 1): Guidelines for the Preparation of Conformity Clauses in Programming Language Standards as Proposed DTR (type 3).

Resolution WP-11: Work Item 97.22.15.01 (WG12)

TRM Project 97.22.15, the subproject 97.22.15.01, Part 2

ISO/IEC JTC1/SC22 instructs its secretariat to request JTC1 Secretariat to delete from ~~the SC22 Programme of Work the project under work item relating to~~ "Guidelines For The Preparation of Programming Language Processor Specifications Which Facilitate Testing".

Resolution WP-12: Convenor of WG14, Programming Language C

ISO/IEC JTC1/SC22 acknowledges the work done by WG14 and thanks the acting convenors, Mr. Hersee and Mr. Plauger, for their efforts in developing a working draft and synchronizing the "C" standardization work between ANSI and ISO.

ISO/IEC JTC1/SC22 appoints Mr. Plauger as convenor of WG14.

Resolution WP-13: Appreciation - Convenors

ISO/IEC JTC1/SC22 thanks the convenors and project editors of its working and ad-hoc groups for their dedicated work thereby facilitating the advancement of the projects assigned to SC22.

Resolution WP-14: External Liaisons Concern

ISO/IEC JTC1/SC22 reconfirms its wish to have strong external liaisons and expresses its concern that the external liaisons are not working well.

ISO/IEC JTC1/SC22 asks the Chairman and the Secretariat to try to revitalize the appropriate external liaisons, especially those with IFIP and CCITT.

Resolution WP-15: Programming Language - SIMULA

Noting the Swedish National Activity Report ~~and the intention~~ to submit SIMULA for International Standardization, ISO/IEC JTC1/SC22 suggests that Sweden forwards the Swedish National Standard on the Programming Language - SIMULA to the ISO Central Secretariat for processing according to the "fast track" procedure.

Resolution WP-16: Liaison to ISO/IEC JTC1/SC21

ISO/IEC JTC1/SC22 accepts the offer of the U.S. Member Body to investigate providing the liaison representative to ISO/IEC JTC1/SC21 - Information Retrieval, Transfer and Management for Open Systems Interconnection (OSI).

Resolution WP-17: Appreciation - Mr. Holka

ISO/IEC JTC1/SC22 wishes to express its gratitude to Mr. Holka for a very interesting and thorough presentation of the problems encountered with character sets and the use of standardized character sets produced by ISO/IEC JTC1/SC2.

Resolution WP-18: Form Interface Management System

ISO/IEC JTC1/SC22 thanks the U.S. Member Body for the submission at an early stage, of the proposed NWI on Form Interface Management System (document TC97/SC22 N358) and instructs its Secretariat to circulate the proposed NWI for comments and for an SC22 letter ballot.

If the New Work Item Proposal for Form Interface Management System is approved by SC22, the SC22 Secretariat shall forward document 97/22 N358 to the JTC1 Secretariat for processing.

Upon approval of the New Work Item Proposal for Form Interface Management System by JTC1, and its assignment to JTC1/SC22, ISO/IEC JTC1/SC22 shall:

- establish a Working Group for Form Interface Management System with the terms of reference to be to coordinate the content of an International Standard for Form Interface Management System.

- appoint Mr. Dan Frantz as Convenor with ANSI providing secretarial support and project editor.

Resolution WP-19: Administration of Working Group for Form Interface Management System

ISO/IEC JTC1/SC22, subject to approval of the NWI on FIMS and its assignment to SC22:

(a) Requests ANSI to forward to the SC22 Secretariat for circulation within SC22 for comment and recommendation, working drafts of the International Standard, Form Interface Management System.

(b) Requests ANSI to make all appropriate efforts to see that the draft proposed standard reflects the consensus of SC22 comments.

(c) Requests ANSI at the appropriate time to provide the SC22 Secretariat with the draft proposed standard for circulation to the member bodies of SC22 for letter ballot on the DP.

(d) Instructs the Working Group on Form Interface Management System to review and comment on the content of the working drafts of the standard produced by ANSI; coordinate any contributions made by SC22 member bodies; and coordinate the comments received during the SC22 ballot period(s).

(e) Instructs this Working Group to establish liaisons with ISO/IEC JTC1/SC18/WG3, Document Architecture; WG5, Content Architecture; WG8, Text Description and Procedural Languages; WG9, User Systems Interface and Symbols, and with the WG's of SC21 which deal with Virtual Terminals and Open Distributed Processing.

Resolution WP-20: Call for Members of SC22 Working Groups

In accordance with document TC97/SC22 N296 and N310, ISO/IEC JTC1/SC22 invites its Member Bodies to nominate experts to the following Working Groups established at the SC22 Plenary meeting in Washington, September, 1987:

- WG15 - POSIX
- WG16 - LISP
- WG17 - Prolog

Resolution WP-21: International/National Synchronization of Standardization

ISO/IEC JTC1/SC22 directs its Secretariat to seek approval from the ISO/IEC JTC1 Secretariat for the following procedure for DP - registration; and upon approval to put it into effect.

"When a member body is responsible for the development of a language standard, ISO/IEC JTC1/SC22 directs its secretariat to submit to ISO Central Secretariat a draft language standard for DP registration if:

(a) previous draft(s) have been circulated to SC22 Member Bodies, and

(b) the SC22 language working group has recommended DP registration, and

(c) the responsible member body has requested DP registration."

Resolution WP-22: Basic, 97.22.11

ISO/IEC JTC1/SC22 directs its WG8 to prepare for registration as a DP a single document for an ISO BASIC standard which:

(a) Establishes by reference ANSI X3.113-1987, Programming Language BASIC, as one section of ISO BASIC

(b) Completes the definition of ISO BASIC with ~~a second~~ another section specifying the conformity rules and the subsetting rules of both ANSI X3.113-1987 and ECMA-116-19xx.

(c) States in an informative Annex that programs and implementations that conform to ANSI X3.113-1987 or ECMA-116 conform to ISO BASIC.

Resolution WP-23: Establishment of Working Group on LISP

ISO/IEC JTC1/SC22 establishes Working Group 16 - LISP with the following terms of reference:

- to coordinate the content of an International Standard for LISP.

SC22 assigns Work Item 97.22.22 to this working group, and refers WG16 to documents ISO/TC97/SC22 N266 and ISO/IEC JTC1 N16 for resolution of comments accompanying the letter ballot of the NWI proposal.

SC22 appoints Mr. Christian Queinnec as the Convenor of WG16, with AFNOR providing secretarial support, and ANSI providing Mr. Richard Gabriel and Mr. William Klinger as the project editors.

Resolution WP-24: Establishment of WG on Prolog.

ISO/IEC JTC1/SC22 establishes Working Group 17 - Prolog with the following terms of reference:

- to coordinate the content of an International Standard for Prolog

- to resolve the comments received in the ballot (ISO/IEC JTC1 N15)

The following project is assigned to WG17: 97.22.23

Mr. Roger Scowen is appointed convenor and project editor with BSI providing secretarial support.

Resolution WP-25: Liaison to ISO/IEC JTC1/SC24

ISO/IEC JTC1/SC22 appoints Mr. Jens Kolind (Denmark) as its liaison representative to ISO/IEC JTC1/SC24 - Computer Graphics.

Resolution WP-26: Establishment of Working Group 15 - POSIX

ISO/IEC JTC1/SC22 establishes Working Group 15 - POSIX with the following term of reference:

- Coordinate the content of an International Standard on the Portable Operating System Interface (POSIX).

Mr. Jim Isaak is appointed convenor, with the U.S. Member Body providing secretarial support and project editor.

Project 97.22.21 is assigned to WG15. The scope for this project shall be:

> The Standard will define a standard operating system interface and environment based on the UNIX Operating System (UNIX is a registered trademark of AT&T in the United States and other countries). The standard will describe the external characteristics and facilities of the operating system with emphasis on those functions and facilities that are needed in a variety of commercial applications. Initially the main focus of the standard will be on the C language interface to the operating system to support application portability at source level.
>
> This standard will consist of several parts.

The first part will provide a functional definition of the interface. Following parts will provide the bindings of the interface to different programming languages.

The initial version of Part I may heavily use C language features. These will be replaced by more abstract, programming language independent specifications in a later version.

Part II will consist of the description of the C language binding. Subsequent Parts may be added as a result of letter ballots indicating the need for other language bindings.

Resolution WP-27: SC22/WG15 Liaison to SC21/WG5

ISO/IEC JTC1/SC22 instructs its WG15 to establish close liaison with the SC21/WG5 concerning OSCRL and consider a joint meeting with the OSCRL Rapporteur Group.

Resolution WP-28: POSIX Administration

ISO/IEC JTC1/SC22:

(a) Requests the U.S. Member Body to forward to the SC22 Secretariat for circulation within SC22 for comment and recommendation, working drafts of the International Standard relating to POSIX.

(b) Requests the U.S. Member Body to make all appropriate efforts to see that the draft proposed standards reflect the consensus of SC22 comments.

(c) Requests the U.S. Member Body, at the appropriate time, to provide the SC22 Secretariat with the draft proposed standards for circulation to the member bodies of SC22 for letter ballot.

(d) Instructs the Working Group on POSIX to review and comment on the content of the working drafts of the standards provided by the U.S. Member Body; and coordinate the comments received during the SC22 ballot period(s).

Resolution WP-28A: Draft Proposal-POSIX

ISO/IEC JTC1/SC22:

(a) Requests the U.S. Member Body to forward document IEEE P1003.1, Draft 12, Standard for a Portable Operating System Interface for

Computer Environment (POSIX), to the SC22 Secretariat as soon as it is available.

(b) Instructs the SC22 Secretariat to forward that document to ISO/CS for registration as a Draft Proposal and to SC22 member bodies for comments.

(c) Requests all SC22 Member Bodies to begin immediate work on this project by considering document TC97/SC22 N363, and sending comments to the Working Group convenor (Jim Isaak, Digital ZK03, 110 Spit Brook Road, Nashua, NH 03062-2698.USA).

Resolution WP-29: Coordination with IEEE

ISO/IEC JTC1/SC22 requests the U.S. Member Body to inform the IEEE Standards Board of the formation of WG15, and the resolutions of this meeting associated with it (e.g. terms of reference, scope, administration and instructions). In line with the principles of operation of SC22 (SC22 N 168R), SC22 wishes to ensure that the outcome of this project will result in a single worldwide POSIX standard.

Resolution WP-30: OSCRL, SSI and Related Matters

ISO/IEC JTC1/SC22 instructs the SC22 Secretariat to inform JTC1 of SC22's willingness to consider any language-related existing or new work items (including OSCRL) arising from JTC1's reconsideration of OSCRL, SSI, and related matters.

Resolution WP-31: POSIX Trademark

ISO/IEC JTC1/SC22 thanks IEEE for relinquishing all claims to the trademark for POSIX.

Resolution WP-32: Contributions to Special Working Group (SWG) on Formal Description Techniques (FDT)

ISO/IEC JTC1/SC22 instructs its secretariat to forward document SC22 N249, A View of Formal Semantics, to the JTC1 secretariat as a contribution from SC22 to the discussions of the SWG on FDTs.

Resolution WP-33: SC22 Delegate to SWG on FDT

ISO/IEC JTC1/SC22 endorses the appointment of Mr. Brian Meek (U.K.) as its delegate to the TC97 SWG on FDTs, and urges its member bodies and WG convenors to send to Mr. Meek, by 1987 October 20, any comments or suggestions on the use of FDTs in language standards which they would wish him to take into account.

Resolution WP-34: Character Handling in Programming Languages,

ISO/IEC JTC1/SC22 requests its member bodies to communicate to the SC22 Secretariat their requirements for character handling in programming languages, including multi-octet character sets.

ISO/IEC JTC1/SC22 instructs convenors of its WGs to report to the next SC22 AG meeting the extent to which their work items could meet those requirements.

Schedule of Project Stages

Resolution WP-35: ~~Establishment of Targets~~

ISO/IEC JTC1/SC22 draws the attention of its working group convenors to document TC97/SC22 N404, Synchronization of ISO/ANSI Activities. ISO/IEC JTC1/SC22 directs its convenors:

(a) to develop in consultation with the related Member Body activity, a ~~set of targets~~ that reflects both national and international processing, and the coordination of these procedures

Schedule of project stages

(b) to monitor the accomplishment of these ~~targets~~ project stages and make suitable reports to SC22, ~~and~~

project stages of

(c) to provide these ~~target~~ schedules to the SC22 Secretariat for distribution to its member bodies.

Resolution WP-36: Convenor for Working Group 8 - BASIC

ISO/IEC JTC1/SC22 appoints Mr. Thomas E. Kurtz as convenor of its WG8, Basic with ANSI providing secretarial support and project editor.

Resolution WP-37: Appreciation

ISO/IEC JTC1/SC22 thanks:

* CBEMA, particularly Catherine Kachurik, for the excellent organization of the second plenary of SC22.

* Ms. Harway and Ms. Bruns-Thepaut for their excellent interpretation.

* Mr. Sorsen for keeping up with those delegates wishing to use the microphone

* Gwendy Phillips, Patti Steiner, Caressa Williams, and Katrina Gray for this help to the subcommittee in some unusual office hours.

* Mr. Brannon of ISO Central Secretariat for his assistance and his excellent presentation to project editors.

* Mr. R. Kearney for the excellent way he has conducted this SC22 meeting and kept all delegations in order.

* Mr. Joe Cote for the excellent secretariat support.

* Mr. Genuys, Mr. Follett, Mr. Meek and Mr. van Wingen for ensuring that the Ad Hoc groups completed their reports in a timely manner.

* Mr. Anton Bickle for his dedicated leadership of the drafting committee and to all members of the drafting committee for their contributions:

  - Mr. J. Kolind
  - Mr. Q. Tran
  - Ms. P. Schwartz
  - Mr. J. Cote
  - Mr. R. Scoven

* CBEMA, and Accredited Standards Committee X3, for the excellent reception.

Resolution WP-38: Special NWI Submission to JTC1 by Member Bodies

ISO/IEC JTC1/SC22 recommends that in the case where a draft proposal for a New Work Item has gained substantial support at an SC22 AG meeting, that:

1. the member body initiating the NWI be encouraged to send the proposal directly to JTC1 for approval

2. at the same time, the SC22 Secretariat issues a ballot, conditional on the approval of the NWI, for the administration of the NWI, assignment to an existing WG or creation of a new WG, designation of a convenor and secretariat, etc.

Where there is no clear evidence of such widespread support, SC22 recommends the use of an SC22 consultative ballot for approving the sending of a NWI proposal to JTC1.

Resolution WP-39: Appreciation - Japan

ISO/IEC JTC1/SC22 wishes to express its gratitude to the Japanese Member Body for kindly accepting to host the next Advisory Group meeti in Tokyo in October 1988.

Resolution WP-40: Multi-Octet Character Sets

ISO/IEC JTC1/SC22 instructs its Working Groups to ensure that future standards shall contain proper provisions for handling non-English, single-octet, and multi-octet character sets.

Resolution WP-41: Appreciation - Dr. Klensin

ISO/IEC JTC1/SC22 expresses its appreciation to Dr. John Klensin for his excellent work as the PL/I Project Editor. He has clarified the particularly difficult situation regarding PL/I and provided SC22 with a viable solution to the problems identified by the commentors such that the document can proceed to registration as a Draft Proposal.

Resolution WP-42: Registration of PL/I Document as Draft Proposal

ISO/IEC JTC1/SC22 instructs its Secretariat to forward document TC97/SC22 N290 to the ISO/CS for registration as a Draft Proposal after changing the title to "Programming Language - General Purpose PL/I".

ISO/TC97/SC22
Languages
Secretariat: CANADA (SCC)

N357

JULY 1987

TITLE: Report from the SC22 liaison representative to SC2, describing the status and relationship between SC2 character set standards and draft standards.

SOURCE: Secretariat ISO/TC97/SC22

WORK ITEM: N/A

STATUS: NEW

CROSS REFERENCE: 97/22 N169(Recommendation 19)

DOCUMENT TYPE: SC2 liaison report to SC22

ACTION: FOR REVIEW BY SC22 MEMBER BODIES. THIS DOCUMENT WILL BE DISCUSSED AT THE FORTHCOMING SC22 PLENARY MEETING.

Address reply to: ISO/TC97/SC22 Secretariat
J.L. Côté, 140 O'Connor St., 10th Floor
Ottawa, Ont., Canada K1A 0R5
Telephone: (613)957-2496 Telex: 053-3336

SEP - 5 1987 JC

SOURCE : T. HOLKA

SUBJECT : SURVEY OF THE SC 2's STANDARDS RELATIVE TO CHARACTER SETS AND COMMANDS.

REF : Resolution 19 of the SC 22 Advisory Group Meeting

(Vienna - November 1986)

The attached document shows that the coding system is based today on ISO 2022 and cannot be directly used by programming languages.

As this system and these codes exist and are or will be used, it seems necessary to determine a method for using the system and the codes.

Is that work to be done by SC 22 ? answering at that question is not obvious, SC 22 has at least to provide a set of requirements.

Anyway, the issue seems to be important enough to be submitted to the next SC 22's plenary.

Note : Even if it is not intended here to recommand any position to SC 22, the attached document tries to give some clarifications and comments that may help during the SC 22 discussions.

THE ISO/TC 97/SC 2 STANDARDS RELATIVE TO CHARACTER CODING

AN INTRODUCTION TO THE "ISO 2022 WORLD"

As there is no philosophy explicitly provided by SC 2, trying to define such a philosophy by examining the set of standards already existing, under development or in project, is a rather difficult duty.

It is anyway possible to distinguish four classes of standards :

- a shopping list of commands : 6429 (see 1.2) ;
- methods ie 646 or 2022 ;
- free-standing codes, such as 6937 and each part of 8859 ;
- application of some codes to media.

Relationship between these four classes are not always and not clearly established.

The ISO 2022 rules and the attached code tables registration system, as described below, allow such combinations that the free-standing codes are not so free-standing as they seem to be, that has as a consequence to be a little confusing for a non SC 2 expert.

A last point has to be raised : In the opinion of SC 2 itself, these standards have been designed for the purpose of interchange, and may then not be appropriate fot the purpose of processing.

Note : Trying to be exhaustive, two standards need to be mentionned.

- 6936 : Transcodification between some ISO codes and the CCITT ITA 2 (International Telegraphic Alphabet number 2).

- 2047 : Graphic representation of control characters
  These two standards do not belong to one of the four classes already defined above.

1. GENERAL CONSIDERATIONS ON THE CODED ENTITIES

In the concerned area, limited to characters and to the more general use of characters, two families of entities are coded by SC 2.

- graphical signs, name given by SC 2 experts to a character which is juged too restrictive (ie A, B but also commas, # etc...) ,

- commands, also called control characters (ie blackspace, carrige return).

1.1 Graphical signs

It is clear that SC 2 has never made a choice between coding meanings or coding shapes. It is possible to find, in the same standard (ie 6937/2 or 8859/1) exemples of the two possible options (see annex 2).

234

It is obvious that processing needs a coding based on meanings. Adopting this option will facilitate the graphic sign selection and the code tables organisation. Even if it is necessary to complete these standards by shape coding oriented ones providing also relationship with meanings, these two types of standards may be used for interchange, depending on the purpose of the interchange.

1.2 Commands

The 6429 standard provides an organized list of commands, it seems that an important distinction has been omitted, and that it is necessary to provide a supplementary criterium defining simply two classes :

- Classe 1 : the commands having a permanent impact on the text

  For instance commands specifying character size and shape, characters or character strings positionning and some other attributes like colour ; code table selection commands are also part of this class.

  Only this class of commands has to be maintained in the command part of the code tables (see below for definition of code tables).

- Classe 2 : the other commands

  The transmission control characters (ie ACK), the device control characters (ie commands to activate a printer attached to a visual display terminal) are of course in this class, as also some commands affecting immediately the text like "insert, delete", these commands, once excecuted, have not to be kept with the text.

  These commands need also to be interchanged, but their nature is different and it is necessary to define for them another coding structure or level.

  This important distinction shall require some reconsideration of the existing standards to be applied.

2. THE "ISO 2022 WORLD"

ISO 2022 defines implicitly an architecture.

2.1 Base principles of the ISO 2022 architecture

The sytem is based on the use of a set of 128 positions (7 bit) code tables.

These code tables having two possible status.

- a permanent status, as an element of a library of code tables, the number of tables is unlimited ;

- an active status depending on the application, 1 to 4 tables may be active simultaneously. A special care has to be taken to the first of them numbered 0 (zero).

Each of these code tables is divided into two independent parts who can be selected separately.

- Part 1 : commands, the C sets ;
- Part 2 : graphics, the G sets.

Notes : 1. The code tables library contains as a matter of fact a set of C sets and a set of G sets.

2. Only two C sets can be active.

As illustrated in figure 1, SC 2 has defined two different types of C sets and G sets, the combination rules are not so simple.

Note : The coding technique is independent of the type of the table.

Examples : 8859 : a biunivoque relationship between a character and an element of code.

6937 Part 2 : a character is coded by using a combination of code elements.

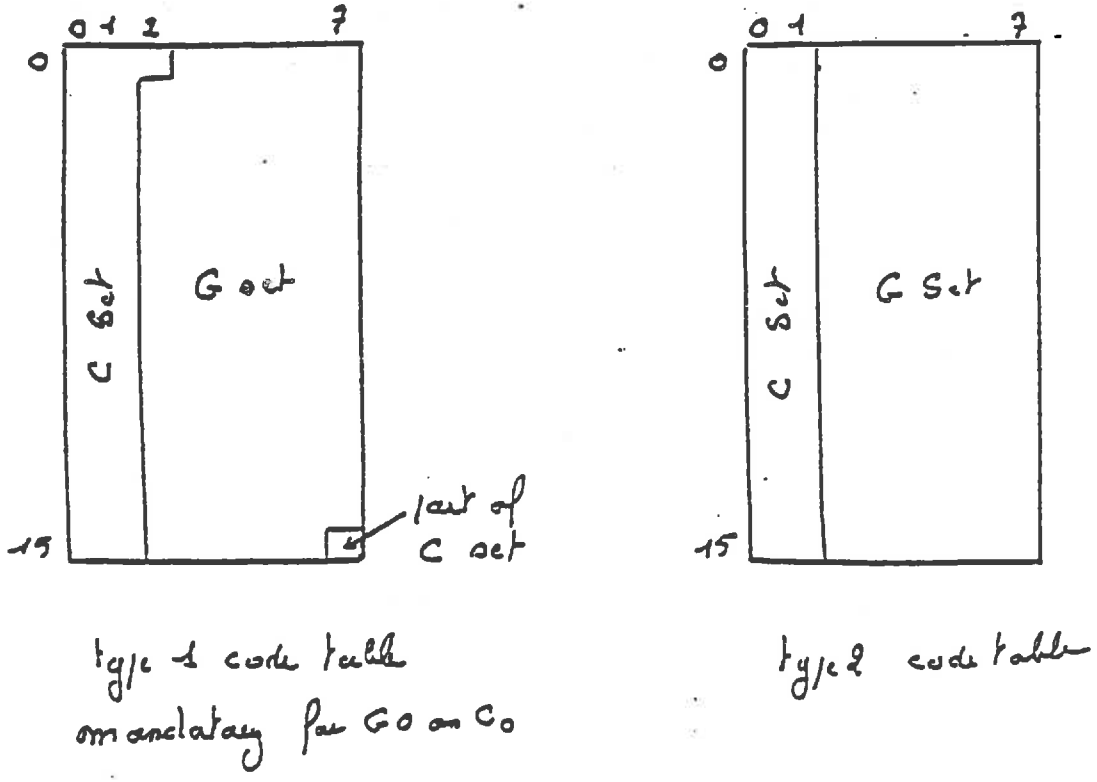


FIGURE 1 : Code tables types

236

2.2 The ISO 2022 architecture

Fig. 2 illustrates how the ISO 2022 system works,

Fig. 3 shows the relative position of the different standards with regard to 2022.

this architecture describes a system made of two parts :

- a driver ;
- the different areas controlled by the driver.

2.2.1 The driver

It is composed of two standards and a register.

ISO 2022 : Information processing - ISO 7 bit and 8 bit coded character sets code extension technique.

The standards provides of description of the code tables (see 2.1) and two commands.

- The Escape command : to select one of the code tables in the library and make it active in one of the four (4) possible positions.

An Escape command remains valid until the occurence of an other escape command on the same active code table position.

Note : This command may also be used to refer to other codes tables than the ones described in 2.1, but there is no standardized reference system in that case.

- The shift command : to select on of the active code tables, two shifts exit as a matter of fact :

  - locking shift, valid until the occurence of an other locking shift.

  - single shift, valid only for the next element of code (ie character).

Notes : 1. ISO 2022 does not provide any indications about the initial conditions,

2. Even if the number of active code tables is limited, it is possible to activate at any time a new table by substituting it to one of the active tables,

3. No restriction is made concerning the contents of the set of active tables, that has as a consequence that a same character may have more than one code.

ISO 2375 : Data processing - procedure for registration of escape sequence.

With the procedure, this standard describes the register.

Register : it provides a bi-univoque relationship between a code table (G or C sets) and the escape command parameters identifying the code.

The register is maintained by ECMA on behalf of ISO/TC 97/SC 2.

Note : The register is designed for the registration of C and G sets, but it exists escape commands going from the character world to other worlds. It seems necessary to undertake a systematic study of what are the other worlds and how to organise them.

A redesigning of the register may be a solution for that issue.

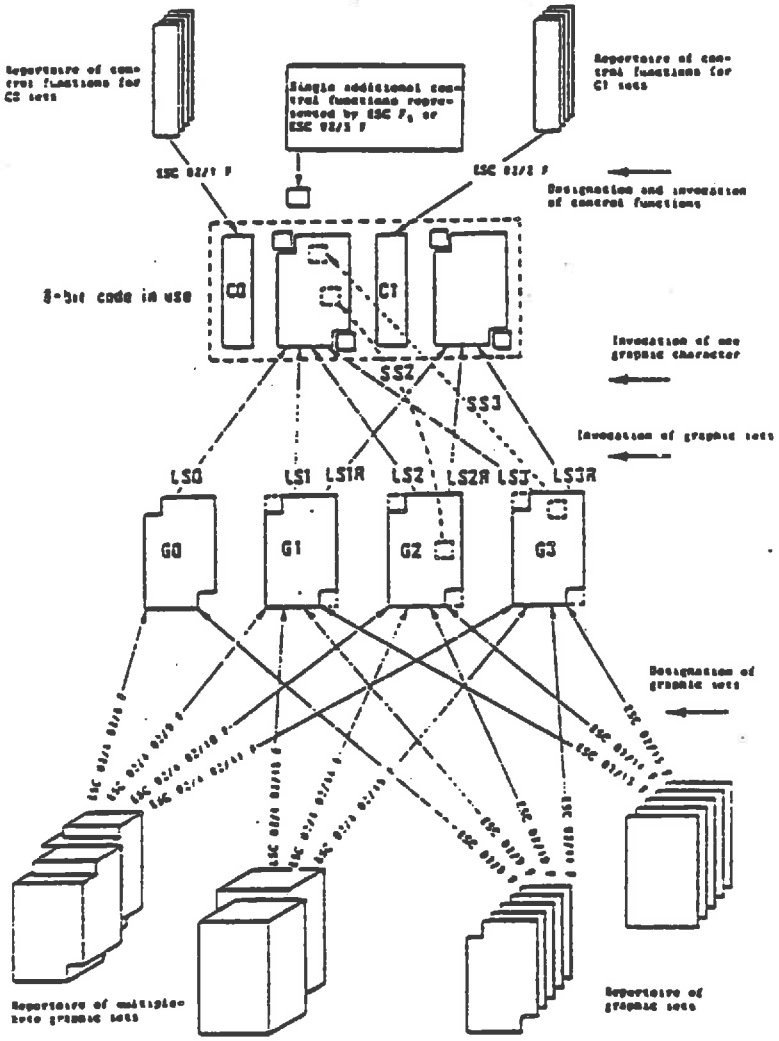


FIGURE 2 : Code extension in 8-bit environment
(Showing all shift facilities)

238

2.2.2 the different areas

Three of them are identified

- 7 bit codes
- 8 bit codes
- 7 or 8 bit codes

A X bit code simply means that the code element has X binary digits.

2.2.2.1 The seven bit codes

Latin alphabets. The principles and rules of coding are described in the following standard :

ISO 646. Information processing - ISO 7 bit coded character set for information interchange.

- remark : it was the first standard produced by TC 97.

This standard is a standard of method, it defines type 1 code tables (see 2.1) intended to be used ALONE, comprising a G set and a C set.

Note : ISO 2022 allows the use of the G set as one of the possible supplementary tables (up to 3).

This standard is an extension of the telegraphic code and preserves.

- The coding principles : a bi-univoque relationship between a code element (position in the table) and a graphic sign or a command.
- A base set of graphic signs and commands that cannot be changed.
- The adaptation system to local (national) or specific needs by defining rules and code table positions where the needed graphic signs or commands may be substituted to the existing ones.

ISO 646 also provides a complete code table IRV (International Reference Version).

IRV is not used in pratice by applications, what is used is a Version of ISO 646.

ISO 646 Versions

These standards apply ISO 646 rules to define a code table, the version may be National version (ie ASCII, the USA version of ISO 646) or specific to some requirements.

To be visible thru ISO standards, these versions must be registred in conformance with the 2375 rules.

Notes : 1. However ISO 646 has been defined for latin alphabets, it is possible to find in the register latin-other languages code tables (ie latin-greek) or pure other alphabets (ie.greek).

It is obvious that these tables do not conform with the ISO 646 substitution rules (only 12 positions allowed to substitution).

2. In the case of very large character sets like the japanese or chinese ones already registred, the ISO 646 substitution rules are not followed, but also the ISO 2022 rules for code table constitution are not followed either.

Non Latin alphabets

A single standard ISO 9036

This standard is the equivalent of ISO 646 for Arabic alphabet, no national or specific version are registred up to now.

Application to medias

	ISO 962	Tapes
	ISO 1113	Punched tapes
*	ISO 1177	Transmission
	ISO 3275	381 mm Magnetic cassettes
	ISO 6586	Punched cards

* Not developped by TC 97/SC 2.

2.2.2.3 Eight bit codes

Preliminary remark : what does exactly mean an 8 bit code in a system based on 128 positions tables (7 bit).

Simply that, to use the first supplementary table their is no need for a shift command, the table is identified by setting to one the first bit of the octet.

The area is composed of two standards :

- a standard of method 4873
- a standard of coding 8859

1) ISO 4873 - ISO 8 bit code for information interchange. Structure and rules for implementation.

It differs from ISO 2022 by providing some restrictions to the ISO 2022's functionalities :

- a base for the C0 set,
- a base for the G0 set,
- a same graphic or a same command cannot be present in two different tables.
- once the four active tables have been selected it is not allowed to change one of them.

No command has been provided to select by a single command a complete 4873 version (1 to 4 tables).

2) ISO 8859 - 8 bit single byte coded graphic character sets.

A multiple part standard, each part defining a complete code for graphic signs only.
Each code is composed itself of two parts :

- As G0 a type 1 table : ASCII
- As supplementary table, a type 2 table.

Notes : 1. 8859 part 1 is also known as 8 bit ASCII.

2. A combination of the ASCII code table with two or three of the supplementary tables is always valid when following the 2022's rules, but may not be valid according to 4873's rules.

The coding system of 8859 is a bi-univoque relationship between a graphic sign and an element of code.

2.2.2.3 Seven and eight bit codes

A single standard in that area : ISO 6937

This multi-part standard includes its own rules, and often specific rules for each part.

Some parts refer only to graphic signs, the basic one beeing

6937 part 2 : Latin alphabetic and non-alphabetic graphic characters

It offers a repertoire of more than 350 signs coded into two tables. The repertoire was defined by selecting all the signs existing on the most common type-writers keyboards.

The coding system is a combination of one to three code elements depending on the environment (7 or 8 bit).

Other graphic parts provide extensions to part 2 by defining :

- specialized area (ie scientific, editing)
- other alphabets (Greek, cyrillic)

The principle of coding is then a bi-univoque relationship between a sign and a code element.

Some parts are purely commands, like

- Part 3 - commands for Page Image Format
- Part 4 - commands for processable text Format.

ISO 6937 part 2 has a companion standards : ISO 7350

This standards defines a register of 6937/2 subrepertoires.
the register is maintained by NCC. There is no standardized system to identify each element of this register.

2.3 GENERAL APPRECIATION ON THE 2022 SYSTEM

- Even if some standards are arguing of an 8 bit coding the system is based on a SEVEN BIT CODING.

- If more than 190 signs are needed the use of escape and shift commands implies a variable length coding.

- The code tables possible arrangements do not guarantee that a same sign has always the same code in the same application.

- The small size of the code tables does not facilitate the use of very large character sets (ie chinese).

As a consequence, the ISO 2022 system is not suitable for processing and some supplementary methods are at least necessary to its use by programming languages.

This system has an advantage, it allows in theory the coding of an infinite character set.

Relative Positions of SC 2 Standards

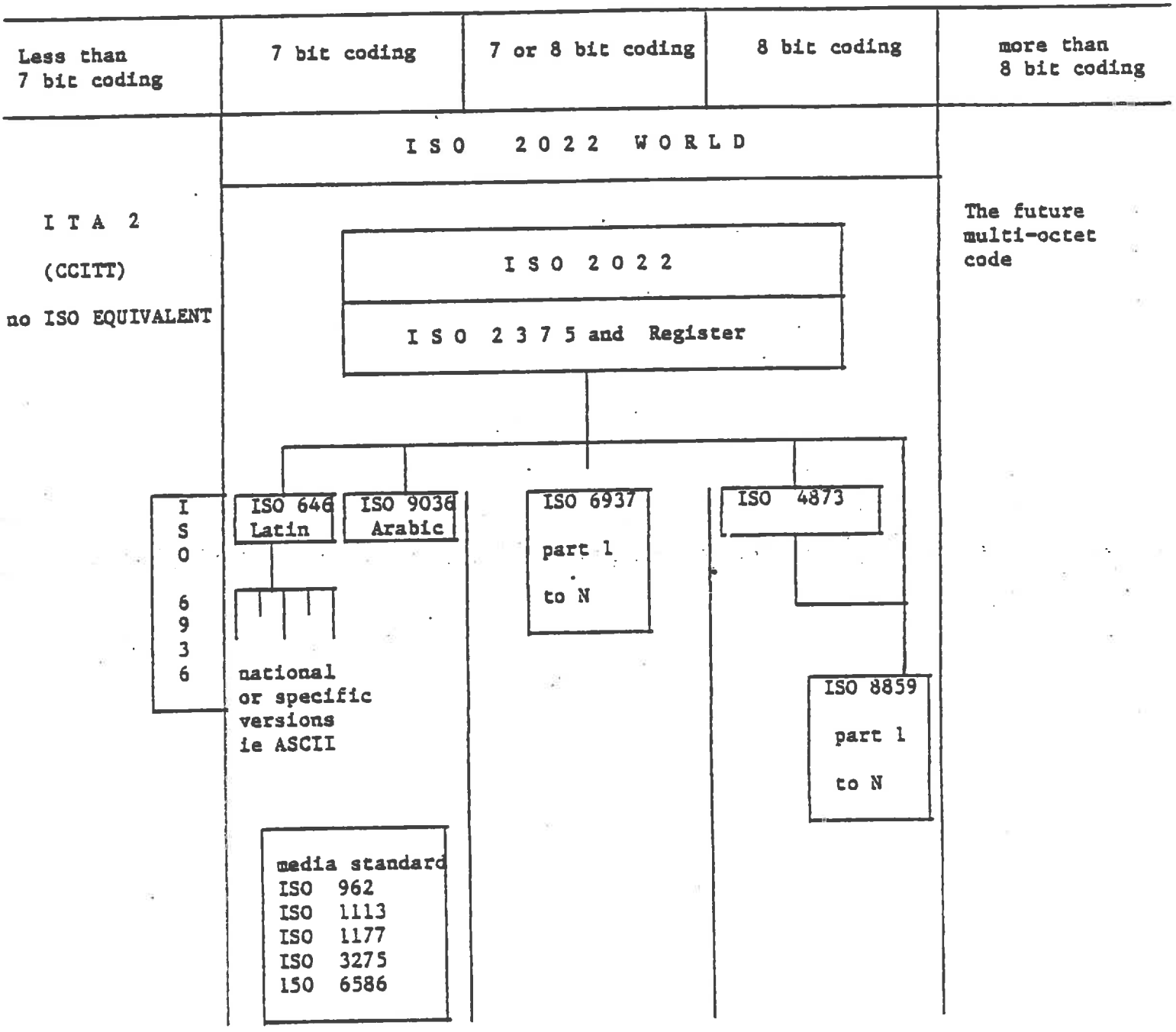


FIGURE 3

243

3. THE MULTI OCTET-CODE

This new code will be once more, an interchange code, the main purpose being to facilitate the interchange of very large character sets.

Two secondary objectives have been added :

- to remain as compatible as possible with the current standards driven by ISO 2022 ;
- to transmit the code by using 8 bit (octet) code elements.

The question is : will that code be more adequate to the processing requirements than the already existing ones ?

To provide a response is not so easy :

- to remain compatible with existing codes will perpetuate the current system of selecting the graphic signs and the commands (ie coding shapes or meanings),
- to transmit 8 bit code elements has as a consequence a variable-length coding for the interchange.

These two points bring rather a disadvantage for processing; but :

- to suppress the escape and shift commands ;
- to provide a single coding for each graphic sign or command ;

  is rather an advantage.

It seems to be very important for the SC 22 business to create a close liaison with SC 2/WG 2 developping this new standard.

Note : The proposed code table organisation provides less 36.000 usable positions in a 256 x 256 table (more than 64.000 see fig. 4).

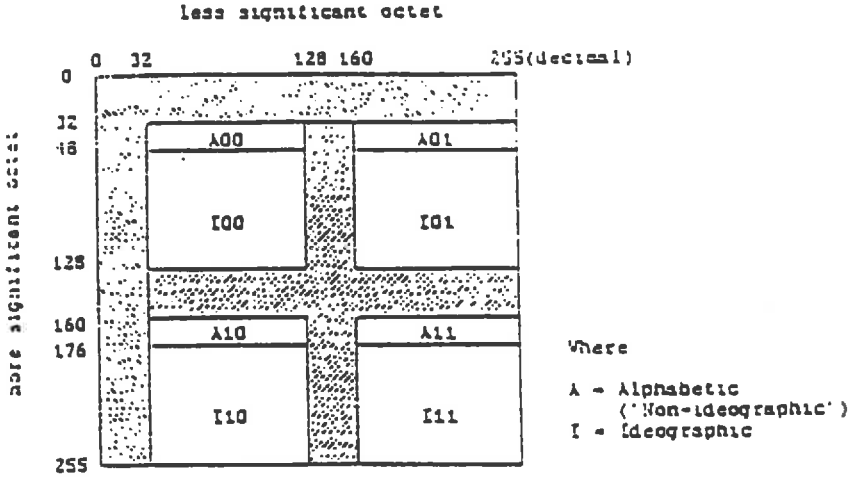


FIGURE 4 : The multi octet code code table

244

4. RELATIONSHIP BETWEEN THE CHARACTER CODED AREA AND OTHER AREAS

At present SC 2 is dealing will two others of these areas :

- image coding itself subdivided into two parts,
  - graphic coding,
  - "photographic" coding,
- audio information coding.

Each of these areas or sub-areas being by itself the equivalent of the character coded one.

The purpose here is not to provide an analysis of the standards developped within each of these areas, but the simple fact they exist raises a question :

How to know which of these areas I am facing to ?

That is specially important in the case of a mixed content document.

SC 2 does not provide really a solution to that issue and is not studying any system to allow the identification of any possible new area that might be required.

CONCLUSION

It is obvious that the adequacy of the existing or under development codes does not exactly match the requirements programming languages can have.

But since these requirements are not already expressed, and that the different standardized codes are more or less used, SC 22 should undertake the preparation of :

- a detailed expression of requirements in this area ;
- a method for using the existing or under preparation codes

In so doing it might be of some interest to consider the codes used by the already commercialized micros.

ANNEX 1

SOME CONSIDERATION ON SC 2 TERMINOLOGY

Graphical character

A sign having a graphical representation basicaly defined when representing a text an a shit of paper or a screen.

Includes letters, decimal digits, punctuation, etc ...

Command character

Has no graphical representation, it identifies a command, the command may have some effect an the global text representation (ie CR: carriege return).

Note : In many Programming language standards a character is simply a fixed length bit string.

Note : SPACE is considered by SC 2 as being a command character and a graphical character.

Escape : ESC

Is not considered by SC 2 as being a command.

Note : The idea an concept of active code table used within the paper is not an SC 2 concept.

In the SC 2 terminology ESC invoques a code table which is then known by the application.

ANNEX II

CODING SHAPES OR CODING MEANINGS

Preliminary note :

Shape has to be used with care, what is considered as "shape" here is very generic, the letter A or a commas have a generic shape.

Their is an other level of shape which is the detail of the font used for printing, that level is not considered within this paper.

Coding shapes (restricted to generic shape)

Definition : Two characters having the some generic shape have a some code.

ie. - Capitol Latin A and Capitol Greek alpha
- Minus sign or dash or hypen.

This coding is appropriate for on imaging devices.

Coding meanings

Definition : give a different code to each different meaning without consideration to the shape.

This coding is appropriate for processing but needs a complement for imaging.

ie a different code for minus, hypen, dash.

Note : The distinction between shape and meaning may be some times a little tricky.

ANNEX III

LIST OF SC 2 STANDARDS

(ISO catalogue 1987)

646 - 1983	Information Processing - ISO 7 bit coded character set for information interchange.
962 - 1973	Information Processing - Implementation of the 7 bit coded character set one its 7 bit and 8 bit extensions on 9 track 12,7 mm (0,5 in) mangetic tape.
963 - 1973	Information Processing - Guide of the definition of 4 bit character sets derived from the 7 bit coded character set for information processing interchange.
1113 - 1979	Information Processing - Representation of the 7 bit coded character set on punched tape.
1177 - 1985	Information Processing - Character structure for start/stop and synchronous character oriented transmission (nost an SC 2 standard).
1672 - 1977	Hardware representation of ALGOL basic symbols in the ISO 7 bit coded character set for information processing interchange.
2022 -1986	Information Processing - ISO 7 bit and 8 bit coded character sets - Coded extension techniques.
2047 - 1975	Information Processing - Graphical representations for the central characters of the 7 bit coded character set.
2375 - 1985	Data Processing - Procedure for registration of escape sequences.
2530 - 1975	Keyboard for international information processing interchange usinf the ISO 7 bit coded character set - Alphanumeric area.
4873 - 1986	Information Processing - ISO 8 bit code for infoamtion interchange - Structure and role for implementation.
6429 - 1983	Information Processing - ISO 7 bit and 8 bit coded character sets - Additional control functions for character - imaging devices.
6586 - 1980	Data Processing - Implementation of the ISO 7 bit and 8 bit coded character sets an punched cards.
6936 - 1983	Data Processing - Conversion between the ISO 7 bit coded character set (ISO 646) and the CCITT international alphabet N° 2 (ITA 2).

ANNEX III

LIST OF SC 2 STANDARDS

(Suite)

6937/1- 1983	Information Processing - Coded character sets for text communication - Part 1 : General introduction.
6937/2-1983	Information Processing - Coded character sets for text communication - Part 2 : Latin alphabetic and non-alphabet graphic characters.
Note :	Other parts are at different levels of approval.
7350 - 1984	Text communication - Registration of graphic character subrepertoires.

STANDARDS UNDER DEVELOPMENT (ISO technical programme 1987

- DP or DIS LEVEL)

6937/3	Information Processing - Coded character sets for text communication - Part 3 : Control function for page image format.
6937/5	Information Processing - Coded character sets for text communication - Part 5 : Scientific and Technical graphic characters.
6937/6	Information Processing - Coded character sets for text communication - Part 6 : Publishing and base discussing graphic characters.
6937/7	Information Processing - Coded character sets for text communication - Part 7 : Greek graphic characters.
6937/8	Information Processing - Coded character sets for text communication - Part 8 : Cyrillic graphic characters.

Note : other parts are under development or in project.

249

ANNEX III

LIST OF SC 2 STANDARDS

(Suite)

8859/1	Information Processing - 8 bit single byte coded graphic character sets - Part 1 : Latin alphabet N° 1.
8859/2	Information Processing - 8 bit single byte coded graphic character sets - Part 2 : Latin alphabet N° 2.
8859/3	Information Processing - 8 bit single byte coded graphic character sets - Part 3 : Latin alphabet N° 3.
8859/4	Information Processing - 8 bit single byte coded graphic character sets - Part 4 : Latin alphabet N° 4.
8859/5	Information Processing - 8 bit single byte coded graphic character sets - Part 5 : Latin/Cyrillic.
8859/6	Information Processing - 8 bit single byte coded graphic character sets - Part 6 : Latin/Arabic.
8859/7	Information Processing - 8 bit single byte coded graphic character sets - Part 7 : Latin/Greek alphabet.

Note : 1) Some of these parts are already approved as IS but not yet published.

2) Other parts are under development.

9036	Information Processing - Arabic 7 bit coded character set for information interchange.

250

ISO TC97/SC22
Programming Languages
Secretariat: CANADA (SCC)

SOURCE: ISO/TC97/SC22/WG10 - Guidelines for the Preparation of Standards Within SC22

SUBJECT: Interim Liaison Report to ISO/TC97/SC22 on TC97/SC2 - Character Sets and Information Coding

Address reply to: ISO. TC97. SC22 Secretariat.
J.L. Côté. 66 Slater St.. Room 328
Ottawa Ont.. Canada K1A 0S5

251

LIAISON REPORT

From ISO/TC97/SC2 - CHARACTER SETS & INFORMATION CODING

For ISO/TC97/SC22 - APPLICATIONS SYSTEMS ENVIRONMENTS AND PROGRAMMING LANGUAGES

A. STRUCTURE

The secretariat of SC2 is held by AFNOR. At present, the chairman is Mr. Jean Dubos of Bull and the secretariat is Mr. Quyon Tran of AFNOR. It consists of six working groups listed below with the standards that each is responsible for:

WG1 - Code Extension Techniques	- ISO 2022
WG2 - 2-Byte Graphic Character Sets	- new project
WG4 - Coded Character Sets for Text Communications	- ISO 6937
WG6 - Additional Control Functions	- ISO 6429
WG7 - 8-Bit Coded Character Sets (1-byte sets)	- ISO 4873, DIS 8859
WG8 - Coded Representation of Pictures	- new project

B. MAJOR STANDARDS

ISO 646, ISO 7 bit Input/Output Coded Character Set
Specifies the 32 character C0 set of control functions, the 94 character G0 set of graphic characters, SPACE and DELETE to be used in 7-bit and 8-bit environments. 12 positions are reserved for national use characters.

ISO 2022, Code Extension Techniques
Describes structure for coded character sets in 7-bit or 8-bit environments and for extending the number of characters that can be represented to 128 and 256 respectively. It also specifies the complete syntax of escape sequences that can be used to designate and invoke control and graphic character sets and to represent other control functions.

ISO 2047, Graphical Representations for the Control Characters of the 7-bit Coded Character Set
Specifies graphical representations for the 32 control functions of ISO 646.

ISO 2375, Procedure for Registration of Escape Sequences
ISO registration of escape sequences for designating character sets. ECMA operates the registration authority.

ISO 4873, 8-bit Code for Information Interchange, Structure and Rules for Implementation
Specifies an 8-bit code derived from and compatible with the 7-bit coded character set specified in ISO 646. It specifies three levels of conformance which are subsets of the structure of ISO 2022.

ISO 6429, Additional Control Functions for Character-Imaging Devices
Specifies a C1 set of control functions to be used with ISO 646 to facilitate data interchange with two-dimensional character imaging input-output devices.

DIS 8859/1 8-bit Single Byte Coded Character Sets, Latin Alphabet Nr. 1
Specifies a multilingual supplemental graphic set for use in a 7-bit or 8-bit environment.

ISO 6937 Character Sets for Text Communication
Is an application specific standard with three parts:
- ISO 6937/1 - General
- ISO 6937/2 - Graphic Character Set for Latin Languages
- ISO 6937/3 - Page Image Format (PIF)

C. FUTURE STANDARDS

Some of the areas that SC2 is currently working on are:
- -technical graphic character set
- -publishing graphic character set
- -2-byte graphic character sets - the aim is to represent all the major characters in the world in one 2-byte character set.
- -text communications and publishing
- -code character sets for pictorial graphics - methods of transmitting pictures.
- -more additional controls (revision to ISO 6429).

D. CHARACTER SETS

The two appendixes to this report describe two separate character sets, the one from ISOs 646, 6429 and 8859/1, and the one from ISO 6937. This section will attempt to describe the differences between the two sets and where each would be used.

Appendix A shows the character set resulting from the combination of ISOs 646, 6429 and 8859/1. This is a 1-byte character set. Each character is represented by a 1-byte code. Codes are provided for all the common accented characters. However it does not include such characters as the oe and ij diphthongs. It is designed to be used with character strings that have to be processed by programs. Having only one byte per character makes comparison of character strings and sorting easier (than the set of ISO 6937). Diphthongs are usually processed as two characters.

The Latin Alphabet Nr. 1 (ISO 8859/1) should handle most requirements for Western European Languages. It is expected that there will shortly be a large number of terminals supporting this character set.

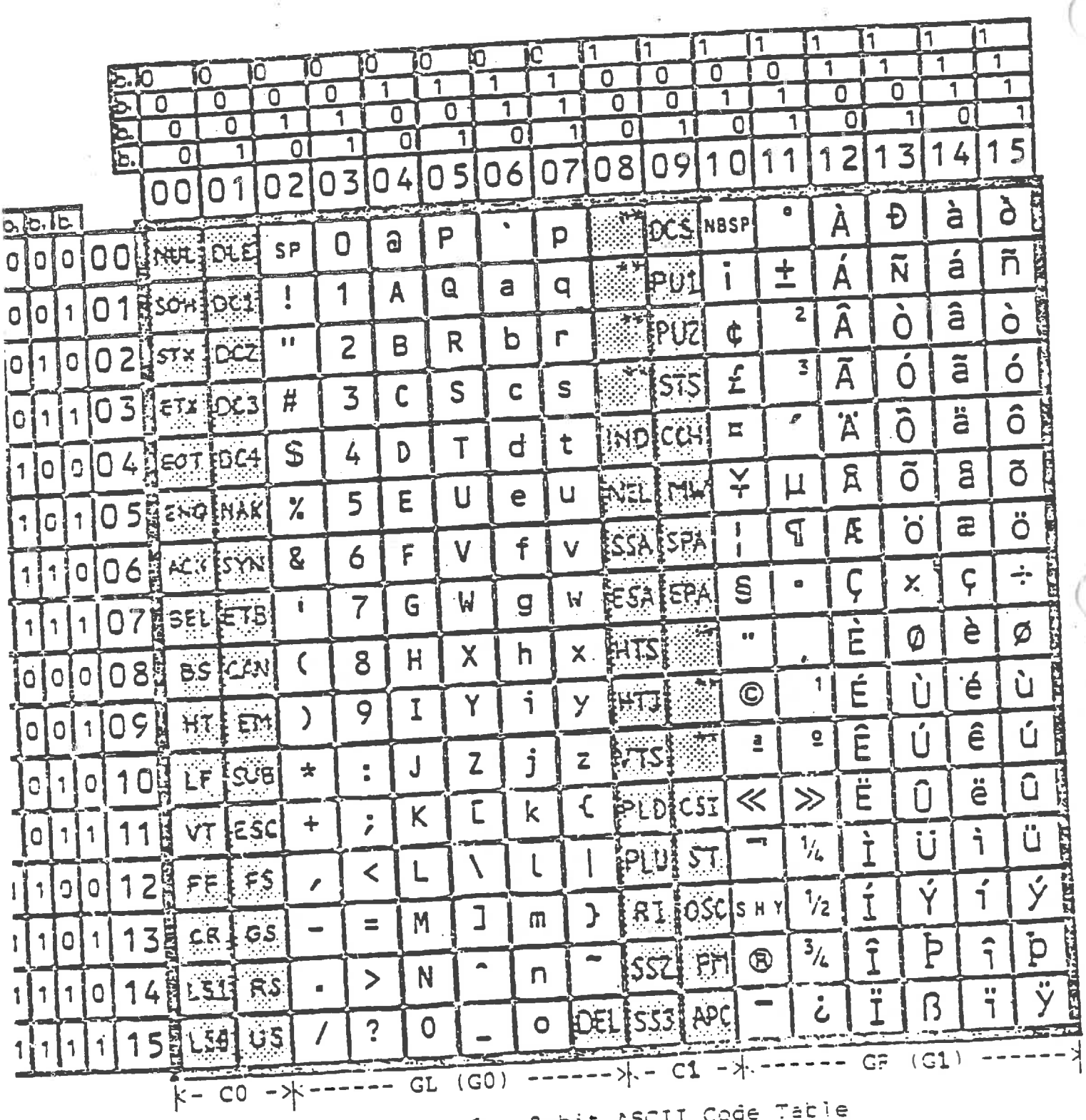
The character set from ISO 6937 is shown in Appendix B. It is designed to be used for text being sent to printers. It produces accented characters by a non-spacing accent followed by the letter (i.e., a 2-byte character), and it includes such characters as the oe and ij diphthongs. Character strings in this character set - where some characters are one byte long and some are 2-byte, are very difficult to handle inside a program.

E. PROBLEMS

As mentioned in item B above, SC2 is developing a 2-byte character set. In order to be able to handle character strings using such a character set, programming languages should provide a character data type which has 2 bytes per character. In the future there may be the requirement for more than 2 bytes per character.

ANSI & ISO Information Interchange Standards

3-Oct-85



b7	b6	b5	b4	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0	0	0	0	NUL	DLE	SP	0	@	P	`	p		DCS	NBSP	°	À	Ð	à	ð
0	0	0	1	SOH	DC1	!	1	A	Q	a	q		PU1	¡	±	Á	Ñ	á	ñ
0	0	1	0	STX	DC2	"	2	B	R	b	r		PU2	¢	²	Â	Ò	â	ò
0	0	1	1	ETX	DC3	#	3	C	S	c	s		STS	£	³	Ã	Ó	ã	ó
0	1	0	0	EOT	DC4	$	4	D	T	d	t	IND	CCH	¤	´	Ä	Ô	ä	ô
0	1	0	1	ENQ	NAK	%	5	E	U	e	u	NEL	MW	¥	µ	Å	Õ	å	õ
0	1	1	0	ACK	SYN	&	6	F	V	f	v	SSA	SPA	¦	¶	Æ	Ö	æ	ö
0	1	1	1	BEL	ETB	'	7	G	W	g	w	ESA	EPA	§	·	Ç	×	ç	÷
1	0	0	0	BS	CAN	(	8	H	X	h	x	HTS		¨	¸	È	Ø	è	ø
1	0	0	1	HT	EM	)	9	I	Y	i	y	HTJ		©	¹	É	Ù	é	ù
1	0	1	0	LF	SUB	*	:	J	Z	j	z	VTS		ª	º	Ê	Ú	ê	ú
1	0	1	1	VT	ESC	+	;	K	[	k	{	PLD	CSI	«	»	Ë	Û	ë	û
1	1	0	0	FF	FS	,	<	L	\	l	|	PLU	ST	¬	¼	Ì	Ü	ì	ü
1	1	0	1	CR	GS	-	=	M	]	m	}	RI	OSC	SHY	½	Í	Ý	í	ý
1	1	1	0	LS1	RS	.	>	N	^	n	~	SS2	PM	®	¾	Î	Þ	î	þ
1	1	1	1	LS0	US	/	?	O	_	o	DEL	SS3	APC	¯	¿	Ï	ß	ï	ÿ

Table 1 - 8-bit ASCII Code Table

ANSI: X3.4 (7-bit ASCII) — X3.64 — BSR X3.134.2

ISO: ISO646 — 646/8859 — 6429 — DIS 8859/1

D54

ISO 6937/2 Text Communication
Latin Alphabetic and non-alphabetic graphic characters

Table 3 – Primary and supplementary sets of graphic characters for text communication (coding when represented by bit combinations 2/1 to 7/14 and 10/1 to 15/14 of an 8-bit code)

"Non-spacing" column →

Accented letters are coded as two bytes:
Eg, á is 12/02 06/01

b4	b3	b2	b1		00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
			b8		0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
			b7		0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
			b6		0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
			b5		0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	0	0	0	00				0	@	P	`	p				°	See 4.3.4	—	Ω	κ
0	0	0	1	01			!	1	A	Q	a	q			¡	±	`	¹	Æ	æ
0	0	1	0	02			"	2	B	R	b	r			¢	²	´	®	Đ	đ
0	0	1	1	03			#	3	C	S	c	s			£	³	^	©	ª	ð
0	1	0	0	04			¤	4	D	T	d	t			$	×	~	™	Ħ	ħ
0	1	0	1	05			%	5	E	U	e	u			¥	µ	¯	♪	See 4.3.4	ı
0	1	1	0	06			&	6	F	V	f	v			See 4.3.3	¶	˘	See 4.3.4	Ĳ	ĳ
0	1	1	1	07			'	7	G	W	g	w			§	·	˙	See 4.3.4	Ŀ	ŀ
1	0	0	0	08			(	8	H	X	h	x			See 4.3.3	÷	¨	See 4.3.4	Ł	ł
1	0	0	1	09			)	9	I	Y	i	y			‘	’	See 4.3.4	See 4.3.4	Ø	ø
1	0	1	0	10			*	:	J	Z	j	z			“	”	˚	See 4.3.4	Œ	œ
1	0	1	1	11			+	;	K	[	k	{			«	»	¸	See 4.3.4	º	ß
1	1	0	0	12			,	<	L	\	l	|			←	¼	_	⅛	Þ	þ
1	1	0	1	13			-	=	M	]	m	}			↑	½	˝	⅜	Ŧ	ŧ
1	1	1	0	14			.	>	N	^	n	‾			→	¾	˛	⅝	Ŋ	ŋ
1	1	1	1	15			/	?	O	_	o				↓	¿	ˇ	⅞	ŉ	

1. See 4.3.1

The ISO/CCITT agreed definition of "text" (6937/1):
"A representation of information for human comprehension that is [illegible] in a two dimensional form, for example [illegible] for presentation in a two dimensional form."

МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ
ORGANISATION INTERNATIONALE DE NORMALISATION

Document ISO/TC 97/SC 22/WG10
N 204

British Standards Institution
Secretariat of ISO/TC 97/SC22/WG10
Guidelines for the preparation of standards
Our ref. no.: ISO/TC97/SC 22/ WG10

Date 12 February 1986

To ISO/TC97/SC 22/WG10

From: BSI
Originator: R. Scowen
(+ B. L. Meek)

ISO/TC 97/SC 22/WG 10

Guidelines for the preparation of standards within SC 22

Character sets

Here is Roger Scowen's draft of guidelines on character sets. An electronic mail version has successfully arrived and is being incorporated into the update to N200. Any immediate comments which I can take into account when doing that will be welcome, but please send them immediately since time is short.

Brian Meek

86/61424

Contact point: H. Walker, BSI Head Office
B.L. Meek, Computer Centre, King's College (KQC), Kensington campus, Campden Hill Road, London W8 7AH

Topic: Character sets - programs

Objective: To provide a common basis for representing programs

Methods:

(1) As far as possible, avoid the use of characters that are in national use positions of ISO 646. If such characters are used, specify an alternative representation for all such symbols.

(2) Require a conforming processor to be capable of accepting a program represented using the minimal character set.

Notes

(1) The characters that are available in all national variants of ISO 646 cannot represent programs in many programming languages in a way that is acceptable to programmers who are familiar with ASCII. In particular, square brackets, curly brackets and vertical line are unavailable.

(2) The characters that are available in the International Reference Version of ISO 646 cannot represent programs in many programming languages in a way that is acceptable to programmers who are familiar with a particular national variant of ISO 646. For example, neither the pound nor dollar symbol may be available.

(3) The characters that are available in ASCII (the USA version of ISO 646) cannot represent programs in many programming languages in a way that is acceptable to programmers because their terminals support some other national variant of ISO 646.

(4) The characters that are available in the eight-bit character sets ISO 4873 with DIS 8859, or ISO 6937/2 would be sufficient to represent programs in a way that looks familiar to most (but not APL) programmers. However, at the beginning of 1986, these standards are not yet widely supported on printers and display terminals.

Topic: Character sets - character literals

Objective: To provide a common basis for representing character literals in programs

Method:

In a character literal: represent each character using one or more of the methods:

(1) The character represents itself, eg A, B, 3, +, (.

(2) A character is represented by a pair of characters: an escape character followed by a graphic character, e.g. if & is the escape character, &' to represent apostrophe, && to represent escape, &n to represent newline.

(3) A character is represented by three characters: an escape character followed by two hexadecimal digits that specify its internal value.

Inside a program store each character as a single byte.

Notes

(1) Programs must be representable on as many different peripherals and systems as possible; there is therefore a need to reduce the number of characters required to represent a program to the minimum that is consistent with general practice and readability. On the other hand, programs themselves must be able to represent and process as many different characters as possible.

These two requirements make it impossible to represent every character by itself in a literal character string. Languages where this is the case are inadequate for general processing of character data.

(2) The representation of a space in a character or string literal arouses strong emotions in many languages. Some claim it should be represented by a visible graphic character so that spaces in programs do not affect the meaning. Others claim that it must be represented by itself because that is the only natural possibility.

(3) The ISO 8-bit character set 6937/2 represents some graphic characters as a pair of 8-bit bytes.

Topic: Character sets - data

Objective: To provide a common basis for processing data

Method:

A programming language standard should not assume that character data is anything other than a sequence of bytes whose meaning depends on the context.

Notes

(1) Many programs will assume their data is expressed in ASCII or some other variant of ISO 646. But if the standard assumes that all data is expressed in any one particular character set, it will cause difficulties for many users.

(2) Collating sequence. The collating sequence determines the order of records sorted on a key that contains a general sequence of characters. Without a definition, programs and data are not portable, but no single collating sequence seems to be generally acceptable.

For example, is

a < A < b < B ... < z < Z

or

a = A < b = B ... < z = Z

Do digits precede or follow letters? Is space ignored? Where do other punctuation characters fit between letters and digits.

Type transfer functions that convert an integer to a character and vice versa provide a partial solution. But some graphic characters have different integer values in different character sets.

LABORATORY
ARCHITECTURE
AND PLANNING

Room 4-209, Massachusetts Institute of Technology, Cambridge, MA 02139 USA 617/253-1355

March 12, 1986

B.L.Meek
Computer Centre
King's College (KQC)
Kensington Campus
Campden Hill Road
London W8 7AH
United Kingdom

Dear Brian,

Several items have appeared in recent WG10 mailings that are motivating me to comment both as an individual and as someone with some experience in the front lines of trying to do language standards, rather than writing rules about it. Let me address three issues in particular, without trying to attach relative importance to them.

I. Character sets.

I am personally extremely sympathetic to the reasoning behind a set of guidelines that suggest or require ISO 646 IRV (no extensions or national use characters) support. Such a rule would make my own [non-standards-related] work considerably easier. At the same time, it is, perhaps, important to remember that there is a principle about Standards that is more important than any set of guidelines: that they represent a consensus and common practice. That principle is written, quite explicitly, into the procedures and rules of many national bodies, including ANSI, and is at least implicit in the ISO directives.

Most of our existing Standard languages require characters outside the IRV set (or representation sequences that few experienced programmers in those languages would recognize). As an aside, while I find the conventions that must be adopted by an implementer, such as the use of "!" to represent the "or" operator, visually offensive, PL/I is one of the exceptions: the standard form of the language can be written entirely within the IRV set. To encourage a rule in a draft Standard that moves against the history and usage of a particular language is to encourage a Standard that will be, in that portion, ignored. In those cases where the Standard acquires legislative authority, such rules encourage implementations purely for conformance purposes and the corresponding resentments against the standardization process. The latter would do no one much good in the long term.

I believe that this type of recommendation about character sets would better be taken as a guideline or suggestion for the potential designers of new languages than as a guideline for specification and standards for languages for which several, typically fairly similar, implementations exist already. By the time a language reaches the stage at which it comes under the aegis of the SC22 program of work, it is typically too late to have much impact on issues such as character sets, at least within the guidelines implied by the "consensus of common practice" principle.

86/62206

201

Let me call WG10's attention to another approach to the character set problem. It is not an approach that I particularly like, and it makes things difficult for the casual user in interchange, but it is a possibility to be aware of. It was agreed in the PL/I Standard in the early 1970s to define an abstraction in which the mapping of the character set used in the Standard to the character set of the implementation was left outside the language Standard itself. The language Standard says only that there shall be at least a certain number of distinct characters and that different members of that set have particular definitions. Given that state of affairs, ISO or a national body could write a *separate* standard, or extend a character set standard, to say "whatever mapping is used between characters in interchange and the abstract characters of the language standard, it must conform to these rules" or even "the mapping shall be defined to be...". But in no event is the language Standard drawn into making prescriptions about character set Standards, which is arguably an advantage.

The principle of keeping the language standards out of the concrete character set business is reinforced by two other trends. For my position as a distant observer, it appears that we are beginning to see an increasing number of character-set standards for alphabets that are not Latin-based and movement toward "multiple byte" character sets for some languages, including some non-alphabetic ones. At least in the first case, it is sensible to expect that sooner or later, a national body will adopt a standard with the character of "within this country, the standard character set will be xyz", where "xyz" is not, by any reasonable stretch of the imagination, ISO 646 IRV. Such a decision would require local deviation from any language standard that specified the IRV or violation of either the language standard or the [national] character set one. It would not, however, compromise a language standard that said "some mapping must be defined, the nature of that mapping should be as specified by other applicable standards" and stopped there.

INTERNATIONAL
ORGANIZATION FOR
STANDARDIZATION

ISO/TC97/SC22 N

September 1987

J. W. van Wingen, the Netherlands

SOME COMMENTS ON N 357

Ø. General Remarks

Developments in hardware and software have, in recent years, created an increasing sophistication in handling text by computers. This evolution had hardly a parallel in facilities provided by programming languages. In particular, producing text output by several popular languages is still severely restricted due to syntactical rules derived from dealing with situations that no longer exist, like input on punched cards, or line printers supporting only capital letters.

1. A Hierarchy of Concepts

A program written in a certain language touches the character aspect in two areas:
--- the way the source text is expressed in characters (traditionally called the"hardware representation" problem, but "transliteration would be a better term)
--- the way it prepares output or reads input

A computer works internally only in "bytes" (what that may be), so a convention was required to establish a unique relation between bytes and graphic characters. Some bytes may have no graphic "representation" and are thus "unprintable". Even if there is one, it may differ between nations. But, in fact, ISO 646 and its national derivates rule strictly what is permitted. This was a base for defining character handling in PL's. Unfortunately, this situation was almost from the beginning mostly outward appearance. What happens with output is decided by the operating system, it may be printed or kept or mailed, outside the control of the program. Things have become worse since. The user can specify not only when output is printed, but also with which letter fonts, and this he achieves by specifying options to the operating system. Thus for producing output a decision is needed on two levels, on that of the PL, and on that of the OSCL.

This leaves the PL designer with a choice between two approaches:
--- low-level: the program deals only with bytes, and provides some means of handling all possible (256 or 128) codes in the program If you want a new line, you put a certain character into your program (or more).
--- high-level: the program deals only with visible characters, all other effects are the result of (perhaps hidden) function calls.
On the operating system level it makes little difference, what it gets, it interpretes.

This way of approach is quite similar to that of OSI. Decisions are ordered according to "layer". In our context, font selection belongs to the OS layer, not to the PL layer. If PL's deal with bytes, not with representations of them, there is little problem.

2. The ISO 2Ø22 system

The essence of ISO 2Ø22 is that a byte may produce a graphic symbol dependent on context, that is, the same byte if preceded by certain other bytes, can produce several different graphics. If is required that this process be controlled by the program, those changing bytes must be produced by the program, either by allowing them for inserting in the source text or by providing appropriate functions. In the last case we have a "binding" problem.

J M van Wingen

SEAS

SHARE European Association

Incorporated in The Netherlands
Founded in 1961

SEAS
National Character
Task Force

White Paper on national character, language and keyboard problems, September 1985

265

Copies of this White Paper may be obtained from

SEAS Headquarters
University of Nijmegen
Toernooiveld
6525 ED Nijmegen
The Netherlands

Phone +31 80 558833 ext. 3357

Enquiries and suggestions regarding the content of th
Paper should be directed to the above address and mar

'Attention SEAS Office Automation Project'

SEAS
National Character
Task Force

White Paper on national character, language and keyboard problems, September 1985

Abstract

The SHARE European Association National Language Task Force (SEAS NCTF) was inaugurated in 1982 to investigate and report on the problems encountered using IBM software and hardware in non-English-speaking countries with a written language based on the Latin alphabet. This White Paper is the result of NCTF work in the period 1982-85 and covers the areas of traditional dataprocessing, word processing and personal computing.

Introductory sections deal with the history of the problem area, followed by sections describing the actual problems and their required solutions as regards 'national characters', 'national languages' and 'keyboards'. The final section outlines the general characteristics of future implementations of software and hardware which are required by the SEAS user community. These implementations are expected to give equal value for money for both English-speaking and non-English-speaking IBM customers, together with significantly improved product performance, quality and ease of use.

Current Members of the Task Force

Anders Berglund, CERN, Switzerland
Bernard Chombart, Promodès, France
Günther Krysmanski, Germany
Jerry Andersen, IBM Kingston, USA
Klaus Daube, Oerlikon Bührle RZ AG, Switzerland
Peter Gardner, Kommunedata, Denmark (chairman)

Earlier Members of the Task Force

Hubert Ickes, IBM Kingston, USA
Knud Nielsen, Denmark (chairman)
Michael Täschner, Germany

Management Summary

European users of IBM hardware and software products find ir these products do not meet their requirements in the areas of:

- National character support
- National language support
- Keyboards
- Inter-product compatibility

IBM hardware and software products are primarily designed : language and the 26 letter latin alphabet. When installed : speaking countries most IBM products suffer a degradation in user-friendlyness and/or performance. The user often has to b some sort of compromise or has to put up with continual inconvenience. In other words. European users experience poorer and get less value for their money.

The White Paper describes in detail the problems encountered asks IBM to give top priority to the solving of these problems c and comprehensive (architectural) basis.

The White Paper has been designed as a source document for t organization at all levels.

The contents will also be of interest to the various national a standards organizations since they reflect the current state o largest single computer user community in Europe.

SEAS National Character Task Force

Table of Contents

371

1 Introduction

This White Paper has been prepared by a 'National Character Task Force' under the auspices of the SHARE European Association, SEAS. The White Paper addresses the problems experienced by non-English-speaking countries when using the hardware and software products currently available from IBM.

These products are primarily designed for the English language and the 26-letter Latin alphabet. When installed in non-English-speaking countries most IBM products suffer a degradation in function and/or user friendliness and/or performance. The user often has to be satisfied with some sort of compromise or has to put up with continual irritation and inconvenience.

SEAS believes that the time has come for IBM to provide the non-English-speaking user with systems of the same quality as is attainable in the English-speaking user community.

Section 2 of the White Paper describes the history of the problem area. It explains why the situation is becoming critical now, despite the fact that most of the problems to be described have existed for at least the past 10 to 15 years.

The prime objective of the White Paper is to be found in section 3. This section provides IBM with a basic source document describing problems as experienced by users and stating the required solutions. There is no attempt to suggest details of implementation, that is to say details of how the solutions should be achieved. However the general characteristics of the implementation as a whole are of great importance to the user community. Section 4 sets out SEAS' requirements for the implementation.

1.1 History of the National Character Task Force.

The events which led to the establishment of the Task Force had their origins in a paper given by SEAS Executive Board member Michael Martin at the 1980 Spring Meeting. The paper was entitled 'User Experiences with Communications' and in it Michael Martin outlined many of the problems facing a typical Danish installation using IBM hardware and software products. By far the majority of problems were caused by product design which made it difficult, if not impossible, to use (or 'implement') the Danish alphabet and the Danish language.

The problem area was also discussed in presentations at the SEAS meeeting in Antwerpen in 1981 given by B. Mölenhof (W.Germany), B. Chombart (France) and by IBM representatives Loizides and Schwinge.

Michael Martin's paper merely brought together a number of problem areas which had been plaguing the non-English-speaking user community for many years. The problems described were to be found to a greater or lesser extent in all European countries.

The points raised in this paper were incorporated into a resolution from the SEAS OS project (088-0680, 03.06.80) which received a sympathetic response from IBM - but a response which did not imply that IBM were really aware of the seriousness of the situation in Europe. Since Martins' paper had stated that European installations got in effect less value for money than their American counterparts. the SEAS board decided to raise the national character problem as one of several 'major areas of concern' at their next regular meeting with the management of IBM Europe.

Executive Board member Frédéric Roux (France) presented the problems on January 8th 1982 and proposed the formation of a task force consisting of 3 SEAS members from different European countries plus an IBM specialist.

IBM agreed to participate in a task force and the inaugural meeting of the SEAS 'National Character Task Force' took place in Noordwijkerhout in the Netherlands on May 4th 1982. The Task Force at that time consisted of SEAS members

Knud Nielsen	(Denmark, Chairman)
Bernard Chombart	(France)
Michael Täschner	(West Germany)
and IBM specialist	
Hubert Ickes	(Kingston, USA)

The Task Force held a 2 day meeting in Paris on August 9-11 1982, by which time Hubert Ickes had been replaced by Jerry Andersen (also from IBM Kingston) and Michael Täschner by Günther Krysmanski (also from West Germany). The contents of the White Paper were agreed on and draft versions for several sections were produced shortly afterwards.

The departure of Knud Nielsen from SEAS resulted in no new work being done until early 1984. The Task Force, now under the chairmanship of Peter Gardner (Denmark), held a 2 day meeting in Paris on 25-26 January 1984. Further work was done here on the structure of the White Paper but again there were difficulties of a practical nature - perhaps understandable given the 15-year history of the problem and its all-embracing aspects.

The next series of working meetings was held in the Netherlands in Veldhoven, in conjunction with the SEAS Spring Meeting 15-19 April 1985. The task force had by this time acquired two new members, Klaus Daube and Anders Berglund, both from Switzerland.

At Veldhoven it was agreed that the subject matter be 'frozen' so that a final version of the White Paper could be presented at the SEAS Anniversary Meeting in Zürich in September 1985.

1.2 Objectives.

The objectives of this White Paper are twofold:

Firstly to make IBM aware of the magnitude of the problems encountered in non-English-speaking countries (section 3).

Second to make IBM aware of the need for solutions at the system architecture level together with the need for stepwise. user-controllable and low-cost conversion procedures when the time comes to implement these solutions (section 4).

1.3 Scope.

The Task Force has found it necessary to limit the scope of section 3 in two ways:

> The problem descriptions will be limited to problems arising in countries using the Latin alphabet as the basis for their written language. This means that additional problems arising in, for example, Arabic, Cyrillic, Greek, Hebrew, Kanji and Katakana user communities will not be described.
>
> The problem descriptions will be further limited to those arising through the use of locally standardised character sets where the attributes of a character are limited to its graphic value, the code point in the EBCDIC table and, for alphabetic characters, the case (upper or lower). We are therefore addressing problems in the traditional Data Processing (DP) and Word Processing (WP) environments together with problems in the new Personal Computer (PC) environment.

Both limitations are felt to be reasonable at the time of writing and have been necessary in order to keep the work of the Task Force to a manageable level.

The first limitation illustrates the essentially global nature of the problem area and requires further comment. We have no illustrations in section 3 of problems specific to, for example, the Greek user community - unfortunately, since Greece is in Europe. We do, however, require the use of the Greek alphabet everywhere in Europe - for both scientific and commercial work.

Similarly we can state that a solution which does not allow use of the Cyrillic, Greek etc. alphabet will increasingly come to be seen as a hindrance. The same arguments obviously apply to Arabic and Japanese. These aspects are not taken up specifically in section 3, but they resurface in section 4.

The second limitation which is purely technical in nature puts additional problems

which may occur in the fields of, for example, computer graphics, publishing and image processing, outside the scope of the White Paper. It is hoped that IBM will by inference recognize the same (or new) types of problem in these fields and take action accordingly.

The scope of section 3 is thus a 'level of ambition' for the problem descriptions in this section. In contrast, the scope of section 4 has wider boundaries. Here the Task Force feels able to outline a set of requirements which are generally applicable to any geographical or technical area.

Whilst international standards for coded character sets are refered to in several places in section 3 it should be realised that there is no organized attempt in this White Paper to relate problem descriptions or required solutions to existing international or national standards. We are in effect criticising IBM for its lack of standards rather than for its deviation from existing standards which may in themselves be restricted or of dubious usefulness in regard to the problem areas described. (Witness for example the many registered national character sets where important delimiter characters such as { } [ ] / \ are replaced by national use characters...)

Finally on the subject of scope we would note that it has been necessary to 'freeze' the subject matter at a certain point in time in order to concentrate on publishing the White Paper. This document should thus be assessed as a reasonably well-formulated attempt at presenting a very complex problem area, but not as an exhaustive thesis.

1.4 Intended Audience.

The intended audience for this White Paper is primarily the IBM organization at all levels, both inside the USA and in Europe.

We also hope that the White Paper will be an eye-opener for the many DP-professionals who have 'learned to live' with character, language and keyboard problems. Now is the time to react - to bring all the old problems up to the surface and to work for a common solution.

The White Paper may also be of more than passing interest to the various standards organizations. The problems described reflect a sad state of affairs after many years of standardisation effort. We are aware that these conditions are by no means confined to the IBM product world, though we will not attempt comparisons with other suppliers here. What must not be overlooked is the commanding position IBM holds in both hardware and software markets. A standard which is unacceptable to IBM or only partly acceptable (for whatever reasons - technical, financial or strategic) has little chance of wide acceptance. As a result, IBM customers are often unable to reap the benefits of existing standards, whilst the standards organizations must see their work neglected in practice by the largest user community. This state of affairs

has certainly dominated in the past. The NCTF hopes that both sides will be able to cooperate more productively in the future .

As mentioned in the Introduction, most of the problems described in section 3 have been with us for a long time and it is hard to see why so little has been done to solve them over the years. In fact the problems have had a tendency to multiply rather than disappear. We can only assume that both IBM and its European user community have had a blind spot here - or have been harassed by other problems of a graver nature.

Whilst it is fairly easy to understand the conservative attitude of IBM product development organizations in the USA with respect to national language and character problems, it is difficult to understand the inaction or lack of leverage of the IBM organizations in each European country. These organizations will, between them, recognise each and every problem in section 3. Most of them will have had first hand experience of these problems in their service bureau and consulting work.

2 Changes in Computing Environment

The Task Force believes that an understanding of the changing information processing environment is essential to put coded character set requirements and problems in perspective. The following discussion of the changing environment is offered in that spirit.

2.1 History

Both the cost and size of computers have decreased by several orders of magnitude over the past quarter century. These reductions have lead to equally dramatic changes in the types of applications processed on computers, and in the nature of the end-users of those computers. The result has been a significant evolution of the coded character set requirements levied against hardware and software products.

Twenty five years ago, the cost of a computer was high enough to be prohibitive for all but the largest businesses and government agencies. Applications were limited to those that could be justified on such an expensive tool. Payroll and accounting were typical, as were various scientific programs that performed particularly complex and lengthy calculations. Computer users were data processing professionals who accepted that communication with a computer had to be on its terms and in its language.

Input was usually from punched cards or paper tape, and data was output on a line printer or card punch. Character sets typically contained 30 or 40 characters, and 5 and 6-bit codes were common. European requirements were satisfied by setting aside a few 'national use' code points, so that graphic characters could be assigned based on each country's need.

As requirements grew, and technology improved, character sets stabilized at 94 characters. IBM standardized on the 8-bit-Extended Binary Coded Decimal Code (EBCDIC), with increasing European requirements being met through expansion of the 'National Use' code-points to the current 13.

The environment described above continued to change markedly through the ensuing years. Input from remote terminals became virtually universal. Keyboards grew in size, and techniques were adopted that allowed 3 or more different graphic characters to be selected from a single key. Terminals with the ability to handle larger, and multiple, character sets became common. Matrix printers also overcame character set size limitations, and new character codes were adopted to meet various application needs.

As a direct result of the dramatic reduction in price, computers have become

relatively common in small businesses and even private homes, with today's end user as likely to be a teller, secretary, student, or accountant as he or she is to be a DP professional. While computers still process payrolls, they are increasingly being used to draft letters, play games, modify recipes, and prepare income tax returns.

This dramatic evolution in the information processing environment has continued to put steady pressure on coded character set support. While character sets are significantly larger, the expansion has fallen short of the demand. European customers are increasingly requesting that products allow them to use all the legitimate letters of their native language. Text and Word Processing customers find that they need additional characters on their keyboards and in their character sets in order to correctly spell various words and names. It is clear that the 94 character set can no longer meet the requirements of many countries whose languages require large repertoires of accented letters.

In addition to problems of character set size, multinational and multilingual requirements are increasing. Multi-national companies are finding that they have an increasing requirement to be able to correctly transmit data across national borders. Customers in multi-lingual countries, such as Belgium and Switzerland, are increasingly insistent about the need to supply coded character set support that spans the requirements of two or more languages.

These problems are aggravated by difficulties created by differences between various coded character sets, as well as software design that incorrectly treats the differences between coded character sets of the various countries. Users of products spanning two subsystems are often frustrated to find that data entered from one product cannot be retrieved by another. Others find that characters that they require are effectively unusable with some programs, even though available at the terminal.

In sum, the coded character set support that was designed to meet the requirements of 10 and 15 years ago is now needlessly diverse, as well as increasingly non-responsive to modern needs. Coded character support needs to be updated to bring it into line with today's end-users and their needs.

2.2 Future Evolution

The trends observed in section 2.1 can be expected to continue in the forseeable future. As the price of computers continues to drop, small business and personal computers will continue to proliferate. New application areas will be discovered, and casual and non-professional end-users will become an even more important factor. Ease-of-use will be stressed, with subsequent pressure on coded character set enhancements. Two trends in particular appear especially significant in regard to character set support.

First, the Task Force believes that the future will see continued growth in the area of Text/Word Processing. This is important because it underscores the requirement for enhancement of European character sets so as to include all of the characters required for each language. Combined with increasing requirements for international communication of data, the net result is continued pressure to supply character sets that not only meet the requirements of one country, but of all countries sharing a common basic alphabet.

Second, the Task Force believes that product distinctions between data processing and word processing are artificial and misguided. The trend is expected to be in the direction of merging these two areas into one 'information processing' application area. Terminals, printers, and software may then be anticipated to interwork across both DP and WP, resulting in the requirement for one work-station on a user's desk, not two. Although the trend can be verified in many newer products, coded character set support continues to be divided into three families, one for DP, one for WP and a third for PC. We believe this hinders and delays the inevitable merger.

Finally on the subject of future evolution, we would like to emphasize that solutions are already long overdue. The piecemeal solutions currently available compound the entire problem area. New ad-hoc solutions in specific areas are constantly being devised at installation, company and national levels - resulting in an overall situation which is difficult if not impossible to describe, let alone understand. Section 4 emphasises the need for a speedy statement of intent from IBM which will point the way to long-term solutions and allow best possible construction of interim solutions.

The user community is also increasingly aware of the fact that other suppliers are already in the market with operational solutions to many of the problems which exist in the IBM-world.

3 Problem descriptions

This section is divided into three major parts:

Section 3.1 describes hardware problems encountered when trying to enter or reproduce national use characters, together with other hardware problems encountered through usage of national characters. These problems are called 'national character problems'.

Section 3.2 describes software problems encountered either because a national character set is in use or because of the requirement that software products must be able to 'converse' in the language or languages of the countries in which they are executing. These problems are called 'national language problems'.

Section 3.3 is devoted to the specific problem area of keyboards. The keyboard of a terminal is by now the primary input device in most installations. It is the primary man-machine-interface in all installations and it is when using the keyboard that many of the problems described in sections 3.1 and 3.2 first become apparent.

Some problems may be on the border-line of these categories - or may be a mixture of them. In these cases the seemingly best category has been chosen and this rather loose approach underlines the basic policy of section 3. Our objective is description rather than analysis. The technically expert reader will quickly recognize that several problems have the same fundamental cause, but he should be on his guard in any critisism of the White Paper for banality, naivety or lack of technical insight in this respect.

By describing what the IBM customer in Europe identifies as his problems we are simply passing on 'the facts as they are' to the IBM technical organization. It is up to this organization to perform the analysis and formulate the necessary strategies for solving these problems within the constraints described in section 4.

We feel obliged to state at this point that many European installations (often members of SEAS) possess experts in the problem areas described. These people have a knowledge level at least as high as, and in some cases higher than, the IBM organizations in their respective countries. It is therefore not through lack of expertise that the problem descriptions in this section have been kept at a fairly simple technical level - but as policy.

We have attempted to adhere to the following pattern for each description:

- A non-commital title
- The problem description
- Examples where appropriate
- The required solution.

We have also felt obliged to grade the problems roughly according to the scale:

A Result incorrect or not as intended,

B Impossible to do but the intent is obvious and reasonable,

C Impossible to do but would like to be able to do,

since the problems described range from direct errors to facilities which are at present missing but are urgently required.

3.1 National character problems

This section describes problems in connection with hardware, chiefly manual input devices and screen and print output devices. Problems resulting from the diversity of character sets have also been placed here. since the character set is essentially a function of the hardware capabilities of the available input devices. For example the de-facto character set of a typical commercial (DP) installation is identical with the character set installed in its 3270 network.

3.1.1 Dual 3270 character sets (grade A)

Several countries have had their 3270 national character sets split into two almost identical versions (the 'almost' is the problem). The 'old' version dating from the first introduction of the 3270 hardware is know known as the 'alternate' character set. The 'new' version dates from the introduction of the 3278. Some of the countries affected and the graphics/code points changed are shown in the following small section from GA27-2837 (figure 10-32).

National use number I/O Hex Code Controller Language Device	1 4A	2 5A	3 6A	4 79	5 5B	6 7B	7 7C	8 5F	9 A1	10 C0	11 D0	12 E0	13 4F	14 7F
Austrian/German	Ā	Ū	ō	`	$	#	§	^	ß	ā	ū	Ō	!	"
Austrian/German (alternate)	ō	ū	ß		Ū	Ā	Ō	¬					|	ä
Danish/Norwegian	#	¤	ø	`	Å	Æ	Ø	^	ü	æ	å	\	!	"
Danish/Norwegian (alternate)	å	å	¦		Å	Æ	Ø	¬					|	æ
Finnish/Swedish	§	¤	ō	é	Å	Ā	Ō	^	ü	ā	å	É	!	"
Finnish/Swedish (alternate)	ō	å	¦		Å	Ā	Ō	¬					|	ä

The 'new' character set was developed/introduced without any form of customer or user-group consultation. The result is that old users (3277 converted to 3278) have had a tendency to stay with the alternate character set - the new version having no obvious advantages of any kind, and in some cases several disadvantages. Note here

that there was already a very widespread use of 3277's at the time the 3278 was introduced. Customers (and their customers in in turn) had no incentive to replace the terminal equipment in nets with several hundred terminals, often purchased and not rented. Over the years the 3277 has of course been phased out - but the alternate character set is still with us.

New customers (post 3277) have acquired the new character set, in most cases without having been informed of the alternative.

IBM has been very quiet on the whole subject of the change, and there is no conversion/comparison documentation available to our knowledge.

The differences between the old and new character sets are essentially:

- the re-positioning of some or all national characters in the code table.
- the disappearence of 'vertical bar' and 'not sign' which are replaced by 'exclamation point' and 'circumflex accent'.

Note that both special symbols are heavily used in PL/I, a popular programming language in Europe. Also the vertical bar is an indispensable component for producing flow-charts, block-diagrams and low-resolution graphics in general.

To add to the confusion, the 3278 keyboard is only available with key-tops corresponding to the new character set. This means that 'exclamation point' must be hit for 'vertical bar' etc. - a constant source of irritation especially for end-users. It also means that some of the keys have no effect when hit - more irritation.

There are other peculiarities. For example in Denmark the u-umlaut from neighbouring Germany was a new graphic in the new character set. Danish users have always required this character and in fact it had been available on '3277-compatible' equipment for some years in both upper and lower case versions. The IBM offer is a single case u-umlaut (unknown whether upper or lower case) in a code point different from either of the two which were already widely in use, and incidently also different the German/Austrian code points for u-umlaut in both the 'alternate' and new versions for these countries (see table above).

The existence of dual 3270 character sets in some countries further complicates the entire national character problem area, causing confusion for all parties involved - not least IBM themselves. The ultimate in confusion arises when 'alternate' and 'new' TP-networks are tied together in an SNA/MSNF configuration.

The requirement is that this sort of unilateral uncoordinated change never be repeated in the future.

3.1.2 Character sets are incomplete (grade B)

Multinational companies and agencies find themselves unable to enter and display correspondence which is written in different languages. in the DP environment.

To take a simple example, it is only in the UK and Italy that standard DP equipment will be able to reproduçe the sentence

The exchange rate is £1 = $0.75

Only these two countries have the £ symbol and several countries do not even have the $ (and incidently the assignment of these two graphics to code points in the UK and Italy is such that there is no possibility of the sentence being correctly reproduced after transmission from one country to the other).

The requirement is that each country have access to a unique and complete code table and that all codes be enterable and displayable.

3.1.3 Diacritical marks in latin alphabet based languages (grade B)

Currently IBM products are only available with the national use characters of selected European countries. Teletext (the successor of Telex), however, supports all the European languages with their special symbols and diacritical marks. With the increased linking of computers for electronic mail and linking of computers to the Tele(te)x network the lack of support from IBM for many European languages is seen as an upcoming severe problem area.

It is impossible to translate a text from Videotex or Teletext correctly to EBCDIC. Also with the multilingual codepage there are missing accented characters.

Turkey belongs to NATO and is strongly related to the European Common Market. So it is necessary to write Turkish in Europe also on DP equipment. Also writing Polish is necessary for international institutions and organizations. The multilingual codepage does not support all the needed characters - and with the concept of "fully formed" characters in one codepage it is not possible to define all of the accented characters which are needed.

The Turkish alphabet uses the special characters

Ç ç Ğ ğ Ş ş Ö ö Ü ü İ ı

The accents are not only for correct spelling and pronounciation, they also can change the meaning or sense of a word completely!

Olumlu bir haber	a good message
ölümlü bir haber	a mortal message

Havalar ılık geçecek	the weather will stay warm
havalar ilik geçecek	the weather will go through a button hole
çam yeşildir	the firtree is green
cam yeşildir	the glass is green.

There are the following special letters in the Polish alphabet. Not all of them can be printed or displayed on IBM equipment:

ą Ć ć ę Ł ł ń Ó ó Ś ś Ź ź Ż ż

Ona jest mężatką	She is married
Łódź jest przemysłowym miastem	Lodz is an industrial city.
To jest żmudna praca!	What a troublesome work!

Also the Czech, Slovenic and Serbocroatic languages use additional charcters in their alphabets:

č Č č Č ď Ď ň Ň ř Ř š Š ť Ť ž Ž

The French language uses some accented letters, which are especially vowels, plus the c with cedilla:

à â ç é è ê ē î ī ô ù û ū

For instance the French sentence "Les bûches brûlent tête-bêche dans l'âtre de la cheminée: c'est Noël" is not a strange sentence, but a very typical case.

A concept of flying accents would save many codepoints. It also would allow the writing of many latin transliterations of non-latin-based scripts. Since there are output capabilities for these things (see this paper), an adequate solution for storing and input of these texts is needed.

3.1.4 Inconsistent code points across DP/WP/PC (grade A)

It is not possible to transfer documents from a WP-environment (e.g. IBM 5520) to a DP-environment without loosing information.

In the WP-environment there are a lot of code points for graphics which have no corresponding graphic in DP-environment. For example there is no µ in the DP-environment. National characters which are available in both environments are coded differently. For example the ū has code point X'D0' in DP-environment in

Germany but X'DC' in WP-environment.

Similar problems also occur when transfering out of the PC-environment.

The requirement is that no such diversity of code tables exist.

3.1.5 Codepoints for same graphic differ internationally (grade A)

If letters are sent from one country to another, lets say from Germany to Denmark, they may be written in a language not spoken in either of these countries (e.g. English). For the text of the letter there will be no problem. To sign the document the national characters of the sending country are needed and for addressing those of the receiving country are needed.

In the case of one of the members of the SEAS National Character Task Force, Günther, problems arise. ü is a German national character, but it is also a Danish 'neighbouring country' character. In Denmark this name appears on the screen or the printer as Günther.

ü is coded as X'A1' in Denmark and as X'D0' in Germany. X'D0' is displayed as å in Denmark.

It is not a good solution to translate the code points at the borders of the countries because there are files, which may contain both text and non-text information. There is no way to decide which part of the file has to be translated and which part has to be left untranslated.

The requirement is that the same code points be used for the same graphics in all countries.

3.1.6 Graphic values for same code point differ internationally (grade A)

This is the reverse situation to the one mentioned before. A letter sent from France to Germany by electronic mail or tape cannot be read, because the French national characters are displayed as German national characters.

The è is coded in France as X'D0'. This code point is displayed on a German 3278 terminal as an ü. Notice that ü both is a German and a French national character.

The requirement is that the same as for 3.1.5

3.1.7 Hardware limitations (grade A)

The requirement to be able to use a specific national character set may impose hardware limitations on the customer which seriously reduce the value for money invested in a piece of equipment.

For example users of a so called 'alternate' character set (see 3.1.1) will find that IBM 3274 cluster controllers cannot be configured for 7 colours. graphics or the APL character set. This means that it is necessary to acquire extra controllers if any of these facilities are to be used.

It is also impossible to install different "languages" (keyboard support for different languages) in one 3274 controller.

An IBM 7436 letter-quality printer configured to use an alternate character set cannot print the vertical bar '|' which is included in the character set. It prints as a minus '-' which makes nonsense of any kind of block chart or form.

The IBM 6670 laser printer originally allowed only 2 concurrent fonts in Denmark, whilst American users were allowed 4 concurrent fonts.

The IBM Quietwriter printer as supplied is unable to reproduce the Danish/Norwegian Ø ø. This is probably a side effect of the fact that these two graphics are (incredibly!) missing from the IBM PC character set. (They can be implemented on the PC by installing IBM's 'National Supplement, Scandinavia' which in turn causes problemens for third party PC-software.)

Obviously the requirement is that no such limitations exist. They are completely artificial and stem not from technical problems but from either IBM policy or lack of knowledge of the European market.

3.1.8 Neighbouring countries (grade A)

Each country having a neighbouring country with a different written language (and this means most countries) has a requirement to be able to data process some or all of the national-use characters of its neighbours. In most cases there will be a relatively large number of people from neighbouring countries resident for shorter or longer periods in the country in question and DP installations have to be able to store and reproduce names which may include neighbouring national-use characters. At present there is either no support for this, or very rudimentary support.

For example in Denmark it is mandatory to be able to reproduce all national-use characters from Norway, Sweden and Germany. At present there is no problem with Norway, which has the same character set, but the Swedish and German Ä, ä, Ö, ö are non-existent. The German Ü, ü are either non-existent (alternate character set) or present only in a single case version (at a code point which differs from all other implementations of Ü, ü).

The Danish prime-minister at the time of writing is Poul Schlüter. Here is his name as reproduced at one installation in Denmark:

UPPER CASE	lower case	Equipment
SCHLÜTER	Schlüter	Non-IBM. 3270 compatible
SCHL≤TER	Schl-ter	IBM 3270 alternate
SCHLüTER	Schlüter	IBM 3270 standard

Note that only the non-IBM equipment can do the job correctly. There are many ad-hoc 'standards' in different installations which attempt to solve the problem. The latest development in Denmark is a series of RPQs introduced by banking and finance installations to provide some sort of homogenity - at least in their world where financial transactions and customer names and addresses are exchanged on a network basis. Needless to say, this is just one more example of an ad-hoc solution which further confuses the total picture. Next after problems concerning a country's own national-use characters, the 'neighbouring countries' problem is probably the worst one for most DP-installations.

The requirement is that no such problem should exist.

3.1.9 Entry and editing of "scientific" text (grade B)

Greek characters and mathematic symbols are not part of the normal IBM character set. It is thus today impossible to type and edit a text containing greek characters and mathematic symbols with these characters visible on the screen.

The IBM strategic in-house-publishing product DCF "solves" the problem of printing such text by allowing the change of font command. However, typing

```
.bf greek
p
.bf special
+
.bf roman
p
.bf special
A
.bf roman
K
.bf special
+
.bf greek
S
.bf special
+
```

to obtain $\pi^+p \rightarrow K^+\Sigma^+$ is extremely clumsy compared to many (non-IBM) systems

having these symbols as part of their normal character set. This results in

- decreased productivity in typing the original text
- increased number of cycles for printing/correcting drafts

and thus in increased costs.

Looking at the market offerings of non-IBM equipment and the program offerings developed by the Research Division of IBM it is clear that the market segment having these requirements is large enough to be taken very seriously.

It should, however, be clear that this requirement should not be confused with the formatting of mathematical formulae in a text processor. Here we speak about typing and seeing the actual characters and not how they should be laid-out on the finished document.

3.1.10 Entering and editing text using a foreign alphabet, or special diacritical marks used in transliterations (grade B)

With the invention of electronic printing techniques (laser printers, dot matrix printers, ink jet printers, ...) the demands of the end user for flexibility in the use of foreign alphabets (e.g. cyrillic) mixed in with roman text, special diacritical marks etc. are met in many devices.

What is not met, and the reasons for it appear ridiculous to the end user, is the possibility to enter and edit such texts with the characters visible in a recognizable shape on the terminal used. To use DCF again as an example:

```
.bf cyrillic
ABV
```

is not really recognized as АБВ resulting in a much more error-prone text entry.

The customer base for the requirement of foreign alphabets includes a number of large export oriented manufacturing firms and banks. Requirement of the special diacritical marks is very large in computerized library catalogues - it is very sad to read the introduction to the new British Library catalogue, comparing it to the old manually typeset catalogue!

3.1.11 User defined code points (grade B)

There is no space left in the code pages for user assigned graphics.

Some installations require graphics like "diameter" ⌀ or "horizontal" ⊓. But where to place them in the code table?

A solution should provide for some codepoints not to be used by general software (printer drivers, compilers, etc.) but assignable by the user (installation).

3.1.12 System use of code points (grade C)

Besides the controls with codes X'00' to X'3F' some products require additional codepoints for internal use.

Fortran compilers (G1, H, H-extended) for example use X'FF' as an internal string delimiter. Thus it is impossible to define this code within a string to be transmitted to a file. Also within a FORMAT statement this codepoint cannot be used.

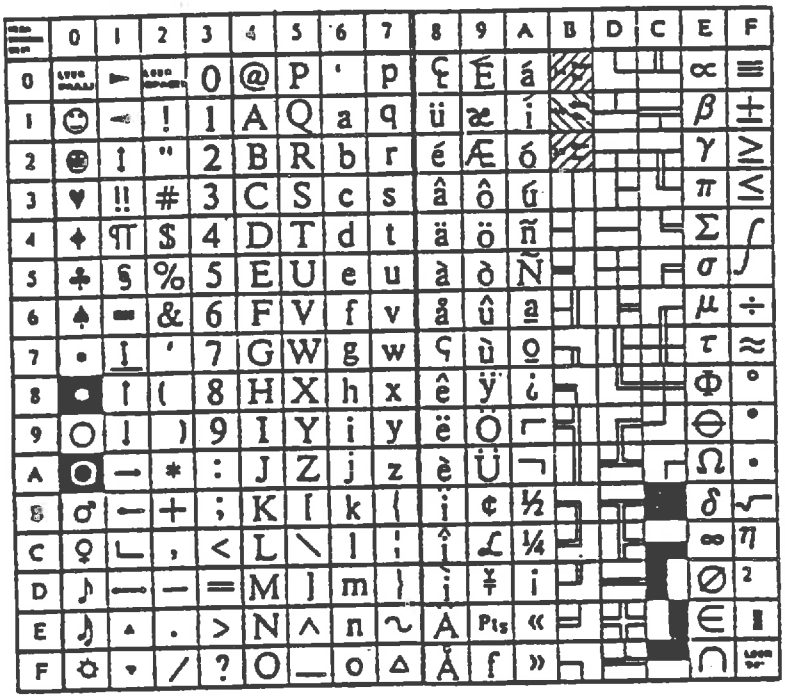
The use of this code point for this purpose is not bad but it should be known and clearly stated.

3.1.13 ASCII to EBCDIC Translation (grade B)

For reading and writing ASCII tapes on an MVS system there is only one translate table in the system. This table is based on the "old" EBCDIC assignment or "character set 103 of code page 256". An installation has no choice of installing several of these tables to support the various EBCDIC assignments. The final solution, however, should be that only one standard table is needed.

Using only a translate table rather than a real translate process does not allow the use of ANSI control sequencies (e.g. to switch to different character set, set tab stops etc.). In a mixed environment with both IBM- and "ASCII"-machines a more flexible generalized conversion mechanism is needed. With growing networks the number of "mixed environments" is also increasing.

The PC behaves in many aspects as an ASCII device, but the use of the control characters there is highly dependant on the device (screen, printer, ...). There is also no way to make the special graphics of the upper part of the code table availble to the EBCDIC environment, since the 8-bit code cannot be represented in two 7-bit code planes with a switch (SO and SI). This de-facto standard is not supported by any non-PC or host based software known to us. So the micro to mainframe link is still restricted to the US-ASCII graphics.



Code table of PC

3.1.14 Error graphic (grade B)

There is a strong need for an "error graphic". Currently undefined codes (no graphic assigned to that code) are presented on most devices as blank or minus '-'. This is completely misleading.

Assume for instance an Assembly language program (still in use!) specifying an area by DS C' ' (define storage) rather than DC C' ' (define constant). The DS defines (by action of the linkage editor) an area of X'00'. During program developement nothing happens because the stuff is printed only. But sent to a 327x this will be interpreted as control (buffer address)!

The 'error' character would be useful for displaying any 'text' that has codepoints with no graphic assigned on the outputdevice.

It may also be advantageous to be able to specify the 'error-fill' character or characters to be used in a particular output.

3.1.15 Currency Symbols (grade A)

The $, ¢ and £ code points and graphics lead an interesting life outside the U.S.A. To take just the $ as an example:

- The code points vary from country to country. X'4A' for Hebrew speakers, X'E0' in Japan, X'5A' in Brazil and X'5B' as 'standard'.
- In the English language area the UK has special problems. The £ occupies the same code point as the US $ (X'5B') whilst the $ occupies the same code point as the US ¢ (X'4A'). Since these are almost the only 3270 character set differences between the US and the UK, there is a tendency for suppliers to provide US instead of UK equipment. Thus £ frequently prints as $ and $ as ¢. Obviously an undesirable state of affairs.
- In Denmark, Norway, Sweden, Finland and Spain the $ is not available, its code point being occupied by Å or Pts.
- In Austria and Germany the $ is available to standard character set users, but not to those using the alternate set.
- The X'5B' 'dollar' position has been historically overworked as a delimiter character and is in fact still the default for JES2 - with unpleasing results in many countries.

The requirement is that if discrete currency symbols are to be implemented in the future, then they must occupy internationally fixed positions in the code table.

The NCTF cannot hazard a guess as to how many discrete currency symbols are neccessary. Presumably we require the $, ¢, £ and ¥ for international work. Note that many countries do not use discrete symbols, i.e. the currency symbol is compounded from upper and lower case alphabetics (thus Swedish kroner are 'kr' inside the country and 'SKr' or even 'SEK' outside).

3.1.16 Text and code

Text is a flow of characters. Their meaning is highly dependent on the context. Also the used graphic has different meanings in different environments. AM is an abbreviation both for Amplitude Modulation and Ante Meridiam! For coded characters it's even worse. Once in the computer storage the intention of the human is lost - if the meaning is not determinable by the context. With the current practice of many code tables used in different devices, the meaning is lost totally.

In the ANSI world (the mini computer world) a common practice is the code extension technique. Within the text the meaning of the code is defined. To switch to another meaning of the code a control sequence is used. The only thing needed is an authority defining the graphics with the associated control sequence... And this authority already exists. For example the ISO character set used in Videotex (identical to ISO 6937 part 2 with the exception of 2 flying accents not present there) is identified by ESC) b for use as G1-set.

There should be a distinction between input, storage and output of graphics (this term is used for punctuations, latin characters, semigraphics etc.). The current practice leaves out the intention of the text. Text with national-use-characters in it becomes meaningless, if this text was entered using different devices. To edit this White Paper we had to set up conventions on "national characters". A tape written in Denmark, read into a system in Switzerland cannot be interpreted without an accompanying manual description!

The input of special characters may be done with two keystrokes on an international keyboard or with special keys on a national keyboard. But the code generated and transferred to storage must reflect the complete meaning (e.g. base graphic and accent or visual interpretation of diphthong etc.).

On output the capabilities of a device must decide what to do with the internal representation of a graphic:

- place the accent onto the base character by overprint
- leave the accent out, if there is only a limited character set (which would be a bad option for Turkish)
- substitute the two byte sequence (e.g. print ae instead of ä)
- select a combined graphic (e.g. for 3800 printer)

3.2 National language problems

Once entered into the system, the various national use characters combine with the Latin alphabet to form text in a national language. This section describes the problems encountered in processing national language text. Also included are problems encountered in getting software products to 'converse' in the desired national language. These problems are of special concern for products which reach the end-user directly - for example 'information center' products.

3.2.1 Messages (grade B)

Messages are not generally available in the national language of the country in which a given software product is being used. In most cases there is no easy, well-documented method for the customer to translate these messages. In some cases the text length available for a possible translation is too small.

The term 'messages' in this context is to be understood as

- Automatic error messages
- Automatic informational and action messages
- 'HELP' information explicitly requested by the user.

The following examples are taken from IMS messages. Although not classed as an end-user product IMS (like CICS) provides the end-user of a DP application system with a bewildering variety of messages of a highly technical nature which are difficult enough to understand for English-speaking users. The first step is to be able to see these messages in one's own language. Here are some literal translations :

```
DFS053 TERMINAL RESTARTED - PLEASE REFORMAT SCREEN
DFS053 Votre terminal vient d'être re-démarré.
       Veuillez ré-initialiser votre contenu d'écran
       (French)
DFS053 Terminal wieder gestartet -
       bitte Bildschirminhalt neu aufbauen
       (German)
DFS053 Terminalen er genstartet -
       vær venlig, at genformatere skærmen
       (Danish)

DFS064 NO SUCH TRANSACTION CODE
DFS064 Le code-transaction n'existe pas
       (French)
```

```
DFS249 NO INPUT MESSAGE CREATED
DFS249 Keine Eingabe erhalten
       (German)

DFS058 EXIT COMMAND COMPLETED
DFS058 Exit kommando genomfört
       (Swedish)
```

The IMS system has no formal procedure for translating these messages. Some messages are in separate load modules, others are contained in program modules and the actual text content of a given message may be run-time variable.

As already mentioned, the content of messages and help information may be so full of DP jargon as to be unintelligable even when literally translated. So the translation process is required not only to achieve the desired national language but also to

- correct spelling from US to UK English
- provide dual case text
- provide more lucid and jargon-free explanations
- provide application-orientated text

Here are some more examples to illustrate this, all in English in order to show that the problem is not confined to non-English-speaking countries :

```
JOB CANCELED
This job has been cancelled.
```

(Dual case, spelling corrected, same content but more user-friendly)

```
DFS064 ?
DFS249 ?
```

(In reality both messages, as shown in the introductory example, simply state that IMS cannot understand or accept the input. The installation has replaced both texts with a "?").

```
DFS058 PRESS PA2 TO GET LOGON SCREEN
```

(The user is told what to do next. This translation could only be achieved by using a fairly complicated exit routine, since the first word of the original message, 'EXIT' in the introductory example, is run-time variable)

Note that in some installations there has been a tendency to remove messages completely, in order to protect the end user from sudden bursts of 'foreign language'. Needless to say, this approach has abvious disadvantages when errors have to be diagnosed!

The requirement is that all messages be capable of customer-translation according to the individual customer's needs. The message may be required to be shortened or (considerably) lengthened by translation. The verbal content (meaning) of a message must be unique for a given message number (processing situation) and must be held separate from any run-time variable information. The method of translation must be well-documented and easy to carry out in practice - for example editing of a file or re-assembling of modules containing the text.

Note also that the customer may only wish to translate a frequently used subset of all possible messages for a given product (For IMS, under 5% of possible messages ever reach the application user's screen).

Thought should also be given to the customer's maintenance problem when installing new versions of the product. the requirement is that previous 'translations' should be useable without the need for complete retranslation. This means that new versions should specifically document

- Altered meanings to existing messages
- New messages

in relation to the previous version.

3.2.2 Reserved words (grade B)

It is not generally possible for the customer to specify alternative reserved words or keywords in his own national language. This is of special concern in products which reach the end-user directly as for example Information Centre type products.

For instance :

```
CREATE VIEW MIJNZICHT (WKNR,NAAM,AFD,JOBK)
  AS SELECT WERKNEMERNR,FAMILIENAAM,AFDELING,JOBKODE
  FROM DSN8.TEMPL
WHERE JOBKODE = 52
   OR JOBKODE = 54
   OR JOBKODE = 56
SELECT *
  FROM MIJNZICHT
```

may be required to be written as :

```
MAAK ZICHT  MIJNZICHT (WKNR,NAAM,AFD,JOBK)
  ALS KIES  WERKNEMERNR,FAMILIENAAM,AFDELING,JOBKODE
  VAN  DSN8.TEMPL
WAAR  JOBKODE = 52
   OF JOBKODE = 54
   OF JOBKODE = 56
KIES   *
  VAN  MIJNZICHT
```

which is now completely in Dutch. The requirement is that such specification of 'synonyms' be possible and that the specification process be well-documented and easy to do in practice - for example by re-assembling a module containing the text or by having the text in a separate file which is accessed during the product's initialisation phase.

3.2.3 Fixed text (grade A/B)

It is not generally possible for the customer to specify fixed text in his own national language - and it is mandatory that all user-directed output from an execution be entirely in the desired national language.

By 'fixed text' is meant the type of (usually short) text which has a tendency to be hard-wired into a product, often in several different modules. Examples are

- 'PAGE'
- 'DATE'
- 'CONT.' or 'CONTINUED'
- The names of the days
- The names of the months
- The names of colours
- Text accompanying graphs, charts and tables
- etc. etc.

The requirement is that fixed text be specifiable by the customer as for example in 3.2.2.

3.2.4 Menus and Prompts (grade B)

This is presented as a separate problem area since although analagous to the previous subsections, the technical aspects are different.

It is not always possible for the customer to specify the fixed text fields of a menu or a prompt in his own national language. Even when this can be done, there may be

problems with fields which require fixed input in English.

For example

```
Copy complete
Copy another (Y/N)?
```

Must be translatable to

```
La copie est terminée
Encore une copie (O/N)?
```

and the product must be able to accept the French Oui instead of Yes.

The requirement is that such translation be possible, that it be easily performed and the method well-documented. As with 3.2.1, the customer must be helped in the maintenance situation.

3.2.5 Dates (grade A)

It is usually impossible to specify the form of the date which is to appear automatically on user-directed output. It is mandatory that dates supplied automatically by a product be in a nationally or locally understood format.

To cut an otherwise lengthy description of the problem to a minimum, it can be stated that the automatic date format should be entirely at the customer's control, especially since the format may be application-specific (even the ISO standard may not be acceptable!)

For example the American date format for February 4th 1985, supplied automatically by 'product X' would typically be

02-04-85

which is understood as 2nd April 1985 in most of Europe. The requirement is that the customer be able to specify

02/04-85	USA
1985-02-04	ISO standard
04.02.85	Europe
4 2 1985	Europe
4-FEB-85	Europe
4 FEV 1985	France
4 février 1985	France
Den 04/02 1985	Denmark
LUNEDI 040285	Italy

or whatever.

More problems will be solved by allowing freedom in this respect than by assuming national standards. The examples above are not all national standards but they are reasonable and nationally or locally understandable in each case.

The 'fixed text' problem of day and month names has been covered earlier.

Note, by the way, that a global IBM 'utility solution' to the date formatting problem would be the best long-term goal. The automatic date format would then reduce to a simple special case. The global solution would of course be 'callable' from any programming environment and would include the necessary new developments required to distinguish dates before and after year 2000.

3.2.6 Thousand and decimals (grade A)

It is generally not possible to specify the delimiting character for editing thousands and decimals when the output is produced under the control of a standard product, or when input is interpreted under the control of a standard product. For example

10.000 yards	is ten thousand yards in the USA.
10.000 meters	is only ten meters in Europe.
1.000 hours	is one hour in the USA, but nearly a month and a half in Europe.
113.500 guilders	may leave the recipient in doubt as to whether he is rich or not.

It is required that the customer be able to specify the character to be used to delimit thousands and decimals when the output and/or input is under the control of a standard product. This must be 'parameter option' and not a 'standard for Europe' since usage varies from country to country. For example Sweden uses the decimal point whilst Denmark, Germany, Austria and Switzerland use the comma.

Note that the input situation raises interesting problems, since the comma has become generally accepted as an inter-field delimiter for 'stream' input. The implication is thus that the inter-field delimiter(s) also be user-specifiable. Currently it may be impossible to enter a comma in a numeric field on certain terminal equipment.

3.2.7 User-specified names (grade A)

National characters are not always eligable for use in user-specified names.

For example MVS JCL only allows use of the 3 'traditional' national use characters, with graphics corresponding to code points X'5B,7B,7C', in names on JCL-statements. This means that users of the standard Austrian/German 3270 code cannot include the national characters Ä, Ö, Ü in their JCL (they could with the original 'alternate' code) :

Country	JCL	Result
U.S.A.	//DD1 DD DSN=ANDERSEN.#69K,...	OK
Denmark	//DD1 DD DSN=KMD.ÅRHUS,...	OK
Germany	//DD1 DD DSN=OERLIKON.BÜHRLE,...	JCL error

It has also been suggested that lower case should be accepted, together with some special symbols such as the useful '_' for binding words together.

The requirement is that all software must at mimimum accept valid national characters and that the present over-restrictive rules in some products be relaxed wherever posssible.

3.2.8 Lengths of names (grade C)

It has been suggested that the 8 character limit on the lengths of some variable names may be too short to allow reasonable mnemotechnic use in some countries.

- member names in partitioned data sets
- elements of data set names
- keywords
- etc.

It would appear to be a reasonable requirement, especially in end-user directed products, that names could be longer than 8 characters (16, 32 ?) or in fact variable-length up to a fairly high limit.

3.2.9 Delimiters (grade A)

This is a typical 'grey area' for many countries. The problems which arise can be roughly grouped into two areas :

a) Use of the three 'traditional' national use code points X'5B,7B,7C' (with US graphics $, #, @) as syntax elements or delimiters.

   A typical example here is the very popular Statistical Analysis System (SAS) which makes heavy use of all three of these US graphics in its input/output syntax. This leads to programs which are confusing to read and in which the use of national characters in variable names is prohibited.

   Although not an IBM product, SAS has nevertheless been recommended by IBM as a useful info-center product.

b) Use of other special symbols such as [ ] { } \ .

The EBCDIC code points for these graphics are X'AD,BD, C0,D0,E0'. Looking at the 3270 codes, however, we find [ ] assigned to X'4A' and X'5A' for Belgian, Portuguese, Spanish and Swiss users. All 5 code points, X'4A,5A,C0,D0,E0', are also candidates for national characters. It is therefore certain that a mainframe software product using these symbols will run into some sort of trouble in Europe.

Using a national ISO standard coded character set on a PC, one or more of these graphics may be replaced at the standard ISO code point by a national character. Examples of software products which are adversely affected by this are PASCAL compilers and the IBM-marketed MULTIPLAN.

The requirement is that no such 'dualling' of the graphics for a given code point (and vice versa) exist. A national character graphic must never carry accidental syntactical significance by virtue of its code point.

3.2.10 Case conversion (grade A)

Some products may still employ 'folding' to convert lower case entered at terminals to upper case for further processing. This has obvious repurcussions for national use characters which have code points between X'3F' and X'80' and are therefore unchanged by folding (OR-ing with X'40').

Lower case to upper case translation only respects the 26 letters of the english alphabet. For example, 'é' stays as an 'é' instead of an 'E' with no accent. Furthermore, if there is a hardware folding facility on some 3278 screens, it concerns its whole contents, and seems to ignore some special graphics e.g. these associated with the dead key if the PRPQ is installed.

Another consequence is the translation of national characters to blanks or punctuation marks when automatically folded to upper case in many software products.

Further there are usages that can differ from one country to another. For example, 'European French' people do not convert accented letters into upper case accented letters, as the Canadians do. Thus the 'European French' upper-case version of the sentence

"Les bûches brûlent tête-bêche dans l'âtre de la cheminée: c'est Noël" is

"LES BUCHES BRULENT TETE-BECHE DANS L'ATRE DE LA CHEMINEE: C'EST NOEL".

Whereas the Canadian French is:

"LES BÛCHES BRÛLENT TÊTE-BÊCHE DANS L'ÂTRE DE LA CHEMINÉE: C'EST NOËL"

There are also cases in application systems where upper to lower case conversion must be done and a new problem is obviously encountered here when a non-accented BUCHES must convert to the accented bûches.

No products can do this now and the conversion function is usually achieved by

software with loss of accents from lower-case to upper-case. Some display hardware, however, is equipped to do the right job. This is to say that it is possible.

The requirement is that case conversion can be done in both directions and without loss of information.

3.2.11 Translating (grade A)

Incorrect results are often obtained via the use of translate tables incorporated into products. These tables are used variously for

- case conversion
- code conversion
- screening of terminal I/O
- etc.

The tables are invariably set up for US standard usage and do not reflect national use characters. When the customer realises what is wrong, he begins the search for the translate tables, which may not be easy to find and may not be documented.

The ultimate requirement is of course that no manipulation of translate tables be necessary - that is that the required national tables are automatically selected and used, or that one standard table is sufficient. The interim requirement is that tables are well documented as to position and usage so that the customer can make any required modifications easily.

3.2.12 Double characters

There are several problems associated with the existence of 'double characters' in written languages. All have a tendency to be special cases and the only common denominator is the fact that the resulting two contiguous character positions must be treated as one unit during processing. There is no IBM hardware or software solution for any of these problems.

Example 1: Single character and equivalent double character may coexist in same text.

> The Danish 'long a' is correctly written as Å, å. However this usage stems from a fairly recent spelling reform which allowed continued use of the old forms AA, Aa, aa in proper names.
>
> Hence AALBORG, Aalborg, Århus, Aargaard, Åse, Aase, mål etc.

Example 2: Single character lower case character converts to double upper case character.

> The German lower case 'double s' is correctly written as 'ß' which becomes 'SS' in upper case.

Hence Hauptstraße, HAUPTSTRASSE.

Example 3: Unable to enter a national-use character, which therefore has to be split into a (strictly illegal) double character.

The French (and latin) dipthongs Æ, æ, Œ, œ cannot be entered as such on a French keyboard. The only alternative is to enter them as the double characters ÆE, ae, OE, oe.

Hence the correct œuvre becomes oeuvre.

An interesting (and deplorable) side effect of the double character problem is introduced in the last example. Many DP-personnel have given up the unequal struggle of transmitting national-use characters outside 'the country of origin' and use artificial double character combinations instead.

This is specially prevalent in names and addresses:

Günther Krysmanski	becomes	Guenther Krysmanski
CH 8050 Zürich	becomes	CH 8050 Zuerich
DK 8200 Århus N	becomes	DK 8200 Aarhus N

All of these are glaring inaccuracies caused by the lack of proper international character support in current systems.

3.2.13 Sorting single case text (grade A).

The order of items sorted alphabetically on one or more character fields is incorrect.

Example from France

EBCDIC collating	correct sequence
frêle	frêle
frêne	frelon
frelon	frémissement
fret	frêne
frémissement	frère
frère	fret

Examples from Denmark :

EBCDIC collating	correct sequence
ÅGÅRD	ANDERSEN
ÆGIDIUS	NIELSEN
ØRSTED	ZEUTHEN
ANDERSEN	ÆGIDIUS
NIELSEN	ØRSTED
ZEUTHEN	ÅGÅRD

3270 alternate	3270 standard	correct sequence
ørsted	ørsted	andersen
ågård	andersen	nielsen
ægidius	nielsen	zeuthen
andersen	zeuthen	ægidius
nielsen	ægidius	ørsted
zeuthen	ågård	ågård

Example from Germany :

3270 alternate	3270 standard	correct sequence
Buße	Busfahrt	Busfahrt
Busfahrt	Buße	Buße
Busse	Busse	Busse
Gõfis	Gõfis	Gasse
Güte	Gasse	Gäßlein
Gäßlein	Güte	Gõfis
Gasse	Gotte	Gotte
Gotte	Guthaben	Güte
Guthaben	Gäßlein	Güthaben

If the product which executes the sort uses the Sort/Merge utility it may be possible to specify a national 'ALTSEQ'. If this parameter cannot be specified or if Sort/Merge is not used, the sequence will be incorrect (e.g. sorting in the EDITOR of ISPF).

Even when ALTSEQ can be specified (e.g. when using Sort/Merge directly), some extra user action is usually/always (?) required. When the sorting algorithm is coded directly in the application program, complicated procedures are necessary in order to ensure correct sequence (at least one translate of all sort fields, or two translates, depending on the methods used).

SEAS National Character Task Force

It is an accepted fact that correct sort sequence cannot be achieved without causing some degree of CPU overhead (for example between 2 and 5% for typical sorts using Sort/Merge and ALTSEQ).

The requirement is that correct national sort sequences be achieved automatically (as for 3.2.11 Translating). It should be noted, however, that it must still be possible for the user to manipulate the sort sequence (as with ALTSEQ) for special applications.

3.2.14 Sorting mixed case text (grade B/C).

It is impossible to sort mixed case text correctly unless some type of 'ALTSEQ' mechanism is used. The code points for the lower and upper case latin alphabet in EBCDIC cause the difficulty. The situation is further complicated by the presence of national characters.

Sorting mixed case text thus requires special mechanisms and involves CPU overhead.

National double characters require very complicated pre-processing of the input to the sort if correct sequence is to be achieved. For example :

Germany Lower case double s is a single character ß which must sort in company with upper case 'SS'. A-umlaut Ä and a-umlaut ä must sort in company with 'ae', 'Ae', 'AE' or with 'a' 'A'.

France The (single character) dipthongs Æ, æ and Œ, œ must sort in company with 'ae', 'Ae', 'AE' and 'oe', 'Oe', 'OE'.

Denmark Å,å 'AA', 'Aa' and 'aa' are all functionally equivalent and must sort in each other's company. Note that the equivalence of Å and AA also gives problems when sorting in upper case!

There is currently no IBM support for correct mixed case sorting with double characters. There are, however, non-IBM systems on the market which handle this situation correctly. The problem area is of increasing concern as more applications make use of mixed case text data.

The requirement is that sorting of mixed case text be automatically and correctly achievable as for 3.2.13.

3.2.15 Multiple languages (grade B)

The problems described in section 3.2 relate to the situation in a single European country using a single written language. However, as has been noted in section 3.1, most installations require the ability to process some words containing national use characters from neighbouring countries and many installations require the ability to process complete texts in other languages. These problem areas are in fact only subsets of the general problem which is seen at its worst in multilingual countries, where several different languages must be processable at all installations. Major examples of these countries are:

Switzerland	(German, French, Italian)
Belgium	(Dutch, French)
Luxemborg	(French, German)
Spain	(Spanish, Catalan)

In addition we must not forget the increasing number of 'international facilities', for example information retrieval systems which may be accessed from all countries via international tele-networks.

An additional and extremely important requirement for the whole of section 3.2 is thus the ability to operate software in the language of one's choice.

The 'switch' from one language to another should be at minimum possible on a job-step or on-line session basis:

not	`ispf`
but	`ispf lang=german`
not	`// EXEC PGM=SORT`
but	`// EXEC PGM=SORT,LANG=SWEDISH`

For on-line sessions it would obviously be desirable in multilingual countries to be able to switch languages in mid-session, without the need to logoff.

Note that the language switch must initiate all aspects of national language and national character support such as translate tables, sort sequences, error messages etc. etc.

3.3 Keyboard related problems

It is difficult to talk about the national character aspect of keyboards without mentioning the many other deficiences in current keyboard implementations. We therefore make no excuse for expanding the subject matter of this section to include aspects which are not specifically related to the national character problem area.

The important point to note is that the national character aspects of keyboard operation and geography must be integrated into a much needed 'international standard keyboard solution' such that national keyboards appear as well defined and easy-to-use subsets of a specific standard.

IBM offers a bewildering variety of keyboards with different layouts, but the tendency nowadays is that a given keyboard may in fact be in use for one or more application types arbitrarily lelected from

- Programming and computer operations
- Data entry
- On-line DB/DC preprogrammed applications
- Information center (including APL)
- Text processing
- Graphics
- etc. etc.

For most of these keyboards it can be stated that

- The geography is wrong for a given application (for example a typist asked to use a 3278 keyboard, or anyone trying to activate 'clear screen' or 'PF1' on this keyboard with one hand, holding a telephone or a pencil in the other!)
- National character support may be poor for the country in question, bad for neighbouring countries and probably non-existent for international use.
- Confusion and low productivity result when a user has to deal with a number of different keyboards at the same site (watch a PC-user trying to manipulate the cursor control keys on a 3278!).

3.3.1 Non consistent character sets across products (grade B)

Look at these two IBM products: two keyboards designed for France.

French IBM 3278 keyboard

```
!  "  §  $  %  +  /  (  )  =  ?  ~
1  2  3  4  5  6  7  8  9  0  '  ^

A  Z  E  R  T  Y  U  I  O  P  ç  *
a  z  e  r  t  y  u  i  o  p  à  &

Q  S  D  F  G  H  J  K  L  è  °  £
q  s  d  f  g  h  j  k  l  é  ù  ¨

>  W  X  C  V  B  N  M  ;  :
<  w  x  c  v  b  n  m  ,  .  -
```

French IBM typewriter keyboard

```
1  2  3  4  5  6  7  8  9  0  °
&  é  "  '  (  §  è  !  ç  à  )  -

A  Z  E  R  T  Y  U  I  O  P  ¨
a  z  e  r  t  y  u  i  o  p  ^

Q  S  D  F  G  H  J  K  L  M  %
q  s  d  f  g  h  j  k  l  m  ù

W  X  C  V  B  N  ?  .  /  +
w  x  c  v  b  n  ,  ;  :  =
```

The first keyboard comes from my 3278 terminal, the second one represents the keyboard of my secretary's IBM typewriter. It is obvious that decimal digits are in lower case on the 3278 and upper case for the typewriter but they are at least on the same keys.

Special characters are not placed at the same location. One key, a dead key on the typewriter with the circonflex accent (â,ô,û) and the trema (ë,ü), does not work in the same way on my 3278 terminal which has no dead key facility. Even the PRPQ which allows this facility does not give the full flexibility of the dead key. But the worst is the M key, an important alphabetic character, which is not on the same row in the two keyboards.

This simple example explains why French secretaries are very reluctant to use IBM 3278 terminals for Word Processing. The French case is specific to the French character set but similar situations exist in other European countries. German secretaries, Scandinavian secretaries and so on are also reluctant to move from their typewriter keyboard to a WP terminal which changes their habits, decreases their

productivity, adds more mistakes and does not contain all the needed features.

Even UK, an English, not American speaking country, is concerned with the pound key (£) and the dollar ($) key when the internal representation is exchanged. Just note also that my French keyboard contains the pound (£) as the national monetary symbol and not the French franc (F) but the IBM 3203 printer chain prints the pound sometimes as a big F, the french franc, or as the US number symbol # which is no longer available on my screen.

3.3.2 National and international keyboards

Specialized hardware exists, e.g. for newspaper composition, but in fact, we don't know any "infocenter" keyboard able to enter in a simple way the whole set of characters that it would be advisable to have under the hand for current national work, and furthermore for international needs. The alternatives offered are very poor:

- if one uses for example DCF/GML, one can enter text, but one has to remember to enter some complex expression for each character combination provided and to spend extra software overhead when interpreting it, and to spend new human time to verify it in output, and to rerun in case of mistakes ...
- if one is not entering text, one can choose a technique (well-known in APL) which consists of selecting a needed character from a matrix by indexing, but this is not possible in all contexts, and doesn't allow character combinations.

A better approach is given with use of the Programmed Symbol Sets 3279-2b or 3b (and also some 3278 special models), which enable one to load and select the character sets required. They are named "Programmed Symbol Sets", and one can define them oneself (using a special editor) if necessary. But there are several major inconveniencies:

- In fact, one needs a special environment, installed and maintained by trained personnel (not cheap by itself !), which does not give any immediate support.
- One is producing codes that can be recognized and processed only (today) in the same environment, so it's very difficult to merge them with other sources and produce an external text output.
- One has to remember your codes rather than find them on the key-tops. The keyboard template supplied with some models is not usable for serious work and it doesn't fit on the handy models.

3.3.3 Lack of grouping of functions (grade B)

Existing keyboards for 327x devices try to group several functions like screen control, programmed functions, cursor and field control. Since the functions of terminals have become more complex (colours, screen sizes, ...) or a device has become multi functional (3270 PC or common PC with emulation etc.) a good grouping of functions becomes very necessary. But basic design principles are often missing:

3270 PC

- HOME only in conjunction with ALT
- PC functions intermixed with terminal functions on same key (CAPS LOCK / SHIFT LOCK). Shift lock is a reminescence from the stone age of typewriters and should be abandoned completely.
- different graphics assigned to a key in terminal and PC mode (!¢ with {[ ).
- mixing of cursor control in PC mode with numeric cluster
- etc. etc.

3278

- HOME only in conjunction with ALT
- extra key for < > but comma and period twice (upper and lower case) on same key

3279

- PF1-PF12 in one row on top rather than in a PF-cluster and PF13-PF24 on shift-(PF1-PF12). The linear arrangement may be good for placing a template, but it is very hard to pick a specific key without reading the keytops. The geometrical arrangement 3x4 can be used by intuition.

In general there are too many keys for normally sized hands in the central area of the keyboard (typewriter area). Although this is according to an ISO standard, the key left of Z is very annoying. Also the rightmost key in the "home row" with { } is not good. They both increase the distance to the SHIFT and RETURN keys. This is at least true for devices sold in Switzerland.

Currently we have to deal with many keyboard layouts:

- typewriter keyboard
- data entry keyboard
- data entry keypunch keyboard
- APL keyboard
- text keyboard

In many cases one person is responsible for several jobs and thus has to become familiar with all of these layouts. Of course the usage of so many different keyboards is error prone and decreases productivity.

4 Required characteristics of solution

This section lays out the conditions proposed by the SEAS user community for future solutions to the problems described in section 3. The technically expert reader will have realised that the general solutions for the three problem areas in section 3 are

National character problems

A character set which completely satisfies the needs of a particular language and which additionally allows the use of characters for all European countries without compromise of any kind. Each character has to have a unique representation in order to enable communication with other countries.

Hardware devices which support this character set.

National language problems

Software products and processes which are language independent and which can be switched to the desired language by a simple and automatic process.

Keyboard related problems

Keyboards which are nationally standardised so as to give optimal efficiency for entering text in the primary language or languages in a given country, and which also allow true multilingual use, i.e. all national use characters can be entered.

Standard layout and operation for other, non-language-dependent, functions.

For all three problem areas there is also the requirement that the present differences in character set and function between data processing (DP), word processing (WP) and personal computing (PC) be resolved into one 'system image'.

The chaotic state of affairs described in section 3 demands some sort of integrated approach to problem-solving. Many problem areas are so interlinked that further blatantly ad-hoc development (for example the PC character set) or piecemeal 'solutions' (for example the alternate character sets) must be banished.

SEAS believes that all future development in the national character, national language and keyboard areas must be coordinated in a well defined direction and that the objective should henceforth be a new level of system architecture.

SEAS requests that IBM give top priority to the developement and implementation of this architecture. SEAS also requests IBM to provide its customers with one or more 'statements of intent' as early as possible in the developement process.

These statements must be such as to enable customers to protect their considerable present and future investments in character set dependent hardware and software during the period until products based on the new architecture are generally available.

We will take the liberty of naming the new architecture 'National Language Architec-

ture' or 'NLA' in the remainder of section 4. Section 4.1 further outlines the required concepts in NLA. Sections 4.2 to 4.5 outline requirements for the conversion process.

We would like to emphasize here that these sections are very much 'open-ended'. We feel that a useful dialogue has been started with IBM in this very difficult and complex area. We hope that IBM will see the need to proceed with an untraditionally high level of customer consultation when planning for NLA. The present White Paper is a starting point and we believe that SEAS and the other IBM user organizations, GUIDE and SHARE, will be able to provide a focus for future discussions with IBM.

We do not specify here that NLA should be based on existing international standards. However we would strongly recommend that IBM consider the use of international standards wherever applicable. We would also ask IBM to take note of SEAS' special interests in this area, together with SEAS' expressed interest in participating in international standards work in the future.

The Task Force has also discussed the possible nature of the conversion process from the current status quo to NLA. There seem to be two extremes - an evolutionary process and a revolutionary process.

Again we cannot at this stage recommend either extreme; this decision must also be the subject of further analysis and consultation with the user community.

In practice we suspect that the process will be mixed - evolution over longer periods to protect customer investment coupled with one or more revolutionary stages to attain the final goal. In this light it is difficult to lay down requirements for the conversion process. Suffice it to say that sections 4.2 to 4.5 are based on the supposition of a single revolutionary change. Some items will obviously be redundant in an evolutionary process, but we feel obliged to guard customer interests in a 'worst case' situation (and here we think that the traumatic experience may easily be several degrees worse than a VSE to MVS/XA conversion).

Many of the items discussed will apply to any type of change and we are confident that IBM does its best in most cases to protect customer interests - which is what sections 4.2 to 4.5 are concerned with. However the Task Force is of the opinion that IBM's track record to date on protecting customer interests in the problem areas of section 3 is so bad that all possible initiatives must be taken to improve on this in the future.

As regards the specific area of keyboard design, we include the section 'Keyboard requirements' as an appendix to the White Paper. The literature is rich with proposals and new developments in this area and, as mentioned already, the NCTF has felt obliged to state its point of view here - keyboard design for NLA must be coordinated with the many other aspects of keyboard use.

Having said this, we acknowledge that the subject matter is in many ways peripheral to the main stream of the White Paper. 'Keyboard requirements' is thus to be seen as an 'appetizer' for very necessary future work and not as a set of requirements on the same level as the present section.

SEAS National Character Task Force

4.1 System architecture

As already stated, SEAS believes that the ultimate goal should be a new level of system architecture - National Language Architecture or NLA.

Amongst the characteristics of NLA should be

- A set of well defined rules at the highest level regarding the entering, displaying, storage, processing and interchange of data containing national-use characters, Greek characters, mathematical symbols and foreign alphabets.
- A well defined open-endedness to allow future development without the need for expensive hardware or software changes. It is acceptable that some expensive hardware features - like multiple character sets on an impact printer - are an optional enhancement.
- An end-to-end protocol for IBM system sender and receiver without the need for translation - even on an international basis.
- Transmission into and out of the IBM system environment (from and to equipment using other character standards, either supplier based or internationally agreed) via an open ended set of automatic translation processes.
- All graphics of all countries available everywhere and assigned to internationally unique code points.
- Provision of new hardware and/or software basic functions where necessary.
- Intelligent treatment of lacking graphic support for a given code point on all output devices. The action to be taken in each case to be user selectable/definable as an integral part of the architecture.
- Standardisation of keyboard geography and method of operation for all system types (DP, WP, PC etc.).

Once the basic concepts of NLA are agreed on and statements of intent or planning guides are available, the necessary software can be developed.

Note though, that many of the problems outlined in section 3.2 can in fact be tackled at once and without compromising future use of NLA facilities.

Amongst the characteristics of NLA-based software should be

- Character set independence.
- Language independence and language tailorability in all products and to a degree which can be controlled by the customer.
- National language versions for all 'major' end-user products and languages without the loss of tailorability if so desired.
- Full documentation for all language facilities and the process for tailoring them.

- Choice of language on at least a per session and per job-step basis without incurring the overhead of having to run multiple copies of the same product.
- National 'typography' (prefered quotation marks, date format, decimal point/comma, currency symbols etc.) and national sort sequence to be considered part and parcel of the language in effect at any time.
- Current 'language in effect' to be available to executing programs via a system variable. The syntax and value set of this 'language indicator' (and other necessary indicators) to be internationally agreed.
- Guidelines for writing language independent and language tailorable software. These guidelines to be enforced for all third party products marketed by IBM.

4.2 Coexistence of existing and new solutions

Existing methods and NLA will have to live alongside each other in a typical installation for many years, both during the period of NLA development and after NLA becomes generally accepted (witness the timelags experienced with conversions from BTAM to VTAM and BSC to SDLC). This coexistence of different methods must be capable of being controlled by the customer by means of existing (or new) methods of parameterization such as

- Default software installation parameters.
- Run-time execution parameters

```
// EXEC....PARM=
// DD......RECFM=
/*JOBPARM....
Session parameters
```

- VTOC/LABEL information
- On-line commands in TP-systems, System macro calls in application programs
- etc. etc.

The required mechanisms must be installed in all environments: OS, VSE, VM etc. and in all products: CICS, IMS, TSO, CMS, SORT/MERGE, SQL, COBOL, PL/I etc.

Traditional or NLA support must be definable for all I/O paths which are not directly specifiable via JCL such as

- JES writers

- Individual TP-nodes under control of CICS or IMS
- etc.

New hardware with NLA-capability must be designed so as to be mode-selectable by simple switching (or must be easily field upgradeable) from traditional support to NLA support. This applies specially to TP- equipment which may by distributed over a wide geographical area and which will introduce severe logistical problems if switching is a complicated process.

Needless to say, traditional (existing) and NLA support must be able to coexist on the same TP-network under control of one TP-system. It must be possible to drive NLA-devices from existing application systems without the need to modify user coding and without incurring loss of function or efficiency.

It is of course accepted that some degree of program modification will be necessary in order to fully utilise the new functionality in NLA.

4.3 Continuance of support

The user must not be forced into making conversion decisions because of the sudden unavailability of a product or the withdrawal of existing IBM support.

Hardware/software products must allow continued use of existing methods for a time period which is commensurable with the user's investment in these products. The Task Force has discussed periods of the order of 5 years but it should be noted that for example 3270 equipment on large DP networks often has a lifespan of over 10 years. The investments to be protected are not only those of the IBM customer-installation but also those of the installation's own customers in turn.

It is accepted that IBM will implement new functionality in new product releases or versions which support NLA, but every reasonable effort should be made to avoid situations where a customer is forced to accept NLA in order to benefit from new non-NLA functions. It is suggested that products be developed so as to allow 'NLA/NONLA' parameterization, rather than offering two versions of the same product.

Support for existing data representation must never be completely dropped - at least the required conversion products must always be available. This to guarantee the survival of large data archives which it would be impracticable to convert other than when actually accessing them.

4.4 Cost of new solutions

IBM should note that the European user community has already paid a high price for living with the present day chaos of character and language problems and product incompatibilities. SEAS would therefore suggest that every effort be made to reduce conversion costs, plus the cost of the actual NLA-support, to an absolute minimum.

Basic data and software conversion tools should be provided free of charge (or very cheaply) by IBM.

Conversion will under the best circumstances probably be a major upheaval for the typical installation. Potential 'cost-barriers' should therefore be reduced to an absolute minimum.

Ultimately we expect both customers and IBM itself should experience a marked reduction in effective costs as conversion to NLA is achieved.

4.5 The need for flexible conversion tools

IBM tools must be available for the conversion of stored data (disk, diskette, tape,....) to NLA format. These tools must be able to convert data organized for all access methods (QSAM, VSAM, ...).

Tools should also be provided for the conversion of existing customer-written source code.

Tools must be flexible, cheap in operation and easy to use - also by untrained personnel (end-users for example).

Conversion must be possible without corrupting existing levels of data security and integrity.

Full details of the hardware/software implementation of NLA together with the conversion tools must be available in the public domain.

Planning information must be released as early as possible and at suitably frequent intervals so that IBM customers and third party suppliers can plan for conversion.

5 Epilogue

The chairman of a task force can, I hope, be allowed the luxury of appending a few words of his own.

This White Paper has taken form over a period of roughly three years. The process has never been easy. Looking back on the result, I think I speak for all of us when I say that there are still a number of loose ends and of course there are errors and misunderstandings here and there. We have all had problems finding time to do the work, which has had to be squeezed out of the usual tight schedules which are the constant burden of all DP people.

On the other hand I would like to emphasize that the subject matter has come 'straight from the horse's mouth', from people in many different countries who experience the problems of section 3 as very real frustrations in their daily work. I hope that this rather direct approach is sufficient to offset any lack of technical completeness in the White Paper.

I am confident that our primary objective has been reached - a concentrated survey of a widespread set of related problems has seen the light of day and the IBM organization has been made aware in no uncertain terms of the plight of its European customers.

On my own behalf and on behalf of SEAS I would like to record many thanks to the following people:

- NCTF member Klaus Daube who has single-handedly been responsible for the layout and typography of the White Paper (using the SUSI text formatter and a XEROX 2700 printer at Oerlikon Bührle Rechenzentrum).
- Lisbeth Dreyer who word-processed the greater part of the text at Kommunedata's Hospital Datacenter.
- Jens Lynge, Handelsbanken, whose untiring efforts finally succeeded in getting an NCTF team together.
- Knud Nielsen, Handelsbanken, the first chairman of the NCTF.
- Jerry Andersen, the IBM representative on the NCTF, who has been indespensible in many ways. Jerry's expert kwowledge of character sets has carried us over many difficulties and he has pointed us in the right direction on several occasions.
- The managers of the SEAS installations who have provided the services of the NCTF members - and also the necessary machine time.
- Last but not least, the many SEAS people who have contributed with ideas and examples.

Finally I would like to thank Daud R. Matthews of the University of Petroleum and Minerals, Dhahran, Saudi Arabia, for the excellent and comprehensive documentation he has provided. This deals with the special problems encountered with IBM products

in the Arabic character set and language areas and was compiled by ARAMCO. As noted elsewhere, the NCTF has felt unable to include these problem areas in the White Paper, but we are confident that IBM is aware of the special needs of this user community and will cater for it in a future 'NLA'.

Peter Gardner 05.06.85

SEAS National Character Task Force

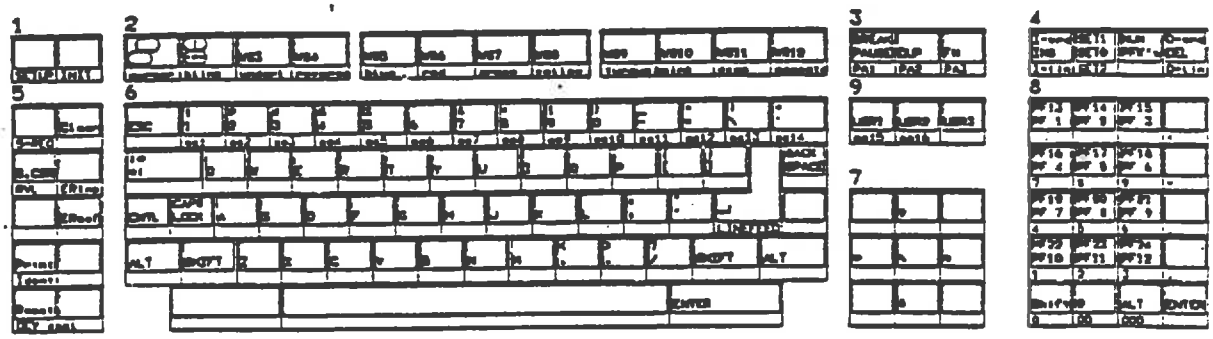
6 Appendix: Keyboard requirements

Our problem is not here to consider how to internally handle and process the entered codes, but only to give a user point of view on the new facilities that a good keyboard should offer for fair work; that is without forgetting the background of the general need to access various character sets and not only the national set.

For general purposes, we recommend that a standard keyboard offers:

- Immediate keys for a number of immediate functions such as Clear, Programmed Functions, etc. which are frequently used. These functions should be single keystroke functions.
- The same topology for 'classic keys' as in the traditional national disposition of the normal typewriter of each country. This implies that the function sub-keyboard(s) should be arranged separately from the classic keyboard. A further advantage of this would be that some of these sub-keyboards could be standard across all countries.
- Having a set of "scratch" keys that one could use for any desired internal code would be very useful. Today this is only achievable by software and results in loss of the normal usage of the affected key.
- For special usages like APL, it should be considered as a requirement that the whole range of characters be available. Also, the APL use should not modify the behaviour of the normal keyboard. An acceptable method of use must be implemented. Currently a true chinese game is required to find certain graphics when swapping normal mode to APL mode and vice-versa.

The following functions could be arranged into groups of keys with easy-to-learn locations as shown in the following diagram.



1) Local hardware functions such as SETUP and INIT (reset the "terminal" to its initial state). These functions should be performed only in conjuntion with an additional key (e.g. ALT)

2) Support of local software like workstation control on PC 3270, windowing. setting attributes and colours as on the 3279 etc. The various functions should be envoked by normal, SHIFT. ALT and SHIFT-ALT, so that a clear grouping of the functions is possible.

3) Miscellaneous like DUP and FIELD MARK and (only in conjunction with ALT) PA1..PA3

4) Mode setting, including INSERT. DELETE. INSERT should work as a flip-flop as on the PC.

   To have a more unique keyboard for different applications (PC. 327x, ...) the mode keys should activate the function of some other keys and the behaviour of the "workstation", for example:

   - normal mode
   - character set 0 (normal) or 1(special graphics) is active
   - character set 1 may be APL
   - real graphics mode (needs an application)
   - etc.

5) Terminal functions as used for 327x like ERASE EOF, ERASE INPUT etc. Dangerous functions like ERASE INPUT should be performed only in conjunction with an additional key (ALT)

6) Central typewriter functions. Keycaps of CR and LF as well as SHIFT need to be larger than those for the graphics. The small key caps of the PC let the fingers slip between the keys. Keys left or right of 1-Q-A-Z at left and BS-]-'-/ at the right are not easily reachable with normal sized hands.

   National keyboards must have the same layout including "dead" keys as the national standards for typewriters. As on typewriters only the most used special characters (e.g. in Switzerland ï is used, but is not on the keyboard) are on own keys. The other specials are produced by "dead" keys.

   An international or multilingual keyboard should allow the entering of all the special national graphics (see ISO 6937 part 2). This could be achieved by using ALT and SHIFT-ALT in conjunction with the normal keys. All the flying accents are defined preferably on the top row (1..0 - = USR1..USR3) - a total of 16 keys. This could work like this:

normal	SHIFT	ALT	SHIFT-ALT
a	A	æ	Æ
y	Y	ø	Ø
6	¬	¯	¯

   Instead of "SHIFT LOCK" the function CAPS LOCK should be performed. This function also works for the special characters of ISO 6937 part 2 like æ Æ or ø Ø.

This function must be indicated by status line or LED.

7) Cursor control. Separate keys in a meaningful arrangement are necessary. Here again the PC is the worst case. More keys than shown in the sketch may be useful (e.g. "next page"). Here again the keys may serve several functions:

normal	SHIFT	ALT	SHIFT-ALT
-> home	->> bottom	-> word	-> sentence

Of course some of these functions may be present only under certain software support (e.g. text applications on PC etc.). But the defined functions should be homogeneous between various applications!

8) Programmed function keys. The arrangement of PF keys should be geometric (3x4) to allow intuitive orientation (for CADAM there is a board of 32 PFK's in a more or less square arrangement operated by the left hand - imagine this in one row of keys!). A switch to the second set (PF13..PF24) should be performed by SHIFT.

In mode "numeric keypad" only PF1..PF3 are available. The numeric pad is (in our opinion) used only for data entry (where only few functions are necessary) or for few PC applications.

9) User-definable keys (USRx). These keys should allow a user to set up local code generation. When pressing one of these keys (with 3 keys a total of 12 meanings are possible) code as defined in a setup is sent to the host or used on the local workstation. This may be a single character or a sequence of characters. Only the total length of the sequences should be limited. And no restriction to the generated code should be made (except for conflicts with the communication).

National considerations

Our feeling is that the major emphasis, as stated above, has to be put on the need for a 'national standard area' very close to the national typewriter geography of each keyboard.

It's also necessary to have "dead" keys in order to merge accents with normal characters. A dead key, when typed on a normal typewriter, does not move to a new position, the mechanism stays in the same place, and so two (or more) characters can be merged. This is certainly a need in France and also in other countries.

International considerations

It's also mandatory to think of the international aspects of the work in a modern office. It's now often necessary to be able to enter new graphics into a letter, or other document, i.e. foreign characters, greek characters, or even mathematical symbols.

One could think of a scheme whereby ALT+a key in 6) generates characters in an alternate set - e.g. ALT-p would give a π. The meaning of this alternate set should be user selectable (greek, cyrillic, ...). A local hardware function should be available to show the keyboard layout for this alternate set. An ambitious scheme would be to

implement an actual display of the set on the key-tops of the characters (see IBM Technical Disclosure Bulletin, Vol 22, No 4).

As stated above, a set of free scratch keys could give this kind of flexibility to the keyboard without changing the customs of the national user, provided he has his usual standard keyboard.

What is SEAS?

SEAS - SHARE European Association is an independent organization of computer users and computing centres with members in Europe, the Middle East and Africa, who utilise or contemplate the use of large and medium IBM machines. SEAS works for the benefit of its membership by promoting the mutual exchange of information and experience and by liaising with IBM on matters of joint interest, in order to facilitate the effective use of computers.

The purposes of SEAS are:

To establish a forum for:

- exchange of know how,
- education in information processing techniques,
- development of improved information processing techniques,
- contributions to advances in the computer community and
- a dirct dialogue with IBM product development staff.

To represent the membership's interests by:

- influencing the development and improvement of products and services,
- communicatingn strategic concerns to IBM at the Executive Level and
- soliciting information about computer industry trends.

SEAS fulfills this mission through:

- The organization of meetings, which include an Anniversary Meeting and a Spring Meeting with programmes covering broad aspects of current and future data processing techniques.
- Project work on specific topics such as Office Automation, High Performance Processing, Interactive Computing, Operating Systems, Networking, Graphics.
- Publications such as the SEAS Newsletter, Proceedings of SEAS Meetings, Project White Papers.

(91)

106(*)JTM-2

TO: X3J3

FROM: Jeanne T. Martin

SUBJECT: Regularization of Fortran 8X, Part I - RANGE

WG5, at its recent meeting in Liverpool, voted (12-10-12) to delete RANGE and SET RANGE from the Fortran 8X language. The vote was ambivalent, but it indicates that perhaps there are some problems with the facility that should be examined by X3J3.

What's irregular about RANGE? The major irregularity is that only non-deferred shape array objects can appear in range lists. A deferred-shape array object can only be ranged independently. Fortran 8X has dynamically allocatable arrays, but these arrays cannot be ranged as a group.

My conception of users' needs is based on the way programmers at LLNL make use of LRLTRAN (an extended Fortran dialect that permits dynamic allocation). They view the data space as an open-ended configuration with the scalars and fixed-size entities packed into the top part of the space. Usually there are few of these. The bulk of the space is utilized for an unknown number of entities whose sizes may be either input-determined or calculated. These entities may be arrays, character strings, or structured objects. See Figure 1.

If Fortran 8X programmers employ the same techniques that LRLTRAN programmers do, and I see no reason why they will not, then there will be very few static arrays in 8X programs. If a range list can only contain static arrays, it will not be of much use.

I believe there is a need to range dynamic arrays since in many applications, calculations that are applied to the center of a grid do not work on the boundaries. Other applications require a moving window that selects adjacent neighbors in a grid. For these applications RANGE is extremely convenient.

What else is wrong? Many people see a drawback in that as currently described, RANGE does not permit a stride to be specified. Another drawback is that there is no indication in an executable statement that an array name does not refer to the entire range. Triplet notation certainly indicates this, but it is so bulky that it can obscure the operations that are performed.

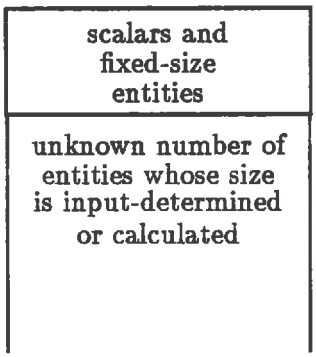


Figure 1: Design for the layout of Data Space

Perhaps a compromise is in order. If the range mechanism could be associated with a special subscript, rather than an array or list of arrays, then this range subscript could actually appear in the statement. It would indicate that it is not the whole array that is taking part in the calculation, but it would not be so bulky as to obscure the calculation.

with triplet notation:

A(ILOW:IHIGH,JLOW:JHIGH)=B(ILOW:IHIGH,JLOW:JHIGH)+C(ILOW:IHIGH.JLOW..

with RANGE:

A = B + C

witn a range subscript:

A(RANGE) = B(RANGE) + C(RANGE)

Another advantage of the range subscript is that there would not be a need for multiple bounds, shape, and size inquiry intrinsics (ELBOUND, DLBOUND, ESIZE, DSIZE, etc.). The intrinsics could all refer to the allocated, identified, or declared attributes of the objects but, when a range subscript is the argument, its currently set attributes could be returned.

A range subscript could easily contain a stride. What this would mean for the SIZE and SHAPE intrinsics would have to be defined.

A range subscript could be used with any array of the appropriate rank, so long as the set ranges were not out of bounds for the allocated, identified, or declared array bounds. Fortran has always had a bounds rule,

32

even though implementations frequently did not enforce it. With the range subscript mechanism, more than one range could be applied to the same array name appearing more than once in the same statement, if the resulting arrays were conformable. This is a feature the current facility lacks.

The range subscript mechanism could work as follows:

```
REAL, ARRAY(:,:), ALLOCATABLE :: A, B, C
ILOW = 2; JLOW = 2; IHIGH = MAX-1; JHIGH = MAX-1
DO
    SET RANGE /RANGE1 = (ILOW:IHIGH, JLOW:JHIGH)/
    A(RANGE1) = B(RANGE1) + C(RANGE1)
    ...
    ILOW = ILOW+1; JLOW = JLOW+1
    IHIGH = IHIGH-1; JHIGH = JHIGH-1
    IF (ILOW.EQ.IHIGH) EXIT
END DO
...
DO N = 1, MAX-4
    SET RANGE /RANGE2 = (N:N+4, N:N+4)/
    A(RANGE2) = B(RANGE2) + C(RANGE2)
    ...
END DO
```

At first glance this would seem to introduce a new entity - the range subscript - but the range subscript could be thought of as just a shorthand notation, a macro that expands to a subscript expression in triplet notation. The range-subscript name would have the same scope as other variable names. It would be a local name since it is "declared" in a SET RANGE statement that is executable, and thus could not be in the declaration part of a module. There really seems to be little need for making a range subscript global. The entities that determine the range (ILOW, IHIGH, etc.) can of course be global. A SET RANGE statement is required before the range subscript can be used in any case. When an array subscripted with a range subscript is passed to a procedure, the effect is the same as if the array with the same subscript expression in triplet notation were passed.

Conclusion This mechanism seems to solve the problems of the current RANGE facility in a very simple way. It eliminates the RANGE statement and several of the inquiry intrinsics. It extends to deferred-shape array

objects, permits a stride, provides an indication in the source that the whole array is not involved, has more flexibility than the current mechanism, and adds very little new syntax. Allowing the inquiry intrinsics to accept range subscript names is the only extension (a minor one) to facilities already in the language.

This is merely a suggestion. There are undoubtedly undisclosed interactions with the current language description that are overlooked. If the suggestion is favorably received, perhaps it could be more formally described and proposed during the public comment period.

106(*)LJM-1

42

S L A C M E M O R A N D U M

August 20, 1987

To: Interested FORTRAN users

From: L. Moss

Subject: Trip Report on WG5 & X3J3 Meetings, 3-14 August 1987

Note: This is a personal report of these meetings and in no sense does it constitute an official record.

PART 1: ISO/TC97/SC22/WG5 MEETING

Working Group 5, which has responsibility for revising the international Fortran standard, met in Liverpool, England, from 3 through 7 August 1987. Delegations from nine nations were present: Austria, Canada, France, Germany, Japan, Italy, the Netherlands, the United Kingdom, and the United States (the total number of individual delegates was 38).

WG5 is the ISO counterpart of X3J3 in the U.S. and has delegated the actual drafting of the revised standard to X3J3. Technical Committee 97 is the ISO counterpart of X3. Sub Committee 22 of TC97, which has no direct counterpart in the U.S., has overall responsibility for programming languages.

Resolutions passed at the Liverpool meeting will be referred to as LRn, i.e., "Liverpool Resolution n". Similarly, resolutions passed at the previous meeting in Halifax, Nova Scotia will be referred to as HRn. Where appropriate, voting results will be indicated as (yes-no-abstain/yes-no-abstain), where the first group refers to the vote by individuals and the second group to the vote by national delegation.

STATUS OF HALIFAX RESOLUTIONS

Following various items of opening business and the activity reports of the individual national bodies, WG5 considered the status of the resolutions passed at the previous meeting in Halifax. Following that meeting, X3J3 adopted a new procedure of formally voting on a response to each WG5 resolution. For the most part, WG5 felt that the new procedure was a significant improvement. However, there was a request (LR6) that in future X3J3 include an explanation of any actions it takes which are contrary to the sense of a resolution (this had in fact been done for some of the Halifax resolutions but not for others). Some additional refinements to this procedure were also discussed and became the subject of LR5 and LR6.

STATUS OF THE DRAFT

Since the meeting last summer in Halifax, the draft (S8/104) has been forwarded to X3, which, at the time of the Liverpool meeting, was balloting on whether to release it for public review in the United States. HR1 (31-1-0/7-0-0), had urged, in part, that the draft be released for public review as soon as possible. Subsequently, an informal letter ballot of WG5 was conducted in parallel with the second X3J3 letter ballot on forwarding the draft. The WG5 results were (25-2). Based on these results, and in hopes of keeping the ISO and ANSI review periods synchronized, the convener forwarded the draft to SC22 between WG5 meetings (this was in accord with ISO procedures). LR1 (28-7-0/7-2-0) confirmed this action by the convener. It was hoped that SC22 would act on the draft at its September meeting in Washington, D.C.

LIVERPOOL RESOLUTIONS

A number of issues raised in various WG5 letter ballots were discussed, as were those Halifax resolutions which X3J3 had not completely accepted. The result was a number of new resolutions, the most significant of which are summarized below (a complete list of the Liverpool resolutions, along with the voting results, may be obtained from me on request). However, WG5 made it clear (LR3) that these resolutions were not intended to delay the public review process.

Pointers and IDENTIFY
LR8 (22-3-10/8-0-1) reaffirmed WG5's request that some form of pointer facility be included in the final draft. A paper in the distribution for the immediately following X3J3 meeting suggested extending the IDENTIFY facility in order to provide pointers. LR9 (13-9-13/3-2-4) recommended pursuing this approach to pointers, or else deleting the IDENTIFY facility.

Significant Blanks
LR12 (26-8-1/9-0-0) reaffirmed WG5's request that blanks be made significant in the new source form.

Multi-Byte Character Sets
WG5 spent considerable time reviewing the request from the Japanese delegation for a facility to manipulate data from several, possibly large, character sets. LR19 (24-1-10/7-0-2) recommended that X3J3 cooperate with the Japanese in adding such a facility. Since this is really a cross-programming-language problem, LR20 (34-0-0/9-0-0) further requested that its convener bring it to the attention of SC22.

National Use Characters
LR21 (31-0-4/9-0-0) expressed concern over the use of square brackets in the standard. A possible solution that was discussed was to make support for square brackets an implementation option, just as is done for lower case characters.

BIT Data Type
LR22 (22-3-10/7-0-2) requested that BIT data type be restored to the language.

Deprecated Features
In LR10 (30-0-5/7-0-2), WG5 withdrew its request (HR13) to return to a single list of deprecated features. WG5 noted that, though the concepts of both obsolescence and deprecation are mentioned in the main text, the deprecated features are only indicated in an appendix. LR11 (28-4-3/8-0-1) requested that a list of deprecated features be added to the main text.

Passed-On Precision
The German member body (DIN) pointed out some possible problems with passed-on precision (i.e., "REAL(*,*)"). LR23 (25-2-8/8-0-1) requested X3J3 to investigate these potential problems.

Name-Directed I/O
In LR13 (23-6-6/5-0-4), WG5 withdrew its request (HR22) that NAMELIST I/O be replaced by name-directed I/O.

RANGE and SET RANGE
LR24 (12-10-12/4-3-2) requested that the RANGE facility be deleted from the language.

AVAILABILITY OF THE DRAFT

A number of delegates were quite concerned that it would prove difficult for reviewers outside the U.S. to obtain copies of the draft in a timely and economical fashion. It was hoped that arrangements could be made between CBEMA and the various national bodies to allow the latter to distribute copies of the draft within their own countries.

CLOSING BUSINESS

PHIGS Fortran Binding
A draft PHIGS Fortran binding was available for comments (WG5 document N234). Comments were to be sent to the convener, Jeanne Martin, by 21 August 1987.

Future Meetings

The 1988 meeting will be held in the AFNOR building in Paris, France. The dates will probably be 19-23 September 1988, though there is a possible clash with an SC22 meeting. It is strongly recommended that hotels be booked at least 1-2 months in advance. Typical hotel prices range from $35-$70 or more. Note that the Summer, 1988 meeting of X3J3 will NOT be held in conjunction with this WG5 meeting.

The 1989 WG5 meeting will probably be held in Italy, with the X3J3 meeting the following week in Vienna. Dates have not yet been determined.

PART 2: X3J3 MEETING

X3J3 met in Liverpool, England, from 10 through 14 August 1987. Approximately 28 members or alternates attended, along with about 13 observers, primarily delegates from the WG5 meeting which was held during the preceding week.

This meeting was primarily devoted to planning how to handle the public review comments, beginning an audit of Fortran 77 against the draft document, and tutorials and subgroup work on a few major issues which it is anticipated will be raised in public comments. No substantive, and only a few editorial, proposals were passed.

All working documents are assigned numbers of the form, "mmm(wg)aaa-n", where:

mmm is the meeting number (the August 1987 meeting was number 105).

wg is the number of the working group assigned responsibility for the proposal, or "*" for individual proposals.

aaa are the initials of the author.

n is a small number to distinguish different documents from a single author at one meeting.

The results of straw votes (SV) are, unless otherwise noted, given as: (yes-no-undecided), with an asterisk next to my vote; formal votes (FV) are always strictly (yes-no).

[I missed the Tuesday and Wednesday sessions of this meeting due to illness, and so the following report is incomplete.]

OPENING BUSINESS

Status of S8

The Standards Planning And Requirements Committee (SPARC) has reviewed X3J3's work on the draft (S8/104), and has certified it as in compliance with the authorizing document (SD-3). The vote was not, however, unanimous, and the minority opinion has been made available to X3.

At the time of the meeting, X3 was conducting a 30-day letter ballot on whether to release the draft for public review. The ballot period was to end on 20 August (or perhaps 22 August), but was expected to be followed by a 10-day reconsideration ballot since there was likely to be at least one "NO" vote. On the other hand, as far as anyone could remember, X3 has never rejected a request to release a draft for public review (which only requires a simple majority in X3). The best guess at the time was that the four month public review might begin around 1 October 1987.

Reorganization of X3J3
The committee will be reorganized as of the next meeting. Under the new organization, the steering committee will be primarily concerned with administrative matters. A group consisting of the chair and vice-chair and the subgroup heads and assistant heads will take on the planning responsibility formerly held by the steering committee. This group will normally meet on Monday and Thursday afternoons during an X3J3 meeting.

There will be three standing working groups:

- WG30 Fortran 77 Issues and Interpretations
- WG31 Editorial
- WG32 Public Review Processing

The Public Review Processing subgroup will coordinate the committee's responses to comments received during the public review period(s).

The old subgroup on Expression Concepts (SG18) will be dissolved, and responsibility for its sections of S8 reassigned among the remaining technical subgroups. The new technical subgroups are:

- SG20 General Concepts (Sections 1, 2, 3, 7, and 14, and appendices A, B, C, and F).
- SG21 Data Concepts (Sections 4, 5, and 6).
- SG22 Control Structures and I/O (Sections 8, 9, and 10).
- SG23 Procedures and Program Units (Sections 11, 12, and 13).

Appendices D, E, G, and H of S8 will be the responsibility of the Editorial working group.

International Representative
Andy Johnson has been appointed International Representative by the Secretariat Management Committee (SMC).

Availability of the Draft
If and when the draft is released for public review by X3, copies may be ordered by phoning Global Engineering. The numbers are:

East Coast	--	(800) 248-0084
West Coast	--	(800) 854-7179
International	--	(714) 540-9870

The cost was not known at the time of the meeting, but was expected to be fairly high -- perhaps $100 or more (this price would include not only S8, but also the comments from X3J3 letter ballots and the committee responses). There was considerable discussion of the negative effect such a price could have on the standardization process.

Membership
At the beginning of this meeting there were 38 members of X3J3, with three on provisional status. The quorum was 12 members.

MULTI-BYTE CHARACTER SETS

There was extended discussion of how to handle large character sets, such as Kanji, in Fortran 8x. Carl Burch presented an outline of a standard module using a derived type to handle such character sets. This approach had some problems with handling I/O, however.

A delegation from Japan then explained in some detail both the current Japanese standard ("NCHARACTER") and their proposal ("CHARACTER (KIND=n)") for Fortran 8x.

The NCHARACTER approach simply adds another intrinsic data type, namely NCHARACTER, which is defined in a way analogous to CHARACTER. Expressions may not mix the two types (e.g., one cannot concatenate CHARACTER and NCHARACTER strings within the program itself), but input and output of both types of entities may be done in a very natural and consistent fashion.

he "KIND=n" proposal is really just a generalization of the NCHARACTER approach. Instead of just two varieties of CHARACTER data, there would now be an open-ended set, with KIND=1 being the usual, default CHARACTER type. Other values of KIND would not necessarily correspond to the number of bytes per character -- rather, they would simply identify alternative character or, more generally, symbol sets. In the Japanese proposal, the connection between a particular value of KIND and a specific character set would be made external to the Fortran program -- perhaps via a compiler switch, or possibly just pre-defined for some implementations. Another possibility, of course, would be to have the KIND parameter directly identify a specific character set from some international registry.

The Japanese visitors made the following additional points:

- Although to be of practical use in Japan a compiler would have to be able to deal with Kanji characters in a source program (if for no other reason than to handle constants of the Kanji data type), they did not think it was appropriate, at least at this time, to standardize such source code support.

- They saw no need to concatenate strings of different kinds of characters within the program, but only in input and output.

- Any approach in which the programmer must keep track of the escape sequences needed to shift in and out of Kanji would probably be unacceptable (this was another problem with the MODULE approach).

- In their proposal, implementers would not be required to support CHARACTER KINDs other than 1.

Several vendors pointed out that NCHARACTER is rapidly becoming a de facto international standard, and questioned the need for replacing it with such an open-ended approach.

Several straw votes were taken on this subject. Should X3J3 do something about large character sets in Fortran 8x? SV (17*-11-8). This same question was repeated, but with voting members only responding: SV (10*-5-5).

Is the problem best solved with the MODULE approach? SV (1-27*-5).

Is the problem best solved with the KIND=n approach? SV (20-4-10*).

Is the problem best solved with the NCHARACTER approach? SV (4-17-12*).

The I/O subgroup was asked to assist the Japanese in developing a detailed proposal, including text, for the next meeting.

WG5 RESOLUTIONS

Significant Blanks
There was a short discussion of some of the arguments in favor of significant blanks in the new source form which were not touched upon in X3J3's response to the WG5 Halifax resolution on this subject, such as improved reliability and ease of construction of software tools. However, it was also pointed out that public comments will probably have a major effect on the final resolution of this issue. A straw vote was taken on deferring a full debate on this subject at least until the next meeting: SV (16*-7-4).

Pointers
The data concepts subgroup was charged with developing a proposal for a pointer facility. At the time of the meeting, they planned to pursue the approach outlined in Carl Burch's paper in the pre-meeting distribution (105(*)CDB-3) -- namely to adapt the IDENTIFY facility to provide pointer capabilities.

Multi-Byte Character Sets
As mentioned above, the Japanese agreed to bring a detailed proposal to the next meeting.

RANGE and SET RANGE
Alex Marusak agreed to redistribute some of the examples shown during past debates on this subject.

Interfaces
The procedures and program units subgroup was asked to clarify the explanation of interfaces and interface blocks.

BIT Data Type
A suggestion was made that the functionality contained in the BIT proposal in Appendix F might be more acceptable if it was provided instead by a different "flavor" of LOGICAL, i.e., one that is not associated with numeric storage units. An outline of such a proposal was presented as a tutorial, and contained the following features:

- An optional "KIND=n" parameter would be added to the LOGICAL type specification statement. The default KIND would be processor dependent. KIND=1 would imply a more compact representation, though there would be no explicit binding to physical bits. Other values of KIND could be defined by a processor as well.

- Alternate forms of LOGICAL constants would be defined: B'1' and B'0'. Either could be used in source code.

- Some new edit descriptor (B?) could be used to select "1" and "0" instead of "T" and "F" for input and output. For list-directed I/O, "LOGICAL (KIND=1)" would probably default to "1" and "0", while "LOGICAL" would default to "T" and "F".

- The usual LOGICAL operators, .AND., .OR., .EQV., .NEQV., and .NOT. would be overloaded to apply to any combination of different kinds of LOGICAL. The result would presumably be promoted to the less compact representation, but an intrinsic coercion routine could force the result to another KIND if necessary.

- A small number of additional intrinsic functions would be added.

- Passed-on KIND would not be permitted (i.e., dummy arguments could not be declared as "LOGICAL (KIND=*)").

One advantage claimed for this approach was that it left the keyword BIT available in case the committee someday wants to add a true BIT string facility. Another was the removal of the redundant .BAND., .BOR., .BXOR., and .BNOT. operators (though these were not really essential in the BIT proposal in Appendix F). There was some discussion about whether to try to explicitly bind "LOGICAL (KIND=1)" to physical bits. Most people seemed to think implementers could be trusted to do what was intended if a non-storage associated flavor of LOGICAL was added to the language. However, there was a little uneasiness with the similarity of "LOGICAL (KIND=1)" and the fairly common extension, "LOGICAL*1", which usually means one bit per byte. A suggestion was made instead to add a keyword rather than a parameter, e.g., PACKED LOGICAL or SHORT LOGICAL.

A straw vote was taken on whether a detailed proposal along these lines should be brought to the next meeting: SV (6-7-14*). A request was made for another straw vote, asking for a similar proposal, but with "LOGICAL (KIND=1)" explicitly bound to physical bits: SV (6-14-8*).

OTHER ISSUES

Distribution of Public Comments
The Public Review Processing subgroup, anticipating that the volume of public comment could be quite large, asked whether every member needed a complete set of all public comments. After some discussion, a straw vote was taken: SV (24*-0-7). A suggestion was made that it might be acceptable to distribute the public comment via microfiche. Most people present already had access to microfiche readers at home, and inexpensive hand-held readers are also available. The consensus was that the subgroup could consider microfiche as an option.

Distribution of Minutes
Jeanne Martin announced that Livermore could no longer distribute the minutes after this meeting, so a new system would be necessary. It was proposed that the larger organizations represented on X3J3 should take turns doing this chore. Jeanne Adams agreed to prepare a list of candidates to be included in the rotation, who would then be asked to get commitments from their management. Cray, which already handles the pre-meeting distributions, and Livermore, which still handles the WG5 distributions, were excused. A straw vote was requested on distributing the minutes on microfiche: SV (5-9-6*).

105(*)MBM-1 Edits to S8
This paper contained a short list of additional non-substantive edits to S8 which were recommended by the editorial committee. There were some questions on points 8 and 11, so they were separated out and a vote was taken on the remaining points: FV (15*-0) -- PASSED. After some discussion, point 8 was withdrawn. Point 11 suggested putting the complete text of two examples which involved some obsolescent features into the obsolescent font: FV (4*-12) -- FAILED.

105(*)JHM-01 Derived-Type I/O
At the Monterey meeting there were some discussions of improving the facilities for handling I/O of derived-type objects. At the time, a straw vote suggested postponing any action until a later time. This paper reviews some of the previously discussed approaches for derived-type I/O. Although several relatively minor extensions were discussed that could help with some derived types, any completely general solution would probably be very difficult. A straw vote was requested on whether to pursue improvements to derived-type I/O any further at this time: SV (2-11*-7).

105(*)JLW-4 MODULE for Variable-Length Strings
This was a first cut at providing variable-length string capability via a standard MODULE. It appeared to be reasonably complete, given the limitations of the derived-type approach. The main disadvantages were:

- A compact substring notation would not be available, so a function notation would have to be used.
- This in turn would make it impossible to assign to a substring (although this may actually be an advantage due to some of the possible ambiguities that might result if it were possible).
- I/O would be awkward.

105(*)RCA/CDB-1 Exponent Letter
This paper suggested two possible alternative ways of declaring an exponent letter to be used in writing constants with a particular combination of precision and range. Both of these suggestions involved deleting the EXPONENT LETTER statement, and instead indicating the letter in some fashion on the REAL type specification statement, either alone or together with declarations of variables.

1. The letter could be indicated in the entity-decl-list in the form of a single-character, alphabetic literal. For example:

```
REAL (10,50) :: ARRAY(30),X,'L',Y
   ...
X = 3.141592654L0
```

2. The letter could be indicated as part of the precision-selector, e.g.,

```
REAL (10,50,EXPONENT_LETTER='L') :: ARRAY(30),X,Y
   ...
X = 3.141592654L0
```

Both of these would certainly work, but they both would be somewhat irregular. The first would insert what appears to be a CHARACTER literal in a list of REAL variables. The second would place one of the things being declared, namely the exponent letter, on the left of the "::" instead of the right. A third possibility was suggested from the floor which would probably answer both these objections:

3. Since the entity-decl-list is a list of REAL objects being declared with the specified precision, the set of all constants of that precision could be included in the list via a "wild card" notation. Using the asterisk as the wild card character this would look as follows:

```
REAL (10,50) :: ARRAY(30),X,*L*,Y
   ...
X = 3.141592654L0
```

106(*) LJM-1

Alternatively, the question mark might be used as the wild card character.

Finally, a fourth possibility was suggested in a paper placed on the table (item #62, by Leo ter Haar).

4. In this approach, the EXPONENT LETTER statement, or something like it, would be retained, but the PRECISION and EXPONENT_RANGE parameters of the precision-selector in the REAL type specification statement would be replaced by a single parameter that would refer to an exponent letter previously declared. The subgroup looked at this idea, and suggested using the term "APPROXIMATION" in place of "EXPONENT LETTER". The above example might then look as follows:

```
APPROXIMATION (10,50)   :: L
REAL (APPROXIMATION=L)  :: ARRAY(30)
REAL (L)                :: X,Y
   ...
X = 3.14159265L0
```

An advantage of this approach would be that the precision and range would only be explicitly specified once, thus reducing errors and making it easier to change the approximation. It might also simplify the discussion of passed-on precision and range. The big disadvantage, however, would be a major perturbation to the text.

A straw vote was taken on whether the current syntax was a problem: SV (6*-8-4).

CLOSING BUSINESS

The registration fee for the next meeting (Fort Lauderdale, FL) will be $70.

Future Meetings

106th: 9-13 November 1987, Ft. Lauderdale, FL (host: David Phillimore, Gould Inc.).

107th: 8-12 February 1988, Monteleone Hotel, New Orleans, LA (host: Jerry Wagener, Amoco Production Research).

108th: 9-13 May 1988, University of Illinois, IL (host: Kurt Hirchert).

109th: 8-12 August 1988, probably in Estes Park, Colorado.

Membership
At the end of the meeting, one of the members on provisional status (Ted Crowley) was dropped as a member, leaving a total membership of 37. The other two provisional members (Murray Freeman and Werner Shenk) had previously obtained excused absences from the chair, and so will again be on provisional status at the next meeting. They will not be able to vote at that meeting.

Next Distribution
The closing date for the next pre-meeting distribution is 5 October 1987. To get an item into the distribution it should be sent to:

Richard A. Hendrickson
Cray Research Incorporated
1345 Northland Drive
Mendota Heights MN 55120
(612-681-5804)

SLAC MEMORANDUM

(73)

106(16)LJM-2

24 September 1987

TO: X3J3

FROM: Len Moss

SUBJECT: Rewrite of DO construct description

I was asked by the editorial committee at the Liverpool meeting to try to clarify the section on the DO construct. The idea was to give the primary description in terms of the properly block-structured (and core-conforming) forms of the DO, then cover the obsolescent versions afterward. During discussion in full committee, it was also suggested that I revise the examples of DO constructs.

The first proposal below contains my attempt to separate the description of the DO construct into two pieces, as suggested by the editorial committee. It also contains a number of other suggested improvements for this portion of Section 8, such as moving the BNF for the CYCLE and EXIT statements later in the section. I have not attempted to make these other changes into separate proposals because many of them are dependent on one another. Instead, I have added notes within the text of the proposal explaining my reasons for each group of changes.

* * *

PROPOSAL 1

On page 2-3, delete the constraint in line 41.

> **COMMENT:** This constraint should be covered in Section 8, since there is an additional requirement if the EXIT or CYCLE statement refers to a DO construct name. **END COMMENT.**

On page 8-1, replace line 10 with "There is also a non-block-structured form of the DO construct."

On page 8-1, in line 15, change "embedded." to "embedded; however, in some forms of the DO construct a sequence of executable constructs without a terminating boundary statement obeys all the other rules governing blocks (8.1.1)."

> **COMMENT:** I want also to consider the *do-body* to be a block when the loop ends with a non-shared CONTINUE; and *do-termination* is not quite the right term since I use *end-do* for block-structured DOs. In any case it seems worthwhile to avoid forward references to the these BNF terms so early in the section. **END COMMENT.**

On page 8-5, replace lines 21-22 with the following:

8.1.4 Iteration Control. Iteration control is provided by the DO construct and the EXIT and CYCLE statements. The DO construct specifies the repeated execution of a sequence of executable constructs. Such a repeated sequence is called a **loop**. The EXIT and CYCLE statements may be used to modify the execution of a loop.

The number of iterations of a loop may be determined at the beginning of execution of the DO construct, or may be left indefinite ("DO forever"). In either case, an EXIT statement (8.1.4.4.4) may be executed anywhere within the DO construct to immediately terminate the loop. A particular iteration of the loop may be abbreviated by executing a CYCLE statement (8.1.4.4.3).

Replace Section 8.1.4.1 with the following:

8.1.4.1 Forms of the DO construct. The DO construct can be written in either a block-structured form or a non-block-structured form.

R816 *do-construct* **is** *block-do-construct*
or *non-block-do-construct*

8.1.4.1.1 Form of the Block-Structured DO construct.

R817 *block-do-construct* **is** *do-stmt*
do-block
end-do

R818 *do-stmt* **is** *label-do-stmt*
or *nonlabel-do-stmt*

R819 *label-do-stmt* **is** [*do-construct-name*:] DO *label* [*loop-control*]

R820 *nonlabel-do-stmt* **is** [*do-construct-name*:] DO [*loop-control*]

R821 *loop-control* **is** [,]*do-variable* = *scalar-numeric-expr*, ■
■ *scalar-numeric-expr* [, *scalar-numeric-expr*]
or (*scalar-int-expr*) TIMES

R822 *do-variable* **is** *scalar-variable*

Constraint: The *do-variable* must be a scalar integer, default real, or default double precision real named variable.

Constraint: Each *scalar-numeric-expression* in *loop-control* must be of type integer, default real, or default double precision real.

R823 *do-block* is *block*

R824 *end-do* is *end-do-stmt* [*do-construct-name*]
or *continue-stmt*

Constraint: If the *do-stmt* of a *block-do-construct* is identified by a *do-construct-name* the corresponding *end-do* must be an *end-do-stmt* referring to the same *do-construct-name*; and if the *do-stmt* of a *block-do-construct* is not so identified, the corresponding *end-do* must not refer to a *do-construct-name.*

Constraint: If the *do-stmt* is a *nonlabel-do-stmt*, the corresponding *end-do* must be an *end-do-stmt.*

Constraint: If the *do-stmt* is a *label-do-stmt*, the corresponding *end-do* must be identified with the same *label.*

8.1.4.1.2 Form of the Non-Block-Structured DO construct.

R825 *non-block-do-construct* is *label-do-stmt*
do-body
do-termination

R826 *do-body* is [*execution-part-construct*]...

R827 *do-termination* is *do-term-stmt*
or *non-block-do-construct*

R828 *do-term-stmt* is *action-stmt*

Constraint: A *do-term-stmt* must not be a *goto-stmt, return-stmt, stop-stmt, exit-stmt, cycle-stmt, arithmetic-if-stmt*, nor *assigned-goto-stmt.*

Constraint: If the *do-term-stmt* is a *continue-stmt,* the corresponding *do-construct* must be the *do-termination* of another *non-block-do-construct.*

Constraint: If the *do-termination* is a *do-term-stmt,* it must be identified with the label referred to in the corresponding *do-stmt*; and if it is another *non-block-do-construct,* both of the corresponding *do-stmt*s must refer to the same label.

Within a scoping unit, all DO constructs whose DO statements refer to the same label are non-block-structured, and are said to **share** the statement identified with that label. The statement so identified must be a CONTINUE statement or a *do-term-stmt* that serves as the *do-termination* of the innermost of these DO constructs.

COMMENT: I believe the above BNF and constraints are equivalent to the version in S8.104, although this needs to be carefully reviewed by a number of people. Not counting the rules and constraints involving the CYCLE and EXIT statements, which have been moved to Sections 8.1.4.4.3 and 8.1.4.4.4, it results in a net total of 5 additional rules and 1 less constraint; however, the block-structured portion has only 1 additional rule and 4 fewer constraints. Moreover, all the core features are defined first and most of the complexity is concentrated in the obsolescent section. **END COMMENT.**

Replace Section 8.1.4.2 with the following:

8.1.4.2 Range of the DO construct. The **range** of a block-structured DO construct is the *do-block*, and satisfies the usual rules for blocks (8.1.1). In particular, transfer of control to the interior of such a block from outside the block is prohibited. It is permissible to branch to the END DO statement or CONTINUE statement terminating a block-structured DO construct only from within the range of that DO construct.

The **range** of a non-block-structured DO construct consists of the *do-body* and the *do-termination*. The end of such a range is not bounded by a particular statement as for the other executable constructs (e.g., END IF); nevertheless, the range satisfies the rules for blocks (8.1.1). Transfer of control into the *do-body* or to the *do-termination* from outside the range is prohibited; in particular, it is permissible to branch to the *do-term-stmt* of a non-block-structured DO construct only from within the range of that construct.

COMMENT: The above two paragraphs take advantage of the different BNF terms, *do-block*, *do-body*, *and do-termination*, to precisely define the non-BNF term **range** for each form of the DO construct in two parallel paragraphs.

The definition of the term **share** for DOs has been moved out of the section on range and up to the above section on the form of the DO. **END COMMENT.**

On page 8-6, line 36, change "loop. A **loop** is " to "loop, i.e.,".

COMMENT: I have moved the definition of **loop** to the beginning of Section 8.1.4. **END COMMENT.**

On page 8-6, line 38, change "loop body" to "loop range".

On page 8-7, line 22 change "DO construct" to "loop".

COMMENT: After going to the trouble to define both range and loop, I thought we should try to make more use of them. **END COMMENT.**

On page 8-7, line 17, change "*do-construct*" to "loop terminates and the DO construct".

On the same page, in line 18, change "*do-constructs*" to "DO constructs"; and in line 19, change "the construct" to "all of these constructs".

On the same page, in line 23, change "*do-variable*" to "DO variable".

COMMENT: In different places, we talk about DO constructs becoming inactive and about loops terminating, so I think it's important to use both phrases in describing what happens when the iteration count is zero. I also tried to be more consistent about avoiding BNF terms in text if good alternatives are available. **END COMMENT.**

On page 8-7, lines 27-31, delete the last two sentences of Section 8.1.4.4.2.

COMMENT: I have moved the gist of these two sentences into the next section. **END COMMENT.**

Replace Section 8.1.4.4.3 with the following:

8.1.4.4.3 Cycle Abbreviation. Step (2) in the above execution cycle may be abbreviated by executing a CYCLE statement from within the range of the loop.

R829 *cycle-stmt* **is** CYCLE [*do-construct-name*]

Constraint: If a *cycle-stmt* refers to a *do-construct-name*, it must be within the range of that *do-construct*; otherwise, it must be within the range of at least one *do-construct*.

A CYCLE statement is said to **belong** to a particular DO construct. If the CYCLE statement refers to a DO construct name, it belongs to that DO construct; otherwise, it belongs to the innermost DO construct in which it appears.

Execution of a CYCLE statement causes immediate execution of step (3) of the current execution cycle of the DO construct to which it belongs. If this construct is not block-structured, the *do-termination* is not executed.

In a block-structured DO construct, a transfer of control to the *end-do* has the same effect as execution of a CYCLE statement belonging to that construct. In a non-block-structured DO construct, transfer of control to the *do-termination* causes the *do-termination* itself to be executed. Unless a further transfer of control results, step (3) of the current execution cycle of the DO construct is then executed.

COMMENT: I like the word "abbreviation" a whole lot better than "interruption". The latter seems to connote slowing down or even exiting the loop, whereas the former suggests shortening by leaving out a portion. Another possible word might be "truncation".

The syntax for the CYCLE and EXIT statements never seemed to me to fit properly with that for the DO construct itself, so I have moved them to the sections where their semantics are discussed. These are two separate sections, so there will be a little redundancy; on the other hand, the long block of syntax for the DO construct itself is shortened a bit which helps some.

I think pointing out explicitly that a non-block-structured DO's *do-termination* is skipped when a CYCLE statement is executed helps to clarify both what the behavior actually is, and why such forms of the DO have been made obsolescent. **END COMMENT.**

Replace Section 8.1.4.4.4 with the following:

8.1.4.4.4 Loop Termination. The EXIT statement provides one way of terminating a loop.

R830 *exit-stmt* **is** EXIT [*do-construct-name*]

Constraint: If an *exit-stmt* refers to a *do-construct-name,* it must be within the range of that *do-construct*; otherwise, it must be within the range of at least one *do-construct.*

An EXIT statement is said to **belong** to a particular DO construct. If the EXIT statement refers to a DO construct name, it belongs to that DO construct; otherwise, it belongs to the innermost DO construct in which it appears.

The loop terminates, and the DO construct becomes inactive, when any of the following occurs:

(1) The iteration count is determined to be zero when tested during step (1) of the above execution cycle.

(2) Execution of an EXIT statement belonging to the DO construct.

(3) Execution of an EXIT statement or CYCLE statement that is within the range of the DO construct, but that belongs to an outer DO construct.

(4) Transfer of control to a statement that is neither the *end-do* nor within the range of the DO construct.

(5) Execution of a RETURN statement within the range of the DO construct.

(6) Execution of a STOP statement anywhere in the program; or termination of the program for any other reason.

COMMENT: As explained above, I have separated the syntax for EXIT and CYCLE from that for the DO construct itself. The rest of the changes in this section are relatively minor rephrasings, except for the change to item (1) in the list. The previous version of this item could possibly, I think, be interpreted to mean that the loop terminates as soon as the iteration count is decremented to zero in step (3) of the execution cycle, before the do variable is incremented. **END COMMENT.**

END PROPOSAL 1

COMMENT: One change I considered and still have mixed feelings about would be to substitute "associated with" for "belong to" to describe the relationship between a CYCLE or EXIT statement and a DO construct. "Belong" has never sounded quite right to me, especially not as a technical term with a formal (i.e., boldface) definition. The ordinary English word "associated" (e.g., "...the CYCLE statement...is associated with that DO construct...") seems to me to be a much better choice. On the other hand, Jeanne Martin pointed out that the word "associated" with a technical meaning is already heavily used in S8, and adding yet another meaning to it, albeit an informal meaning, might be more confusing than helpful. I'd be happy to hear suggestions for alternative words or phrases to use in this context. **END COMMENT.**

* * *

The following proposal is a complete rewrite of the DO construct examples section. I have kept the important examples copied from Fortran 77 but have modified some of the existing Fortran 8x examples, and have added quite a few new ones. It should be noted that I have included a few examples of **INVALID** syntax which I think help to clarify some of the BNF rules and constraints. I believe this proposal is independent of the previous one.

PROPOSAL 2

Replace Section 8.1.4.5 with the following:

8.1.4.5 Examples of DO constructs. The following are all valid examples of block-structured DO constructs:

Example 1:

```
READ (IUN, '(1X,G14.7)', IOSTAT=IOS) X
DO; IF (IOS .NE. 0) EXIT    ! A "DO WHILE" loop
   IF (X .GE. 0.)  THEN
      CALL SUBA (X)
      CALL SUBB (X)
         ...
      CALL SUBZ (X)
   ENDIF
   READ (IUN, '(1X,G14.7)', IOSTAT=IOS) X
END DO
```

The above program fragment contains a DO construct that does not have an iteration count. The loop will continue to execute until an end-of-file or I/O error is encountered, at which point the EXIT statement terminates the loop. When a negative value of X is read, the program skips immediately to the next READ, bypassing most of the range of the loop.

Example 2:

```
      DO                              ! A "DO WHILE + 1/2" loop
         READ (IUN, '(1X,G14.7)', IOSTAT=IOS) X
         IF (IOS .NE. 0) EXIT
         IF (X .LT. 0.)  CYCLE
         CALL SUBA (X)
         CALL SUBB (X)
            ...
         CALL SUBZ (X)
      END DO
```

This program fragment behaves exactly the same as the previous one. However, the EXIT statement has been moved to the interior of the range, so that only one copy of the READ statement is required. A CYCLE statement has also been used to avoid an extra level of IF nesting.

Example 3:

```
      SUM = 0.
      READ (IUN) N
      OUTER: DO (N) TIMES          ! A DO with a construct name
         READ (IUN) IQUAL, M, ARRAY(1:M)
         IF (IQUAL .LT. IQUAL_MIN) CYCLE OUTER   ! Skip inner loop
         INNER: DO 30 I = 1, M    ! A DO with a label and a name
            CALL CALCULATE(ARRAY(I),RESULT)
            IF (RESULT .LT. 0) CYCLE
            SUM = SUM + RESULT
            IF (SUM .GT. SUM_MAX) EXIT OUTER
30          END DO INNER
         END DO OUTER
```

The outer loop has no DO variable; however, it does have an iteration count of MAX(N,0), and will execute that number of times unless SUM exceeds SUM_MAX, in which case the EXIT OUTER statement terminates both loops. The inner loop is skipped by the first CYCLE statement if the quality flag, IQUAL is too low. If CALCULATE returns a negative RESULT, the second CYCLE statement prevents it from being summed. Note that both loops have construct names, and that the inner loop also has a label. A construct name is required on the EXIT statement in order to terminate both loops, but is optional on the CYCLE statements since each belongs to its innermost loop.

Example 4:

```
      N = 0
      DO 40, I = 1, 10
         J = I
         DO  K = 1, 5
            L = K
            N = N + 1        ! This statement executes 50 times
         END DO              ! Non-labelled DO inside a labelled DO
   40 CONTINUE
```

After execution of the above program fragment, I = 11, J = 10, K = 6, L = 5, and N = 50.

Example 5:

```
      N = 0
      DO I = 1, 10
         J = I
         DO 50, K = 5, 1    ! This inner loop is never executed
            L = K
            N = N + 1
   50    CONTINUE           ! Labelled DO inside a non-labelled DO
      END DO
```

After execution of the above program fragment, I = 11, J = 10, K = 5, N = 0, and L is not defined.

* * *

The following are all valid examples of non-block-structured DO constructs:

Example 6:

```
      DO 60
         READ (IUN, '(1X,G14.7)', IOSTAT=IOS) X
         IF (IOS .NE. 0) EXIT
         IF (X .LT. 0.)  GOTO 60
         CALL SUBA (X)
         CALL SUBB (X)
            ...
         CALL SUBY (X)
         CYCLE
   60    CALL SUBNEG(X)     ! SUBNEG only called when X < 0.
```

This DO construct is not block-structured since it ends with a statement other than an ENDDO or CONTINUE. The loop will continue to execute until an end-of-file or I/O error occurs. Note that the CYCLE statement does not result in SUBNEG being called.

Example 7:

```
      SUM = 0.
      READ (IUN) N
      DO 70, (N) TIMES
         READ (IUN) IQUAL, M, ARRAY(1:M)
         IF (IQUAL .LT. IQUAL_MIN) M = 0    ! Skip inner loop
         DO 70 I = 1, M
            CALL CALCULATE(ARRAY(I),RESULT)
            IF (RESULT .LT. 0) CYCLE
            SUM = SUM + RESULT
            IF (SUM .GT. SUM_MAX) GOTO 71
   70       CONTINUE        ! This CONTINUE shared by both loops
   71    CONTINUE
```

This example is similar to Example 3 above, except that the two loops are not block-structured since they share the CONTINUE statement with the label 70. The *do-termination* of the outer DO construct is the entire inner DO construct. The inner loop is skipped by forcing M to zero. If SUM grows too large, both loops are terminated by branching to the CONTINUE labelled 71. The CYCLE statement in the inner loop may still be used to skip negative values of RESULT.

Example 8:

```
      N = 0
      DO 100 I = 1, 10
         J = I
         DO 100 K = 1, 5
            L = K
  100       N = N + 1       ! This statement executes 50 times
```

In this example, the two loops share an assignment statement. After execution of this program fragment, I = 11, J = 10, K = 6, L = 5, and N = 50.

Example 9:

```
      N = 0
      DO 200 I = 1, 10
         J = I
         DO 200 K = 5, 1       ! This inner loop is never executed
            L = K
  200       N = N + 1
```

This example is very similar to the previous one, except that the inner loop is never executed. After execution of this program fragment, I = 11, J = 10, K = 5, N = 0, and L is not defined.

* * *

The following are all examples of invalid skeleton DO constructs:

Example 10:

```
      DO I=1, 10
         ...
      END DO LOOP              ! No matching construct name
```

Example 11:

```
      LOOP: DO 1000 I=1, 10    ! No matching construct name
         ...
 1000 CONTINUE
```

Example 12:

```
      LOOP1:  DO
         ...
      END DO LOOP2             ! Construct names don't match
```

Example 13:

```
      DO I=1, 10               ! Label required or...
         ...
 1010 CONTINUE                 ! ...END DO required
```

Example 14:

```
      DO 1020 I=1, 10
         ...
 1021 END DO                   ! Labels don't match
```

Example 15:

```
      FIRST: DO I=1, 10
         SECOND: DO J=1, 5
            ...
         END DO FIRST          ! Improperly nested DOs
      END DO SECOND
```

END PROPOSAL 2

SLAC MEMORANDUM

(44)

106(16)LJM-3

30 September 1987

TO: X3J3

FROM: Len Moss

SUBJECT: Rewrite of DO construct description, revisited

[This is an alternate version of 106(16)LJM-2, "Rewrite of DO construct description". The only significant differences are in the syntax (BNF and constraints) for the non-block-structured forms of the DO. I apologize for sending two separate proposals, but I had already mailed the first version before Jeanne Martin convinced me I should present this version of the syntax in the pre-meeting distribution as well.]

I was asked by the editorial committee at the Liverpool meeting to try to clarify the section on the DO construct. The idea was to give the primary description in terms of the properly block-structured (and core-conforming) forms of the DO, then cover the obsolescent versions afterward. During discussion in full committee, it was also suggested that I revise the examples of DO constructs.

The first proposal below contains my attempt to separate the description of the DO construct into two pieces, as suggested by the editorial committee. It also contains a number of other suggested improvements for this portion of Section 8, such as moving the BNF for the CYCLE and EXIT statements later in the section. I have not attempted to make these other changes into separate proposals because many of them are dependent on one another. Instead, I have added notes within the text of the proposal explaining my reasons for each group of changes.

* * *

PROPOSAL 1

On page 2-3, delete the constraint in line 41.

> **COMMENT:** This constraint should be covered in Section 8, since there is an additional requirement if the EXIT or CYCLE statement refers to a DO construct name. **END COMMENT.**

On page 8-1, replace line 10 with "There is also a non-block-structured form of the DO construct."

On page 8-1, in line 15, change "embedded." to "embedded; however, in some forms of the DO construct a sequence of executable constructs without a terminating boundary statement obeys all the other rules governing blocks (8.1.1)."

> **COMMENT:** I want also to consider the *do-body* to be a block when the loop ends with a non-shared CONTINUE; and *do-termination* is not quite the right term since I use *end-do* for block-structured DOs. In any case it seems worthwhile to avoid forward references to the these BNF terms so early in the section. **END COMMENT.**

On page 8-5, replace lines 21-22 with the following:

8.1.4 Iteration Control. Iteration control is provided by the DO construct and the EXIT and CYCLE statements. The DO construct specifies the repeated execution of a sequence of executable constructs. Such a repeated sequence is called a **loop**. The EXIT and CYCLE statements may be used to modify the execution of a loop.

The number of iterations of a loop may be determined at the beginning of execution of the DO construct, or may be left indefinite ("DO forever"). In either case, an EXIT statement (8.1.4.4.4) may be executed anywhere within the DO construct to immediately terminate the loop. A particular iteration of the loop may be abbreviated by executing a CYCLE statement (8.1.4.4.3).

Replace Section 8.1.4.1 with the following:

8.1.4.1 Forms of the DO construct. The DO construct can be written in either a block-structured form or a non-block-structured form.

R816 *do-construct* **is** *block-do-construct*
or *non-block-do-construct*

8.1.4.1.1 Form of the Block-Structured DO construct.

R817 *block-do-construct* **is** *do-stmt*
do-block
end-do

R818 *do-stmt* **is** *label-do-stmt*
or *nonlabel-do-stmt*

R819 *label-do-stmt* **is** [*do-construct-name*:] DO *label* [*loop-control*]

R820 *nonlabel-do-stmt* **is** [*do-construct-name*:] DO [*loop-control*]

R821 *loop-control* **is** [,]*do-variable* = *scalar-numeric-expr* , ■
■ *scalar-numeric-expr* [, *scalar-numeric-expr*]
or (*scalar-int-expr*) TIMES

R822 *do-variable* **is** *scalar-variable*

Constraint: The *do-variable* must be a scalar integer, default real, or default double precision real named variable.

Constraint: Each *scalar-numeric-expression* in *loop-control* must be of type integer, default real, or default double precision real.

R823 *do-block* is *block*

R824 *end-do* is *end-do-stmt* [*do-construct-name*]
or *continue-stmt*

Constraint: If the *do-stmt* of a *block-do-construct* is identified by a *do-construct-name* the corresponding *end-do* must be an *end-do-stmt* referring to the same *do-construct-name*; and if the *do-stmt* of a *block-do-construct* is not so identified, the corresponding *end-do* must not refer to a *do-construct-name.*

Constraint: If the *do-stmt* is a *nonlabel-do-stmt*, the corresponding *end-do* must be an *end-do-stmt.*

Constraint: If the *do-stmt* is a *label-do-stmt*, the corresponding *end-do* must be identified with the same *label.*

8.1.4.1.2 Form of the Non-Block-Structured DO construct.

R825 *non-block-do-construct* is *action-term-do-construct*
or *shared-term-do-construct*

R826 *action-term-do-construct* is *label-do-stmt*
do-body
do-term-action-stmt

R827 *do-body* is [*execution-part-construct*]...

R828 *do-term-action-stmt* is *action-stmt*

Constraint: A *do-term-action-stmt* must not be a *continue-stmt, goto-stmt, return-stmt, stop-stmt, exit-stmt, cycle-stmt, arithmetic-if-stmt*, nor *assigned-goto-stmt.*

Constraint: The *do-term-action-stmt* must be identified with a label and the corresponding *label-do-stmt* must refer to the same label.

R829 *shared-term-do-construct* is *outer-shared-do-construct*
or *inner-shared-do-construct*

R830 *outer-shared-do-construct* is *label-do-stmt*
do-body
shared-term-do-construct

R831 *inner-shared-do-construct* is *label-do-stmt*
do-body
do-term-shared-stmt

Constraint: A *do-term-shared-stmt* must not be a *goto-stmt, return-stmt, stop-stmt, exit-stmt, cycle-stmt, arithmetic-if-stmt*, nor *assigned-goto-stmt.*

Constraint: The *do-term-shared-stmt* must be identified with a label and all of the *label-do-stmts* of the *shared-term-do-construct* must refer to the same label.

The *do-term-action-stmt, do-term-shared-stmt,* or *shared-term-do-construct* following the *do-body* of a non-block-structured DO construct is called the **DO termination** of that construct.

Within a scoping unit, all DO constructs whose DO statements refer to the same label are non-block-structured, and are said to **share** the statement identified with that label. The statement so identified must be the *do-term-shared-stmt* of the innermost of these DO constructs.

COMMENT: I believe the above BNF and constraints are equivalent to the version in S8.104, although this needs to be carefully reviewed by a number of people. Not counting the rules and constraints involving the CYCLE and EXIT statements, which have been moved to Sections 8.1.4.4.3 and 8.1.4.4.4, it results in a net total of 8 additional rules and no additional constraints; however, the block-structured portion has only 1 additional rule and 4 fewer constraints. Moreover, all the core features are defined first and most of the complexity is concentrated in the obsolescent section. **END COMMENT.**

Replace Section 8.1.4.2 with the following:

8.1.4.2 Range of the DO construct. The **range** of a block-structured DO construct is the *do-block*, and satisfies the usual rules for blocks (8.1.1). In particular, transfer of control to the interior of such a block from outside the block is prohibited. It is permissible to branch to the END DO statement or CONTINUE statement terminating a block-structured DO construct only from within the range of that DO construct.

The **range** of a non-block-structured DO construct consists of the *do-body* and the following DO termination. The end of such a range is not bounded by a particular statement as for the other executable constructs (e.g., END IF); nevertheless, the range satisfies the rules for blocks (8.1.1). Transfer of control into the *do-body* or to the DO termination from outside the range is prohibited; in particular, it is permissible to branch to a *do-term-shared-stmt* only from within the range of the corresponding *inner-shared-do-construct*.

COMMENT: The above two paragraphs take advantage of the different BNF terms, *do-block*, *do-body*, *do-term-action-stmt*, *do-term-shared-stmt*, and *shared-term-do-construct*, to precisely define the non-BNF terms **DO termination** and **range**. for the various forms of the DO construct.

The definition of the term **share** for DOs has been moved out of the section on range and up to the above section on the form of the DO. **END COMMENT.**

On page 8-6, line 36, change "loop. A loop is " to "loop, i.e.,".

COMMENT: I have moved the definition of loop to the beginning of Section 8.1.4. **END COMMENT.**

On page 8-6, line 38, change "loop body" to "loop range".

On page 8-7, line 22 change "DO construct" to "loop".

COMMENT: After going to the trouble to define both range and loop, I thought we should try to make more use of them. **END COMMENT.**

On page 8-7, line 17, change "*do-construct*" to "loop terminates and the DO construct".

On the same page, in line 18, change "*do-constructs*" to "DO constructs"; and in line 19, change "the construct" to "all of these constructs".

On the same page, in line 23, change "*do-variable*" to "DO variable".

COMMENT: In different places, we talk about DO constructs becoming inactive and about loops terminating, so I think it's important to use both phrases in describing what happens when the iteration count is zero. I also tried to be more consistent about avoiding BNF terms in text if good alternatives are available. **END COMMENT.**

On page 8-7, lines 27-31, delete the last two sentences of Section 8.1.4.4.2.

COMMENT: I have moved the gist of these two sentences into the next section. **END COMMENT.**

Replace Section 8.1.4.4.3 with the following:

8.1.4.4.3 Cycle Abbreviation. Step (2) in the above execution cycle may be abbreviated by executing a CYCLE statement from within the range of the loop.

R832 *cycle-stmt* **is** CYCLE [*do-construct-name*]

Constraint: If a *cycle-stmt* refers to a *do-construct-name*, it must be within the range of that *do-construct*; otherwise, it must be within the range of at least one *do-construct*.

A CYCLE statement is said to **belong** to a particular DO construct. If the CYCLE statement refers to a DO construct name, it belongs to that DO construct; otherwise, it belongs to the innermost DO construct in which it appears.

Execution of a CYCLE statement causes immediate execution of step (3) of the current execution cycle of the DO construct to which it belongs. If this construct is not block-structured, the DO termination is not executed.

In a block-structured DO construct, a transfer of control to the *end-do* has the same effect as execution of a CYCLE statement belonging to that construct. In a non-block-structured DO construct, transfer of control to the DO termination causes that statement or construct itself to be executed. Unless a further transfer of control results, step (3) of the current execution cycle of the DO construct is then executed.

COMMENT: I like the word "abbreviation" a whole lot better than "interruption". The latter seems to connote slowing down or even exiting the loop, whereas the former suggests shortening by leaving out a portion. Another possible word might be "truncation".

The syntax for the CYCLE and EXIT statements never seemed to me to fit properly with that for the DO construct itself, so I have moved them to the sections where their semantics are discussed. These are two separate sections, so there will be a little redundancy; on the other hand, the long block of syntax for the DO construct itself is shortened a bit which helps some.

I think pointing out explicitly that a non-block-structured DO's terminal construct is skipped when a CYCLE statement is executed helps to clarify both what the behavior actually is, and why such forms of the DO have been made obsolescent. **END COMMENT.**

Replace Section 8.1.4.4.4 with the following:

8.1.4.4.4 Loop Termination. The EXIT statement provides one way of terminating a loop.

R833 *exit-stmt* is EXIT [*do-construct-name*]

Constraint: If an *exit-stmt* refers to a *do-construct-name*, it must be within the range of that *do-construct*; otherwise, it must be within the range of at least one *do-construct*.

An EXIT statement is said to **belong** to a particular DO construct. If the EXIT statement refers to a DO construct name, it belongs to that DO construct; otherwise, it belongs to the innermost DO construct in which it appears.

The loop terminates, and the DO construct becomes inactive, when any of the following occurs:

(1) The iteration count is determined to be zero when tested during step (1) of the above execution cycle.

(2) Execution of an EXIT statement belonging to the DO construct.

(3) Execution of an EXIT statement or CYCLE statement that is within the range of the DO construct, but that belongs to an outer DO construct.

(4) Transfer of control to a statement that is neither the *end-do* nor within the range of the DO construct.

(5) Execution of a RETURN statement within the range of the DO construct.

(6) Execution of a STOP statement anywhere in the program; or termination of the program for any other reason.

COMMENT: As explained above, I have separated the syntax for EXIT and CYCLE from that for the DO construct itself. The rest of the changes in this section are relatively minor rephrasings, except for the change to item (1) in the list. The previous version of this item could possibly, I think, be interpreted to mean that the loop terminates as soon as the iteration count is decremented to zero in step (3) of the execution cycle, before the do variable is incremented. **END COMMENT.**

END PROPOSAL 1

COMMENT: One change I considered and still have mixed feelings about would be to substitute "associated with" for "belong to" to describe the relationship between a CYCLE or EXIT statement and a DO construct. "Belong" has never sounded quite right to me, especially not as a technical term with a formal (i.e., boldface) definition. The ordinary English word "associated" (e.g., "...the CYCLE statement...is associated with that DO construct...") seems to me to be a much better choice. On the other hand, Jeanne Martin pointed out that the word "associated" with a technical meaning is already heavily used in S8, and adding yet another meaning to it, albeit an informal meaning, might be more confusing than helpful. I'd be happy to hear suggestions for alternative words or phrases to use in this context. **END COMMENT.**

* * *

The following proposal is a complete rewrite of the DO construct examples section. I have kept the important examples copied from Fortran 77 but have modified some of the existing Fortran 8x examples, and have added quite a few new ones. It should be noted that I have included a few examples of **INVALID** syntax which I think help to clarify some of the BNF rules and constraints. I believe this proposal is independent of the previous one.

PROPOSAL 2

Replace Section 8.1.4.5 with the following:

8.1.4.5 Examples of DO constructs. The following are all valid examples of block-structured DO constructs:

Example 1:

```
READ (IUN, '(1X,G14.7)', IOSTAT=IOS) X
DO; IF (IOS .NE. 0) EXIT    ! A "DO WHILE" loop
   IF (X .GE. 0.)  THEN
      CALL SUBA (X)
      CALL SUBB (X)
         ...
      CALL SUBZ (X)
   ENDIF
   READ (IUN, '(1X,G14.7)', IOSTAT=IOS) X
END DO
```

The above program fragment contains a DO construct that does not have an iteration count. The loop will continue to execute until an end-of-file or I/O error is encountered, at which point the EXIT statement terminates the loop. When a negative value of X is read, the program skips immediately to the next READ, bypassing most of the range of the loop.

Example 2:

```
DO                              ! A "DO WHILE + 1/2" loop
   READ (IUN, '(1X,G14.7)', IOSTAT=IOS) X
   IF (IOS .NE. 0) EXIT
   IF (X .LT. 0.)  CYCLE
   CALL SUBA (X)
   CALL SUBB (X)
      ...
   CALL SUBZ (X)
END DO
```

This program fragment behaves exactly the same as the previous one. However, the EXIT statement has been moved to the interior of the range, so that only one copy of the READ statement is required. A CYCLE statement has also been used to avoid an extra level of IF nesting.

Example 3:

```
SUM = 0.
READ (IUN) N
OUTER: DO (N) TIMES          ! A DO with a construct name
   READ (IUN) IQUAL, M, ARRAY(1:M)
   IF (IQUAL .LT. IQUAL_MIN) CYCLE OUTER    ! Skip inner loop
   INNER: DO 30 I = 1, M    ! A DO with a label and a name
      CALL CALCULATE(ARRAY(I),RESULT)
      IF (RESULT .LT. 0) CYCLE
      SUM = SUM + RESULT
      IF (SUM .GT. SUM_MAX) EXIT OUTER
```

```
30        END DO INNER
        END DO OUTER
```

The outer loop has no DO variable; however, it does have an iteration count of MAX(N,0), and will execute that number of times unless SUM exceeds SUM_MAX, in which case the EXIT OUTER statement terminates both loops. The inner loop is skipped by the first CYCLE statement if the quality flag, IQUAL is too low. If CALCULATE returns a negative RESULT, the second CYCLE statement prevents it from being summed. Note that both loops have construct names, and that the inner loop also has a label. A construct name is required on the EXIT statement in order to terminate both loops, but is optional on the CYCLE statements since each belongs to its innermost loop.

Example 4:

```
      N = 0
      DO 40, I = 1, 10
         J = I
         DO  K = 1, 5
            L = K
            N = N + 1        ! This statement executes 50 times
         END DO              ! Non-labelled DO inside a labelled DO
   40 CONTINUE
```

After execution of the above program fragment, I = 11, J = 10, K = 6, L = 5, and N = 50.

Example 5:

```
      N = 0
      DO I = 1, 10
         J = I
         DO 50, K = 5, 1     ! This inner loop is never executed
            L = K
            N = N + 1
   50    CONTINUE            ! Labelled DO inside a non-labelled DO
      END DO
```

After execution of the above program fragment, I = 11, J = 10, K = 5, N = 0, and L is not defined.

* * *

The following are all valid examples of non-block-structured DO constructs:

Example 6:

```
      DO 60
         READ (IUN, '(1X,G14.7)', IOSTAT=IOS) X
         IF (IOS .NE. 0) EXIT
         IF (X .LT. 0.)  GOTO 60
         CALL SUBA (X)
         CALL SUBB (X)
            ...
         CALL SUBY (X)
         CYCLE
   60    CALL SUBNEG(X)     ! SUBNEG only called when X < 0.
```

This DO construct is not block-structured since it ends with a statement other than an ENDDO or CONTINUE. The loop will continue to execute until an end-of-file or I/O error occurs. Note that the CYCLE statement does not result in SUBNEG being called.

Example 7:

```
      SUM = 0.
      READ (IUN) N
      DO 70, (N) TIMES
         READ (IUN) IQUAL, M, ARRAY(1:M)
         IF (IQUAL .LT. IQUAL_MIN) M = 0    ! Skip inner loop
         DO 70 I = 1, M
            CALL CALCULATE(ARRAY(I),RESULT)
            IF (RESULT .LT. 0) CYCLE
            SUM = SUM + RESULT
            IF (SUM .GT. SUM_MAX) GOTO 71
   70       CONTINUE        ! This CONTINUE shared by both loops
   71    CONTINUE
```

This example is similar to Example 3 above, except that the two loops are not block-structured since they share the CONTINUE statement with the label 70. The terminal construct of the outer DO is the entire inner DO construct. The inner loop is skipped by forcing M to zero. If SUM grows too large, both loops are terminated by branching to the CONTINUE labelled 71. The CYCLE statement in the inner loop may still be used to skip negative values of RESULT.

Example 8:

```
      N = 0
      DO 100 I = 1, 10
         J = I
         DO 100 K = 1, 5
            L = K
  100       N = N + 1       ! This statement executes 50 times
```

In this example, the two loops share an assignment statement. After execution of this program fragment, I = 11, J = 10, K = 6, L = 5, and N = 50.

Example 9:

```
      N = 0
      DO 200 I = 1, 10
         J = I
         DO 200 K = 5, 1      ! This inner loop is never executed
            L = K
  200       N = N + 1
```

This example is very similar to the previous one, except that the inner loop is never executed. After execution of this program fragment, I = 11, J = 10, K = 5, N = 0, and L is not defined.

* * *

The following are all examples of invalid skeleton DO constructs:

Example 10:

```
      DO I=1, 10
         ...
      END DO LOOP             ! No matching construct name
```

Example 11:

```
      LOOP: DO 1000 I=1, 10   ! No matching construct name
         ...
 1000 CONTINUE
```

Example 12:

```
      LOOP1:  DO
         ...
      END DO LOOP2            ! Construct names don't match
```

Example 13:

```
      DO I=1, 10              ! Label required or...
         ...
 1010 CONTINUE                ! ...END DO required
```

Example 14:

```
      DO 1020 I=1, 10
         ...
 1021 END DO                    ! Labels don't match
```

Example 15:

```
      FIRST: DO I=1, 10
         SECOND: DO J=1, 5
            ...
         END DO FIRST           ! Improperly nested DOs
      END DO SECOND
```

END PROPOSAL 2

MEMORANDUM

(45)

106( )MBM−1
14 September 1987

To: X3J3

From: Mike Metcalf

Subject: Edits

I have noted the following points which I consider would improve the document. Only the change #6 is of any substance.

1. **P. 3-1, l. 26-39:**
   Box the Table, (better layout), and the Tables 6.1, 6.2, 7.1, 7.2, 7.5 (renumber 7.3), 7.6 (renumber 7.4 and set on one page), 7.7 (renumber 7.5), 7.8 (renumber 7.6 and set on one page), 7.9 (renumber 7.7 and set on one page), 7.10 (renumber 7.8), 7.11 (renumber 7.9), 12.1, F.1, F.2, F.3, and F.4; also the tables on P. 10-6, l. 23-35, P. 10-7, l. 18-25, P. 10-8, l. 1-10, and P. 14-5, l. 1-28.

2. **P. 4-5, l 21:**
   After "The length is" add "specified by", (clearer wording).

3. **P. 5-1 l. 11-12:**
   Change "EXPEND" to "EXPENDITURE", (better use of long name).

4. **P. 5-11, l. 16 and 18:**
   Add "value" after "default", ("default" is an adjective).

5. **P. 5-14, l. 36 and 39:**
   Delete (superfluous).

6. **P. 5-15, l. 20:**
   Replace
   is RANGE [/*range-list-name*/]*array-name-list*
   by
   is RANGE/ *range-list-name*/*array-name-list*
   or RANGE[::] *array-name-list*

   (make use of double colon permissable, in order to be more regular with respect to other attribute statements).

7. **P. 8-3, l. 39+:**
   Add "Constraint: The *case-value-ranges* in different *case-selectors* may not overlap." (a modified repetition of P. 8-4, l. 15-16).

8. **P. 8-6, l. 3:**
   Change the first "the" to "a", (better grammar, and consistency of wording).

9. **P. 9-5, l. 26:**
   Delete the whole sentence beginning "If connected...", (is stated better in sentence beginning on l. 27).

10. **P. 10-2, l. 7:**
    Change "It" to "The integer literal constant *r*"; move this sentence out of the constraint onto the following line, (a definition does not belong in a constraint).

11. Section 13 Global edits. The use (or absence) of the underscore character in intrinsic procedure names is irregular. Thus, LEN_TRIM is written with an _, but MATMUL without; the English words in SETEXPONENT are not separated, but those in DATE_AND_TIME are. I suggest that the rule be

    a. English words are always separated by underscores,

    b. artificial words are never separated by underscores.

    (We can think up other rules, but they should be clear and memorable.) The introduction of these rules would result in the following global changes:

    LEN_TRIM becomes LENTRIM
    SETEXPONENT becomes SET_EXPONENT
    DOTPRODUCT becomes DOT_PRODUCT
    RANDOMSEED becomes RANDOM_SEED

The following point comes from John Reid, and is necessary to correct the BNF to make possible the indexing of components of a derived type in an IDENTIFY statement:

P. 6-8 l. 33 add ■ at end of line
l. 33+ add ■[% *component-name*[(*mapping-subscript-list*)]]...

The last set of points were presented in outline to the Editorial Committee in Liverpool by David Williams. The last one seems serious, and needs work by subgroup.

1. Remove all constraints from Section 2 — they are sometimes repeated (differently!) in the other Sections.

    a. P. 2-1 l. 26-29 move to P. 12-9 l. 23+;

    b. P. 2-1 l. 34-36 move to P. 11-2 l. 26+;

    c. P. 2-1 l. 40-42 delete;

    d. P. 2-3 l. 1 delete;

    e. P. 2-3 l. 2-4 delete;

    f. P. 2-3 l. 38-39 move to P. 12-11 l. 17+, changing 'executable construct' to *'executable-construct'*.

    g. P. 2-3 l. 40 delete

    h. P. 2-3 l. 41 delete

2. P. 12-14 l. 7+ add 'Constraint: A given *dummy-arg-name* may appear only once in any *dummy-arg-name-list*', and delete sentence beginning "A given..." on l. 8-9.
   Move l. 10-12 to l. 8−, to become a constraint.

3. David(?) raised the question: if A is a *named constant*, what is A(10) (or A(5:7))? Upon investigation, we have no term, BNF or otherwise, for these objects. Any suggestions?

Can such an object appear as a *primary* (R701) in an *expr* (R712)? A constraint on R601 forbids them to be *variables*, and R305 and R307 say they aren't *constants*; by the same token A(I) is also not a *variable*. What is it? Is

```
REAL, ARRAY(10),PARAMETER :: A=[1:10]
B = A(5)          ! B real
```

legal? It seems not.

Two proposed editorial changes:

1. Example at top of page 5-5 has syntax error; line 11, change "M" to "LEN=M".

2. Page 8-5, line 21, change "Iteration Control" to "DO Construct." Make similar change in Table of Contents, page ix.

   This is to make this title consistent with the others in Section 8 (IF Construct, CASE Construct).

   Currently, the words "DO Construct" do not occur in the Table of Contents, but most other, less significant, statements are identified by name.

To: X3J3

From: John Reid

Subject: Meeting minutes

Date: 23rd September 1987

I would like to thank all the scribes for their support at the Liverpool meeting. Only two were too late for inclusion in the minutes and these have now arrived and are attached to this paper.

I have decided that I will always include a one-page set of guidelines for scribes in the pre-meeting distribution and number it JKR-1, so that people can find it easily. I have revised the guidelines slightly since the last meeting because of minor problems that I found this time.

Do not forget that amended copies of proposal papers must be handed to Neldon during the meeting. In general, the amendments are not scribed.

Specimen scribe notes

Discussion leader: Reid Scribe: Another

Reference:	104(*)JKR-3 AGENDA
Reid:	Begin with a brief overview of the topic by summarizing the presentation. Omit this if if the title already does it.
Straw Vote:	The scribe notes must be posted by air mail to the secretary (J. K. Reid, Blg 8.9, Harwell Laboratory, Oxon OX11 0RA, England) in the week following the meeting. (30-0-0)
Jones:	What format do you want?
Ans:	Copy these notes as closely as possible. If you do not have bold, use underlining. Please send a top copy, printed on one side of the paper, with adequate margins (line length at most 7 inches), avoid a grey tone (worn ribbon), and do not fold it.
Motion:	If a proposal in a paper is amended, the discussion leader must provide an amended copy of the paper to the librarian (Marshall) before the end of the meeting (Reid, Adams).
Formal Vote:	**27-0. Passed.**

7 Glossary

Discussion leader: Metcalf

Scribe: Hoover

Reference: 105(*) JCA-[illegible] Editorial Items for General Submission

Summary: Proposal A proposes to interchange Appendix E and Appendix F in the document. Proposal B proposes adding some subsection headings to Appendix C.

Metcalf: There were two proposals contained in this document. Proposal A proposed interchanging appendices E and F. The editorial committee discussed this but decided the change would result in endless confusion. Removed extensions are most commonly referred to as "Appendix F."

Adams: Will this be true for the final standard?

Metcalf: We do not want the final standard to contain Appendix F.

Adams: Are there going to be two indices or just one?

Paul: We voted on that at one time. We also voted on Appendix F.

Metcalf: Take a straw vote on the issue. Yes means we want to elim[illegible]te Appendix F from the final standard. (17-4-11).

Adams: I think it would be appropriate to have a proposal on this. Refer to WG5 Resolution R7.

Weaver: If this proposal is passed then it becomes the first document in S14.

Adams: I failed to report on one of the planning items from the sub-group heads meeting. We should not have a document related to public review until it goes out. S14 should be all documents relating to WG5 Liverpool resolutions.

[What follows is a discussion of S14 and changes made to the new version of S8.

Hirchert: We don't officially have control of the document until the public review is open.

Adams: But we do have control over S15.

Hirchert: WG5 resolutions are of a general nature. R7 is the only one that changes the document. There seems to be no point in [S14].

Metcalf: I agree. The editorial committee is unlikely to bring any proposals at this meeting.

Adams: We've been charged by WG5 to produce a document. We will eventually have a modified document called S15.

Hirchert: I suggest that the bulk of WG5 resolutions be sent to ANSI to be registered as official public comments.
[Scribe comment: I think Jeanne Martin is going to do this but I'm not sure. I was trying to catch up with the discussion.]

Adams: Then redo the straw vote.

Williams: I'm concerned that all the good work on condition handling will disappear from the public arena.

Schonfelder: Brian Meek voted no on this resolution because of things like exception handling. Vendor extensions will likely follow Appendix F.

Marusak: All those things were important to some people at some time and they've now crossed some plateau of interest.

Weaver: We might want to update a JOD more than every 10 years. Perhaps we want to issue a Technical Report.

Metcalf: We should either put these features in the standard or not have Appendix F at all since we don't want subsets or supersets.

Paul: But we want to guide those vendors who decide to add more features.

Burch: Subsetting in F77 was a drastic impairment to portability. There's nothing in Appendix F that is current practice.

Hirchert: If Appendix F is to be retained, significant technical and editorial work needs to be done. We have enough to do with the main document and we should issue Appendix F as a Technical Report.

Wagener: My impression was that our motivation for Appendix F was to get that information in front of the public and get a reaction to those features. It's explicitly there to get the public's comments. Then either put them in the standard or in some other document. It's not appropriate to have them in the standard because of the possibility that they might change.

Adams: Here are two considerations:
1. Eliminate Appendix F from the document.
2. Move Appendix F into a JOD.
I think Appendix F should go somewhere.

[Back to editorial matters.]

Metcalf: Proposal B. We think it's an excellent idea but we also feel there should be a consistent implementation of subsection headings throughout the whole of Appendix C. The problem is

37

that Appendix C has been put together by various individuals and is not consistent. Tracy is going to investigate this and submit a proposal for the next meeting.

Adams: My motivation was to start the editorial committee on general editing tasks.

Metcalf: Glossary Issues. Reference: 105(*) JCA-4 -- p. 177 of the premeeting and item 34 on the table.

The chair of X3K5 (Dictionary) has contacted us and wants to include a Fortran glossary in the next edition. He has based it on an earlier version of John Reid's glossary.

We wondered whether he wanted a glossary for F77 or F8x and decided he wanted one for both. He thinks that John's glossary is the union of these things.

We don't find it interesting to do a glossary for F77. We're not ready to do something for him for F8x. We think it's wrong of us to support the publication of incorrect information. Discussion in editorial committee has revealed that Wood has done some fine work and it will be very useful to us.

Straw vote: Eliminate Appendix F from the final Standard (17-4-11).

Straw vote: Establish discussion document (24-6-6).

16 Public Review

Discussion leader: Burch Scribe: Philips

Carl Burch	Does everyone need a copy of every letter? The volume is so large - not everyone can read it. Could mail copies at requesters expense.
John Reid	Microfiche is a possibility.
Jim Matheny	Cannot make copies.
Carl Burch	Every Subgroup will have a complete printed set of public comments.
Miles Ellis	Microfiche is acceptable if copies are available.
Dick Weaver	Supports microfiche. Small viewers are available.
Len Moss	Likes Microfiche.

Jim Matheny	Cobol had lots of comments. C had few comments. Because of $100. cost of manual we will receive few comments.
Kurt Hirchert	Public relations disaster if it is known that not every letter was received by every member.
George Paul	Want complete set. Microfiche acceptable. Evaluations can be subjective.
Alex Marusak	Microfiche is best solution.
Jeanne Martin	Essential that everyone has a copy of everything. Cross referencing is important. Agree with Jim - will be relatively few comments.

Straw vote: It is important that all members get a copy of all comments, either on paper or on microfiche (24-0-7).

To: X3J3
From: John Reid
Date: 13 September, 1987
Subject: Derived-type I/O

Introduction

It has been recognized for a long time that we have a deficiency in respect of I/O for derived types. What is needed is a means of specifying the conversions between scalars of the derived type and character strings. Adequate flexibility, with some loss of convenience, is provided on output by calling a function in the output list. However, there is no way to call a procedure on input.

I propose that the external form should be a character string, exactly as in list-directed I/O. On input, the processor will pass this string to a 'derived-edit' subroutine which will return a scalar value of derived type. On output, the processor will call a derived-edit function that converts a scalar of derived type to a character string that the processor outputs as in list-directed output.

I propose that such procedures have a suffix of the form

EDIT [(*letter*)]

and have explicit interfaces. If (*letter*) is absent, the procedure defines the conversion for list and name directed I/O, and if it is present, the letter specifies the name of the edit descriptor. Overloading of existing letters is acceptable because the type of the item in the I/O list resolves the overload. The first argument of a function must be a scalar input argument of derived type and the function must return a scalar character result. The first argument of a subroutine must be a scalar output argument of derived type and the second argument must be a scalar character input argument. Where an edit letter is provided, there may be further arguments; they must be scalar input arguments and are given values from integer constants in the edit descriptor.

To illustrate these ideas, consider a module for extended precision arithmetic that uses the type

```
TYPE EXTENDED
    INTEGER MANTISSA(4), EXPONENT
END TYPE
```

The external form is a real constant that lies within a single record and has no embedded spaces or a real constant enclosed in quotes or apostrophes that may have embedded spaces and may lie on more than one record.

For list-directed input, a suitable subroutine might be

```
SUBROUTINE INL(E,C) EDIT
    TYPE(EXTENDED), OUT :: E
    CHARACTER(*),    IN :: C
        :    ! Code that converts to extended type.
END SUBROUTINE
```

and for list-directed output, a suitable function might be

```
FUNCTION OUTL(E) EDIT
    TYPE(EXTENDED), IN :: E
    CHARACTER(50)      :: OUTL
        :   ! Code to convert to a character string.
END FUNCTION
```

For formatted input with descriptor E*w.d*, a suitable subroutine might be

```
SUBROUTINE INF(E,C,W,D) EDIT(E)
    TYPE(EXTENDED), OUT :: E
    CHARACTER(*),   IN  :: C
    INTEGER,        IN  :: W, D
        :    ! Code that converts to extended type.
END SUBROUTINE
```

and for formatted output with descriptor E*w.d*, a suitable function might be

```
FUNCTION OUTF(E,W,D) EDIT(E)
    TYPE(EXTENDED), IN :: E
    CHARACTER(W)       :: OUTF
    INTEGER,        IN :: W, D
        :   ! Code to convert to a character string.
END FUNCTION
```

If we need to be able to read back a value that has previously been output, it will be best to output delimitted strings, that is use a file that has been opened with a DELIM specifier. I have considered and rejected the idea of providing more control on the length of the external string, as in the *w* field of many formats. It would make the facility more complicated and add little extra functionality. In any case, we may not necessarily want to count in characters (see 106(*)JKR-3 where a Kanji module is considered, for example).

To: X3J3
From: John Reid
Date: 19 September, 1987
Subject: A derived string type for large characters

At the last meeting, I was alone in voting in favour of continuing to look at the module approach for the Kanji problem. The main difficulty was in respect of I/O, which we should address anyway. and a possible approach is suggested in 106(22)JKR-2. Another problem is associated with substrings. Again, our derived-type mechanism has been shown to be inadequate. Here I explore repairing the deficiency.

Substrings are akin to another array dimension. We allow arrays to be constructed for derived types in exactly the same way as for intrinsic types, so why not allow substrings too? My suggestion is very simple. I propose that a derived type with a single type parameter called LEN and a single component that is a rank-one array of length LEN be accessible with substring notation. For example, for Kanji we might use the type

```
TYPE LCHARACTER (LEN) ! Type for a string of large characters
   CHARACTER(2) WORD(LEN)
END TYPE
```

Given such a type, we would be permitted to declare a variable such as

```
TYPE(LCHARACTER(10)) S
```

and then access substrings such as S(3:3) and S(:I).

A module for such a data type is given below. It mimics the facilities for CHARACTER. Inspection of procedures such as VERIFY shows that the new type is very similar to CHARACTER in use. The module 'cheats' once; the length of the result of TRIM needs the new function LEN_TRIM, which would be legal only in an intrinsic module. The one thing it lacks is a readable form for Kanji constants. My suggestion here is that a compiler intended for use in Japan should permit forms such as '漢字' and interpret them as of type LCHARACTER. Different versions would be needed for other countries with very large character sets. And I do not think we should be concerned about escape sequences. For instance '漢字' would perhaps be held in the source as a 3-byte escape character, two 2-byte characters, and another 3-byte escape character, but the program would regard it as LCHARACTER(2). For I/O, I assume that we always know whether we are reading small or large characters, just as we need to know whether we are reading reals or integers. If the module is intrinsic, it could have more complicated versions of the I/O procedures that process excape characters when necessary.

I would like us to adopt this extension of substringing to derived types as part of the standard, and adopt a module such as LSTRING, below, as an intrinsic module in a collateral standard. We have built a extension mechanism into Fortran 8x. Let's use it, rather than adding yet more to an already very complicated language.

```
MODULE LSTRING

     TYPE LCHARACTER (LEN) ! Type for a string of large characters
        CHARACTER(2) WORD(LEN)
     END TYPE

     CHARACTER(2), PARAMETER, PRIVATE :: SPACE = CHAR(63)//CHAR(65)
     TYPE(LCHARACTER(1)), PARAMETER :: LSPACE = LCHARACTER(1)([SPACE])
        ! LSPACE holds the large space character for use inside and
        ! outside the module. SPACE holds its CHARACTER(2) form for
        ! use inside the molule only.

CONTAINS

! Assignment subroutines

     SUBROUTINE ASSIGNLL(S,T) ASSIGNMENT ! LCHARACTER from LCHARACTER
        TYPE(LCHARACTER(*)), OUT :: S
        TYPE(LCHARACTER(*)), IN :: T
        L = MIN(LEN(S),LEN(T))
        S%WORD = [T%WORD(:L),(LEN(S)-L)[SPACE]]
     END SUBROUTINE

     SUBROUTINE ASSIGNLC(S,T) ASSIGNMENT ! LCHARACTER from CHARACTER
        TYPE(LCHARACTER(*)), OUT :: S
        CHARACTER(*), IN :: T
        L = MIN(LEN(S),LEN(T)/2)
        DO I = 1, L
           S%WORD(I) = C(2*I-1:2*I)
        END DO
        S%WORD(L+1:) = SPACE
     END SUBROUTINE

     SUBROUTINE ASSIGNCL(S,T) ASSIGNMENT ! CHARACTER from LCHARACTER
        CHARACTER(*), OUT :: S
        TYPE(LCHARACTER(*)), IN :: T
        L = MIN(LEN(T),LEN(S)/2)
        DO I = 1, L
           S(2*I-1:2*I) = T%WORD(I)
        END DO
        S(L+1:)=''
     END SUBROUTINE

! Concatenate operator

     FUNCTION CONCATENATE(S,T) OPERATOR(//)
        TYPE(LCHARACTER(*)) S,T
        TYPE(LCHARACTER(LEN(S)+LEN(T))) CONCATENATE
           CONCATENATE%WORD = [S%WORD,T%WORD]
     END FUNCTION
```

```
! Comparison functions (only == and > are shown)

    LOGICAL FUNCTION EQUAL(S,T) OPERATOR(==)
       TYPE(LCHARACTER(*)) S,T
       TYPE(LCHARACTER( MAX(LEN(S),LEN(T)) )) :: CS,CT
       CS = S; CT = T ! Uses subroutine ASSIGNLL
       EQUAL = ALL(CS%WORD==CT%WORD)
    END FUNCTION

    LOGICAL FUNCTION GREATER(S,T) OPERATOR(>)
       TYPE(LCHARACTER(*)) S,T
       TYPE(LCHARACTER( MAX(LEN(S),LEN(T)) )) :: CS,CT
       CS = S; CT = T ! Uses subroutine ASSIGNLL
       DO I = 1,LEN(CS)
          IF(CS%WORD(I)<>CT%WORD(I))GO TO 10
       END DO
       GREATER  = .FALSE.
       RETURN
    10 GREATER = CS%WORD(I)>CT%WORD(I)
    END FUNCTION

! Edit descriptor functions

    SUBROUTINE IN(S,T) EDIT(A) ! Descriptor A
       TYPE(LCHARACTER(*)), OUT :: S
       CHARACTER(*), IN :: T
       S = T ! Uses subroutine ASSIGNLC
    END SUBROUTINE

    SUBROUTINE INW(S,T,W) EDIT(A) ! Descriptor A w
       TYPE(LCHARACTER(*)), OUT :: S
       CHARACTER(*), IN :: T
       INTEGER, IN :: W
       S = T(:W*2) ! Uses subroutine ASSIGNLC
    END SUBROUTINE

    SUBROUTINE INL(S,T) EDIT ! List-directed input
       TYPE(LCHARACTER(*)), OUT :: S
       CHARACTER(*), IN :: T
       S = T ! Uses subroutine ASSIGNLC
    END SUBROUTINE

    FUNCTION OUT(S) EDIT(A) ! Descriptor A
       TYPE(LCHARACTER(*)), IN :: S
       CHARACTER(2*LEN(S)) OUT
       OUT = S ! Uses subroutine ASSIGNCL
       END SUBROUTINE

    FUNCTION OUTW(S,W) EDIT(A) ! Descriptor A w
       TYPE(LCHARACTER(*)), IN :: S
       INTEGER, IN ::  W
       CHARACTER(2*W) OUTW
       OUTW = S ! Uses subroutine ASSIGNCL
    END SUBROUTINE

    FUNCTION OUTL(S) EDIT ! List-directed output
       TYPE(LCHARACTER(*)), IN :: S
       CHARACTER(2*LEN(S)) OUTL
       OUTL = S ! Uses subroutine ASSIGNCL
    END SUBROUTINE
```

```
! Analogues of the intrinsic functions for CHARACTER type

    FUNCTION ILCHAR(C) ! Collating position of given large character
       TYPE(LCHARACTER(1)) C
       CHARACTER(2) CC
       CC = C ! Uses subroutine ASSIGNCL
       ILCHAR = ICHAR(CC(1:1)*256) + ICHAR(CC(2:2))
    END FUNCTION

    FUNCTION LCHAR(I) ! Large character in given collating position
       TYPE(LCHARACTER(1)) LCHAR
       LCHAR = CHAR(I/256) // CHAR(I-(I/256)*256)
    END FUNCTION

    FUNCTION INDEX(STRING,SUBSTRING,BACK)
       TYPE(LCHARACTER(*)) STRING, SUBSTRING
       LOGICAL, OPTIONAL ::  BACK
       LOGICAL FORWARD
       FORWARD = .TRUE.
       IF(PRESENT(BACK))FORWARD = .NOT.BACK
       IF(FORWARD)THEN
          DO INDEX = 1,LEN(STRING)-LEN(SUBSTRING)+1
             IF(STRING(I:I+LEN(SUBSTRING)-1) == SUBSTRING) RETURN
          END DO
          INDEX = 0
       ELSE
          DO INDEX = LEN(STRING)-LEN(SUBSTRING)+1,1,-1
             IF(STRING(I:I+LEN(SUBSTRING)-1) == SUBSTRING) RETURN
          END DO
          INDEX = 0
       END IF
    END FUNCTION

    INTEGER FUNCTION LEN_TRIM(S) RESULT I
       TYPE(LCHARACTER(*)) S
       DO I = LEN(S),1,-1
          IF (S(I:I)<>LSPACE) EXIT
       END DO
    END FUNCTION

    FUNCTION REPEAT(STRING,NCOPIES)
       TYPE(LCHARACTER(*)) STRING
       INTEGER NCOPIES
       TYPE(LCHARACTER(LEN(STRING)*NCOPIES)) REPEAT
       REPEAT%WORD = [NCOPIES[STRING%WORD]]
    END FUNCTION
```

```
      INTEGER FUNCTION SCAN(STRING,SET,BACK)
         TYPE(LCHARACTER(*)) STRING, SET
         LOGICAL, OPTIONAL ::  BACK
         LOGICAL FORWARD
         FORWARD = .TRUE.
         IF(PRESENT(BACK))FORWARD = .NOT.BACK
         IF(FORWARD)THEN
            DO SCAN = 1,LEN(STRING)
               DO J = 1,LEN(SET)
                  IF(STRING(SCAN:SCAN) == SET(J:J)) RETURN
               END DO
            END DO
            SCAN = 0
         ELSE
            DO SCAN = LEN(STRING),1,-1!
               DO J = 1,LEN(SET)
                  IF(STRING(SCAN:SCAN) == SET(J:J)) RETURN
               END DO
            END DO
            SCAN = 0
         END IF
      END FUNCTION

      FUNCTION TRIM(STRING)
         TYPE(LCHARACTER(*)) STRING
         TYPE(LCHARACTER(LEN_TRIM(STRING)) ! Illegal statement in a
                                           ! nonintrinsic module.
         TRIM = STRING
      END FUNCTION

      INTEGER FUNCTION VERIFY(STRING,SET,BACK)
         TYPE(LCHARACTER(*)) STRING, SET
         LOGICAL, OPTIONAL ::  BACK
         LOGICAL FORWARD
         FORWARD = .TRUE.
         IF(PRESENT(BACK))FORWARD = .NOT.BACK
         IF(FORWARD)THEN
            OUTERF: DO VERIFY = 1,LEN(STRING)
               DO J = 1,LEN(SET)
                  IF(STRING(VERIFY:VERIFY) == SET(J:J)) CYCLE OUTERF
               END DO
               RETURN
            END DO
            VERIFY = 0
         ELSE
            OUTERB: DO VERIFY = LEN(STRING),1,-1
               DO J = 1,LEN(SET)
                  IF(STRING(VERIFY:VERIFY) == SET(J:J)) CYCLE OUTERB
               END DO
               RETURN
            END DO
            VERIFY = 0
      END FUNCTION
END MODULE
```

To: X3J3

From: John Reid

Subject: Glossary

Date: 30th September 1987

50

106(13)JKR-4

The following is the product of work done by the editorial group at the Liverpool meeting and by me afterwards. The intention is to provide a replacement for Appendix H of S8 that would be an improvement on the present version and would also be suitable to place in ANSDIS (American National Standard Dictionary for Information Systems).

Glossary of Technical Terms Used in Fortran 8x

The following is a list of the principal technical terms used in the draft standard and their definitions. A reference in parentheses immediately after a term is to the section number in the draft standard where the term is defined or explained. A reference in parentheses that starts 'R' is to a syntax rule number in the draft standard. The wording of a definition here is not necessarily the same as in the draft standard.

action statement. A single *statement* specifying a computational action (R223).

actual argument (12.4). An *expression*, *variable*, *procedure*, or alternate return specifier that is specified in a procedure *reference*.

alias (5.1.2.4.3). A *named data object* whose *type*, *type parameters*, and *rank* are specified in a *type declaration statement* containing an ALIAS *attribute*. It may not be *referenced* or *defined* unless it is *alias associated* with a data object that may be referenced or defined. If it is an *array*, it does not have a *shape* unless it is alias associated.

alias array (5.1.2.4.3). An *array* that is an *alias*.

alias associated (6.2.6). The relationship between an *alias* with a nonalias *object* following a valid execution of an IDENTIFY *statement*.

alias association (14.7.1.3). The process of establishing an *alias* and the resulting relationship.

allocatable array (5.1.2.4.3). A *named array* whose *type*, *type parameters*, and *rank* are specified in a *type declaration statement* containing an ALLOCATABLE *attribute*. Only when it has space allocated for it does it have a *shape* and may it be *referenced* or *defined*.

argument (12). An *actual argument* or a *dummy argument*.

argument association (14.7.1.1). The relationship between an *actual argument* and *dummy argument* during the execution of a *procedure reference*.

argument keyword (2.5.2). A *dummy argument name*. It may used in a *procedure reference* ahead of the equals symbol (R1209) provided the procedure has an *explicit interface*.

array (2.4.7). A set of *data*, all of the same *type* and *type parameters*, whose individual elements are arranged in a rectangular pattern. It may be a *named* array, an *array section*, a *structure component*, a *function* value, or an *expression*. Its *rank* is at least one.

3.51

array element (2.4.7, 6.2). One of the *scalar data* that make up an *array*.

array section (6.2.4.3). A *subobject* of an *array* consisting of regularly spaced *array elements* or *substrings* of regularly spaced array elements.

array-valued. Having the property of being an *array*.

assignment statement (7.5.1.2). A *statement* of the form '*variable* = *expression*'.

association (14.7). *Name association* or *storage association.*

assumed-size array (5.1.2.4.4). A *dummy array* whose *size* is assumed from the associated *actual argument.* Its last upper bound is specified by an asterisk.

attribute (5). A property of a *data object* that may be specified in a *type declaration statement* (R501).

belong (8.1.4.1). If an EXIT or CYCLE *statement* contains a *construct name*, the statement belongs to the DO construct using that name. Otherwise, it belongs to the innermost DO construct in which it appears.

block (8.1). A sequence of *executable constructs* embedded in another executable construct, bounded by *statements* that are particular to the construct, and treated as an integral unit.

block data program unit (11.4). A *program unit* that provides initial values for *data objects* in *named common blocks.*

bounds (5.1.2.4.1). For a *named array*, the limits within which the values of the *subscripts* of its *array elements* must lie.

character. A letter, digit, or other symbol.

characteristics (12.2)

(1) Of a *procedure*, its classification as a *function* or *subroutine*, the characteristics of its *dummy arguments*, and the characteristics of its *function result* if it is a function.

(2) Of a *dummy argument*, whether it is a *data object*, *procedure*, or an alternate return indicator; and whether it has the OPTIONAL *attribute.*

(+ more)

character string (4.3.2.1). A sequence of *characters* numbered from left to right 1, 2, 3, ...

character storage unit (14.7.2.1). The unit of storage for holding a *datum* of *type* character.

common block (5.5.2). A block of physical storage that may be accessed by any of the *scoping units* in an *executable program.*

component (4.4). A constituent of a *derived type.*

conformable (2.4.7). Two *arrays* are said to be **conformable** if they have the same *effective shape.* A *scalar* is **conformable** with any array.

conformance (1.4). An *executable program* conforms to the standard if it uses only those forms and relationships described therein and if the executable program has an interpretation according to the standard. A *program unit* conforms to the standard if it can be included in an executable program in a manner that allows the executable program to be standard conforming. A *processor* conforms to the standard if it executes standard-conforming programs in a manner that fulfills the interpretations prescribed in the standard.

3 1C

connected (9.3.2).

(1) For an *external unit*, the property of referring to an *external file*.

(2) For an *external file*, the property of having an *external unit* that refers to it.

constant (2.4.4). A *data object* whose value must not change during execution of an *executable program*. It may be a *named constant* or a *literal constant*.

constant expression (7.1.6.1). An *expression* satisfying rules that ensure that its value does not vary during program execution.

construct (8). A sequence of *statements* starting with a CASE, DO, IF, or WHERE statement and ending with the corresponding terminal statement.

data. Plural of *datum*.

data entity (2.4.3, 4.2). An *entity* that has or may have a data value. It may be a *constant*, a *variable*, an *expression*, or a *function*.

data object (2.4.3.1). A *datum* or a set of *data* of the same *type* and *type parameters* that may be *referenced* as a whole. It may be *named* or it may be a *subobject*.

data type (2.4.1). A *named* category of *data* that is characterized by a set of values, together with a way to denote these values and a collection of *operations* that interpret and manipulate the values. A data type may be parameterized, in which case the set of data values depends on the values of the parameters.

datum. A single quantity that can take any of the set of values specified for its *data type*.

declared (6.2.1.2). Adjective applied to the *range*, *bounds*, *extents*, *size*, or *shape* of an *array*. It always refers to the whole of the array, even in the case of a *named* array with the RANGE *attribute*.

deferred-shape array (5.1.2.4.3, 6.2.6). An *allocatable array* or an *alias array*.

definable (2.5.4). A *variable* is **definable** if its value may be changed by the appearance of its *name* or *designator* on the left of an *assignment statement*. An *allocatable array* that has not been allocated is an example of a *data object* that is not definable. An example of a *subobject* that is not definable is C(I) when C is an *array* that is a *constant* and I is an integer variable.

defined (2.5.4). For a *data object*, the property of having or being given a valid value.

defined assignment statement (7.5.1.3). An *assignment statement* that is not an *intrinsic* assignment statement and is defined by a *subroutine subprogram* which has an *explicit interface*.

defined operation (7.1.3). An *operation* that is not an *intrinsic* operation and is defined by a *function subprogram* which has an *explicit interface*.

deleted feature (1.6). A feature in Fortran 77 that is considered to have been redundant and largely unused. No features in Fortran 77 have been deleted from the standard. Note that a feature designated as an *obsolescent feature* in the standard may become a deleted feature in the next revision.

deprecated feature (1.6). A feature in Fortran 77 that is considered to have become redundant by the inclusion of certain new features in the standard. Such a feature may become an *obsolescent feature* as the new features become widely used.

derived type (2.4.1.2). A *type* whose *data* have *components* each of which is either of intrinsic type or of another derived type.

designator. See *subobject designator.*

dummy argument (12.5.2.2, 12.5.2.3, 12.5.4). An entity whose *name* appears in the parenthesized list following the *procedure* name in a FUNCTION *statement*, SUBROUTINE statement, or *statement function* statement.

dummy procedure (12.1.2.3). A *dummy argument* that is specified or *referenced* as a *procedure.*

dummy array. A *dummy argument* that is an *array.*

effective (6.2.1.2). Adjective applied to the *bounds*, *extents*, *size*, or *shape* of an *array*. For a *named* array with the RANGE *attribute*, it implies qualification to the subarray determined by the most recently executed SET RANGE *statement*. For other arrays, no qualification is implied.

elemental (12.4.3, 12.4.5). An adjective applied to an *operation*, *function*, or *assignment statement* that is applied independently to the elements of an *array* or corresponding elements of a set of *conformable* arrays and *scalars.*

entity (2.1). The term used for any of the following: a *program unit*, a *procedure*, an *operator*, an *interface block*, a *common block*, an *external unit*, a *statement function*, a *type*, a *named variable*, an *expression*, a *component* of a type, a *named constant*, a *statement label*, a *construct*, an *exponent letter*, a namelist group, or a range list.

executable construct (2.1). A CASE, DO, IF, or WHERE *construct* or an *action statement* (R222).

executable program (2.2.2). A set of *program units* that includes exactly one *main program.*

executable statement (2.3.1). An instruction to perform or control one or more computational actions.

explicit interface (12.3.1). For a *procedure referenced* in a *scoping unit*, the property of being an *internal procedure*, a *module procedure*, an *intrinsic procedure*, an *external procedure* that has an *interface block*, or a *dummy procedure* that has an interface block.

explicit-shape array (5.1.2.4.1). A *named array* that is declared with *explicit bounds.*

exponent letter (4.3.1.2). A letter that is used in the representation of a real or complex *literal constant* to indicate its *type parameter values.*

expression (7.1). A sequence of *operands*, *operators*, and parentheses (R712). It may be a *variable*, a *constant*, a *function reference*, or may represent a computation.

extent (2.4.7). The size of one dimension of an *array.*

external file (9.2.1). A sequence of *records* that exists in a medium external to the *executable program.*

external procedure (2.2.4.1). A *procedure* that is defined by an *external subprogram* or by means other than Fortran.

external subprogram (2.2). A *subprogram* that is not contained in a *main program*, *module*, or another subprogram.

external unit (9.3). A mechanism that is used to refer to an *external file.* It is identified by a nonnegative integer.

file (9.2). An *internal file* or an *external file.*

function (2.2.4). A *procedure* that is invoked in an *expression.*

function result (12.5.2.2). The *data object* that returns the value of a *function.*

function subprogram (12.5.2.2). A sequence of *statements* from a FUNCTION statement that is not in an *interface block* to the corresponding END statement.

global entity (14). An *entity* identified by a *lexical token* whose *scope* is an *executable program.* It may be a *program unit*, a *common block*, or an *external procedure.*

host (2.2.4.3). A *main program* or *subprogram* that contains an *internal procedure* is called the host of the internal procedure. A *module* that contains a *module procedure* is called the host of the module procedure.

host association (11.2.2). The process by which an *internal subprogram*, *module subprogram*, or *derived type* definition accesses *entities* of its *host.*

implicit interface (12.3.1). A *procedure referenced* in a *scoping unit* is said to have an implicit interface if the procedure is an *external procedure* that does not have an *interface block*, a *dummy procedure* that does not have an interface block, or a *statement function.*

instance of a subprogram (12.5.2.4). The copy of a *subprogram* that is created when a *procedure* defined by the subprogram is *invoked.*

interface block (12.3.2.1). A sequence of *statements* from a INTERFACE statement to the corresponding END INTERFACE statement.

interface of a procedure (12.3). See *procedure interface.*

internal file (9.2.2). A character *variable* that is used to transfer and convert *data* from internal storage to internal storage.

internal procedure (2.2.4.3). A *procedure* that is defined by an *internal subprogram.*

internal subprogram (2.2). A *subprogram* contained in a *main program* or another subprogram.

intrinsic (2.5.7). An adjective applied to *types*, *operations*, *assignment statements*, and *procedures* that are defined in the standard and may be used in any *scoping unit* without further definition or specification.

invoke (2.2.4).

(1) To call a *subroutine* by a CALL *statement* or by a *defined assignment statement.*

(2) To call a *function* by a *reference* to it by *name* or *operator* during the evaluation of an *expression.*

keyword (2.5.2). *Statement keyword* or *argument keyword.*

label. See *statement label.*

length of a character string (4.3.2.1). The number of *characters* in the *character string.*

lexical token (3.2). A sequence of one or more characters with an indivisible interpretation.

literal constant (2.4.4). A *constant* without a *name.*

local entity (14). An *entity* identified by a *lexical token* whose *scope* is a *scoping unit.*

main program (2.2.3, 11.1). A *program unit* that is not a *module*, *subprogram*, or *block data program unit.*

module (2.2.5, 11.3). A *program unit* that contains or accesses definitions to be accessed by other program units.

module procedure (2.2.4.2). A *procedure* that is defined by a *module subprogram.*

module subprogram (2.2). A *subprogram* that is contained in a *module* but is not an *internal subprogram.*

name (3.2.2). A *lexical token* consisting of a letter followed by up to 30 alphanumeric characters (letters, digits, and underscores).

name association (14.7.1). *Argument association, alias association, use association,* or *host association.*

named. Having a *name.*

named constant (2.4.4). A *constant* that has a *name.*

numeric storage unit (14.7.2.1). The unit of storage for holding a *datum* of *type* default real, integer, or logical.

object (2.4.3.1). *Data object.*

obsolescent feature (1.6). A feature in Fortran 77 that is considered to have been redundant but that is still in frequent use.

operand (2.5.8). An *expression* that precedes or succeeds an *operator.*

operation (7.1.2). A computation involving one or two *operands.*

operator (2.5.8). A *lexical token* that specifies an *operation.*

parent of an alias (6.2.6). The *data object* with which an *alias* appears in an IDENTIFY *statement.*

present (12.5.2.8). A *dummy argument* is **present** in an *instance of a subprogram* if it is *associated* with an *actual argument* and the actual argument is a dummy argument that is present in the invoking *procedure* or is not a dummy argument of the invoking procedure.

procedure (2.2.4, 12.1.2). A computation that may be *invoked* during program execution. It may be a *function* or a *subroutine.* It may be an *external procedure,* a *module procedure,* an *internal procedure,* a *dummy procedure,* or a *statement function.* A *subprogram* may define more than one procedure if it contains ENTRY *statements.*

procedure interface (12.3). The *characteristics* of a *procedure,* the *name* of the procedure, the name of each *dummy argument,* the *operator* (if any) by which it may be *referenced* (*functions* only), and whether it may be referenced by an *assignment statement* (*subroutines* only).

processor (1.2). The combination of a computing system and the mechanism by which *executable programs* are transformed for use on that computing system.

program. See *executable program* and *main program.*

program unit (2.2). The fundamental component of an *executable program.* A sequence of *statements* and comment lines. It may be a *main program,* a *module,* an *external subprogram,* or a *block data program unit.*

range (6.2.1.2). The set of elements in an *array* that lie within its *effective bounds.*

rank (2.4.7). The number of dimensions of an *array.* Zero for a *scalar.*

record (9.1). A sequence of values that is treated as a whole within a *file*.

reference (2.5.5). The appearance of a *data object name* or *subobject designator* in a context requiring the value at that point during execution, or the appearance of a *procedure* name, its *operator* symbol, or a *defined assignment statement* in a context requiring execution of the procedure at that point. Note that neither the act of defining a *variable* nor the appearance of the name of a procedure as an *actual argument* is regarded as a reference.

scalar (2.4.6).

(1) A single *datum* that is not an *array*.

(2) Not having the property of being an *array*.

scope (14). That part of an *executable program* within which a *lexical token* has a single interpretation. It may be an *executable program*, a *scoping unit*, a single *statement*, or a part of a statement.

scoping unit (2.2.1). One of the following:

(1) A *derived type* definition,

(2) An *interface block*, excluding any interface blocks contained within it, or

(3) A *program unit* or *subprogram*, excluding *derived type* definitions, *interface blocks*, and subprograms contained within it.

section subscript (6.2.4). A *subscript* or *subscript triplet* in an *array section selector*.

selector. A syntactic mechanism for designating

(1) Part of a *data object*. It may designate a *substring*, an *array element*, an *array section*, or a *structure component*.

(2) The set of values for which a CASE *block* is executed.

sequence array (12.4.1.4). An *array* without the RANGE *attribute* that is an *allocatable array*, an *assumed-size array*, or an *explicit-shape array* that is either a *dummy array* associated with a sequence array or is not a *dummy argument*. Note that all Fortran 77 arrays are sequence arrays.

shape (2.4.7). For an *array*, the *rank* and *extents*. The shape may be represented by the rank-one array whose elements are the extents in each dimension.

size (2.4.7). For an *array*, the total number of elements.

standard module (1.7). A *module* standardized as a separate collateral standard.

statement (3.3). A sequence of *lexical tokens*. It usually consists of a single line, but the ampersand symbol may be used to continue a statement from one line to another and the semicolon symbol may be used to separate statements within a line.

statement entity (14). An *entity* identified by a *lexical token* whose *scope* is a single *statement* or part of a statement.

statement function (12.5.4). A *procedure* specified by a single *statement* that is similar in form to an *assignment statement*.

statement keyword (2.5.2). A word that is part of the syntax of a *statement* and that may be used to identify the statement.

statement label (3.2.5). A *lexical token* consisting of up to five digits that precedes a *statement* and may be used to refer to the statement.

storage association (14.7.2.2). The relationship between two *storage sequences* if a storage unit of one is the same as a storage unit of the other.

storage sequence (14.7.2.1). A sequence of contiguous *storage units*.

storage unit (14.7.2.1). A *character storage unit* or a *numeric storage unit*.

stride (6.2.4.4). The increment specified in a *subscript triplet*.

structure (4.2). A *data object* of *derived type*.

structure component (6.1.2). The part of a *structure* corresponding to a *component* of its *type*.

subobject (2.4.3.2). A portion of a *named data object* that may be *referenced* or *defined* independently of other portions. It may be an *array element*, an *array section*, a *structure component*, or a *substring*.

subobject designator (2.5.1). A *name*, followed by one or more *component selectors*, *array section* selectors, *array element* selectors, and *substring* selectors.

subprogram (2.2). A *function subprogram* or a *subroutine subprogram*. *Modules* and *block data program units* are not subprograms.

subroutine (2.2.4). A *procedure* that is *invoked* by a CALL *statement* or by a *defined assignment statement*.

subroutine subprogram (12.5.2.3). A sequence of *statements* beginning with a SUBROUTINE statement that is not in an *interface block* to the corresponding END statement.

subscript (6.2.4). One of the list of *scalar* integer *expressions* in an *array element selector*. Note that in Fortran 77, the whole list was called the subscript.

subscript triplet (6.2.4). An item in the list of an *array section selector* that contains a colon and specifies a regular sequence of integer values.

substring (6.1.1). A contiguous portion of a *scalar character string*. Note that an *array section* can include a *substring selector*; the result is called an array section and not a substring.

type (4) *Data type*.

type declaration statement (5). An INTEGER, REAL, DOUBLE PRECISION, COMPLEX, CHARACTER, LOGICAL, or TYPE(*type–name*) *statement*.

type parameter (2.4.1). A parameter of a parameterized *data type*.

type parameter values (1.5.3, 2.4.1). The values of the *type parameters* of a *data entity* of parameterized *data type*.

undefined (2.5.4). For a *data object*, the property of not having a determinate value.

use association (14.7.1.2). The association of *names* in different *scoping units* specified by a USE *statement*.

value attribute (5.1.2.1). Whether a *data object* is a *constant* or a *variable* and whether it has a *defined* initial value.

variable (2.4.5). A *data object* whose value can be *defined* and redefined during the execution of an *executable program*. It may be a *named* data object, an *array element*, an *array section*, a *structure component*, or a *substring*. Note that in Fortran 77, a variable was always *scalar* and named.

whole array (6.2.1). A *named array*.

106(*)JKR-5

51

To Fortran Forum, BCS, NAG,...

From: John Reid

Subject: X3J3 meeting in Liverpool

Date: 19 August 1987

Note: This is a personal report of the meeting and in no sense does it constitute an official record of it.

References:

[1] ISO/TC97/SC22/WG5 N254. Liverpool resolutions.

[2] ISO DTR 9547. Test methods for programming language processors - guidelines for their development and acceptability (Type 3).

[3] 105(*)CDB-3. Some pesudo-random thoughts on a pointer facility.

[4] 96(*)JKR-6. Names, alias pointers.

[5] 105(16)CDB-5. MODULE approach to Kanji support.

[6] 105(16) CDB-4. Unbuffered bit I/O.

[7] 105(16) JHM-01. Derived Type I/O.

[8] 105.JLW-4. Module for variable-length strings.

[9] 105(*)RCA/CDB-1. Remove the EXPONENT LETTER statement.

1. Summary

Recent meetings have been concentrating on getting the draft standard into a acceptable state, but this one was quite different. It considered a fair number of technical issues that were either raised by the ISO Working Group meeting in the previous week or which are thought likely to be raised in public comments. The most time was spent considering how multi-byte character sets can be accommodated, and the Japanese observers were mandated to make a detailed proposal. In addition, Lawrie Schonfelder was asked to make a detailed pointer proposal, based on relaxations of ALLOCATABLE and ALIAS.

3KJ

An audit of Fortran 77 to check that it is all included was begun and the glossary was reviewed at the request of another ANSI committee, X3K5.

2. Public comments

SPARC (Standards Planning and Requirements Committee) reviewed the draft standard at its July meeting and found that it complies with its authorizing document. It has therefore been submitted for a 30-day letter ballot of X3, closing on August 20. It seems likely that the 4-month comment period will be October-January. The document will be available from Global Engineering Documents, Inc. by calling (714) 540-9870 at a price that will probably exceed $75. Private reproduction of S8 is allowed within members' organisations.

Three comments have already been received by X3 and passed on to X3J3: Convex and Boeing both argued against public review of the present draft, while IFIP WG2.5 asked for the draft to be sent out as soon as possible.

3. Meeting of ISO/TC97/SC22/WG5

The ISO Fortran Working Group met at Liverpool in the previous week and its opinions are summarized by its formal resolutions [1]. The voting figures are both by individuals and by countries.

The working group confirmed the action of its convenor in forwarding S8 for processing as a draft proposed standard (28-7-0; 7-2-0). The countries voting NO were Japan and France. It is hoped that ISO and ANSI processes will be in step.

The following resolutions amount to comments for X3J3:

R7: The final standard should not contain an appendix of extensions.

R8: Pointers should be added.

R9: If pointers are adopted, they should be integrated into IDENTIFY or IDENTIFY should be dropped.

R11: Deprecated features should be identified in the text.

R12: Significant blanks should be reconsidered.

R14: The index should be improved and more examples added.

R15: References to the notes should be added to the text.

R16: Each successive draft should indicate what changes are made.

R17: X3J3 should look into the possibility of requiring a report on any size or complexity violations by a program.

R18: Add examples to clarify the use of interfaces.

R19: Add a facility for very large character sets such as those of Chinese or Kanji.

R21: WG5 is concerned about the use of square brackets, which are reserved for national use in ISO 646, in the syntax.

R22: Restore BIT data type.

R23: There are some detailed problems re passed-on precision.

R24: Delete RANGE (votes of 12-10-12, 4-3-2).

4. Responses to WG5

On Resolution 7, X3J3 favoured eliminating Appendix F from the final standard (17-4-11) and replacing it by some kind of discussion document (24-6-6).

On Resolution 12, it was decided (16-7-4) to wait until the February meeting before reconsidering significant blanks, so that the public comments can be gauged.

On Resolution 22, Brian Smith and Jerry Wagener considered the possibility of parameterizing LOGICAL to provide a facility for bits. Though it was agreed that this would provide virtually identical facilities to the BIT data type in Appendix F, it was not favoured by those strongly wanting bits and the committee as a whole was undecided (6-7-4) on whether such a proposal should be prepared for the next meeting.

Resolutions 8,9 and 19 were considered in more detail (see Sections 6 and 7).

5. Test methods for programme language processors

A draft ISO technical report [2] on test methods for language processors was considered by the committee and did not provoke any detailed discussion.

6. Pointers

Carl Burch [3] suggested the provision of pointers through allowing recursive data structures, permitting objects to have both the ALIAS and ALLOCTABLE attributes, allowing an ALIAS to move from host to host, and extending ALLOCATE to scalars. In fact, this is very similar to a proposal [4] that I placed before the committee about two years ago. This approach has the merit of requiring comparatively little change to the draft standard. The committee wanted to see a complete and detailed proposal at the next meeting (23-1-9).

7. Multi-byte character sets

There was a very long discussion on how to accommodate a request from Japan for facilities that handle character sets containing many thousands of different characters. Carl Burch [5] suggested a module containing a type NCHAR(LENGTH) that contains LENGTH 2-byte characters, preceded and succeeded by 3-byte escape sequences. The escape sequences allow CHARACTER and NCHAR data to be mixed freely on output, using A format. However, there are problems over specifying field widths in formats, defining constants, and assigning to substrings. These led the committee to reject the approach (1-27-5). I was the one person to vote in favour, believing that there was room for detailed improvement and that we should look to enhancing the module facility. It was decided that the best approach is to parameterize CHARACTER, and the Japanese were asked to provide a detailed proposal for the next meeting.

8. Unbuffered I/O

There was a discussion of unbuffered I/O, along the lines of CDC's BUFFER IN and BUFFER OUT, but without the asynchronous aspect, see [6]. A straw vote (12-7-13) was mildly in favour of the subgroup pursuing this.

9. Derived Type I/O

Jim Matheny [7] asked the committee to consider whether further facilities for derived type I/O were needed. Currently, a structure is simply treated as a list of its ultimate components. More general output is available by placing a character function in the output list, but nothing comparable is available for input. None of the ideas discussed seemed to be entirely satisfactory, and the committee decided against having a proposal for the next meeting (2-11-7).

10. Glossary

John Wood, chairman of X3K5, wishes to include a Fortran glossary in the proposed American National Standard Dictionary for Information Systems (ANSDIS) and has been looking at the Fortran 8x glossary of May 1986. He would really like a glossary that covers both Fortran 77 and Fortran 8x, but the editorial subgroup felt that working on a Fortran 77 glossary ten years late was inappropriate. It therefore decided instead to take advantage of Wood's comments to improve the Fortran 8x glossary and include notes on terms whose meaning was different in Fortran 77. A revised glossary will be brought to the next meeting.

Kurt Hirchert was appointed as Vocabulary Representative and will liaise with John Wood.

11. Fortran 77 audit

All the subgroups began an audit of Fortran 77 to check that it has all been included.

12. Editorial work

The editorial subgroup considered some suggested editorial changes, most of which were approved by the Committee. A list of such approved changes will be maintained.

The subgroup also reviewed which parts of the document were in serious need of rewriting. The final list consisted of source form, IDENTIFY, DO constructs, and interface blocks.

13. Module for variable-length strings

Jerry Wagener [8] considered building a module for variable-length strings based on a type containing an integer and a character variable of fixed length. The integer holds the current length and the character variable holds the string. The most awkward aspect is I/O, once again illustrating that better facilities for derived-type I/O are needed.

14. EXPONENT LETTER statement

The EXPONENT LETTER statement has been criticized as being a clumsy solution to a minor problem. Bob Allison and Carl Burch [9] therefore considered two possibilities for its deletion, illustrated by the examples

```
REAL(PRECISION=10)::A,B,C,'L'
```

and

```
REAL(PRECISION=10,EXPONENT LETTER='L')A,B,C
```

both of which introduce undesirable irregularities. A straw vote (6-8-4) indicated that the committee did not feel that the current syntax presents a problem.

15. Next meeting of X3J3

The next meeting of X3J3 will be in Fort Lauderdale, November 9-13. The premeeting distribution deadline is October 5.

P.S. For those who wish to see an informal description of Fortran 8x. Mike Metcalf and I have written a book called "Fortran 8x Explained", which will be available in November, published by OUP.

To: X3J3
From: Larry Rolison
Subject: Zuccotto (Mary Morgan's Terrific Chocolate Dessert Recipe)
Date: 24 August 1987

I was charged by popular acclaim to include this recipe in the next premeeting distribution, so here it is, thanks to Mary Morgan (Steve Morgan's wife - Steve is Lawrie's alternate). However, there is some good news and some bad news. The good news is that a lot of us got to sample this dessert at Lawrie's house the week of the X3J3 meeting and can testify to its addictiveness. The bad news is that I have it on good authority (i.e., a female British subject now living in the US) that the recipe will do us US residents no good because all measures and most ingredients are different. That is, a British teaspoon is not a US teaspoon, British flour is not US flour, treacle is basically unobtainable in the US (what on earth IS it anyway?), etc. If anyone on the committee is a secret chef, or knows a capable chef, and can adapt the recipe to US measures and ingredients, Margie, my wife, would be eternally grateful. I assume I speak for several other committee members that also had more than one piece of this dessert!

Part I: The Oil Chocolate Sponge Cake Shell

6 oz flour
2 tablespoons cocoa powder
1 teaspoon baking powder
1 teaspoon bicarbonate of soda
5 oz sugar

Mix above ingredients together then add:
2 tablespoons treacle
2 eggs
5 fluid oz sunflower oil
5 fluid oz milk

Pour into two greased 8 inch sponge tins. Bake at 325F (180C) for about 45 minutes.

Mary added the following comments:

- She uses this sponge cake recipe because it produces a very sticky sort of sponge that blends together around the pudding. Drier sponges tend to stay in layers.
- The two 8" sponge tins aren't important. You just need a pile of thin slices of sponge.
- The cake gets better the older it gets!

Part II: The Filling

1 pint double cream (2 1/2 cups heavy cream) beaten until stiff
1 oz (1/4 cup) icing (confectioner's) sugar
2 oz (1/2 cup) toasted hazelnuts
8 oz fresh cherries ("stoned"; I hope this translates to "pitted")
[or tinned black cherries may used]
4 oz dark dessert chocolate, grated
2 fluid oz (1/4 cup) brandy and 2 fluid oz orange liqueur (or whatever liqueurs might sound interesting to you)
2 8" chocolate sponge cakes sliced in two (Part I)

1. Combine cream, sugar, nuts, cherries, and chocolate; chill.
2. Line a 3 pint (6 cup) pudding basin with 3/4 of the sponge; damp with the liqueur.
3. Fill the sponge case with the cream mixture, finishing with the remaining sponge and any remaining liqueur.
4. Chill for a couple of hours, or it will freeze.
5. Ease away from the sides of the basin and turn out.

(53)

106(*)LRR-3

To: X3J3
From: Larry Rolison
Subject: YAPP (Yet Another Pointer Proposal)
Date: 22 September 1987

Even though the committee has shown its preference via a straw vote for Carl's method of incorporating (the effect of) pointers into the language and even though I'm a member of the subgroup charged with further development of his ideas, I find I must try to present an alternative method. I have repeatedly said that I contend the general FORTRAN [sic] population wants basically nothing more than an address to play with and a simple dynamic storage scheme to accompany pointers but have been guilty of complaining without offering a solution. It is "put up or shut up" time so here goes. . .

My main goal is to present a model that is simple and straightforward. It has been said many times that simplicity is (now, maybe "was") one of the hallmarks of FORTRAN and likely one of the main reasons for its success. I think that the concept of pointers is a fairly simple idea. They've been around for 25 years and anyone other than a novice programmer has a pretty good feel for what they are and how they're used. Such an alleged simple idea, then, should be expressed in a language in simple syntax and semantics. I see no point in spending hours trying to pervert the contents of Fortran 8x (or any other language) to avoid a couple more keywords. In this proposal, I'll use what already exists in 8x if it seems natural and invent new constructs otherwise.

So, OK, let's get to specifics. The first thing needed is a pointer data type. Dabbling in recursive data structures and considering the semantics of combining ALIAS and ALLOCATABLE may be an interesting intellectual exercise for some of us but c'mon, the great unwashed are crying for pointers so let's just give them to them! Let's just bite the bullet right off and call a pointer a pointer!

If we're going to invent a new data type, then the first thing to do is turn to Section 4 to find out what a data type is. To quote the Book: "A data type has (1) a name, (2) a set of valid values, (3) a means to denote such values (constants), and (4) a set of operations to manipulate the values." Let's take these one at a time.

(1) a name

This one's easy. The type specifier for the pointer type is the keyword POINTER.

(2) a set of valid values

The set of values for the pointer type is a subset of all addresses capable of being generated by the processor.

This is the point where the strong/weak typing camps part. The strong typing contingent would insist that the valid values for a particular pointer are only those addresses for objects of the type to which the pointer is bound. The weak typing contingent wants a pointer to be able to contain any address the processor would otherwise be capable of computing. While I'm personally more inclined to the laissez faire brand pointer, I see no reason why a language (in particular, Fortran) should not be able to accommodate both preferences. To do so, a syntax must exist that allows one to simply declare a variable to be a pointer and a syntax must exist to bind a pointer to something. For now, assume that "something" is defined to be "user-defined type". I propose the following type-spec addition to R502:

or POINTER [(type-name)]

For the weak typing camp, to declare a pointer that may point at a (almost) arbitrary data object, you simply write:

```
POINTER arb_ptr
```

I refer to this style of pointer in this proposal as being unbound.

For the strong typing camp, your declaration would be a two-part form:

```
TYPE fruit
  INTEGER color
  INTEGER size
END TYPE
.
.
POINTER(fruit) :: fruit_ptr
```

The declaration of the pointer FRUIT_PTR states that it may only be used to locate data objects of type FRUIT. The pointer is defined to be bound to the type.

I want two points to stand out here:

* Pointers are not inexorably linked to "allocatableness".

  No declaration for a data object of type FRUIT is shown in the above example. The omission was by intent. A bound pointer declaration need not know anything about a data object that the pointer may be used to locate and, in particular, the object need not be in dynamic storage (i.e., it need not be allocatable). I am purposely separating pointers from "allocatableness" because I maintain they are two separate topics. They may interact with each other but they do not depend on each other.

* Pointers may only point at data objects and subobjects.

  Let's dispense with notions like a pointer may point at a procedure right off the bat. A pointer is a data address pure and simple. Enough said.

(3) a means to denote such values (constants)

This one is easy but we can probably argue for hours on the spelling. I suggest NULLPTR. PL/I uses NULL, Pascal uses nil, etc. I suggest NULLPTR because it mnemonically contains both ideas of what it is: a null value and a null value of type pointer. If a future committee wants to use the keyword NULL for some general purpose, it will be available to them.

I imagine some folks will shudder at the prospect of a malicious programmer pulling a stunt like:

```
REAL     nullptr
POINTER  abe
.
.
nullptr = 3.22
abe = nullptr
```

but let me assure them that Fortran is already context sensitive. A compiler can indeed determine that the real value 3.22 is not being propagated to the pointer ABE. This is analogous to declaring an intrinsic function to be a type other than is language-defined type: it is not sufficient to remove it intrinsic and generic properties. As I describe the allowable contexts for pointers later in this paper, I think it will become obvious that a compiler can always disambiguate the syntax.

The null constant is the same for both bound and unbound pointers. There is no need to distinguish between a null-valued unbound pointer and a null-valued bound pointer.

(4) a set of operations to manipulate the values.

I want to make several points in this section.

* The only operations permitted on a pointer are the tests for equality and inequality.

  A pointer may not (let me repeat that: may NOT) be an operand of an arithmetic expression. We have argued this before ad nauseum. Many people decry pointers as being inherently nonportable but it mostly comes down to the fact that they are nonportable when one performs arithmetic on them to cleverly step through a character string or some such nonsense.

  A processor must be capable of generating an address for every valid data object and subobject. Since it can do so, it follows that a pointer value can be constructed for every object and subobject (and, yes, it may require the pointer to be a software simulated pointer but it is a pointer nonetheless). Thus,

  (1) since all Fortran data objects are portable,

  (2) since a processor must be capable of addressing all objects and subobjects, and

  (3) since a pointer is an address,

  the pointer facility is portable. Note that the value of the pointer is not portable but we don't care about that. The code is portable.

* Both pointer operands of a comparison operator must be bound to the same type if at least one of them is bound to a type. An unbound pointer may only be compared to another unbound pointer. The constant NULLPTR may be compared to either a bound or unbound pointer.

  These restrictive rules are a safety net for the strong typing camp. A compiler writer could certainly relax the rules to allow the comparisons I prohibit but I believe it would defeat the purpose of bound pointers.

* A pointer bound to a type may be assigned only to a pointer bound to the same type. An unbound pointer may only be assigned to another unbound pointer. The constant NULLPTR may be assigned to either a bound or unbound pointer.

  As stated above, this is a safety net to those who want to exercise control over the use of pointers.

The following points are not strictly part of the definition of a data type but are needed to clarify my definition of the pointer data type.

* An object of type pointer may not be an I/O list item nor may it be a subobject of an I/O list item.

  In case I have inadvertently left any loopholes, my intent is to bar I/O of pointers completely. This restriction will likely be the proverbial straw that broke the camel's back for diehard bit-twiddlers (after already having had arithmetic operations on pointers taken away from them) but we really must close the door on this issue to maintain portability of a pointer facility.

  The complaint may arise that someone wishes to read a record from a file into a structure and link the structure into a linked list. They want the file record and the linked list structure to have the same declaration. Too bad. The programmer will simply have to declare the file record to be a substructure of the linked list node (the linked list node contains the additional pointer member). The file record can then be written from or read into the substructure.

* An object of type pointer has no storage sequence.

  This is to prevent access to a pointer value via EQUIVALENCE.

* If a pointer is passed as an argument, the interface must be explicit.

  Since pointers do not exist in FORTRAN 77, I think we have to say this, don't we? At any rate, this is an attempt to prevent the passing of a pointer to, say, an integer and thereby opening up a world of wonderful things that could be done with (to) it.

  Argument matching for pointers follows the same rules as for assignment.

I hope it is becoming clear by now that my goal is to prevent access to a pointer in all cases except where it is used as a pointer. I believe it is the only hope we have of producing a portable pointer facility.

OK, I've defined a pointer data type. What can you do with it? The most obvious is, of course, to define a variable that may be used to locate another data object. The symbol I've chosen for this "operation" is "->". The symbol exists in other languages for the same purpose (so it will already be familiar to some programmers), it's intuitive, mnemonic, and (unlike symbols like the "up arrow" that Pascal uses) the individual characters are both commonly found on keyboards.

I impose the syntax that when a pointer variable is used to locate a data object, the pointer variable, the pointer operator, and the subject data object must ALWAYS appear. I require all three to reduce the confusion that has plagued those that apparently first encountered pointers while learning a language like Pascal. My intent is to prevent people from becoming confused over whether it is the pointer that is being referenced or the object to which it points. Pascal-styled syntax has never made sense to me (having come from a PL/I background and a vendor systems language that has PL/I-like pointers). For example, in the simple assignment statement

```
P^ = Q^
```

(see what I mean about character availability?) there is no indication in the syntax of the assignment statement itself what data is being moved. To me, the important thing to know when reading such a line is what data is being moved, not its locator. Thus, in a statement like

```
P->TARGET = Q->SOURCE
```

it is obvious in the syntax itself what data is being moved where. I am a firm believer in syntax being obvious in sympathy for those that need to maintain software written by others.

This restriction is also useful in making it obvious to a reader that

```
P = Q
```

is a pointer assignment where

```
P->TARGET = Q->SOURCE
```

is a data movement using pointers.

Naturally, since the pointer symbol may be thought of as an operator, one may freely stack them. For example,

```
ME->HAND%INDEX_FINGER->YOU%FACE
```

is valid syntax.

Next, we need to determine just what a pointer can point at. Recall

that I said that the subject of a pointer need not be in dynamic (allocatable) storage. That statement should have sparked two questions:

(1) If the pointer's subject is not in dynamic storage, how do you obtain a value for the pointer?

(2) If the pointer's subject is in dynamic storage, how do you create (and delete) the storage and how do you obtain a value for the pointer?

Let's take them in order. To generate an address for an object that is not in dynamic storage, I propose the intrinsic function LOC. Again, we can argue about the spelling for hours. (My heart lies with ADDR but LOC already exists in a couple of existing implementations so in the spirit of standardizing common practice....) I think there is no question on the requirement for an intrinsic function that returns an address. Of course, to make it complete, we need the usual rules that the data object argument must be defined, etc. I propose that the function accept arguments of any storage type. I mean to allow dynamic storage arguments as well as static and automatic.

Usage of the LOC function result is required to follow the same rules as for pointer variables. That is, if the result is assigned to a bound pointer, the argument to LOC must be an object of the type to which the assignment target is bound. However, I propose to allow a window of vulnerability here that may annoy the strong typing camp: the LOC of any data object may be assigned to an unbound pointer without regard to the binding of the argument object's type. That is, if the object is of type X and there is at least one pointer that is bound to type X, the LOC of the object may still be assigned to an unbound pointer. This point caused some confusion among the reviewers in my department so here's an example of what I mean:

```
TYPE fruit
  INTEGER  color
  INTEGER  size
END TYPE

TYPE(fruit)      :: orange

POINTER(fruit)   :: fruit_ptr, citrus_ptr
POINTER          :: apple

apple = LOC(orange)
```

I believe that this "out" is necessary occasionally and that prohibitions of its use should be legislated by the individual programming shop, not the language.

We can argue about whether the syntax

```
LOC(object)->another_object
```

should be permitted or not. I think it is ugly, largely unneeded, and should be prohibited.

Let's move on to dynamic storage. A major use of pointers is in manipulating linked lists. To create such a list, one generally makes use of a dynamic storage scheme to create (and possibly delete) storage for nodes in the list. Fortran 8x already has an ALLOCATABLE attribute and ALLOCATE/DEALLOCATE statements so I propose we generalize them. I can already hear the lamentations of those that think allocatable arrays are on their deathbed, but 'tis not true. I'm going to go way out on a limb here and make the claim that not only do allocatable arrays and pointers live quite well together but there are good reasons for allocatable arrays to remain in the language exactly as they are today.

I propose that the ALLOCATABLE attribute be used for any data object declaration to mean that the declaration is only providing a template for laying out the data object in storage and that storage will be allocated at execution time. I propose that the current ALLOCATE and DEALLOCATE statements become special ("shorthand") cases of my

generalized statement form. To whit:

```
R610 allocate-stmt    is ALLOCATE(allocatable-object-name [ ]
                         [ ] [, STAT=stat-variable]) SET(pointer)
                      or ALLOCATE(array-allocation-list [ ]
                         [ ] [, STAT=stat-variable])

R613 deallocate-stmt  is  DEALLOCATE [ ]
                          [ ] (pointer->allocatable-object-name [ ]
                          [ ] [, STAT=stat-variable])
                      or  DEALLOCATE(array-name-list [ ]
                          [ ] [, STAT=stat-variable])
```

My choice of the metaterm allocatable-object-name may not be 100% correct but take it as what it says for the purposes of this paper. The metaterm "pointer" in the SET option and in the DEALLOCATE statement is defined to mean simple scalar, array element, or a structure member that is a simple scalar or an array element (have I missed any scalar cases?). It must be definable at the time the ALLOCATE statement is executed and defined at the time the DEALLOCATE statement is executed.

First, I'll describe my generalized forms and how one uses them then I'll (attempt to) justify why the current scheme should be retained.

In my generalized scheme, basically the same rules exist for declaring and allocating an arbitrary object as for the current allocatable array. For example, only a data object, not a subobject, may have the ALLOCATABLE attribute (and thus may be allocated). Of course, there are some modifications. Let me illustrate by example. Suppose we want to build a simple linked list of structures and we want the structures to contain data structures that vary in size. Using the current derived-type declaration and my generalized ALLOCATE:

```
TYPE person(name_len, num_children)
  CHARACTER(len=name_len) :: last_name
  CHARACTER(len=name_len),                &
   ARRAY(num_children)    :: child_name
  POINTER(person)         :: next
END TYPE

TYPE(person), ALLOCATABLE :: employee
POINTER(person)           :: new, delete

ALLOCATE(employee(40,num_of_kids)) SET(new)
```

As with the current allocatable array scheme, the values of the type parameters need to be known at the time the ALLOCATE statement is executed.

To delete a node in the linked list, one would first unlink it (code is left to the reader) then free the space:

```
DEALLOCATE(delete->employee)
```

I suppose there could be an argument made to allow a list of names in each statement but I don't think such a list is necessary so I didn't include it in the syntax definition. I don't think groups of things are typically allocated at one time (at least when building linked lists).

The example makes use of a structure. I expect that typically structures will be the main interest of use with pointers but I see no reason why we should prohibit simple scalars from being allocated (I'll get to arrays in a bit). I would expect such simple scalars to be generally used with unbound pointers. Remember that I have imposed the restriction that a bound pointer must be bound to a derived type. So if a user really wanted just to allocate a simple scalar they probabl wouldn't be in the mood to generate a type consisting of just a sing scalar. Anyway, I propose that we allow allocation of scalars such as i the following code fragment:

```
CHARACTER(len=132), ALLOCATABLE  :: line
POINTER                          :: line_ptr

ALLOCATE(line) SET(line_ptr)
```

If the user really wanted to use a bound pointer, the example would become:

```
TYPE chars
  CHARACTER(len=132) :: line_chars
END TYPE
TYPE(chars), ALLOCATABLE  :: line
POINTER(chars)            :: line_ptr

ALLOCATE(line) SET(line_ptr)
```

As with the LOC function, if the pointer in the SET option is bound, the object being allocated must be of the type to which the pointer is bound. If the pointer in the SET option is unbound, the object being allocated may be any object declared to be ALLOCATABLE.

It is conceivable that a user is only interested in a single instance of a data object in dynamic storage as a method of managing temporary data objects. This is my segue into why I think the current form of allocatable arrays should be retained. I believe that the reason allocatable arrays were invented was to manage temporary storage, and in particular large amounts of temporary storage, without having to call a subprogram to create automatic storage. The current scheme works because such temporary storage management problems only need a single instance of the storage, manipulate it, then discard it. They don't need a collection (list) of such instances to exist simultaneously. If the need was sufficiently strong to have caused the invention of allocatable arrays, then the need must still be in existence (even with pointers). If a single instance is all the user requires, then why force them to use a pointer artificially (by eliminating the current ALLOCATE form)? In fact, I think that we should now generalize array-allocation-list in the ALLOCATE BNF to include any allocatable data object. This would provide the power of having the processor manage the pointer if a user only needed one instance of an arbitrary data object. Consider the LINE example above. If a user only needed a single instance of LINE and didn't want to bother with a pointer, why not allow:

```
CHARACTER(len=132), ALLOCATABLE :: line

ALLOCATE(line)
```

This generalized form is, I think, a useful expansion of the current allocatable array scheme and provides a nifty shorthand to my generalized ALLOCATE. Of course, the user may not mix methods; that is, if the object is allocated with a SET option, it must always be accessed using a pointer.

For the user that wants to play with dynamic arrays (and now any other allocatable object), they would have three choices: the current single instance scheme, accessing the array (or other object) with an unbound pointer, or accessing it with a bound pointer.

Having put forth my pointer proposal and having proposed that the current scheme of allocatable arrays be retained in the language (and in fact enhancing the scheme), there is one last item remaining to be addressed (pun intended): IDENTIFY. I think we might all agree that by and large pointers obviate the need for the scalar IDENTIFY. The only thing that the scalar IDENTIFY gains us is locking the alias onto a single host but even the word "single" has taken on, to my way of thinking, a twisted connotation. I'm willing to abandon the scalar IDENTIFY but, on the other hand, keeping it in the language doesn't upset me. Maybe changing an object's name has some usefulness.

However, I do feel fairly strongly that the original array IDENTIFY

should remain in the language. I think that arrays of pointers ar conceptually too difficult to manipulate (people just don't think that way). I like the ability of the array IDENTIFY to compute skewed sections, for example. I think it is sufficiently powerful and general enough to keep in the language and will live quite well with (my proposed) pointers.

This has certainly been a lot of material to digest in one proposal, so let me summarize:

* A new data type is proposed. It is called POINTER.
* The declaration form is POINTER [(type-name)]. If type-name is absent, the pointer being declared is an unbound pointer (and may point at any data object or subobject). If type-name is present, type-name must be the name of a derived type. The pointer being declared is bound to the type type-name (and may only point at objects of type type-name).
* The null pointer value is denoted by the language-defined constant NULLPTR.
* The symbol that represents the pointer "operation" is "->". Pointer operations may be stacked. If an object is accessed via a pointer, the pointer variable name, the pointer symbol, and the name of the subject of the pointer must appear.
* The only operations permitted on pointer values are tests for equality and inequality. A bound pointer may only be compared to a pointer bound to the same type. An unbound pointer may only be compared to another unbound pointer. NULLPTR may be compared to either kind of pointer.
* A bound pointer may only be assigned to a pointer bound to the same type. An unbound pointer may only be assigned to another unbound pointer. NULLPTR may be assigned to either kind of pointer.
* A pointer may not be an I/O list item, directly or indirectly.
* If a pointer is passed as an argument, the interface must be explicit. The argument matching rules are the same as for assignment.
* An object of type pointer has no storage sequence.
* A LOC intrinsic function exists to compute the address of a data object. The result of the function must obey the same rules as for pointer variables. However, the result of the function may be assigned to and used in the context of an unbound pointer.
* The ALLOCATABLE attribute is extended to apply to any data object (but not subobjects).
* The ALLOCATE statement is extended to set a pointer variable and to allocate space for any object declared to be allocatable. If no pointer is provided, the rules remain the same as they are now but are extended to allocatable objects other than arrays.
* The DEALLOCATE statement is extended to free dynamic space according to a pointer and its subject. If no pointer is provided, the rules remain the same as they are now but are extended to allocatable objects other than arrays.
* The scalar IDENTIFY seems to have lessened usefulness given this implementation of pointers and could be abandoned. The array IDENTIFY still has sufficient usefulness to remain in the language.

I have tried to cover the major points of a pointer data type; I recognize work remains. For instance, more thought needs to be devoted to alleviating the addressing inefficiency problems when a LOC is taken of an object of one type and used to access an object of another type

4/14/

that might not start on the same addressing boundary. My objective has been to be reasonably thorough but remaining in overview mode.

I imagine that most, if not all, of the ideas in this proposal are not original in that they have likely been covered by the committee prior to my arrival. Nonetheless, I strongly feel that we should not turn a blind eye to a pointer data type in favor of wedging another mechanism into the language. Again, I contend that when users say they want pointers, they want an easy-to-understand (and -use) pointer data type.

Appendix I: Alternatives Rejected

A. Direct incorporation of attributes

Since strong typing these days seems to equate to binding a pointer to a type (a set of attributes), it is possible to present a syntax that allows a pointer to be bound to an arbitrary set of attributes without the additional burden of defining a type. This is really only practical for nonstructure objects. Consider:

```
TYPE bob(name)
  CHARACTER(len=name), ARRAY(0:9) :: dave
END TYPE

POINTER(bob) :: bob_ptr
```

as opposed to incorporating the attributes directly into the pointer declaration:

```
POINTER(CHARACTER(len=name), ARRAY(0:9)) :: bob_ptr
```

I have not proposed this direct incorporation method mainly because it could lead to long, awkward declarations. Also, it would not be particularly safe in that if one had several declarations of pointers and wanted them all bound to the same attribute set, using a type name would be safer than lists of attributes.

B. Pointer bound to collection of types

Our 1100 series systems programming language has a mechanism to bind a pointer to a collection of types. For example, since our 1100 machines are not hardware paging machines, the UCS (new-generation) compilers run in a software virtual paging environment. We segregate the dictionary (symbol table) information into one "area" and the text entries that represent the executable code into another "area". We have several kinds of virtual entries in the form of 8x-like structures in the dictionary area. Each entry may be thought of as an individual "type". It is sufficient in most cases to pass and use a pointer that points to any kind of dictionary entry. Each entry is identified by a field in the entry itself.

Although we have found this ability to be very useful, I have not included it in this proposal because I think just getting a cohesive pointer facility implemented at this point would be a minor miracle, let alone trying to add more functionality. If this proposal, or another like it, advances in the committee, I would be happy to discuss this type-grouping idea in more detail.

Appendix II: Other points

A. Is LOC a good choice for a name?

In the body of the proposal, I said that we should probably pick LOC as the intrinsic name because the name already exists in some implementations and performs basically the same operation. One of my reviewers pointed out that the "basically" may be problem. Apparently customer programs exist that assign the LOC result to integer variables and other such nasties. Since the 8x definition of the function result may be different than the result currently being returned by a vendor's software and keeping an eye directed toward conversion costs, we perhaps should use another name. I'd like to fall back on ADDR if LOC would cause too many problems.

B. Storage allocation of derived-type objects

While working through this pointer proposal, a related problem with 8x's statement that a derived-type object has no storage sequence reared its ugly head. Recall in appendix section I.B I mentioned that our compiler has a number of dictionary entries that are declared as having different "types" but that we access them via a single pointer. Each dictionary entry has a tag field that identifies the entry. Moving from one entry to another via a pointer depends on the fact that the tag field is always in the same relative location within the entry's storage. Thus, we can have a declaration that covers just enough of the entry to include the tag field (a header portion), interrogate the tag field, and operate on the entry accordingly.

I can not believe that we are the only programmers in existence that wish to do something like this. I am not making an argument to bring back variant types, although this might be the place to do that. I would rather resurrect an idea we discussed around a year ago (Mt. Kisco maybe?): Can we develop an attribute to control the ordering of members of an object of derived type? I think we only need to control the ordering; there should be no need to talk about storage allocation. I think we can have an attribute that tells the compiler to order the members exactly as declared and yet continue to state that a derived-type object has no storage sequence (presumably to avoid EQUIVALENCE). If a user then declares a multitude of derived-type objects each having a header portion declared in exactly the same way, can we not count on a compiler to allocate storage exactly the same way? Is it too much to ask a vendor to "do the right thing"?

In case I'm not making myself clear, here's what I think we need:

```
TYPE header
  INTEGER  tag
  INTEGER  other_information
END TYPE

TYPE entry_1
  TYPE(header)       header_info
  CHARACTER(LEN=20)  name
  ...
END TYPE

TYPE entry_2
  TYPE(header)       header_info
  INTEGER            number
  ...
END TYPE

TYPE(header), ORDERED   :: generic_header
TYPE(entry_1), ORDERED  :: node_type_1
TYPE(entry_2), ORDERED  :: node_type_2
POINTER                 :: gp
```

```
IF (gp->generic_header%tag .EQ. 31) THEN
  gp->node_type_1%name = its_name
ELSE
  gp->node_type_2%number = its_number
END IF
```

I have chosen the attribute ORDERED and applied it to the declaration of the object. I would think we might want to allow such an attribute (statement?) to occur in a type definition, a la PRIVATE, to allow a user to ensure that all objects declared with the derived type are indeed ordered.

If any of you are involved with the PL/I committee, or have associates that are, you might find it interesting to discuss with them the lengths that committee has gone to in order to ensure that structures match in storage allocation. They needed to solve the same problem and did it in a more restrictive manner. In order to appease the anti-storage-association contingent of X3J3, I'm trying to be as general as possible.

54

To: X3J3

From: B. T. Smith, General Concepts (SG 20)

Subject: Liverpool Resolution (R21) On The Use Of National Characters

References: [1] ISO/TC96/SC22/WG5 N254, Liverpool, August 1987.
[2] X3J3 Standing Document S8.104 (June 1897), Section 3.1.4, pp. 3-1 and 3-2.

1. Liverpool Resolution R21

Liverpool resolution R21 [1] states:

That WG5 expresses to X3J3 its concern about the negative effect on the production of standard-conforming processors if characters in the national use positions in ISO 646, such as square brackets, are required in the Fortran 8X character set.

The voting results for R21 at the Liverpool ISO meeting were: 31-0-4 (Individual); 9-0-0 (Country)

2. Subgroup's Recommendation

Upon reviewing the WG 5 concern on the processor's required use of square brackets (as stated in [2]), we propose to allow the use of the square brackets by the processor to be optional and to permit the processor to use alternative graphics for these ASCII characters. The effect of this proposal is to require a portable standard conforming Fortran 8x program to use the alternative forms of the square brackets, namely (/ and /). Programs that use [and] will be standard conforming but can only be processed on processors that support the square brackets [] -- the use of square brackets would be similar to the use of lower case letters as currently described in S8.

Several other alternatives were discussed in the subgroup, including proposals that required the availability of a pair of characters but allowed alternative graphics, and that made the characters [ ] optional but with these graphics. It was decided to propose the combination of these as described above.

Our style in S8 appears to be to avoid the use of optional characters in examples (such as lower case letters). If this convention is to be adhered to, all uses of brackets in examples in S8.104 need to be replaced by (/ and /); however, such occurrences have not been noted in this proposal.

3. Proposal

On page 3-1, lines 37-38, after "Bracket", insert "(optional)" twice.

On page 3-1, line 43, delete the sentence "Where ... permitted.". Begin a new paragraph after line 42. Replace "The" with "The availability of the right or left bracket is optional. If the processor does not support them, the".

On page 3-2, line 1, replace ". The" with "and the".

On page 3-2, line 2, after "symbol", insert ", right bracket, or left bracket".

55

To: X3J3

From: B. T. Smith, General Concepts (SG 20)

Subject: S8 Audit of Chapters 17 and 18 of Fortran 77

References: [1] X3.9-1978 The Fortran Standard, Chapters 17 and 18.

[2] X3J3 Standing Document S8.104 (June 1987).

1. Introduction

The following items list those places in S8 [2] which appear to be inconsistent with Sections 17 and 18 of Fortran 77 [1]. Included with each missing sentence or idea is a suggested change to S8.104(June). I have used the notation (Ed.) or (Tech.) to indicate whether the change is editorial in nature or a substantial technical issue.

2. Possible Inconsistencies

2.1. Total Association Of Arrays

The following statement is made in Fortran 77 (page 17-1, lines 43-44):

Two entities are totally associated if they have the same storage sequence.

The corresponding statement in Fortran 8x is (page 14-6, lines 7-8):

Two scalar entities are totally associated if they have the same storage sequence.

Since in Fortran 77 entities includes arrays, Fortran 77 arrays can be totally associated but according to the definition of Fortran 8x, arrays cannot. However, I cannot find any substantial use of the term "total association" in Fortran 8x (an inconsequential use appears in line 37, page 14-6 in S8.104 (June)).

The need for the concept of total association seems to be obviated by Fortran 8x's more natural description of when objects with subobjects become defined or undefined (S8.104(June), p. 14-7, lines 6-13).

We need a grep on "total" to find out where this term is used. If it is not used, we should remove the definition of totally associated.

2.1.1. Suggested Changes To S8 (Ed.)

Assuming that the term "total association" is unnecessary, the following changes are recommended:

Page 14-6, lines 7-8:
Delete the sentence: "Two scalar entities ... sequence.".

Page 14-6, line 37:

Change "totally" to "storage".

Remove all other references to "total association", wherever they are.

2.2. Changes In Terms

In comparing Fortran 77 (Sections 17 and 18) and Fortran 8x, I discovered several places where the change in terms between Fortran 77 and Fortran 8x leads to potential confusion. I recommend that the following changes be made.

2.2.1. Association Versus Storage Association

Fortran 8x describes association as being of two kinds -- name and storage association -- and name association is either host, use, alias, or argument association. There is a potential misunderstanding in places in Fortran 8x where the term association is used without being qualified. (For example, page 14-8, lines 8-9 of S8.104 (June) could be interpreted as requiring all argument associated character storage units to become defined when the dummy argument is defined by any assignment statement -- this is surely not intented.)

Consider the following changes to Fortran 8x:

2.2.1.1. Suggested Changes To S8 (Ed.)

Page 14-6, line 6:
Before "**Association**", insert "**Storage**"

Page 14-6, line 13:
Before "association", insert "storage"

Page 14-6, line 15 and page 14-8, lines 8, 10:
Before "associated", insert "storage"

2.2.2. Real Versus Default Real

Fortran 8x uses default real where Fortran 77 uses real.

2.2.2.1. Suggested Changes To S8 (Ed.)

Page 14-6, lines 19-21:
On line 20, replace "integer, real" with "integer, default real". On lines 19 and 21, insert "default" before "complex" twice.

2.2.3. Fortran 8x Variable Versus Fortran 77 Variable

The sentence (S8.104(June), page 14-5, line 38) "A variable or array element of a derived type has no storage sequence." seems to suggest that either an array element is not a variable or an array element in this context is special. Neither is true. I recommend that "or array element" be deleted.

2.2.3.1. Suggested Changes To S8 (Ed.)

Page 14-5, line 38:
Delete "or array element".

2.2.4. Implied-DO Versus *io-implied-do*

The term "implied-DO" is a Fortran 77 term which does not appear to be used in 9.4.2 (S8) where input/output implied-dos are defined. The term used there is the BNF term *io-implied-do*. I suggest we use the Fortran 8x term consistently in Section 14 of S8.

2.2.4.1. Suggested Changes To S8 (Ed.)

Page 14-7, line 38:
Change "implied-DO" to "*io-implied-do-object*"

Page 14-7, line 39:
Change "implied-DO variable" to "*do-variable of the io-implied-do*"

2.2.5. Entities Versus Data Objects

On page 14-6, lines 41-42, the term "entities" is used where I believe data objects is correct and more precise. Only data objects can be storage associated (as opposed to types, program units, etc.) and named objects (rather than entities without names such as expressions) can be storage associated. Argument association in S8 appeals to sequence association to describe what associates between expressions and dummy arguments. (Is this true in all cases? If so, read on to the suggested changes.)

2.2.5.1. Suggested Changes To S8 (Ed.)

Page 14-6, lines 41-42:
Change "entities" to "data objects" twice.

2.3. Argument Association

The Fortran 77 text on page 17-2, lines 1-6, states that argument association may cause entities in different program units to become associated. S8 does not say this directly but relies on the reader understanding that the scope of actual and dummy arguments are in different scoping units. I can accept that these statements are equivalent. However, I cannot find the next Fortran 77 sentence in S8. That sentence

is:

Note that association between actual and dummy arguments does not imply association of storage sequences except when the actual argument is the name of a variable, array element, or substring.

The corresponding statement that should be in Fortran 8x is slightly different but does it need to be said? It seems to be a consequence of 12.4.1.4 (Page 12-6 of S8.104(June)). The suggested change below seems to be said more directly than 12.4.1.4 and there is a natural place to put it in S8 as proposed below.

2.3.1. Suggested Changes To S8 (Ed.)

Page 14-4, line 9+:
Insert as a new paragraph:

Argument association does not imply sequence association (12.4.1.4) except when the actual argument is a variable, an array expression, or a substring.

2.4. Association Of Entry Names

Fortran 77 defines the concept of "entry association" -- see Fortran 77, Section 15.7.3. S8 does not define this term as one of the kinds of association but does define the concept on Page 12-11, lines 23-28. Whereas Fortran 77 seems to suggest that entry association is a separate concept, S8 uses two terms -- association (which kind?) when the entry name(s) and function name have the same characteristics (type, type parameters and maybe some more properties?) and storage association when they have different characteristics (limited to those cases where storage association is defined).

Is this really a problem? It appears to me to be a problem of exposition rather than a technical problem. I can see several solutions but I am not sure they are worth the bother. The one I prefer is to define **entry association** as a fifth kind of name association. Its definition is the association that occurs between the entry name(s) and functions name when they have the same characteristics. If this proposal is accepted, the term "entry associated" can be used on page 12-11, line 26 (before the first word), the term "entry association" can be defined in section 14.7.1 and a new section, say "14.7.1.4 **Entry Association**", would be added with appropriate text. But, I feel this should be discussed before any text is prepared.

However, the text of S8 (page 14-2, lines 2-5) is inconsistent with the text of Fortran 77 (page 18-3, lines 16-21). Consider the following change to S8.

2.4.1. Suggested Changes To S8 (Tech.)

Page 14-2, lines 3-5:
Replace the text "function. If a function subprogram ... entry." with the following text:

function or one of the entry names, if any, within the function subprogram.

2.5. Effect On Associated Objects When An Object Is Defined

Point (7) of Fortran 77, page 17-3, lines 3-34 seems to correspond to point (11) of S8, page 14-8, lines 8-14. However, the sentence on lines 13-14 of point (11), namely:

When a scalar variable without a storage sequence becomes defined, all associated variables become defined.

is misleading, I think, and at the same time less general than is required for Fortran 8x.

My concerns are two. First, the term associated in Fortran 8x applies to all kinds of association, including, in particular, argument association. If the word "associated" is interpreted as "argument associated", then the sentence seems to contradict section 12.5.2.9 of S8. I propose to add a further sentence that makes the reader aware of the restrictions of 12.5.2.9. Secondly, the sentence appears to be needed for use, host, alias, and allowed argument associations and applies to array variables as well as scalar variables. Why is this sentence limited to scalar variables?

2.5.1. Suggested Changes To S8 (Tech.)

Page 14-8, lines 13-14:
Replace these lines with the following:

When a variable without a storage sequence becomes defined, all associated variables become defined. Note that there are restrictions on argument association when combined with other associations that limit the use of the associated variables in a scoping unit (12.5.2.9).

2.6. Keywords, Edit Descriptors, Etc. Are Not Names

The sentence in Fortran 77, page 18-1, lines 3-9 does not appear in S8. I believe the BNF implies this statement, namely, that identifiers such as READ, GOTO, A5, etc are not necessarily names in Fortran 8x unless used in certain contexts. Do you agree?

2.7. Five Classes Of Global Entities

On page 18-1, lines 38-50, there are 5 classes of global entities: common block, external function, subroutine, mail program, and block data program. After reviewing chapters 17 and 18 of Fortran 77 and checking the corresponding statements in Fortran 8x, I see no need for this classification. Is this correct?

2.8. A Symbolic Name Identifies Something

The following sentence (page 18-1, lines 52-53 of Fortran 77) does not appear in Fortran 8x explicitly:

The symbolic name of a local entity identifies that entity in a single program unit.

The definition of "name" (which is the corresponding 8x term for symbolic name) says it identifies a program constituent such as a variable, etc (page 2-9, lines 19-20 of S8). I believe this is adequate.

2.9. Classes Of Local Entities

Fortran 77 (page 18-2, lines 10-24) defines 6 classes of local entities and Fortran 8x places these six classes into one class. Nothing seems to be wrong with this reclassification in Section 14 of S8. Is there are problem elsewhere in S8?

2.10. Permitted Dummy Arguments

Fortran 77 (page 18-2, lines 26-29) states that a name that is a dummy argument is either a variable (or array) or a dummy procedure and that the specification and use of such dummy argument must not violate the class rules for local entities. This concept is not stated in Section 14 of S8 and I cannot find it in Section 12 either. Also, I cannot find all of the restrictions related to violations of the Fortran 77 classification rules -- for example, where does it state that a dummy argument name cannot be specified to have the PARAMETER attribute or appear as the name on the left-hand side of an equals in a PARAMETER statement?

I suggest below that a sentence be added to section 12.5.2.3. Should restrictions be placed on dummy arguments, PARAMETER attribute, PARAMETER statement, INTRINSIC statement and statement function statement in the appropriate sections? Should something like the recommended sentence be added to Section 14, page 14-1, line 40+, instead?

2.10.1. Suggested Changes To S8 (Tech.)

Page 12-10, line 25+:
Add the following constraint after line 25:

Constraint: A name that is a dummy argument must be either a variable or a dummy procedure.

2.11. External Function

The section of Fortran 77 (page 18-3, lines 1-14) describing how to tell whether a name is the name of an external function does not appear in S8, as far as I can find. I believe it is needed and can find no better place for it other than Section 14.

2.11.1. Suggested Changes To S8 (Tech.)

Page 14-2, line 1+:
Add the following section after line 1 and renumber the subsequent sections:

14.1.2.2 External Function. A name is the name of an external function if it meets either of the following conditions:

(1) The name appears immediately following the token FUNCTION in a FUNCTION statement when it is the first statement of an external subprogram or the token ENTRY in an ENTRY statement within an external function subprogram.

(2) The name appears immediately following the token FUNCTION in a FUNCTION statement in an interface block and the name is not the name of an internal or module procedure accessible to the scoping unit containing the interface block.

(3) It is not an array name, character variable name, statement function name, intrinsic function name, internal subprogram name, dummy argument, subroutine name, type parameter name, or type name, and every appearance in a scoping unit is immediately followed by a left parenthesis except in a type declaration statement, in an EXTERNAL statement, or as an actual argument.

2.12. Subroutine

Section 18.2.3 (Fortran 77) does not appear in this form in S8. However, the alternative conditions both appear in Section 12 of S8. I believe this is adequate.

2.13. Arrays

Section 18.2.6 does not appear in this form in S8. There appears to be no simple statement listing the ways in which a named array can be declared. On the other hand, I am not convinced it is necessary to have such an explicit statement.

2.14. Variables

Section 18.2.7 does not appear in this form in S8. The BNF seems to cover this list of conditions. Does anyone disagree?

2.15. Intrinsic Functions

The sentence (Fortran 77, page 18-4, lines 26-28) "Note that the use of a name that appears in Table 5 as a dummy argument of a statement function removes it from the class of intrinsic functions" does not appear in S8. It is a consequence of 18.2.10 (Fortran 77) which also does not S8. Consider placing the following text in section 14.

2.15.1. Suggested Changes To S8 (Tech.)

Insert a new section after line 1+ on page 14-2 (note that the above text recommends another section be added here) and renumber the subsequent sections:

14.1.2.3 Intrinsic Procedure. A name is the name of an intrinsic procedure if it meets all of the following conditions:

(1) The name appears in the tables in 13.9 (and all subsections), 13.10, and 13.11 as a procedure name (including specific name) or is a type parameter name.

(2) It is not an array name, statement function name, external or internal subroutine name, type name, or a dummy argument name

(3) Every appearance of the name, except in an INTRINSIC or CALL statement, a type declaration statement, or as an actual argument, is immediately followed by an actual argument list (possibly empty) in parentheses.

Note that the use of a name that appears in the table in sections 13.9, 13.10, or 13.11 as a dummy argument of a statement function removes it from the class of intrinsic functions.

2.16. Statement Function Dummy Arguments

The Fortran 77 text on page 18-4, lines 33-37 corresponds to the S8 text on page 14-3, lines 17-19 but does not seem to say exactly the same thing. One problem is that the word "name" in the line 18 is ambiguous (which name -- the statement entity or the local(global) entity); a second problem is that Fortran 77 states the scope of both names whereas S8 only gives the use of one of the names.

A second issue is that the length type parameter for a statement function dummy argument of type character is the same as the local variable of the same name. S8 does not require this. Is this an oversight or a relaxation of the Fortran 77 rule? I assume it is an oversight.

Consider the following changes to the S8 text.

2.16.1. Suggested Changes To S8 (Tech.)

Page 14-3, line 9:
Insert "and type parameters" after "type".

Page 14-3, lines 17-19:
Replace with the following text:

The name of a statement entity also may be the name of a global or local entity in the same scoping unit; in this case, the name within the statement function statement, IDENTIFY statement, or DATA statement is interpreted as that of the statement variable and all other appearances of the name are the name of the global or local entity.

2.17. Statement Entities

Statement entities are required to be scalar variables in all cases in S8. The Fortran 77 text requires that variables with the same names as statement entities are also scalars (because of the Fortran 77 definition of variable not being an array). However, the S8 text seems to allow the following situation: a name that is the name of a statement entity can also be the name of a local entity that is an array. This interpretation is possible because the S8 definition of variable includes an array.

Does anyone agree with this interpretation of the S8 text? If so, we need to change the text somehow.

2.18. Dummy Procedure

Section 18.2.11 of Fortran 77 does not appear in S8. Consider adding a following text as 14.1.2.4.

2.18.1. Suggested Changes To S8 (Tech.)

Insert a new section after line 1+ on page 14-2 (note that the above text recommends two other sections be added here) and renumber the subsequent sections:

14.1.2.4 Dummy Procedure. A name is the name of a dummy procedure if the name appears in the dummy argument list of a FUNCTION, SUBROUTINE, or ENTRY statement and meets one or more of the following conditions:

(1) It appears in an EXTERNAL statement.

(2) It appears immediately following the token CALL in a CALL statement.

(3) It is not an array name or character variable name, and every appearance is immediately followed by a left parenthesis except in a type declaration statement, in an EXTERNAL statement, in a CALL statement, in an OPTIONAL statement, as a component name, as an argument keyword, as a dummy argument, as an actual argument, or as a common block name in a COMMON or SAVE statement.

56

To: X3J3

From: B. T. Smith

Subject: Liverpool Resolution (R23) On Passed-on Precision

References: [1] ISO/TC97/SC22/WG5 N254, Liverpool, August 1987.
[2] X3J3 Standing Document S8.104 (June 1987).
[3] ISO/TC97/SC22/WG5 N245, Liverpool, August 1987.

1. Liverpool Resolution R23

Liverpool resolution R23 [1] states:

That WG5 draw the attention of X3J3 to the concerns of the German member body (DIN) about passed-on precision contained in paper N245.

The voting results for this resolution at the Liverpool ISO meeting were: 25-2-8 (Individual); 8-0-1 (Country)

2. The Problem

In discussing paper N245 [3] with Karl-Heinz Rotthäuser, I became aware of the source of his concerns over the implementability of the real (*,*) facility as proposed in [2]. His concerns are best illustrated by an example which follows below. My proposed solutions are to recommend a modification of the definition of automatic data object in Section 5.1 (which was actually brought to my attention by Alex Marusak at the second half of Meeting 103) and to provide some notes for Appendix C to explain the semantics of real(*,*) as requiring only the effective precisions.

Consider the following example:

```
! Main program

interface
    subroutine  EXAMPLE( X, Y)
    real(*,*)  X, Y
end interface

real(4)  C, D
real(10)  X, Y
  ...
call  EXAMPLE( X, Y)
  ...
call  EXAMPLE( C, D)
  ...
end

subroutine  EXAMPLE( X ,Y)
real(*,*)  X, Y
real( effective_precision(X), effective_exponent_range(X) )  A, B
integer, save, initial ::  K = 1
  ... (1)
```

```
X = ...
Y = ...
K = K + 1
A = ...
B = ...
  ... (2)
return
end
```

The above program (assuming the dots are replaced by valid code) is a valid Fortran 8x program according to S8.104 (June, 1987). In discussing [3] with Karl-Heinz Rotthäuser, the semantics of the above program were not clear to him. In addition, he asked whether a slightly modified version of the above program, namely, replace the third statement with:

```
real( effective_precision(X), effective_exponent_range(X) ), save ::  A, B
```

was a valid program (answer: no).

To make the semantics of the above program clear, I find it easiest to display a program with equivalent semantics but relies on the machine dependent real and double precision data types to specify it. For this example, I assume a machine that has effective decimal precisions of 6 and 15 decimal digits for its real and double precision data types.

Consider the following program and its references which I claim have equivalent semantics to the above program:

```
! Main program

interface
    subroutine  EXAMPLE1( X, Y)
    real  X, Y
end interface

interface
    subroutine  EXAMPLE2( X, Y)
    double precision  X, Y
end interface

real  C, D
double precision  X, Y
  ...
call  EXAMPLE2( X, Y)
  ...
call  EXAMPLE1( C, D)
  ...
end

subroutine  EXAMPLE1( X ,Y)
real  X, Y
real  A, B
double precision  X2, Y2
double precision  A2, B2
integer, save, initial ::  K = 1
  ... (1)
X = ...
Y = ...
K = K + 1
```

```
A = ...
B = ...
  ... (2)
return
entry  EXAMPLE2( X2, Y2)
  ...  Same as (1) with X, Y, A, B replaced by X2, Y2, A2, B2
X2 = ...
Y2 = ...
K = K + 1
A2 = ...
B2 = ...
  ...  Same as (2) with X, Y, A, B replaced by X2, Y2, A2, B2
return
end
```

In the above program, the processor-dependent declarations real and double precision are used to specify the variables that depend upon the precision type parameters of the arguments. The source code after the ENTRY statement is a copy of executable statements before the ENTRY statement, except that variables such as A, B, X, and Y need to be replaced by the variables A2, B2, X2, and Y2 respectively, and statement labels and construct names need to be similarly modified. Saved local variables such as K are global to both procedures (EXAMPLE1 and EXAMPLE2). Data objects such as K have the same storage representation for both procedures and hence can be saved. Data objects whose storage representation is dependent on the particular reference to the procedure (called automatic data objects in S8.104 (June, 1987), cf. p. 5-2, lines 20-25) cannot be saved.

A similar translation of the references to EXAMPLE is required. This translation can always be done during compilation, because the interface to a procedure such as EXAMPLE is required to be explicit. That is, during the compilation of the reference, the processor knows the characteristics of all of the dummy arguments of EXAMPLE and therefore can translate the reference to either

```
call  EXAMPLE1( ARG1, ARG2)
```

or

```
call  EXAMPLE2( ARG1, ARG2)
```

according to the type and/or precision type parameters of ARG1. The processor either may relax the argument association rules to permit an association between a precision-specified real type and real (and double precision) arguments, or may modify the declarations of ARG1 and ARG2 to real (or double precision) arguments as shown by the references in the example above.

The above example assumes that the procedure EXAMPLE was an external procedure. A similar transformation is possible for a module procedure. However, for a similar internal procedure, the above transformation is not suitable because an ENTRY statement is not allowed in an internal procedure. In this case, the equivalent program is obtained by adding a second internal (named EXAMPLE2) as appropriate with the same body as EXAMPLE except for two situations: first, new type specifications (double precision) for the data objects A, B, X, and Y are required, and secondly, all of the specifications for saved objects (e.g., K) need to be placed in the host scoping unit and appropriately renamed to avoid name conflicts. Renaming of statement labels and construct names is not required in the case of an internal procedure. Another approach is to add a procedure of the same name as an overloaded internal procedure name instead of giving it a new name. An example of latter transformations is given in the recommended text to be added to Appendix C.

3. Changes to S8

I recommend the following two changes to S8.104 (June, 1987). The first change makes it clear that automatic data objects refer to any data object whose size or type parameters depend upon a dummy argument. The second change adds some further notes that make it clearer that only the effective precision and exponent range attributes need to be used to implement real(*,*) dummy arguments.

Page 5-2, line 20:
Delete "nonprecision".

Page C-16, line 26+:
Insert the following text after line 26 (similar to the text above but changed to suit the surrounding text of Appendix C):

To illustrate that only the effective attributes of the actual arguments need be used, consider the module below containing the module subprogram EXAMPLE with real(*,*) dummy arguments and a reference to the module subprogram EXAMPLE:

```
module  SAMPLE

CONTAINS
subroutine  EXAMPLE( X ,Y)
real(*,*)  X, Y
real( effective_precision(X), effective_exponent_range(X) )  A, B
integer, save, initial ::  K = 1
  ... (1)
X = ...
Y = ...
K = K + 1
A = ...
B = ...
  ... (2)
return
end

end module  SAMPLE


! Main program
use  SAMPLE
real(10)  X, Y
  ...
call  EXAMPLE( X, Y)
  ...
end
```

Suppose the effective precision and effective exponent range type parameters for a particular machine's real data types are (6,75) and (15,75). Then, on such a machine, the following module and reference to the module subprogram are equivalent to the previous module subprogram and references:

```
module  SAMPLE
integer, save, initial, private ::  K = 1
```

```
CONTAINS
subroutine  EXAMPLE( X ,Y)
real  X, Y
real  A, B
  ... (1)
X = ...
Y = ...
K = K + 1
A = ...
B = ...
  ... (2)
return
end
subroutine  EXAMPLE( X ,Y)
double precision  X, Y
double precision  A, B
  ... (1)
X = ...
Y = ...
K = K + 1
A = ...
B = ...
  ... (2)
return
end

end module  SAMPLE


! Main program
use  SAMPLE
double precision  X, Y
  ...
call  EXAMPLE( X, Y)
  ...
end
```

The equivalent module is obtained by replacing the module subprogram with real(*,*) arguments with two module subprograms, both with names EXAMPLE, whose bodies are essentially copies of the original module subprogram. The body of the first module subprogram EXAMPLE is the same as the original subprogram except that the declarations of real(*,*) dummy arguments (and arguments that depend on the type parameters of such arguments) are replaced by real declarations, the specifications for all saved variables are placed in the specification part of the module, and such saved variables are specified as private. The body of the second module subprogram EXAMPLE is the same as example except that the dummy arguments (and related data objects) are declared with double precision declarations. If the name of any saved variable is the same as a host variable, the saved variable needs to be renamed in both the specification part of the module and the two version of EXAMPLE. The reference to EXAMPLE remains the same, but the declarations of the arguments are changed to double precision.

Appendix To 106(*)-BTS-3

A. Use Of REAL(*,*) In Practice

The following subprogram is an example of the use of REAL(*,*) dummy arguments to specify the elementary function COTAN(X). The algorithm in this Fortran 8x software (consistent with S8.104 (June, 1987)) is taken from the *Software Manual For The Elementary Functions*, by W. J. Cody and W. Waite, published by Prentice-Hall Inc, Englewood Cliffs, New Jersey, 1980.

```
! Main program

interface
    function  COTAN( X )
    real(*,*), intent(in) ::  X
    real( effective_precision(X), effective_exponent_range(X) )  COTAN
end interface

real(4)  X, RESULT
real(10)  C, D
  .
  .
  .
X = 0.0
RESLT =  COTAN( X )
write(6,*)  'COTAN to at least 4 digits -- X, COTAN(X) = ', X, RESLT
  .
  .
  .
C = atan( real(1.0, C) )
D = COTAN( C )
write(6,*)  'COTAN to at least 10 digits -- X, COTAN(X) = ', C, D
  .
  .
  .
end

function  COTAN( X )

real(*,*), intent(in) ::  X
real( effective_precision(X), effective_exponent_range(X) )  COTAN

! This external function, written in Fortran 8x, version S8.104 (June, 1987)
! computes the COTAN(X).

! The algorithm used below is described in "Software Manual For The
! Elementary Functions" by W. J. Cody and W. Waite.  This program has
! been written by Brian T. Smith, Argonne National Laboratory, Sept. 1987.

! Local variables.

integer  N
real( effective_precision(X), effective_exponent_range(X) )  &
      C1, C2, F, G, SQRT_EPS, P1, P2, P3, Q1, Q2, Q3, Q4, &
      X1, X2, XDEN, XN, XNUM, Y, YMAX

exponent_letter( effective_precision(X), effective_exponent_range(X) )  Q
```

```
! Local constants.

integer, parameter :: ERROR_UNIT_NUMBER

real( effective_precision(X), effective_exponent_range(X) ), parameter ::   &
                  HALF = 0.5 Q0, &
                  ONE = 1.0 Q0, &
                  TWO_OVER_PI = 0.63661 97723 67581 34308 Q0, &
                  ZERO = 0.0Q0

intrinsic  abs, aint, effective_precision, epsilon, huge, int, mod, real, &
           sign, sqrt, tiny

Y = abs(X)

if( Y .lt. tiny(Y) )  then

!   Give an error message for argument error.

    write(ERROR_UNIT_NUMBER,*)  &
       'Argument too small in magnitude to evaluate COTAN(X)'
    RESULT = sign( huge(X), X)
    return

endif

SQRT_EPS = sqrt( epsilon( X ) )
YMAX = aint( ONE / ( SQRT_EPS * TWO_OVER_PI ) )

if( Y .gt. YMAX )  then

!   Give an error message for argument error.

    write(ERROR_UNIT_NUMBER,*)  &
        'Argument too large in magnitude to evaluate COTAN(X)'
    RESULT = ZERO
    return

endif

! Compute the quadrant for X.

N = int( X * TWO_OVER_PI + HALF )
XN = real( N, X)

! Determine  F.

X1 = aint( X )
X2 = X - X1

if( effective_precision( X ) .le. 9 )  then

    C1 = 1.5703 125 Q0
    C2 = 4.8382 67948 97 Q-4

else

    C1 = 1.57080 07812 5 Q0
    C2 = -4.4544 55103 38076 86783 08 Q-6
```

```
endif

F = ((X1 - XN*C1) + X2) - XN*C2

if( abs( F ) .lt. SQRT_EPS )  then

    XNUM = F
    XDEN = ONE

else

    G = F * F

!   Compute XNUM = F*PG and XDEN = QG as values of low degree polynomials
!   P(G) and Q(G) whose degrees and coefficients depend on the effective
!   decimal precision of X.

    select case( effective_precision(X) )
    case(:7)

        P1 = -0.95801 7723 Q-1
        Q1 = -0.42913 5777 Q0
        Q2 = 0.97168 5835 Q-2
        XNUM = P1*G*F + F
        XDEN = ((Q2*G + Q1)*G + HALF) + HALF

    case(8:10)

        P1 = -0.11136 14403 566 Q0
        P2 = 0.10751 54738 488 Q-2
        Q1 = -0.44469 47720 281 Q0
        Q2 = 0.15973 39213 300 Q-1
        XNUM = (P2*G + P1)*G*F + F
        XDEN = ((Q2*G + Q1)*G + HALF) + HALF

    case(11:16)

        P1 = -0.12828 34704 09574 3847 Q0
        P2 = 0.28059 18241 16998 8906 Q-2
        P3 = -0.74836 34966 61206 5149 Q-5
        Q1 = -0.46161 68037 42904 8840 Q0
        Q2 = 0.23344 85282 20687 2802 Q-1
        Q3 = -0.20844 80442 20387 0948 Q-3
        XNUM = ((P3*G + P2)*G + P1)*G*F + F
        XDEN = (((Q3*G + Q2)*G + Q1)*G + HALF) + HALF

    case(17:18)

        P1 = -0.13338 35000 64219 60681 Q0
        P2 = 0.34248 87823 58905 89960 Q-2
        P3 = -0.17861 70734 22544 26711 Q-4
        Q1 = -0.46671 68333 97552 94240 Q0
        Q2 = 0.25663 83228 94401 12864 Q-1
        Q3 = -0.31181 53190 70100 27307 Q-3
        Q4 = 0.49819 43399 37865 12270 Q-6
        XNUM = ((P3*G + P2)*G + P1)*G*F + F
        XDEN = ((((Q4*G + Q3)*G + Q2)*G + Q1)*G + HALF) + HALF

    case default
```

```
            write(ERROR_UNIT_NUMBER,*)  'This program  COTAN  does not support&
               & effective decimal precisions larger than 18.  Program terminated.'
            stop

        end select

    endif

    if( mod(N,2) .eq. 0 )  then

        RESULT = XDEN / XNUM

    else

        RESULT = -XNUM / XDEN

    endif

    return
    end
```

B. Equivalent Fortran 77 Software

Using the transformations listed below, this subprogram and its references can be transformed by the following rules into machine-dependent Fortran 77 software.

(1) In the program unit with a reference to COTAN(X), declare real variables with REAL or DOUBLE PRECISION declarations as appropriate. This can be done immediately in cases where the precision type parameter declarations are specification expressions as they must be constant expressions or expressions involving the intrinsics EFFECTIVE_PRECISION or EFFECTIVE_EXPONENT_RANGE which in effect are constants known at compile time. In cases where the type parameters are asterisk, then apply rules (2) through (7) below to the program unit containing the reference, yielding in effect constant values for the type parameters. If the arguments of COTAN are DOUBLE PRECISION data objects, change the reference to COTANS instead of COTAN.

(2) Change the type declaration of all data objects whose precision (and exponent range) type parameters depend on the arguments of COTAN to REAL.

(3) Copy the declaration of all data objects whose precision (and exponent range) type parameters depend on the arguments of COTAN, adding an S to the name of such data objects, and changing their declaration from REAL to DOUBLE PRECISION.

(4) Apply step one to the declarations of the remaining real data objects.

(5) Copy the executable part of COTAN to follow an ENTRY statement of the form:

```
        entry   COTANS( XS )
```

which follows the return statement. Add an S to all variable names in the copied executable part whose type is floating point and which depends on the precision (or exponent range) type parameters of the arguments of COTAN. Rename any statement labels or construct names so that they do not

conflict with such labels and names in the executable code before the entry statement.

(6) Rename any long names (more than 6 characters) so that they have fewer than 6 characters, any names with special characters like 2, provide external functions for the intrinsic functions not in Fortran 77, and change the form of the source to satisfy the rules of Fortran 77 source text (including removing non-Fortran 77 features such as interface blocks).

(7) For those environmental inquiry functions that return constants, substitute their values into the code. (Such functions are: EFFECTIVE_PRECISION, EFFECTIVE_EXPONENT_RANGE, HUGE, TINY, EPSILON, etc.) Where those intrinsic functions that do not return constants, substitute the appropriate Fortran 77 code: for example, real(X, real data object) becomes real(X); real(X, double precision data object) becomes DBLE(X). Where dead code results from these substitutions, delete it.

Assuming a machine that uses the declaration REAL for real data objects with effective decimal precision 6 and DOUBLE PRECISION for real data objects with effective decimal precision 15, the equivalent Fortran 77 code is as follows:

```
c Main program

c       interface
c           function  COTAN( X )
c           real, intent(in) ::  X
c           real  COTAN
c       end interface

c       interface
c           function  COTAN( X )
c           double precision, intent(in) ::  X
c           double precision  COTAN
c       end interface

        real  X, RESLT
        double precision  C, COTANS, D
c         ...
        X = atan(0.1)
        RESLT =  COTAN( X )
        write(6,*)
     +      'COTAN to at least 4 digits -- X, COTAN(X) = ', X, RESLT
c         ...
        C = atan(1.0D0)
        D = COTANS( C )
        write(6,*)
     +      'COTAN to at least 10 digits -- X, COTAN(X) = ', C, D
c         ...
        end

        function  COTAN( X )

        real  X
c       intent(in)  X
        real  COTAN

c This external function, transformed from a program written in Fortran 8x
c (version S8.104, June, 1987) computes the COTAN(X).  This program
c compiles and runs on a Sun 3/50 using BSD 4.3.
```

```
c This algorithm used below is described in "Software Manual For The
c Elementary Functions" by W. J. Cody and W. Waite.  This program has
c been written by Brian T. Smith, Argonne National Laboratory, Sept. 1987.

c Local variables.

      integer  N
      real
     + C1, C2, F, G, SQEPS, P1, P2, P3, Q1, Q2, Q3, Q4,
     + X1, X2, XDEN, XN, XNUM, Y, YMAX

c     exponent_letter( effective_precision(X), effective_exponent_range(X) )  Q

c Local constants.

      integer  ERRNO
      parameter ( ERRNO = 6 )

      real  HALF, ONE, TOVRP, ZERO
      parameter ( HALF = 0.5 E0, ONE = 1.0 E0,
     +            TOVRP = 0.63661 97723 67581 34308 E0,
     +            ZERO = 0.0E0 )

      intrinsic  abs, aint, int, mod, real, dble, sign, sqrt

      double precision  XS
c     intent(in)  XS
      double precision  COTANS
c     intent(out)  COTANS

c This external function, written in Fortran 8x, version S8.104 (June, 1987)
c computes the COTAN(X).

c Local variables.

      double precision
     + C1S, C2S, FS, GS, SQEPSS, P1S, P2S, P3S, Q1S, Q2S, Q3S, Q4S,
     + X1S, X2S, XDENS, XNS, XNUMS, YS, YMAXS

c     exponent_letter( effective_precision(X), effective_exponent_range(X) )  Q

c Local constants.

      real  HALFS, ONES, TOVRPS, ZEROS
      parameter ( HALFS = 0.5 D0, ONES = 1.0 D0,
     +            TOVRPS = 0.63661 97723 67581 34308 D0,
     +            ZEROS = 0.0D0 )

      Y = abs(X)

      if( Y .lt. 5.87747E-39 )  then

c     Give an error message for argument error.

         write(ERRNO,*)
     +     'Argument too small in magnitude to evaluate COTAN(X)'
         COTAN = sign( 3.40282E+38, X)
         return

      endif
```

```
      SQEPS = sqrt( 9.53674E-07 )
      YMAX = aint( ONE / ( SQEPS * TOVRP ) )

      if( Y .gt. YMAX )  then

c        Give an error message for argument error.

         write(ERRNO,*)
     +     'Argument too large in magnitude to evaluate COTAN(X)'
         COTAN = ZERO
         return

      endif

c Compute the quadrant for X.

      N = int( X * TOVRP + HALF )
      XN = real( N )

c Determine  F.

      X1 = aint( X )
      X2 = X - X1

c     Dead code removed.

         C1 = 1.5703 125 E0
         C2 = 4.8382 67948 97 E-4

      F = ((X1 - XN*C1) + X2) - XN*C2

      if( abs( F ) .lt. SQEPS ) then

         XNUM = F
         XDEN = ONE

      else

         G = F * F

c   Compute XNUM = F*PG and XDEN = QG as values of low degree polynomials
c   P(G) and Q(G) whose degrees and coefficients depend on the effective
c   decimal precision of X.

c        Dead code removed.

             P1 = -0.95801 7723 E-1
             Q1 = -0.42913 5777 E0
             Q2 = 0.97168 5835 E-2
             XNUM = P1*G*F + F
             XDEN = ((Q2*G + Q1)*G + HALF) + HALF

c        Dead code removed.

      endif

      if( mod(N,2) .eq. 0 )  then

         COTAN = XDEN / XNUM
```

```
      else

         COTAN = -XNUM / XDEN

      endif

      return

      entry  COTANS( XS )

      YS = abs(XS)

      if( YS .lt. 1.1125369292536D-308 )  then

c     Give an error message for argument error.

         write(ERRNO,*)
     +     'Argument too small in magnitude to evaluate COTAN(X)'
         COTANS = sign( 1.7976931348623D308, XS)
         return

      endif

      SQEPSS = sqrt( 2.2204460492503D-16 )
      YMAXS = aint( ONES / ( SQEPSS * TOVRPS ) )

      if( YS .gt. YMAXS )  then

c        Give an error message for argument error.

         write(ERRNO,*)  'Argument too large to evaluate COTAN(X)'
         COTANS = ZEROS
         return

      endif

c Compute the quadrant for X.

      N = int( XS * TOVRPS + HALFS )
      XNS = dble( N )

c Determine  F.

      X1S = aint( XS )
      X2S = XS - X1S

c     Dead code removed.

         C1S = 1.57080 07812 5 D0
         C2S = -4.4544 55103 38076 86783 08 D-6

      FS = ((X1S - XNS*C1S) + X2S) - XNS*C2S

      if( abs( FS ) .lt. SQEPSS )  then

         XNUMS = FS
         XDENS = ONES

      else
```

```
            GS = FS * FS

c     Compute XNUM = F*PG and XDEN = QG as values of low degree polynomials
c     P(G) and Q(G) whose degrees and coefficients depend on the effective
c     decimal precision of X.

c           Dead code removed.

                P1S = -0.12828 34704 09574 3847 D0
                P2S = 0.28059 18241 16998 8906 D-2
                P3S = -0.74836 34966 61206 5149 D-5
                Q1S = -0.46161 68037 42904 8840 D0
                Q2S = 0.23344 85282 20687 2802 D-1
                Q3S = -0.20844 80442 20387 0948 D-3
                XNUMS = ((P3S*GS + P2S)*GS + P1S)*GS*FS + FS
                XDENS = (((Q3S*GS + Q2S)*GS + Q1S)*GS + HALFS) + HALFS

c           Dead code removed.

          endif

          if( mod(N,2) .eq. 0 )  then

             COTANS = XDENS / XNUMS

          else

             COTANS = -XNUMS / XDENS

          endif

          return
          end
```

The above Fortran 77 program was run on a Sun 3/50 using Unix BSD 4.3 and produced the following output:

```
COTAN to at least 4 digits -- X, COTAN(X) =   9.96687e-02  10.000000
COTAN to at least 10 digits -- X, COTAN(X) =  0.78539816339745 1.00000000000000
```

57

To: X3J3

From: B. T. Smith

Subject: Miscellaneous Revisions To S8.104 (June, 1987)

References: [1] X3J3 Standing Document S8.104 (June 1987).

1. Introduction

In checking S8.104 [1], I have discovered a few typos, missed edits, and editorial changes. Also, there are some of my edits that have been missed or ignored which are important enough to try again with some explanation of why they represent necessary changes. I have classified the recommendations as 'Editorial' (Ed.), 'Technical Changes' (Tech.), and/or 'changes for the Journal of Development' (JoD), if there is one.

Location	Recommendations
1. Page xiv (Ed.)	Add the following lines to the table of FIGURES: Figure A.1 The Fortran Language Standard ..... A-2 Figure A.2 Supplementary Standards ..... A-3 Figure A.3 Secondary Standards ..... A-4 Figure A.4 The Fortran Family of Standards A-6
2. Page xv (Ed.)	Add the following lines to the table of TABLES: Table C.1 Values Assigned to Inquire Specifier Variables ... C-7
3. Page 2-4, lines 20-21 (Ed.)	Replace 'an assignment operation' by 'a defined assignment statement'. Discussion: Nowhere else in the document can I find a reference to assignment as an operation. If we plan to use this term, it should be used throughout the description in 7.5. I believe it should be consistent and the easiest change is to change these lines.
4. Page 2-6, line 46 (Ed.)	Change 'END statement' to 'END statement'.

4. Page 2-6, line 46 (Ed.)

Change 'END statement' to 'END statement'.

Discussion: This statement is a definition and hence the term being defined should be in bold font. It will now get into the index as well.

5. Page 2-7, line 36 (Ed.)

Change '14.7.1, 11.3.1' to '14.7.1.2'.

Discussion: 14.7.1.2 is a more direct reference to use and host association.

6. Page 2-9, line 28 (Ed.)

Change 'An' (bold) to 'An' (roman font).

Discussion: 'An' is not part of the term being defined.

7. Page 2-10, line 14+ (Ed.)

Add a definition for Entity. This definition is a slight modification of the definition found in the glossary (construct name has been added):

2.5.9 Entity. The term entity is used for any of the following: a program unit, a procedure, an operator, an interface block, a common block, an i/o unit, a statement function, a type, a named variable, an expression, a component of a type, a named constant, a statement label, a construct, a construct name, an exponent letter, a namelist, or a range list.

Discussion: This is a term used throughout the text and is used with a different meaning than is given in Fortran 77. It is an important "near-catch-all" term that needs to be defined because it does not catch all possible items — for example, specification statments, shape, exponent letter, input/output unit, and so on. If these "things" that are not included should be included or if "things" such as construct, interface block, etc. should not be included, then let us get the correct definition inserted in this place in the text.

8. Page 4-2, line 3 (Ed.)

Replace 'operator functions' with 'functions with the OPERATOR option'.

Discussion: The term 'operator functions' is not defined anywhere. As it is used in the quoted context, it reads like a term with a special meaning. But all that is needed at this point is

the understanding that we are referring to a function with another option associated with it.

9. Page 4-4, line 25, 29 (Ed.)

Delete 'the' on line 25. On line 29, change the letter 'O' (the letter oh) to the digit '0' (digit zero).

10. Page 4-6, line 46 (Ed.)

Change 'PRIVATE,' to 'PRIVATE, then'.

Discussion: 'then' is helpful to indicate how to parse this sentence before you reach the end of it.

11. Page 4-7, line 25 (Ed.)

Delete line 25.

Discussion: These parameters do not seem to be used anywhere.

12. Page 5-2, line 25+ (Ed.)

Insert the following sentence as a separate paragraph:

An assumed type parameter is a type parameter of a dummy argument whose value is specified as an asterisk.

Discussion: The term 'assumed type parameter' is used on page 7-7, lines 40-42, and page 12-2, lines 35-36 but is not defined. The above change inserts the definition where type parameter values are specified. This will also define an assumed length parameter but not an assumed bound (see recommendation below on page 5-8, line 14+.)

13. Page 5-6, line 37 (Tech.)

After 'ment', add 'of a subprogram'.

Discussion: Otherwise, one is allowed to specify the INTENT(IN) attribute for the dummy arguments of a statement function. Is it intended that this be allowed?

14. Page 5-8, line 14+ (Ed.)

Insert the following sentence as a separate paragraph:

An assumed bound is a lower or an upper bound of an *assumed-shape-spec*.

Discussion: The term 'assumed bound' is used on page 7-7, lines 40-42, but is not defined. The above change inserts the definition where assumed shape arrays are specified.

15. Page 5-11, line 16 (Ed.)

Change 'use-name-list' to '*use-name-list*'.

Discussion: A BNF term is always italicized when used in the text. Why not here as well?

16. Page 5-11, line 36 (Tech.)

After 'appears', insert 'except in the scoping unit of the main program unit'.

Discussion: This change is required to remain compatible with Fortran 77. The original statement implies that a SAVE statement is required in the main program for any common block that is saved in a subprogram. This is not required in Fortran 77 (cf. X3.9-1978, page 8-11, lines 1-4, note the use of subprogram)

17. Page 5-13, line 11 (Ed.)

After 'a deferred-shape array,', add 'an alias scalar object,'.

Discussion: A deferred-shape array includes an alias array but not an alias scalar object. This should also be changed on page D-10, line 10.

18. Page 5-13, lines 34-37 (Tech.)

Delete 'Note that if an object ... constant'.

Discussion: This statement in lines 34-37 is currently in Fortran 77. I am recommending that the statement be deleted. An example of this situation arising is as follows. Suppose a high precision variable (that is, higher precision than default real) is being data initialized to a constant (say, .2) which cannot be represented on the machine exactly (no binary representation can represent .2 exactly, for instance). According to X3.9-1978, page 9-2, lines 14-17, the processor is allowed to supply a value of .2 that is as precise as appropriate for double precision rather than the single precision value.

I find the quoted Fortran 77 (and Fortran 8x) statement very difficult to agree with. The previous statement in X3.9 says that the intrinsic function DBLE (from Table 4) is to be used for the conversion. (The similar statement is made in S8.104,

page 5-13, lines 38-40.) Thus, one statement says that the result is to be DBLE(.2) and the other states that 0.2D0 can be the value of the data initialized item. These are typically different results. Therefore, I find it difficult to justify the original Fortran 77 statement.

Both in Fortran 77 and Fortran 8x, all precision requirements are minimum; that is, a processor is allowed to supply a more precise answer than is required by the minimum requirements. In many cases, there are no solid precision requirements. Why is data initialization being singled out as a place where the processor is allowed to provide more precision than is implied by the constants?

If there is some reason why it is important to retain this statement, then I would like to see it generalized. That is, replace 'double precision real' by 'real of higher precision than that of the constant' on lines 35-36, page 5-13. Or must we all remember this strange exception applies only to double precisions items in the data initialization list!

19. Page 5-16, lines 23-48 (Ed.)

Change all comments to upper case. Similarly, on lines 2-4 of page 5-17, change the comments to upper case.

Discussion: The recognition of lower case letters by the processor is optional, and should not be used in examples in the text of the draft standard.

20. Page 6-3, line 17 (Ed.)

Before 'bounds', insert 'declared'.

Discussion: In the previous two sentences, both effective and declared bounds are used; the phrase 'the bounds' is ambiguous.

21. Page 7-7, lines 41-42 (Tech.)

Change 'not assumed or allocated' to 'defined and not assumed'.

Discussion: This case must cover alias arrays and ranged arrays as well as allocated arrays. Since in these cases the declared or effective bounds are not defined, this change is better than listing all of the cases.

22. Page 7-9, line 19 (Tech.)

After 'is', insert 'either', and after 'expression', insert 'or a variable whose type parameters or bounds inquired about are defined and not assumed'.

Discussion: This change makes a constant expression an instance of a restricted expression which is what I think was intended.

23. Page 7-18, line 13 (Ed.)

Change '2.4.5' to 'Section 6'.

Discussion: This reference is for the BNF term *variable* rather than the prose definition. The BNF rule R601 is in Section 6.

24. Page 7-18, lines 22-23 (Tech.)

Delete 'and have the same type parameter values'.

Discussion: The text on page 7-20, lines 10-12, defines assignment when the type parameter values are not the same.

25. Page 7-18, lines 40-41 (Tech.)

Delete 'and type parameter values'.

Discussion: The text on page 7-20, lines 10-12, defines assignment when the type parameter values are not the same.

26. Page 7-19, lines 17-20 (Tech.)

Delete lines 17-18 and 'DBLE,' on line 20, provided the definition of the intrinsic function REAL is changed as suggested immediately below in the change for page 13-39, lines 35-36+.

Discussion: The definition of the intrinsic function REAL as it is currently written allows an implementation to return different results for DBLE(R) and REAL(R,MOLD=D) where D is declared of type double precision real. Although an implementation is unlikely to implement these functions differently, it is an irregularity that I see no advantage in permitting. Once this change is made, there is no need for lines 17 and 18 of Table 7.11, and in fact, including them suggests that DBLE(R) and REAL(R,MOLD=D) are different. There is no incompatibility with Fortran 77 as REAL(R) in Fortran 77 and Fortran 8x must return a value of type default real.

27. Page 13-39, lines 35-36+ (Tech.)

After 'real', insert 'except double precision real'. After line 28+, insert 'Case (ii): If A is of type double precision real, the result is equal to DBLE(A).' and renumber Case (ii) as Case (iii).

28. Page 8-1, line 31 (Ed.)

Change the '12.4.4' to '12.4.3, 12.4.4, 12.4.5'.

Discussion: All procedure references need to be included. Otherwise, this suggests that elemental function references and elemental assignments are not permitted.

29. Page 8-9, line 23 (Ed.)

Replace 'Statement Labels' with 'A Statement Label', and 'Statement labels provide' with 'A statement label provides'.

Discussion: There is no need to make this definition plural. Rewriting it so that it is singular is both clearer and more consistent with nearly all of the other definitions.

30. Page 8-10, lines 3, 7-8 (Ed.)

After '*format-stmt*', insert 'that appears in the same scoping unit as the *assign-stmt*'. Delete the sentence: 'The statement label ... statement.' on lines 7-8.

Discussion: This makes the BNF rules R828 and R829 consistent.

31. Page 9-1, lines 12-16 (Ed.)

Change 'and' to 'or' in lines 12, 13, and 15, 'are' to 'is a' in lines 13, 14, and 15, and 'statements' to 'statement' in lines 13, 14, and 16.

Discussion: There is no need to make this definition plural. Rewriting it so that it is singular is both clearer and more consistent with nearly all of the other definitions.

32. Page 9-12, lines 28-29 (Ed.)

Delete 'The labeled statement ... statement.'.

Discussion: As stated, it applies only to the situation when an error condition occurs. The constraint has been stated in a more general location, namely page 9-10, lines 25-27.

33. Page 9-12, lines 28-29,36-37 (Ed.)

Delete 'The labeled statement ... statement.' twice.

Discussion: As stated, it applies only to the situation when an error condition occurs. The constraint has been stated in a more general location, namely page 9-10, lines 25-27.

34. Page 10-1, lines 20-24 (Ed.)

Indent these lines another inch.

Discussion: This change in layout should also be done to page D-21, lines -5 to -9.

35. Page 10-10, lines 28-31 (Ed.)

On lines 28-29, change 'nonleading blank characters are interpreted as zeros or are' to 'any nonleading blank character is interpreted as zero or is'. Also on lines 30-31, change 'all nonleading blank characters in succeeding numeric input fields are' to 'any nonleading blank character in succeeding input fields is'.

Discussion: The singular case is easier to understand.

36. Page 11-6, line 14 (Ed.)

Change 'BLOCK DATA' to 'block data'.

Discussion: The term 'block data program unit' is introduced in the section on program unit concepts (page 2-3, lines 42-47) using the lower case spelling. That spelling should be used here.

37. Page 12-1, line 29 (Ed.)

After '**function**', insert '(12.5.4)'.

Discussion: Why not give a forward reference to the section on statement functions? It saves someone the effort of looking for it by scanning for the section in the table of contents or looking for it in the index.

38. Page 12-2, lines 7-8 (Ed.)

The condition 'Where a type ... expression,' is always satisfied in the sense that any variable or constant is an expression. Replace this condition by 'Where a type parameter or bound of an array is an expression that depends upon the value or attributes of another object,'.

Discussion: The point of this phrase apparently is to indicate that the formula for the dependence is a characteristic when there is a formula. That is, the condition is trying to eliminate the case were the type parameter is specified as an * or an array bound is specified as a :. The 'that' clause is the same used on page 12-1, lines 40-41.

39. Page 12-7, line 47 (Ed.)

Replace 'same shape as' by 'type and type parameters (if any) of the *function-name* (12.5.2.2) and the same shape of'.

Discussion: If this change is not made, the type and type parameters of an elemental function reference are not specified anywhere. An appeal to "reasonableness" might be used, but it should be stated specifically.

40. Page 12-8, line 3 (Ed.)

Replace 'the reference' by 'a reference'.

Discussion: The original text suggests that there can be only one kind of reference in a scoping unit or executable program or in any context.

41. Page 12-9, line 30 (Ed.)

Change 'attribute' to 'attributes'.

Discussion: The function result can have more than one attribute; for example, it can have an array attribute, an allocatable attribute, or a type-spec attribute.

42. Page 12-11, line 21 (Tech.)

After 'subprogram.', insert 'The ENTRY statement may have no parentheses if there is no dummy argument list but the function can only be referenced as *entry-name*().'.

Discussion: This is a Fortran 77 restriction, unless the intention is to extend Fortran 77.

43. Page 12-11, lines 42-43 (Ed.)

Replace 'END statement, END FUNCTION statement, or END SUBROUTINE statement' by '*end-function-stmt* or *end-subroutine-stmt*'.

Discussion: The END statement includes the others as well as the *end-program-stmt* which behaves like a STOP statement.

44. Page 13-1, line 39 (Ed.)

After 'The', insert 'transformational'.

Discussion: After the change, the first two sentences will be parallel.

45. Page 13-2, lines 43-44 (Ed.)

Change the sentence 'The function TRANSFER ... argument.' to 'The function TRANSFER specifies that the physical representation of the first argument is to be treated as if it were of the type of the second argument with no conversion.'.

Discussion: The current text reads as if some conversion occurs. The recommended text is clearer.

46. Page 13-4, line 33 (Ed.)

After 'dimension', add the sentence: 'In all cases, the size, shape, or bounds of the arrays inquired about must be defined.'.

Discussion: Allocated arrays before they are allocated have no declared or effective bounds. Similarly, a ranged array has no effective bounds before it is the subject of a SET RANGE statement. The above prohibition is stated here in this general way to avoid stating the restriction in the individual sections describing each array inquiry function.

47. Page 13-5, line 22 (Ed.)

Change 'keyword arguments' to 'argument keywords'

Discussion: My notes indicate that this change was made at Meeting 104.

48. Page 13-7, line 13 (Ed.)

Indent 'of' three spaces.

49. Page 13-8, lines 44-45 (Ed.)

After 'restarts' on line 44, add 'the'. On line 45, indent 'pseudorandom' three spaces.

50. Page 13-15, line 18 (Ed.)

Replace the sentence 'If X is complex, ... AIMAG(X).' with the sentence 'If X is complex and MOLD is absent, the result is CMPLX(REAL(X,X),AIMAG(X)); if X is complex and MOLD is present, the result is CMPLX(REAL(X,MOLD),AIMAG(X),MOLD).'.

Discussion: The original statement contradicts the restriction on Y when X is complex (see lines 11-12, page 13-15). Although replacement text is somewhat verbose, it does not introduce a contradiction.

51. Page 13-18, line 26 (Ed.)

After 'Time', insert '(GMT)'.

Discussion: GMT is used as a short form in the example. It takes so little effort to include the abbreviation for those not familiar with it. On the other hand, this standard is full of special terms that it could conceivably be one such term not recognized by the reader.

52. Page 13-32, lines 34,42 (Tech.)

Before 'extent', insert 'effective'. Similarly, on page 13-35, lines 9 & 17.

Discussion: In the description of the result value, it is not clear whether the extent used is the declared or effective extent. I believe that the effective extents are the only sensible choice but it requires some understanding of how it is implemented to make this conclusion.

53. Page 13-33, lines 8-9 (Ed.)

Delete 'Its shape must be defined.'.

Discussion: This restriction for MAXVAL is stated in a general way in 13.7.1 and has been deleted in most places in the subsections of 13.12, in particular for the description of the function MINVAL.

54. Page 13-33, line 31 (Ed.)

Change 'GT' to 'LT' and 'positive' to 'negative'.

Discussion: The changed example makes more sense with respect to the MAXVAL operation. The current example is meaningless, except when there are no positive elements in C in which case it returns a very negative number -- surprise!!!

55. Page 13-37, lines 33-34 (Ed.)

Delete 'Its shape ... defined.'.

Discussion: It is not set in SUM so why here.

56. Page 13-35, line 25 (Ed.)

After the period, add 'RESHAPE ([2,4], [1:6], [0,0], [2,1]) has the value

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 0 & 0 \end{pmatrix}$$

Discussion: This insertion adds an example that illustrates the use of the optional arguments.

57. Page 13-46, lines 34-35 (Ed.)

Delete 'Its shape ... defined.'.

58. Page 13-47, lines 16-17 (Ed.)

Delete 'Its shape ... defined.'.

Discussion: This restriction is not stated for the inverse operation PACK. Why is it said here? It has already been said in a general place in 13.7.1.

59. Page 14-1, line 4 (Ed.)

After 'tion.', insert 'An entity that has a scope is either a name, a label, an exponent letter, an external input/output unit, an operator symbol, or an assignment symbol.'.

Discussion: The added sentence gives the reader some help in understanding to what the concept of scope applies. For example, it does not apply to constants and expressions.

60. Page 14-2, lines 10,14 (Tech.)

After 'parameters', insert 'with neither an asterisk'.

Discussion: If one of the arguments has a type parameter value of asterisk and the other is a constant, the procedure reference cannot be resolved in some cases.

61. Page 14-3, lines 29,33 (Tech.)

After 'parameters', insert 'with neither an asterisk'.

62. Page 14-5, line 20 (Ed.)

Insert 'S' before 'tatement'.

63. Page 14-5, line 38 (Ed.)

Delete 'or array element'.

Discussion: An array element is a variable. This adds nothing but confusion.

65. Page 14-6, line 7 (Ed.)

After first 'are', insert 'storage'.

Discussion: This makes it clearer that this statement is not discussing other kinds of association such as argument or name association.

66. Page 14-6, lines 19-21 (Tech.)

On lines 19 and 21, insert 'default' before 'complex'. On line 20, change 'integer, real' to 'integer, default real'.

Discussion: Storage association is not permitted for specified precision real or complex data objects, yet as originally stated, there appears to be some way that they can be storage associated.

67. Page A-3, line -2 (Ed.)

On the second line from the bottom of the page, before the first 'Supplementary', insert 'Procedure'.

Discussion: I assume that the title on the left is to be contrasted with the title on the right, namely 'Module Supplementary Standards'.

68. Page B-1, lines 30-44 (Ed.)

Change '*SUBR_NAME*' to roman rather than italic font on lines 30 and 42. Change '$return_1$' to '1' in roman font on line 44. Use the program font for lines 30-39 and lines 42-44.

69. Page B-2, lines 1-4 (Ed.)

Change '$return_2$' to '2' in roman font. Use program font on lines 1-4.

70. Page B-2, line 35 (Ed.)

Change 'Assigned' to 'assigned'.

Discussion: This change is recommended in order that the spelling is consistent with the same phrase on page B-1, line 21.

71. Page B-5, line 20 (Ed.)

Delete 'below'.

Discussion: No where else is the direction of the reference given. If the page break ever occurs between here and the reference section, the word is wrong.

Discussion: No where else is the direction of the reference given. If the page break ever occurs between here and the reference section, the word is wrong.

72. Page C-3, lines 37-38 (Tech.)

Change 'remains undefined' to 'is not standard-conforming'.

Discussion: Fortran 77 (Page 5-5, lines 9-17, X3.9-1978) says that the subscripts must be in bounds for the array. Thus, any program that violates this prohibition is not standard conforming. 'Undefined' is a weaker term. Has the rule in Fortran 8x changed?

73. Page C-8, lines 41,43 (Ed.)

Delete 'count' twice.

Discussion: The juxtaposition of two nouns 'value' and 'count' is awkward and unnecessary.

74. Page C-11, line 22 (Ed.)

Change 'in USE' to 'in the USE statement'.

Discussion: 'in USE' is a short hand for 'in the USE statement' which is a shorthand phrase used nowhere else. Why use it here?

75. Page C-12, lines 48-50 (Tech.)

Replace lines 48-50 with the following line:

```
ELEMENT = ANY( A%ELEMENT_VALUE( 1:CARD(A) ) .EQ. X )
```

Discussion: Particularly in program examples, unnecessary statements always make the reader question whether the statements are needed for some unsuspected reason. As ANY returns .FALSE. for zero-sized arrays, there is no need for the test on the size of A. Why confuse everyone with superfluous statements?

76. Page C-14, lines 9-10 (Tech.)

On line 9, delete 'A = SETF([1:0]); '. On line 10, delete 'IF (ESIZE (A % ELEMENT_VALUE) .GE. N) '.

Discussion: Particularly in program examples, unnecessary statements always make the reader question whether the statements are needed for some unsuspected reason. A and B are of the same type (200 is the fixed size of the array component

of SET); A is not ranged in the subroutine SET_ASSIGNMENT_COERCION and thus ESIZE(...) in the above statement always returns 200. 200 is always greater than or equal to N (or B is incorrectly defined at some earlier step). Therefore ESIZE test is unnecessary. Why confuse everyone with superfluous statements?

77. Page C-17, lines 13-14 (Ed.)

Change '"rank" (number of dimensions) and "shape" (set' to '"shape" (number of dimensions and set'.

Discussion: This looks like the definition of shape, which it is not. Shape is defined as both the number of dimensions and the size of each dimension. The original statement contradicts the definition of shape.

78. Page C-17, line 20 (Ed.)

Delete 'effective'.

Discussion: The term is too easily confused with the specialized meanings of effective range and effective bounds. Let us not confuse readers unnecessarily.

79. Page C-18, line 25 (Ed.)

Change 'Space allocation query' to 'Array allocation status'.

Discussion: This makes the description the same as on page 13-7, line 46.

80. Page C-19, line 6 (Ed.)

Change 'do' to 'perform'.

Discussion: 'do' is colloquial. Let's use a more precise term.

81. Page C-20, line 36 (Ed.)

Change '(N, N, N)' to '(N)'.

Discussion: The function RAND is a function of a single argument N that returns a rank-3 array of shape [N,N,N] -- cf. page C-21, lines 46-49. Note that the name RAND is a local variable in the function RAND and is an automatic array of shape [N,N,N].

82. Page C-21, line 37+ (Ed.)

After line 37, insert the following text:

The intrinsic subroutine RANDOM can be used to generate the random array as follows:

```
FUNCTION  RAND (N)
INTEGER  N
REAL :: RAND (N, N, N)
CALL  RANDOM (RAND)
RETURN
END
```

Discussion: A pseudorandom number generator is an intrinsic function that is available for this application. Why not illustrate how it can be used in this application?

83. Page C-21, line 44-45 (Ed.)

Change the case of the letters in the comments to upper case.

84. Page C-22, line 15+ (Ed.)

After line 15, insert:

```
FUNCTION  RAND (N)
INTEGER  N
REAL :: RAND (N, N, N)
CALL  RANDOM (RAND)
RETURN
END
```

85. Page H-1, line 7 (Ed.)

Delete 'subprogram'. Replace the sentence 'Such entities ... accessible.' with the sentence: 'In addition, a scoping unit that is an internal subprogram, module subprogram, or derived-type definition accesses entities from its host scoping unit.'.

86. Page H-2, line 42 (Ed.)

Replace 'i/o' with 'input/output'.

87. Page F-2, line 10 (JoD,Tech.)

After '10.', insert the sentence:

Bit assignment is defined in F.1.1.5.13.

Discussion: Currently, the referenced section is F.1.1.5.12. When the recommended section on the general form of expres-

sions (see below) is added, then it will become subsection number 13.

88. Page F-3, lines 3-8 (JoD,Ed.)

Use "program" font for the text in the first column as is used in the corresponding tables of Section 7. Similarly, use "program" font on lines 17-18, 35-39; on page F-4, lines 10-14, 24-25; and on page F-5, lines 3-5, 22-27.

89. Page F-4, lines 3-6 (JoD,Ed.)

Change 'RF11', 'RF12', 'RF13' to 'RF03', 'RF04', 'RF05', respectively.

Discussion: These production rules are duplicates of production rules from the page F-1.

90. Page F-4, lines 19, 30 (JoD,Ed.)

On line 19, change 'RF14' to 'RF11'. On line 30, change 'RF15' to 'RF12'.

91. Page F-5, lines 10,11,13 (JoD,Ed.)

On line 10, change 'RF16' to 'RF13'. On line 11, change 'RF17' to 'R709'. On line 13, change 'RF18' to 'RF14'

92. Page F-5, line 29+ (JoD,Ed.)

After line 29, add the following section:

F.1.1.5.9 General Form of an Expression. Expressions are level-6 expressions optionally involving defined binary operators. Rule R712 must be extended as follows:

RF15 *expr* **is** [*expr defined-binary-op*] level-6-expr

R322 *defined-binary-op* **is** . *letter* [*letter*] .

Simple examples of an expression are:

Example	Syntactic Class
A	*level-6-expr* (RF13)
D .BAND. E	*level-6-expr* (RF13)
B .UNION. C	*expr* (RF14)

More complicated examples of an expression are:

4/61

```
(B .INTERSECT. C) .UNION. (X - Y)
A+B .EQ. C * D
.INVERSE. (A + B)
A+B .AND. C * D
E // G .EQ. H(1:10)
```

93. Page F-5, lines 30-34 (JoD,Ed.)

Before 'A' on line 30, insert the heading '**F.1.1.5.10 Bit Intrinsic Operations.**'. On lines 30-31, replace '**, character intrinsic ... logical intrinsic operation** are similarly' with 'is'. On line 32, change 'and' to 'or'. On lines 32-34, replace '*character intrinsic* ... respectively' with 'similar to the definition of numeric intrinsic operation'.

Discussion: This heading is missing from the original text when it was taken from S8 to Appendix F in the compromise.

94. Page F-6, line 47 (JoD,Ed.)

After 'shape.', insert 'Table F.1 gives the permitted uses of operands of type bit for the intrinsic operations.'.

Discussion: A reference to Table F.1 is needed and this provides it, similar to the way it is done in Section 7.

95. Page F-7, line 10-10+ (JoD,Ed.)

Change 'operands' to 'definitions'. After line 9, insert the paragraph:

A bit relational intrinsic operation is interpreted as having the logical value true if the values of the operands satisfy the relation specified by the operator. A bit relational intrinsic operation is interpreted as having the logical value false if the values of the operands do not satisfy the relation specified by the operator. The bit value B'0' is less than the bit value B'1'.

Discussion:

96. Page F-7, line 17 (JoD,Ed.)

Change '7.1.1' to 'F.1.1.5.9'.

97. Page F-8, line 5 (JoD,Ed.)

After 'statements', insert ', appearing in masked array assignments'.

98. Page F-8, line 19 (JoD,Ed.)

On line 11, change 'The shape' to 'The effective shapes'.

99. Page F-9, line 6 (JoD,Ed.)

After 'false.', insert: 'Rules R803 and R804 must be extended.' and after line 44, insert the lines:

RF16 *if-then-stmt* is [*if-construct-name* :] □
□ IF (*scalar-mask-expr*) THEN

RF17 *else-if-stmt* is ELSE IF (*scalar-mask-expr*) THEN

100. Page F-9, line 8 (JoD,Ed.)

After 'false.', insert: 'Rule R807 must be extended.' and after line 2, insert the line:

RF18 *if-stmt* is IF (*scalar-mask-expr*) *action-stmt*

101. Page F-9, line 24+ (JoD,Ed.)

After line 24, insert lines 37-39 of page 8-3, changing 'logical' to 'logical or bit'.

102. Page F-9, line 25 (JoD,Ed.)

Make line 25 a constraint and change 'bit' to 'logical or bit'.

103. Page F-12, line 32+ (JoD,Ed.)

After line 32, insert the constraints:

Constraint: A name must not occur more than once in a *type-param-name-list*.

Constraint: If either PRECISION or EXPONENT_RANGE occurs in a *type-param-name-list*, both must occur.

Discussion: The above two constraints are those of R417 which is repeated in the above text without its constraints.

104. Page F-12, line 39+ (JoD,Ed.)

After line 39, insert:

Constraint: Each bound in the *explicit-shape-spec* (5.1.2.4.1) must be a nonprecision type-parameter expression (7.1.6.2).

Constraint: An *access-spec* or a PRIVATE statement within the definition is permitted only if the type definition is within the specification part of a module.

Discussion: The above two constraints are those of R419 which is repeated in the above text without its constraints.

105. Page F-13, lines 21-22 (JoD,Ed.)

Change 'Each ... statement.' to 'For a given CASE construct or *variant-part*, each *case-value* must be of the same type as the *component-name* preceding the CASE construct or *variant-part*, respectively.

Discussion: The above change makes the text the same as for R815 which is being extended as RF25.

106. Page F-20, line 16 (JoD,Tech.)

After 'point.', add 'This extension is so designed that exceptions need be monitored by the processor only if they are declared, are capable of being recognized by the processor, and are raised in specific blocks of statements or in procedures called from such blocks of statements.'.

Discussion: I am recommending this sentence be added to give the reader some help in understanding the nature of the exception handling extension.

107. Page F-20, line 39 (Jod,Ed.)

Change 'an allocation requested by an ALLOCATE statement (6.2.2).' to 'an allocation or deallocation requested by an ALLOCATE statement (6.2.2) or DEALLOCATE statement (6.2.3) and the STAT= specifier is not provided. If this condition is enabled, it may be handled as described below instead of causing immediate termination of the executable program.'.

108. Page F-22, line 43 (Jod,Ed.)

Replace 'the dummy condition associated with an actual condition' with 'the condition is present (12.5.2.8)'.

Discussion: The phrase 'associated with an actual condition' is incorrect as the actual argument may be a dummy argument of the invoking procedure which corresponds to an optional dummy argument whose actual argument is not present. The 'presence' of an argument is carefully defined in 12.5.2.8

to avoid this ambiguity and therefore should be used here.

109. Page F-25, lines 23-24 (Jod,Tech.)

Replace the sentence 'A condition must not ... main program.' with the sentence 'The handling of a condition propagated from a main program is processor dependent.'.

Discussion: As stated, a processor appears to be prohibited from giving any kind of trace or diagnostics of any unhandled conditions. I think the above wording avoids the question and corresponding concern and states what was intended.

110. Page H-6, line 11 (JoD,Ed.)

After 'subscript', insert ' (F.2.2)'. Delete 'in a named circumstance when' and replace with 'when a specified event or circumstance is detected and'.

111. Page H-6, line 13 (JoD,Ed.)

After 'signal', insert ' (F.4.1.3)'. Delete 'in a named circumstance when' and replace with 'when a specified event or circumstance is detected and'.

112. Page H-6, line 15 (JoD,Ed.)

After 'subscript', insert ' (F.2.2)'.

113. Page H-6, line 14+ (JoD,Ed.)

Insert after line 14:

tag component (F.1.2.1). The (non-variant) component of a derived type immediately preceding the variant part.

variant component (F.1.2.1). The component of a derived type corresponding to a component name in the variant part.

variant part (F.1.2.1). The part of a derived type that begins with SELECT CASE and ends with END SELECT.

variant structure (F.1.2). A structure whose derived type has a variant part.

106(21)JLS/JSM-1

58

To: X3J3

From: Lawrie Schonfelder and Steve Morgan

Subject: Pointer Functionality Sans Data-Type

References: 105(*)CQB-3 Document 21
96(*)JKR-6
90(9)KWH-11
Lindsay and van der Meulen
"Informal Introduction to Algol-68" North-Holland (1977)
Woodward and Bond
"Guide to Algol-68RS" Edward Arnold (1983)
Hogan
"The C Programmer's Guide" Prentice Hall (1984)
Kernigan and Ritchie
"The C programming Language" Prentice Hall (1978)
Wirth
"{{{{Pascal}}}}"
~~{{{{{ADA book}}}}}~~ ADA Reference Manual ANSI/MIL-STD-1815A-1983

(There have been many papers on this subject by Kurt Hirchert John Reid and myself, among others. A pipe burst at my home last year resulted in the destruction of my back copies of the relevant minutes, so this has meant I have had to work from memory except for the above references and from the discussions in subgroup at the Liverpool meeting)

Introduction:

As a result of discussion by the committee of the paper 105(*)CQB-3 at the Liverpool Meeting and the straw votes taken the Data Concepts Subgroup (21) was asked to investigate in detail the provision of most of the functionality of pointers by the suggested approach. That is, by relaxing the restrictions on the use of the attributes ALLOCATABLE and ALIAS, and using the facilities of IDENTIFY and ALLOCATE/DEALLOCATE. This paper is the outcome of lengthy discussions that were held in subgroup. The paper is a general description of the facility and the required syntax and semantics to implement it consistently in F8x. This Paper does not contain detailed "S8" edits. These will be extensive and intricate. It was not considered reasonable to attempt these until a high probability of consensus in X3J3 was achieved.

The general approach of both 105(*)CQB-1 and 96(*)JKR-6 was to allow ALLOCATABLE and ALIAS attributes to be used together and to allow both to be specified for components of a derived type in the definition of the type. A limited degree of recursion was suggested in derived types by both papers. Namely, that a derived type definition could refer to itself but only indirectly through a pointer. This is the basis of this proposal.

4/67

It rapidly became clear in subgroup discussion that in order to fill in the detail a significant number of other restrictions in the existing language needed to be relaxed or changed as well. This proposal is an attempt to follow the general approach but to provide as much of the functionality of pointers as possible without introducing them as an additional data type. The proposal has been produced as the result of many hours of detailed thought and discussion, and the trying of many different detailed options. All but the one presented here were rejected. Some because they produced unworkable syntax and others because we considered the implementation consequences unacceptable. We have not been able to cover all these options in the paper nor give the detailed reasons why we did not continue with them. To have done so would have produced a paper of impossible length and we did not have enough time to write such a document. We think the proposal and the model discussed below provides a high functionality consistent and implementable pointer facility entirely within the spirit of F8x and similar to many suggestions made before by other authors.

In discussion with a number of committee members it became clear that a major issue with pointers was the potential deleterious effect on optimization the presence of pointers might have if unconstrained pointer assignment was allowed. There is also the well known problem of the "dangling pointer" which can occur if a pointer is associated with an object whose lifetime is less than the pointer. After trying a number of possible approaches to resolve these difficulties, none of which worked, this proposal follows the line taken by a number of other languages. It ignores the potential optimisation problem which is in most cases an over stated difficulty, and it simply defines the dangling pointer as something that can only be produced by a non-conformant program; in time honoured Fortran fashion. All attempts to make dangling pointers impossible syntactically and to restrict the effect of pointers on certain existing common optimisations resulted in removing one or other of the major reasons for having pointers in the first place.

The Model:

It is very difficult, if not impossible, to describe or discuss pointers without employing some underlying model of what in fact is going on. (The same is true of the existing ALIAS/IDENTIFY and ALLOCATABLE/ALLOCATE.) It was found to be essential to have a clear agreed model that could be used during discussions of acceptability; particularly when these were about implementability and effect on optimisation.

The model which is the basis of all the subsequent discussion and which was the only viable one considered and agreed by the subgroup is summarised below.

1. When an object is declared with either an ALLOCATABLE or an ALIAS attribute, what is created by the declaration is a descripter for the appropriate actual object. For simple objects the descripter would be a simple address but for more complex objects it would be more complex, for arrays being a dope-vector

2. Such a descripter is associated with an actual object, i.e. the descripter is filled in so that it refers to genuine space, either by allocating new space using an ALLOCATE statement, or by assigning values describing existing space using the IDENTIFY statement. Note: this means that it would be invalid to identify an alias with any object which was not currently associated with space.

3. An ALLOCATABLE,ALIAS component, a pointer component, of a derived type will be represented by the space to hold the relevent descripter in any object of that derived type.

4. When a pointer name appears in an expression it is automatically dereferenced so that the value of the object contained in the space currently associated with the pointer is delivered.

5. A compound object may not be dereferenced. However a pointer to an array or a structure is not a compound object and it can and will be dereferenced to deliver the array or the structure that are its current target. A structure containing pointer components cannot be dereferenced since the structure itself is not a pointer, only the individual components that are pointers can be dereferenced. Likewise an array of pointers cannot be dereferenced only a scalar single pointer, an array must be subscripted before dereferenced.

6. Assignment copies values (at least conceptually). Descripters are copied by IDENTIFY. When a pointer appears on the left of an assignment it is dereferenced and the value on the right is copied into the space associated with the pointer. However when a structure is assigned, non-pointer components have their values copied but pointer components are identified, the descripters stored in these components are copied by the default assignment.

7. The "heap" management model used is that when space is allocated it is taken from the top of the heap. Space becomes free only when no pointer is left associated with is. No attempt is made to mark space as free as soon as it becomes inaccessable. When insufficient space remains on the heap to satisfy an allocation request a garbage collect process is invoked. This searches all pointers targeted on the heap and compacts the space they address, leaving the free space on the top once again. This is by no means the only viable space management model that could support the semantics of this proposal but it has the virtue of being simple to describe and having a low running overhead until garbage collection is caused.

The Proposal:

In this section we give an outline description of the points in the language that need to change, or that we are proposing should change to implement this proposal.

1. The attributes ALLOCATE and ALIAS should be allowed for any declaration. They should not be restricted to only declaration of arrays as at present. The same set of objects may be declared to be both ALLOCATABLE and ALIAS at the same time.

2. An allocatable object is initially undefined and not associated with a definable object. It can be associated with a definable object by an ALLOCATE statement. An ALLOCATE statement creates a new object with the appropriate attributes, and causes it to be associated with ~~with~~ the name of the allocatable object specified in the statement.

3. An alias object is initially undefined and not associated with any definable object. It can be associated with an existing definable object by an IDENTIFY statement.

4. The processor is not required to preserve any information of the route by which an association by identification is established. Even if an alias is associated with a definable object via a chain of previously identified aliases the association is always between an alias and a definable object. Two aliases which happen to be associated with the same object are associated independently and either association may be broken without effecting the other.

5. The ALLOCATABLE and ALIAS attributes may be specified for components of a derived type in the definition for the type. In this case both must be specified. (It would actually be possible for ALIAS to be usefully specified without ALLOCATABLE. It would not be possible to sensibly manipulate a structure containing an allocatable but not identifiable component).

6. An object having both the ALLOCATABLE and ALIAS attributes is refered to as a pointer to allow for greater brevity in description. However most of the properties of such objects stem from their possession of the allocatable and alias attributes.

7. The restriction of the ALLOCATE statement to the creation of arrays must be removed so as to allow scalar objects to be allocated. Since the association of a pointer with a target object can be changed both by allocation and identification there appears to be little point in retaining the restriction that a object must be explicitly deallocated before it is allocated again. The proposal is that an ALLOCATE statement always creates a new definable (but undefined) object and associates the allocatable object with it. Any previous association of the allocatable object, whether as a pointer or not, is removed by the ALLOCATE statement.

8. Two forms of the IDENTIFY statement are permitted and should be

distinguished in the description. The first should be a simple identify which merely establishes an association between an alias and a definable object of the matching attributes without any array mapping. The second should cover the situation where the alias is associated with a definable object by a mapping of the target subscripts. The restriction of the alias to single hosts should not apply. Any definable object whose attributes, type and rank, match those of the alias may be associated with it by an IDENTIFY statement. Any existing association of the alias is removed by the IDENTIFY. If two aliases are associated with the same definable object these associations are considered to be entirely independent.

9. The ALIAS attribute should be specifiable, as is the allocatable attribute for dummy arguments. Such an argument can only be matched by an actual argument which is also specified to have the corresponding attributes. The state of association with a definable object of the dummy argument becomes that of the actual argument when the procedure is invoked. The state of association of the actual argument becomes that of that of the dummy argument when a return is executed. Where a dummy alias has been identified with a local target that is not saved the association must be broken before a RETURN from the procedure is executed.

10. A pointer may become associated with a target definable object by either allocation or identification. Also a pointer dummy argument could be associated with an actual argument that was identified rather that allocated. The question therefore becomes not is a pointer ALLOCATED but is it ASSOCIATED. We are therefore proposing this change in spelling for this function, and defining it to return true provided an allocatable or alias object is currently associated with a definable actual object.

11. There may well be situations where it is advantageous to explicitly remove an association between a pointer and its target, irrespective of whether the association had been established by allocation or by identification. DEALLOCATE does not seem to be an appropriate spelling for this statement. DISASSOCIATE would more accurate, and inspite of a dislike for such long keywords we have chosen to use it for purposes of presentation. The DISASSOCIATE statement breaks the association of the specified allocatable or alias objects with their current targets.

12. When a pointer appears in an expression the value of the associated definable object is referenced, except when the pointer appears as an argument of a procedure, or operand of an operator, where the dummy argument is itself a pointer. In this latter case the association is passed to the dummy pointer. Note that, an array of pointers can only appear in a context which requires an array of pointers. An array of pointers must not appear in a context which requires an array of the target objects. The set of objects associated with an array of pointers do not constitute an array.

13. When a pointer appears on the left of an assignment the value of the expression to the right is assigned to the object associated with the pointer. The IDENTIFY statement operates as an assignment for pointers rather than values. For derived types default assignment which is component by component assignment is interpreted as IDENTIFY for any pointer components.

14. Following the rules for expressions given above, the normal relational operators compare the values of associated objects. It is however necessary in some circumstances to be able to test if two pointers are associated with the same object. For this purpose we propose that the .EQV. and .NEQV. be given the extended meanings. The expression

    pointer .EQV. pointer

is true if both pointers are associated with the same definable object, and the expression

    pointer .EQV. variable

is true if the variable is associated with the pointer, and so on.

15. The rules and interpretation of pointers appearing in I/O lists are similar to those which apply to assignment and expressions. The only things that can be read and written are values of objects. A pointer itself cannot be transfered by I/O. This means a structure containing a pointer component may not appear unqualified in an I/O list.

16. The attributes ALLOCATABLE and ALIAS may not be used with value attributes, PARAMETER or DATA, nor with the INTENT attribute. Since these attributes specify constraints on the definition status and values of objects they are rather inappropriate for pointers which at declaration not only have no defined value they are not yet associated with any object that can have a defined value.

17. An association between a pointer and a definable object is broken explicitly either by the establishment of a new association by an ALLOCATE statement or an IDENTIFY statement, or by a DISASSOCIATE statement.

18. In a procedure if a pointer is accessed via a USE statement or passed as a dummy argument on exit it must not be associated with an unsaved local object. Before exit such a pointer must either be dissasociated or associated with an object whose lifetime is greater than that of the procedure.
{{this defines the dangling pointer problem away as by definition it cannot occur. Unfortunately it could be costly on some machines to check to enforce this rule. However it is not unlike the rule on exceeding array bounds in this respect}}

19. A disassociated pointer may appear in any context that requires a zero-sized array of similar attributes. Allocating a zero-sized array does not create an actual object and so does not establish an association for the pointer that is allocated. Identifying a pointer with a zero-sized section of an array results in a disassociated pointer.

20. In a value constructor for a derived type containing a pointer component the expression corresponding to this component must deliver an object which can be identified with this component rather that assigned to it. A null expression may appear for such expressions, in which case the corresponding pointer component becomes disassociated.

Discussion of Some Issues:

Issue: 1. The handling of chains of pointers

The present rules about when an alias becomes disassociated are such that if A,B,C are all pointers, and we set up the chain where C is allocated, B is identified with C, and A with B, then if B were to have its association changed by identifying it differently or otherwise, A would have its association with the space on the heap broken. A would become disassociated and a reference to the ASSOCIATED function with A as argument would produce the result false. Under present rules the sequence

```
TYPE(any),ALIAS,ALLOCATABLE::A,B,C  !declare pointers
TYPE(any)::D  !declare object
LOGICAL::POINTING
ALLOCATE(C)  !create object and associate with C
IDENTIFY(B=C) !associate B with the same object as C
IDENTIFY(A=B) !associate A with the same object as C
POINTING=ASSOCIATED(A)  !the value is true

IDENTIFY(B=D)  !associate B with the object D

POINTING=ASSOCIATED(A)   !the value is false
```

This I would claim is both counter intuitive and complicated to implement. A consequence is that not only does an identification have to copy the appropriate descripter values for the ultimate target space it has to keep track of the chain of identifications that are involved. When A is identified in the above example the processor must record somehow that it is associating A with space via B which is associated with space via C, which is associated with space via allocation. For an alias/identify that does not allow true pointer capability such chains are not likely to be very long. However for pointers these chains could become very long. In order to allow the ASSOCIATED function to always have a defined result a pointer must not only consist of the descripter for the space pointed at but it would have to record any other pointer that is identified with space via it. Alternatively we could always traverse the identification chain when referencing a pointer target. However this turns an object like A into a pointer to a pointer to a pointer to some space. Again the overheads if the chains get long would be totally unacceptable and would make the use of pointers pointless.

The alternative approach which is the usual one in most other languages is that when B is identified with C, the C descripter values only are copied into B, and when A is identified with B the B descripter values are copied into A; leaving A,B & C all independently refering to the same space. This is strictly similar to variables X,Y & Z after a sequence

```
Z=5
Y=Z
X=Y
```

All three variables now refer to the same value but these are still independent. We could re-assign Y without effecting the values of Z or X.

With this model for pointers the analogue is clear. Each pointer is associated independently with its target space. The sequence

```
ALLOCATE( C )
IDENTIFY(B=C)
IDENTIFY(A=B)
```

sets up a temporary situation where they all refer to the same space (the pointers all have the same "value" in that they have the same values for their descripters). If now we execute a

```
IDENTIFY(B=D)
```

we have had no effect on the association of A or C.

Issue: 2 When is Heap Space Freed and Ready for Re-use

The space in the above example was created by allocation. It is introduced into the program via C. It is now associated with A as well. The existing rules would require that the space on the heap be freed and become available for re-use when C was deallocated. That is the space ceases to exist when the association with C is broken. This either leaves A as a dangling pointer or we again have the requirement that the processor keep track of which pointers are associated with heap space via C. This is a similar situation to that of remembering chains. The whole route is no longer required but an arbitrary number of reverse pointers could be required. Since we need to have a defined result from the ASSOCIATED function applied to A simply leaving A dangling is not a reasonable option. It would be possible to define the operation of disassociating C as illegal if any other pointer is associated with the same space as C, but this could not be checked without the same processor overhead as needed to render A disassociated.

An alternative approach is suggested by the model that both A and C are simply names for descripters. In this case there is no special relationship with either with the space. They are both merely descripters whose current values cause them to refer to overlapping space on the heap. In principle, the association of either with the space could be broken without effecting the other. However this still leaves the question of when does the actual space become free, and when can it be reused. One obvious answer could be that space is only free when it is no longer associated with a currently existing pointer. In this model space is always simply allocated off the top of the heap until insufficient space is left. A garbage collection process is then initiated to recover free space and to consolidate the active heap. This would require the processor to search the associations of all existing pointers; All those pointing into the heap at least. Space currently associated with a pointer would have to be marked as in-use the remainder of the heap would then be free and could be garbage collected. This would clearly be a massive processor overhead whenever it occured, but there would be relatively little heap overhead until and unless it did.

The statement breaking an association simply does that. It has no strange side effects on other pointers. The dangling pointer cannot be caused by heap space being inadvertedly garbage collected while still associated with an active pointer. Hence the suggested name DISASSOCIATE for the statement which would simple break an association. It would not necessarily cause any space to be freed. The ASSOCIATED function would test whether an object was currently associated or not. The mechanism for determining this and that required to manage garbage collection is much the same. A pointer would require a flag bit which was set or unset depending on the status of the pointer.

Issue: 3 Default assignment for a structure with a pointer component

A further feature of this model is that for a derived type containing a pointer component the default component by component assignment operation remains defined as standard assignment for normal variable components but IDENTIFY for the pointer components. Although it is reasonable to define a default that within an expression a pointer is dereferenced to provide the value contained in the space associated with the pointer, it is critical that for any compound object the same level of dereferencing is applied to all components. To see why this is the case it is useful to use a concrete example. Suppose we had a type which is to be used to represent a linked list of integers.

```
TYPE CELL
  INTEGER::val
  TYPE(CELL),ALLOCATABLE,ALIAS::next  !pointer to next cell in list
ENDTYPE CELL
TYPE(CELL)::first,second,third,another
     ! creates 4 objects each of type CELL
     ! causes space to be generated to hold an integer and
     ! and a descripter for a CELL
first%val = 1  ! assign value 1 to val field of the first cell
IDENTIFY ( first%next = second )   !make the pointer in first cell refer
                                   !to the second cell
second%val = 2; IDENTIFY ( second%next = third )
third%val = 3   !this has created a list of three cells values 1,2,3
another = second
     !this copies the value 2 into another%val
     !and identifies, that is, copies the descripter from
     !second%next to another%next
     !another is therefore the head of a list which also terminates
     !on the cell third

{ Note: the result of ASSOCIATED(second%next) would be true but that of }
{       ASSOCIATED(third%next) would be false.                           }
{       The result of second%next.EQV.another%next would be true         }
```

In the previous example if the pointer in second was dereferenced when it appeared in the expression on the left of the assignment it would produce a value which was an integer and a pointer. So on the right of the assignment we would have two integers and a pointer to a cell, a pointer that is unassociated. On the left there is space to hold one integer and a pointer to a cell. For the copying of values required by assignment to occur the pointer components must copy descripters without dereferencing. Hence pointer components must be identified. (This rule is also applicable to contexts such as input output lists). It is reasonable to permit automatic dereferencing pointers to deal with values but problems would arise if differential dereferencing were permitted amoung the components of a derived type. It would be all to easy to produce an infinite loop of dereferences. For example if in the above example we added a statement

```
IDENTIFY ( third%next = first )
```

and then

```
WRITE(*,*) first
```

full dereferencing would produce 1,first%next dereferenced to 1,2,second%next then 1,2,3,third%next then 1,2,3,1,first%next and so on. This is clearly undesirable. Nor is it desirable for undereferenced pointers to be read or written. So this would make a statement such as the above invalid.

Issue: 4 Pointers as arguments

Clearly pointers must be passable as arguments. Currently we have a well defined set of rules for dealing with allocatable arguments. Extending this to alias objects, pointers, is fairly straight forward. If the dummy argument is neither allocatable nor alias, the matching actual argument must be associated when the procedure is invoked. The definition state of the associated object will then be determined by the intent specification of the dummy and the details of the execution of the procedure. Such a procedure cannot alter the association of the actual argument. If the dummy argument is declared to be an alias, a pointer, the matching actual argument must be similarly declared. Now the actual argument need not be associated on entry to the procedure and both the definition status and the association of the actual argument may be changed by the procedure.

Pointers which can be accessed via a pointer dummy argument, or those accessed via a USE statement, have a lifetime that is longer than a number of the possible local targets. If such a pointer is identified with a local target object there is therefore the possibility of this pointer being left dangling on exit from the procedure. (Allocated pointers cause no problem since the associated space is not released until the pointer ceases to exist or the association is broken explicitly; heap space is essentially global). Either all such identifications of a local object with a non-local pointer must be considered illegal or the leaving of such an association in effect on exit from the procedure must be made illegal.

A number of attempts were made to make such identifications syntactically impossible or at least easily detectable. None of these proved feasible unless pointers as derived type components were disallowed. This was not considered to be acceptable as it removed most of the reasons for having pointers in the first place. We finally were forced to adopt the cop-out of simply defining it to be non-standard conforming for a return to be executed with a global pointer or a dummy pointer associated with a non-saved local object. Therefore by definition a standard conforming program does not produce dangling pointers, but there is no proposed mechanism for statically detecting violations of this rule and it could be difficult, or at least costly, to detect dynamically. This situation is unfortunate but no worse in practice than the case of array bounds. There is a general prohibition against exceeding array bounds, but where performance is paramount bounds are not checked.

In order to allow dereferencing to be applied as the default for pointer actual arguments it will be necessary to require an explicit interface for procedures containing pointer dummy arguments. Also since we are not considering a pointer to be a data type the presence or absence of the pointer attributes for an argument cannot be used to resolve an overload.

4/77

Issue: 5 Intent attributes

The question can be raised of can dummy argument pointers be declared with intent attributes. The question was discussed at length at the San Jose meeting related to allocatable dummy arguments. The general difficulty was to decide whether the intent attribute applied to the definition status of the associated object or whether it applied to the association status of the argument. The problem with the former is that in many cases the definition status may be entirely irrelevant. A pointer dummy argument may well be used deliberately to allow a change in association, not in the value of the associated object. If intent applies to the association an INTENT(IN) is useless as it would be equivalent to having the dummy argument as the equivalent non-pointer, to a simple variable. INTENT(OUT) would imply that an association would have to be established before the argument could be used, and INTENT(INOUT) would mean that the argument would have to be associated on entry but the association could be changed. Applying the intent attribute to the association would have some utility but on balance it was considered that to simply disallow intent attributes with pointers was the least confusing option.

Issue: 6 Value attributes

The situation here is similar to that which applies to intent. The attributes at present are phrased in terms of definition of values. This would be entirely inappropriate for pointers which are not definable when declared. Therefore we would again have to define the meaning of PARAMETER and DATA (I still find these keywords entirely ridiculous, confusing and without justification. I wish we could at least use sane keywords for the new attributes even if we must keep the existing lunacies for the old statement forms) attributes in terms of the association not the object value.

If we assume therefore that for pointers the value attributes would have to specify the status of the association of a pointer with a target object then we have the following problems. A pointer with a constant association with its target space is essentially a variable and so it makes no sense to have a pointer with a PARAMETER attribute. A pointer with an initially established association is more sensible. However the syntax to allow ALLOCATE statements or IDENTIFY statements within the object declaration is not pretty, and potentially confusing. The simplest approach is therefore to disallow value attributes with pointers.

Examples:

We include in this section a few explicit examples of code using the proposed syntax for pointers.

1. A STRING module.

In this example we construct a module for a variable length character string which does not require the specification of a maximum character length and which does not end up storing large numbers of rubbish characters.

```
MODULE variable_character

  TYPE STRING
    PRIVATE
    CHARACTER,ARRAY(:),ALLOCATABLE,ALIAS::chars
  END TYPE STRING
  !-- defines a STRING as a pointer to an array of single characters

  INTEGER,PRIVATE,PARAMETER::max_input_length=80
  !-- a parameter such as this must be set since without stream I/O
  !-- there is no way of reading an arbitrary length string of characters
  !-- where end of string is coded in the data

CONTAINS
  FUNCTION LEN_STRING(s)  !-- return the length of the current string
    TYPE(STRING)::s
    INTEGER::LEN_STRING
      IF(ASSOCIATED(s%chars))THEN
        LEN_STRING = SIZE(s%chars)  !-- length of associated chars array
      ELSE
        LEN_STRING = 0  !-- no array associated, null string length zero
      ENDIF
  END FUNCTION LEN_STRING

  FUNCTION CONCAT(ls,rs),OPERATOR(//) !-- concatanate operator for strings
    TYPE(STRING)::ls,rs,CONCAT
    INTEGER::len_l,len_r
    len_l = LEN_STRING(ls); len_r = LEN_STRING(rs)
    ALLOCATE(  CONCAT%chars(1:len_l+len_r)  )  !-- allocate space for result
    CONCAT%chars(1:len_l) = ls%chars  !-- assign left string to result
    CONCAT%chars(len-l+1:len_l+len_r) = rs%chars
                                      !-- assign right string to result
  END FUNCTION CONCAT
```

```
FUNCTION CONCAT(lc,rs),OPERATOR(//) !-- concatanate operator for
  CHARACTER(LEN=*)::lc              !-- char // string
  TYPE(STRING)::rs,CONCAT
  INTEGER::len_l,len_r
  len_l = LEN(lc); len_r = LEN_STRING(rs)
  ALLOCATE(  CONCAT%chars(1:len_l+len_r)  )  !-- allocate space for result
  DO ( I=1,len_l )  !-- assign left characters to result
    CONCAT%chars(I) = lc(I:I)
  ENDDO
  CONCAT%chars(len-l+1:len_l+len_r) = rs%chars
                                    !-- assign right string to result
END FUNCTION CONCAT

FUNCTION CONCAT(ls,rc),OPERATOR(//) !-- concatanate operator for
  CHARACTER(LEN=*)::rc              !-- string // char
  TYPE(STRING)::ls,CONCAT
  INTEGER::len_l,len_r
  len_l = LEN_STRING(ls); len_r = LEN(rc)
  ALLOCATE(  CONCAT%chars(1:len_l+len_r)  )  !-- allocate space for result
  CONCAT%chars(1:len_l) = ls%chars  !-- assign left string to result
  DO ( I=len_l+1,len_l+len_r )  !-- assign right characters to result
    CONCAT%chars(I) = rc(I:I)
  ENDDO
END FUNCTION CONCAT

!-- STRING_STRING assignment does not need to be defined the default
!-- assignment which copies the pointer components is exactly what is
!-- needed.

SUBROUTINE CHAR_STRING_ASSIGN(vs,echar), ASSIGNMENT
          !-- character to string assignment
  TYPE(STRING)::vs
  CHARACTER(LEN=*)::echar
  INTEGER::len_rhs
  len_rhs = LEN(echar)
  ALLOCATE( vs%chars(1:len_rhs) )  !-- allocate space for LHS
  DO ( I = 1,len_rhs )
    vs%chars(I) = echar(I:I)  !-- assign characters to string
  ENDDO
END SUBROUTINE CHAR_STRING_ASSIGN
```

```
SUBROUTINE STRING_CHAR_ASSIGN(vchar,es), ASSIGNMENT
           !-- string to character assignment
           !-- Characters in string es are assigned into the
           !-- character variable left justified and the variable
           !-- is padded on the right with blanks or the string
           !-- truncated on the right to match lengths
  TYPE(STRING)::es
  CHARACTER(LEN=*)::vchar
  INTEGER::len_c,len_s
  len_c=LEN(vchar); len_s=LEN_STRING(es)
  IF(len_c == 0)THEN; RETURN  !-- an attempt to assign a value to a zero
                              !-- character is treated as a no-op not error
  ELSE
    DO ( I=1,MIN(len_c,len_s) )  !-- copy characters from the string into
                                 !-- the character either filling the
                                 !-- character or exhausting the string
      vchar(I:I) = vs%chars(I)
    ENDDO
    vchar(I:len_c) = ' '  !-- pad character if necessary
    RETURN
  ENDIF
END SUBROUTINE STRING_CHAR_ASSIGN

SUBROUTINE GET_STRING(UNIT,str,IOSTAT)
!-- read an arbitrary length string from UNIT=UNIT, connected to a
!-- formatted sequential file. The next record, upto the max_input_length
!-- is assumed to constitute the required string
!-- if the input record is longer than max_input_length the excess are
!-- ignored
  INTEGER,INTENT(IN)::UNIT
  INTEGER,INTENT(OUT),OPTIONAL::IOSTAT
  TYPE(STRING)::str
  INTEGER::len_str
  CHARACTER(LEN=max_input_length)::inbuf
  IF(PRESENT(IOSTAT))THEN
    READ(UNIT,'(A)',IOSTAT=IOSTAT) inbuf
    IF(IOSTAT<>0) RETURN  !--I/O error return unchanged string and IOSTAT
                          !-- value to calling program
  ELSE
    READ(UNIT,'(A)') inbuf  !--I/O error assumed to halt execution
  ENDIF
  !-- inbuf now contains the required string of characters possibly with
  !-- excess trailing blanks
  len_str=LEN_TRIM(inbuf)
  ALLOCATE(str%chars(1:len_str))  !-- if len_str=0 no allocation occurs
                                  !-- there is no point in allocating
                                  !-- space for a zero length array
  DO( I=1,len_str )  !-- assign characters from inbuf to string
    str%chars(I) = inbuf(I:I)
  ENDDO
END SUBROUTINE GET_STRING
```


4/81

```
  SUBROUTINE PUT_STRING(UNIT,str,IOSTAT)
    INTEGER,INTENT(IN)::UNIT
    INTEGER,INTENT(OUT),OPTIONAL::IOSTAT
    TYPE(STRING),INTENT(IN)::str
     ...
{ the inverse operation to GET_STRING  }
     ...
  END SUBROUTINE PUT_STRING

  FUNCTION EQUAL(ls,rs),OPERATOR(==)
    TYPE(STRING)::ls,rs
    LOGICAL::EQUAL
    INTEGER::len_l,len_r
    len_l=LEN_STRING(ls); len_r=LEN_STRING(rs)
    IF(len_l==len_r)THEN
      EQUAL=ALL(ls%chars==rs%chars)
    ELSE
      EQUAL=.FALSE.
    ENDIF
  END FUNCTION EQUAL

{ similar function/operator definitions can be defined for other relations }
{ and so on                                                                 }
{ included should be overloads for functions such a INDEX                   }

  FUNCTION EXTRACT(str,is,if)
  !-- given starting index and finishing index in a string extract the
  !-- contained substring
  !-- if either index is ommitted the corresponding end of the string is
  !-- implied as it is if is<1 or if>LEN_STRING(str)
  !-- if if<is zero length substring, a disassociated string is returned
    TYPE(STRING)::str,EXTRACT
    INTEGER,OPTIONAL::is,if
    INTEGER::len_str,strt,fin
    len_str = LEN_STRING(str)
    IF(PRESENT(is))THEN; strt = MAX(is,1)
                   ELSE; strt = 1;         ENDIF
    IF(PRESENT(if))THEN; fin = MIN(if,len_str)
                   ELSE; fin = len_str     ENDIF
    IDENTIFY( EXTRACT%chars = str%chars(strt:fin) )
  END FUNCTION EXTRACT

  SUBROUTINE INSERT(sub_str,str,is)
  !-- copy the characters of substring into the string starting at index is
  !-- if   is+LEN_STRING(sub_str) > LEN_STRING(str)   the copy process cannot
  !-- take place in situ so a new string is allocated to hold the result
     ...
  { fairly obvious code }
     ...
  END SUBROUTINE INSERT

END MODULE variable_character


{ Further examples may be prepared for tabling at the meeting }
```


4/82

Further Comment:

It is questionable whether the two attributes ALLOCATABLE and ALIAS should be retained. They are so much alternate faces of the same coin and will in practice most commonly be used together. It would simplify and shorten the description significantly if they were collapsed into a single attribute. Jeanne Martin has suggested VIRTUAL as a suitable keyword for such an attribute.

It appears to be maintaining a basic fiction to continue to claim that pointers of this variety do not constitute additional data types. All it appears to achieve is the inconvenience of forcing the use of derived types in order to construct arrays of pointers (to be carefully distinguished from a pointer to an array) or to handle pointers to pointers. It would not be very much more difficult, and in many ways it would be easier to treat the whole approach as a new pointer data type. If this were done I would suggest using the keyword POINTER, e.g.

```
POINTER(attributes-of-objects-pointed-to)
```

to declare the type of the pointer objects.

Neither of these options would necessarily cause major changes in the model or in the major functionality. It would merely simplify the description and the expression of the functionality.

TO: X3J3

FROM: Brian Smith and Jerry Wagener

SUBJECT: Parameterized LOGICAL

Parameterizing the intrinsic logical datatype can provide exactly the same functionality as the bit data type of Appendix F. The changes to S8 to accomplish this are listed on the following pages. Most of the changes occur in Section 13, where array masks play an important role in the intrinsic functions, and where appropriate transfer functions are described. Otherwise, the changes are very few, and many of those are due to the (unnecessary but recommended) addition of B'1' and B'0' as logical constants.

An informal summary of the features of this approach is given on the next page. The detailed text changes to accomplish this functionality begin on page 3.

Summary of Features

1. The KIND=Parameter. Logical data objects may be declared with the KIND= type parameter; such objects are called "specified kind logical" objects, whereas logical objects without the KIND= parameter are "default logical" objects. Specified kind logical objects may not be storage associated (whereas default logical objects may be), allowing bit level implementation. The implementation may use different storage units for objects with different KIND parameters if it chooses, but it must use single physics bits for KIND=1 if the implementation offers single bit storage units for any KIND= parameter.

   Example: LOGICAL(1) L(64) declares a logical array of 64 elements, that could be implemented in 64 bits

2. Expressions and Arguments. Specified kind and default logical objects may be freely mixed in expressions and assignment, but not across procedure calls. This is exactly analogous to the situation with real objects. However, unlike the real case, KIND=* is not provided for logical dummy arguments, and association of logical arguments requires the usual type, type parameter, and shape conformance.

   Example: FLAG = L(K) is allowed, where FLAG and L(K) are logical objects having different KIND parameters

3. Logical Constants. As described here, .TRUE. and .FALSE. remain the constants for default logical, but B'1' (or B"1") and B'0' (or B"0") are the constants for specified kind logical. The "B" form of logical constant has a KIND parameter value of 1. This division is carried over into I/O values, with the L edit descriptor used for both specified kind and default logical I/O. Clearly, this partitioning could easily be relaxed, and either form of constant in any logical context could be allowed. For I/O, this would involve an additional edit descriptor, say a B edit description, so that either edit descriptor could be used with any logical I/O object, and L would specify the T/F form of I/O and B would specify the 1/0 form of I/O.

   Example: L = B'0'

   Intrinsic Functions. The intrinsic functions provided here are:

'TEGER_TO_LOGICAL	converts an integer value to a logical array
'ICAL_TO_INTEGER	converts a logical array to an integer value
'AL	converts between kinds of logical
	the KIND parameter value of a logical object
'ICALS	returns the maximum size logical array that can be used in LOGICAL_TO_INTEGER

4/85

4/86

TEXT CHANGES FOR ALL SECTIONS EXCEPT SECTION 13

2-9/6
14-6/20 insert "default" before "logical"

2-9/16 change "Specified" to "Specified kind logical, specified"

4-1/8,9 replace sentence with:
"For example, the logical data type has a set of two values, true and false (true denoted by .TRUE., B'1', or B"1", and false denoted by .FALSE., B'0', or B"0"), which are manipulated by logical operations."

4-5/42+ add the alternatives: or B'1'
or B'0'
or B"1"
or B"0"

insert before line 43, page 4-5 the following two paragraphs.

The values .TRUE, B'1', and B"1" represent the value true, and the values .FALSE, B'0', and B"0" represent the value false.

The logical type is parameterized by the kind type parameter KIND. There are at least two such values: one is the default logical type with constant values .TRUE. and .FALSE.; a second one is KIND=1 with constant values B'1' (or B"1") and B'0' (or B"0") and corresponds to a processor-dependent representation for logical data. Data objects with a specified kind type parameter are of specified kind logical type; the constants B'1', B"1", B'0' and B"0" are of specified kind logical type, and have kind type parameter value 1.

The kind type parameter value for default logical type is considered to be different from the kind type parameter value for any specified kind logical type.

5-1/24 change "LOGICAL" to "LOGICAL [kind-selector]

5-4/34 insert the following:

R509.5 kind-selector is ([KIND=] type-param-value)

Constraint: The kind-selector type-param-value must not be an asterisk, and must not include an expression that contains a variable.

Specified kind logical entities are non-storage associated, having processor dependent storage units (which may be single physical bits). The number of specified kind logical types that a processor supports is processor dependent, but must include that for which the type-parameter-value is 1.

4/87

5-18/16
5-19/29 after the first comma insert:
"an object of logical type unless of default logical type,"

7-6/42 insert "logical," after "complex,"

7-6/45 insert "or logical" at beginning of line
change "or complex" to ", complex, or logical"

7-7/13 insert the following paragraph:

For an expression x_1 op x_2 where op is a logical intrinsic binary operator, and k_1 and k_2 are the kind type parameters of the operands x_1 and x_2, respectively, the kind type parameter of the result is max (k_1,k_2) if k_1 and k_2 are both specified in the declarations of x_1 and x_2, and of default kind otherwise.

7-14/29 insert "default" after "type"

7-19/26 insert new sentence:

For a logical intrinsic assignment statement, variable and expr may have different type parameters, in which case the expr value is converted to the corresponding value for variable.

10-8/23 add the following sentence:

Alternatively, the input field may contain a single 1 or 0 anywhere in the field, or one of the B forms of logical value (B'1', B"1", B'0', B"0", without interspersed blanks) anywhere in the input field.

10-8/24 after "F" insert "if the output item is of type default logical, or 1 or 0 otherwise"

10-12/41 replace period with "for default logical output items, and 1 for true values and 0 for false values for specified kind logical output items."

14-5/36 insert "default" before "logical"

C-2/2 insert the following new paragraph:

The reason for providing the specified kind logical type is to allow other, and possibly more efficient, storage units for logical data objects. Default logical objects are required to have numeric storage units. Specified kind logical objects do not have this requirement, and may be implemented in single physical bits or any other storage unit the implementation chooses. Objects with different kind type parameter values may be implemented with different

storage units. KIND=1 logical objects, the only specified kind logical an implementation is required to support, should use single physical bit storage units if the implementation chooses to provide such support for any specified kind logical objects. Specified kind logical objects could enhance execution performance in those applications using logical masks in array operations and/or logical arrays for bit processing.

CHANGES TO SECTION 13

13-1/30 after 'Character,' insert 'Logical,'

13-1/44+ insert the following two sections and renumber the current sections 13.4.5 and 13.4.6 accordingly:

13.4.5. Logical Functions. The elemental function LOGICAL converts between objects of logical type with different type parameter values. The transformational functions INTEGER_TO_LOGICAL and LOGICAL_TO_INTEGER convert between a logical array and an integer.

13.4.6. Logical Inquiry Functions. The inquiry function KIND returns the type parameter value of an object of type logical. The inquiry function MAX_LOGICALS returns the maximum size of a logical array that can be converted to an integer.

13-6/51+ add the following two sections and renumber the current sections 13.9.6 through 13.9.14 accordingly:

13.9.6. Logical Functions

INTEGER_TO_LOGICAL (I, SIZE, RL) Optional SIZE, RL	Convert an integer to a logical array
LOGICAL (L, MOLD) Optional SIZE, RL	Convert between objects of type logical with different type parameters
LOGICAL_TO_INTEGER (I, RL) Optional RL	Convert a logical array to an integer

13.9.7 Logical Inquiry Functions

KIND(L)	Kind type parameter
MAX_LOGICALS (I)	Maximum logical array length for conversion

13-31/1+ add the following two sections after line 1 and renumber the subsequent sections accordingly:

13.12.56 LOGICAL (L, MOLD)

Optional Argument. MOLD

Description. Convert between kinds of logical.

Kind. Elemental function.

Arguments.

L	must be of type logical.
MOLD (optional)	must be of type logical

Result Type and Type Parameters. Logical.
If MOLD is present, the type parameter is that of MOLD; otherwise, it is the type parameter of default logical.

Result Value.

Case (i): If MOLD is of type default logical or is absent, the value is .TRUE. if L is true, and otherwise otherwise .FALSE.

Case (ii): If MOLD is present and has a specified type parameter, the value is B'1' if L is true, and otherwise B'0'.

Example. logical (B'1') has the value .TRUE.

13.12.57. LOGICAL_TO_INTEGER (L, RL)

Optional Argument. RL

Description. Convert a logical array to an integer.

Kind. Transformational function.

Arguments.

L must be of type logical and rank one. Its size must satisfy the inequality ESIZE (L) ≤ MAX_LOGICALS(1).

RL (optional) must be of type logical.

Result Type and Shape. Scalar integer

Result Value.

Case (i): If RL is absent or has the value false, the result has value equal to the integer represented by the elements of L, regarded as a binary number where true is treated as the binary digit 1 and false is treated as the binary digit 0 with the element having the largest subscript value being the least significant binary digit of the result.

Case (ii): If RL has the value true, the result has value equal to the integer represented by the elements of L, regarded as a binary number where true is treated as the

binary digit 1 and false is treated as the binary digit 0 with the element having the largest subscript value being the most significant binary digit of the result.

LOGICAL_TO_INTEGER (INTEGER_TO_LOGICAL (J, RL=RL), RL) must have the value J for all nonnegative values of the integer J and any values of RL. INTEGER_TO_LOGICAL (LOGICAL_TO_INTEGER (L, RL), ESIZE (L), RL) must have the value L for any value of a logical array L with type parameter value 1 for which ESIZE (L) ≤ MAX_LOGICALS(1) and any value of RL.

Example. LOGICAL_TO_INTEGER ([B'0',B'1',B'0',B'1']) has the value 5.

13-28/29+ add the following two sections after line 29 and renumber the subsequent sections accordingly:

13.12.48. INTEGER_TO_LOGICAL (I, SIZE, RL)

Optional Arguments. SIZE, RL

Description. Convert an integer to a logical array.

Kind. Transformational function.

Arguments.

I	must be scalar and of type integer. Its value must not be negative.
SIZE (optional)	must be scalar and of type integer with a positive value. If it is omitted, it is as if it were present with the value MAX_LOGICALS(1).
RL (optional)	must be of type logical.

Result Type, Type Parameters, and Shape. The result is a logical array of rank one with SIZE number of elements. Its type parameter has value 1.

Result Value.

Case (i): If RL is absent or has the value false, the result is a logical array containing the binary representation of the argument where a true value is treated as the binary digit 1 and a false value is treated as the binary digit 0. The array element with the largest subscript will contain the least significant digit of the binary representation. Zero extension or

truncation will take place at the low end of the array as necessary.

Case (ii): If RL has the value true, the result is a logical array containing the binary representation of the argument where a true value is treated as binary digit 1 and a false value is treated as the digit 0. The array element with the largest subscript will contain the most significant digit of the binary representation. Zero extension or truncation will take place at the low end of the array as necessary.

LOGICAL_TO_INTEGER (INTEGER_TO_LOGICAL (J, RL=RL), RL), must have the value J for all noninteger values of the integer J and any value of RL. INTEGER_TO_LOGICAL (LOGICAL_TO_INTEGER (L, RL), ESIZE (L), RL) must have the value L for any value of a logical array L with type parameter value 1 for which ESIZE(L) ≤MAX_LOGICALS(1) and any value of RL.

Example. INTEGER_TO_LOGICAL(5,6) has the value [B'0',B'0',B'0',B'1',B'0',B'1']

13.12.49. KIND(L)

Description. Returns the kind type parameter value of a logical entity.

Kind. Inquiry function.

Argument. L must be of type logical. It may be scalar or array valued.

Result Type and Shape. Integer scalar.

Result Value. The result has value equal to the kind type parameter value of L if L is scalar or of an element of L if L is array valued.

Example. If C is declared by the statement
LOGICAL(1) L(100)
KIND(L) has the value 1.

13-33/6+ add the following section after line 6 and renumber the subsequent sections accordingly:

13.12.60 MAX LOGICALS(I)

Description. Returns the maximum size of a logical array that can be converted to a value of type integer.

Kind. Inquiry function.

Argument. I must be of type integer or array value.

Result Type and Shape. Integer scalar.

Result Value. The result has value equal to the maximum size of a logical array L that can be converted to integer using LOGICAL_TO_INTEGER (L, LR).

13-13/5 after "Type", insert ", Type Parameters," and before "logical", insert "default"

13-13/24 after "Type" insert ", Type Parameters," and replace "The result is of type logical with "Same as MASK."

13-13/28-29 replace ".TRUE." with "true" and ".FALSE." with "false"
13-12/29-30
13-34/11
13-47/28

13-29/15,30 insert "and Type Parameters" after "Type", and replace
13-30/3,18 "Logical" with "Default logical"

13-37/32 after "Type", insert ", Type Parameters,", and replace "Logical scalar" with "scalar of default logical type"

13-33/1 replace ".FALSE." with "false"
13-35/20
13-23/39
13-20/25
13-28/4
13-41/30
13-48/36

13-20/14-15 delete "If the ... type logical."
13-31/17

13-31/15 replace "If the ... type, the" with "The"
13-20/20

13-28/9 replace ".TRUE." with "true"
13-41/33
13-48/8

There are similar changes required to Appendix F functions (DIAGONAL, PROJECT, FIRSTLOG, LASTLOC) which are not shown here.

106(*) BPW-1

60

To: X3J3

From: Walt Brainerd
Rex Page
Jerry Wagener

Subj: Syntax for structure component reference

One of the most effective means of managing complexity in a computer program is abstraction. An essential characteristic of abstraction is that the underlying implementation of operations may be changed, while the manner in which the operation is used remains the same. The classic way of doing this in Fortran is to use subprograms; invocations (users of abstractions) may remain fixed, while the computations within the subprograms (implementors of abstractions) change to meet new requirements.

Some mechanisms added to Fortran 88 effectively assist the programmer in utilizing abstractions in programs; these include the derived data type and modules. These tools permit the programmer to define a data type with its operations in a manner that allows the implementation of the data representation and the operations to be changed without altering references to the operations--with one exception. The exception is that if structures are used, their syntax is different from the other schemes available, namely arrays and subprograms.

Look at this problem from another point of view. In Fortran 88, there are three kinds of "aggregate" objects, that is, objects with subobjects; they are character strings, arrays, and structures. Functions also may be viewed as aggregate objects, even though the standard does not describe them as such. With most of these objects, it is possible to reference an elementary component (called an "atom" in the table below); with arrays and strings, it is possible to refer to a nonatomic subobject. The following table summarizes what the standard permits and the notation it uses.

aggregate class	atom	index	atom designator	subobject designator
structure	component	expression	s%c	--
array	element	integer	a(i)	a(i:j)
string	character	integer	--	s(m:n)
function	value	argument	f(a)	--

The table illustrates that all but one of the allowed atom or subobject designators is a name followed by an item enclosed in parentheses. The exception is the subobject designator for structures. This violates the precept that analogous concepts should be denoted similarly and that unique syntax, unused elsewhere in a programming language, should be reserved for a unique feature, unrelated to other parts of the language.

Because structure references are similar to other constructs in the language, they not only do not warrant their own unique notation, but the different notation also prevents the programmer from abstracting an implementation without having to worry about changing (or selecting) the correct notation to refer to an operation, depending on the chosen implementation.

In the table, "--" means the reference is not allowed. Two of the three are not allowed because they make no sense, but there is no reason not to permit a reference to a single character in a character string; this can be done now only by referencing a subobject (substring) of length one. But this is another topic.

The proposal below permits a reference to a component that is an expression, a feature useful, for example, to extract the cardinality of the union of two sets:

```
PRINT *, CARDINALITY ( A + B )
```

If A and B are sets, as in the example in C.11, and union uses the operator +, then in the current version of Fortran, this example would require two statements and another set variable, U:

```
U = A + B
PRINT *, U % CARDINALITY
```

It is this extension that dictates the change in the definition of variable given in the proposal; an arbitrary expression is not permitted on the left of an assignment.

To show in detail how the syntax is now inconsistent, a simple example using three implementations of a data type will be shown. To keep the example simple, suppose that a program deals with U.S. postal zip codes and a frequent operation is to determine the state associated with a particular zip code. Of course, zip codes should form a data type that should be put in a module with the specification of the operations to be performed on zip codes. The major decision to be made (and probably to be changed after some

experience with the program) is the method of calculating the state, given the 5-digit zip code. We will consider doing this with an array, a function, and with a structure component. First, consider the array case:

```
MODULE ZIP_CODE  !SPACE INEFFICIENT; NO ERROR CHECKING
   CHARACTER (LEN = 2), ARRAY (0 : 99999) :: STATE
END MODULE ZIP_CODE
   . . .
INTEGER ZIP
ZIP = 87122
PRINT *, STATE (ZIP)
```

The next example uses a function to determine the state.

```
MODULE ZIP_CODE  ! MORE SPACE EFFICIENT; ERROR CHECKING
   TYPE ZIP_TYPE
      PRIVATE
      INTEGER :: CODE
   END TYPE ZIP_TYPE

   CONTAINS
   SUBROUTINE SETZ (Z, N) ASSIGNMENT
      TYPE (ZIP_TYPE) :: Z
      INTEGER N
      Z % CODE = N
      IF (N < 1 .OR. N > 99999) . . . ERROR . . .
   END SUBROUTINE SETZ

   FUNCTION STATE (Z)
      CHARACTER (LEN = 2) :: STATE
      TYPE (ZIP_CODE) :: Z
      SELECT CASE (Z % CODE)
         . . .
         CASE (87000 : 88999)
            STATE = "NM"
         CASE DEFAULT
            . . . ERROR . . .
         . . .
      END SELECT

   END FUNCTION STATE

END MODULE ZIP_CODE
   . . .
USE ZIP_CODE
TYPE (ZIP_TYPE) ZIP
   . . .
ZIP = 87122
PRINT *, STATE (ZIP)
```

The main thing to notice is that the syntax for referencing the state for zip code ZIP is identical to the first example.

In the third implementation, the state is carried as a component of the structure. This scheme might be used if the number of assignments is expected to be much smaller than the number of times a state is determined for a zip code.

```
MODULE ZIP_CODE  ! MORE EFFICIENT THAN THE PREVIOUS VERSION
   TYPE ZIP_TYPE ! IF NUMBER OF STATE REFERENCES EXCEEDS
      PRIVATE    ! NUMBER OF ASSIGNMENTS
      INTEGER :: CODE
      CHARACTER (LEN = 2) :: STATE
   END TYPE ZIP_TYPE

   CONTAINS
   SUBROUTINE SETZ (Z, N) ASSIGNMENT
      TYPE (ZIP_TYPE) :: Z
      INTEGER N
      Z % CODE = N
      IF (N < 1 .OR. N > 99999) THEN
          . . . ERROR . . .
      ELSE
          Z % STATE = ST (N)
      END IF
   END SUBROUTINE SETZ

   FUNCTION ST (N)
      CHARACTER (LEN = 2), PRIVATE :: ST
      TYPE (ZIP_TYPE) :: Z
      SELECT CASE (N)
         . . .
         CASE (87000 : 88999)
            ST = "NM"
         CASE DEFAULT
            . . . ERROR . . .
         . . .
      END SELECT

   END FUNCTION ST

END MODULE ZIP_CODE
   . . .
USE ZIP_CODE  ! APPLICATION CODE
TYPE (ZIP_TYPE) ZIP
   . . .
ZIP = 87122
PRINT *, ZIP % STATE  ! STRUCTURE REFERENCE
```

In this last case, the programmer must know that the implementation of the data type ZIP_CODE is a structure with component STATE and use the strange syntax with the percent sign. With the proposal below, the reference to the state of ZIP in this last version would be identical to that in the previous two examples, namely STATE (ZIP).

Proposal 1:

2-8,23: "a structure component" ->
"a structure component of a variable"
2-9,22: "component selectors," ->
"parent structure selectors,"
4-7,4: delete "to qualify the name of a structured object
of this type"
6-1,14+:

Constraint: variable must not be a structure component whose parent is not a variable.
6-1,31: "P % AGE" -> "AGE (P)"
6-2,16: "P % NAME (1:1)" -> "NAME (P) (1:1)"
6-2,21-28: replace with
R608 structure-component is component-name (parent-structure)
R609 parent-structure is expression

Constraint: The type of expression must be of a derived type containing a component with the name component-name.

Constraint: Either the parent or the component may be an array, but not both.

6-2,33+: Add paragraph:
If the parent is a variable, the component also is a variable.
6-2,37-39: replace left column of table with
SCALAR_COMPONENT (SCALAR_PARENT)
SCALAR_COMPONENT (ARRAY_PARENT (J))
SCALAR_COMPONENT (ARRAY_PARENT (1:N))
6-2,39+: add:
SCALAR_COMPONENT (ARRAY1 + ARRAY2)
component of array expression parent
6-8,16: "STRUCTURE % COMPONENT" -> "COMPONENT (STRUCTURE)"
6-8,18 and 19: "STRUCTURE % A (J)" -> "A (STRUCTURE) (J)"
6-8,20: "STRUCTURE % B" -> "B (STRUCTURE)"
6-10,13: "STRUCTURE % ARRAY (I)" -> "ARRAY (STRUCTURE) (I)"
6-10,14: "STRUCTURE (I) % ARRAY (J)" -> "ARRAY (STRUCTURE (I)) (J)
7-20,17: "D % S to C % S" -> "S (D) to S (C)"
7-20,17: "D % T to C % T" -> "T (D) to T (C)"
7-20,18: "D % U to C % U" -> "U (D) to U (C)"
7-20,18: "D % V to C % V" -> "V (D) to V (C)"
12-11,3: "STR % LENGTH" -> "LENGTH (STR)"
12-11,4: "STR % VALUE" -> "VALUE (STR)"
14-2,31: "within" -> "as"
C-4,11: "GEORGE % AGE" -> "AGE (GEORGE)"

```
C-4,12: "MARY % ID % LAST_NAME" -> "LAST_NAME (ID (MARY))"
C-4,13: "MARY % ID % NUMBER % SSN" -> "SSN (NUMBER (ID (MARY)))"
C-4,16: "GEORGE%AGE" -> "AGE (GEORGE)"
C-4,17: "GEORGE%ID % NUMBER" -> "NUMBER (ID (GEORGE))"
C-4,26: "A (1) % ELT (3)" -> "ELT (A (1)) (3)"
C-4,27: "A (2:4) % VAL" -> "VAL (A (2:4))"
C-12,18: "SETF" -> "SET"
C-12,20: "CARD" -> "CARDINALITY"
C-12,22: replace with "are included, MEMBER and SUBSET.  MEMBER"
C-12,27: "SETF" -> "SET"
C-12,30-32: delete sentence "An assignment coercion ... if not)."
C-12,34 to C-14,22:

MODULE INTEGER_SETS

INTEGER, PARAMETER :: MAX_SET_CARD = 200

TYPE SET                                    ! DEFINE SET DATA TYPE
   PRIVATE
   INTEGER CARD
   INTEGER ELEMENT (MAX_SET_CARD)           ! COULD BE ANY DATA TYPE
END TYPE SET

CONTAINS

FUNCTION CARDINALITY (A)                    ! RETURNS CARDINALITY OF SET
   INTEGER CARDINALITY
   TYPE (SET) A
   CARDINALITY = CARD (A)
END FUNCTION CARDINALITY

FUNCTION MEMBER (X, A) OPERATOR (.IN.) ! DETERMINES IF ELEMENT X
   LOGICAL MEMBER                           ! IS IN SET A
   INTEGER X
   TYPE (SET) A
   MEMBER = ANY (ELEMENT (A) (1 : CARD (A)) .EQ. X)
END FUNCTION MEMBER

FUNCTION UNION (A, B) OPERATOR (+)          ! UNION OF SETS A AND B
   TYPE (SET) A, B, UNION
   INTEGER J
   UNION = A
   DO J = 1, CARD (B)
      IF (.NOT. (ELEMENT (B) (J) .IN. A)) THEN
         IF (CARD (UNION) < MAX_SET_CARD) THEN
            CARD (UNION) = CARD (UNION) + 1
            ELEMENT (UNION) (CARD (UNION)) = ELEMENT (B) (J)
         ELSE
            ! MAXIMUM SET SIZE EXCEEDED . . .
         END IF
      END IF
   END DO
END FUNCTION UNION
```

```
FUNCTION DIFFERENCE (A, B) OPERATOR (-)  ! DIFFERENCE OF SETS A AND B
   TYPE (SET) A, B, DIFFERENCE
   INTEGER J, X
   DIFFERENCE = SET ([1:0])
   DO J = 1, CARD (A)
      X = ELEMENT (A) (J)
      IF (.NOT. (X .IN. B)) DIFFERENCE = DIFFERENCE + SET ([X])
   END DO
END FUNCTION DIFFERENCE

FUNCTION INTERSECTION (A, B) OPERATOR (*) ! INTERSECTION OF SETS A AND B
   TYPE (SET) A, B, INTERSECTION
   INTERSECTION = A - (A - B)
END FUNCTION INTERSECTION

FUNCTION SUBSET (A, B) OPERATOR (<=) ! DETERMINES IF SET A IS
   LOGICAL SUBSET                    ! A SUBSET OF SET B
   TYPE (SET) A, B
   SUBSET = CARD (A) .LE. CARD (B)   ! THESE TWO STATEMENTS
   IF (.NOT. SUBSET) RETURN          ! IMPROVE EFFICIENCY
   SUBSET = ALL (ELEMENT (A) (1 : CARD (A)) .IN. B)
END FUNCTION SUBSET

FUNCTION VECTOR (A)                  ! TRANSFER THE VALUES OF SET A
   INTEGER, ALLOCATABLE, ARRAY (:) :: VECTOR
   TYPE (SET) A                      ! INTO A VECTOR OF ASCENDING ORDER
   INTEGER I, J, K
   ALLOCATE (VECTOR (CARD (A)))
   VECTOR = ELEMENT (A) (1 : CARD (A))
   DO I = 1, CARD (A) - 1            ! USE A BETTER SORT
      DO J = I + 1, CARD (A)         ! IF CARD (A) IS LARGE
         IF (VECTOR (I) > VECTOR (J)) THEN
            K = VECTOR (J); VECTOR (J) = VECTOR (I); VECTOR (I) = K
         END IF
      END DO
   END DO
END FUNCTION VECTOR

FUNCTION SET (V)                     ! TRANSFER FUNCTION BETWEEN A
   TYPE (SET) SET                    ! VECTOR OF ELEMENTS AND A
   INTEGER V (:)                     ! CORRESPONDING SET OF ELEMENTS
   INTEGER J                         ! REMOVING DUPLICATE VALUES
   CARD (SET) = 0
   DO J = 1, ESIZE (V)
      IF (.NOT. (V (J) .IN. SET)) THEN
         CARD (SET) = CARD (SET) + 1
         ELEMENT (SET) (CARD (SET)) = V (J)
      END IF
   END DO
END FUNCTION SET

END MODULE INTEGER_SETS
```

Examples of using INTEGER_SETS (A, B, and C are sets;
X is an integer variable):

```
! CHECK TO SEE IF A HAS MORE THAN 10 ELEMENTS
IF (CARDINALITY (A) > 10) ...

! CHECK FOR X AN ELEMENT OF A BUT NOT OF B
IF (X .IN. (A - B)) ...

! C IS THE UNION OF A AND THE RESULT OF B INTERSECTED
! WITH THE INTEGERS 1 TO 100
C = A + B * SET ([1 : 100])

! DOES A HAVE ANY EVEN NUMBERS IN THE RANGE 1:100?
IF (CARDINALITY (A * SET ([2:100:2])) > 0) ...
```

End of proposal

TO: X3J3

FROM: Jerry Wagener

SUBJECT: Liverpool Resolutions, August 3-7, 1987

Resolution L1 - Submission of S8 to SC22
individual vote: 28-7-0 country vote: 7-2-0

That WG5 confirms the action of its convenor in forwarding S8.104 to SC22 for processing as a Draft Proposed Standard.

X3J3 Response. X3J3 concurs with this resolution, and notes that it requires no action on X3J3's part.

Resolution L2 - French Translation
individual vote: 34-0-1 country vote: 9-0-0

That WG5 appreciates the offer of the French member body (AFNOR) to be responsible for the French translation of the final standard.

X3J3 Response. X3J3 concurs with this resolution, and notes that it requires no action on X3J3's part.

Resolution L3 - X3J3 Schedules
individual vote: 31-3-1 country vote: 8-1-0

That WG5 explicitly states that all recommendations to X3J3 and requests for additions, deletions, study, etc., of Fortran 8x features contained in the Liverpool resolutions should be processed by X3J3 in a way which does not delay the public comment process.

X3J3 Response. X3J3 appreciates WG5's strong sentiment that no WG5 resolution unduly delay the processing of Fortran 8x.

Resolution L4 - Status of Halifax Resolutions
individual vote: 31-0-4 country vote: 9-0-0

That WG5 thanks X3J3 for its work in response to the Halifax resolutions, acknowledges receipt of N227 (status of Halifax Resolutions) and notes that no further work is requested of X3J3 on those resolutions.

X3J3 Response. X3J3 notes and appreciates this resolution.

Resolution L5 - Resolution Life Cycle
individual vote: 35-0-0 country vote: 9-0-0

That WG5 now establishes a standing operating procedure whereby reporting of actions and responses to resolutions shall not be carried forward beyond the next regularly scheduled WG5 meeting, and that unresolved resolutions from the previous meeting shall be withdrawn unless they are the subject of explicitly reaffirmed or amended resolutions.

X3J3 Response. X3J3 notes that this resolution requires no X3J3 response. However, X3J3 strongly supports this resolution, and commends WG5 on this action.

Resolution L6 - Content of Resolution Response Document
individual vote: 35-0-0 country vote: 9-0-0

That WG5 requests its convenor to ensure that responses to WG5 resolutions shall be recorded in a document which includes the following for each resolution:

1. The full text of the resolution, including the WG5 voting figures.

2. The response, including the voting figures and explanation where relevant.

X3J3 Response. X3J3 concurs with this resolution, and trusts that this response document adequately reflects the requested format.

Resolution L7 - Temporary Nature of an Extension Features Appendix
(cf. Halifax 3) individual vote: 23-9-3 country vote: 8-1-0

That WG5 reaffirms its Halifax resolution 3 (1986), namely that WG recommends to X3J3 that the final published document not contain an appendix of suggested extension features.

X3J3 Response. X3J3 currently intends to remove Appendix F from the final document. The current plan is to develop a separate standing document entitled something like "Candidate Extensions". The initial contents of this document would be Appendix F. This document would then be evolved by X3J3 and made available to users and implementers of Fortran 8x. It would also be the principal source of incremental features during the development of Fortran 9x.

Resolution L8 - Pointers (cf. Halifax 11)
individual vote: 22-3-10 country vote: 8-0-1

That WG5 reaffirms the intent of Halifax resolution 11 (1986), namely that WG5 feels that a major feature that is lacking in the current S8 is that of a pointer facility.

X3J3 Response. Assigned to the Data Concepts subgroup.
This topic continues to receive X3J3 attention, and a tutorial on a recursive data structures form of pointers was given at X3J3 meeting #105 (Liverpool, August 1987). Following the tutorial, a straw vote of 23-1-9 favored this approach to pointers, and consequently a proposal is being developed for discussion at X3J3 meeting #106 (Ft. Lauderdale, November 1987).

Resolution L9 - Pointers and IDENTIFY
individual vote: 13-9-13 country vote: 3-2-4

That WG5 recommends that if pointers are adopted by X3J3 they should either be integrated into the IDENTIFY facility, or else the latter facility should be deleted.

X3J3 Response. Assigned to the Data Concepts subgroup.
The proposal mentioned in the preceding resolution response uses the IDENTIFY statement for pointer assignment.

Resolution L10 - Deprecated Features (cf. Halifax 13)
individual vote: 33-0-5 country vote: 7-0-2

That WG5 withdraw its Halifax resolution 13 (1986).

X3J3 Response. X3J3 notes and appreciates this resolution.

Resolution L11 - Decremental Features
individual vote: 28-4-3 country vote: 8-0-1

That WG5 recognizes the motivation for splitting decremental features in Fortran 8x into two different classes.

However, WG5 requests X3J3 to consider the possibility of identifying both obsolescent and deprecated features in the text of the standard, consistent with the requirement imposed on processors for detecting both classes of decremental features.

X3J3 Response. Assigned to the General Concepts subgroup.

Resolution L12 - Significant Blanks (cf. Halifax 14)
individual vote: 26-8-1 country vote: 9-0-0

The WG5 reaffirms the intent of Halifax resolution 14 (1986), namely that WG5 believes that significant blanks are logically associated with free source form and that the appropriate time to introduce this feature is at the same time as the free source form, even if no syntax in Fortran 8x is dependent on its presence. The presence of significant blanks in Fortran 8x will give greater flexibility for the future development of the language and will simplify development of software tools

WG5 notes the response prepared by X3J3 to Halifax resolution 14 but requests X3J3 to reconsider this matter as part of its processing of the comments received during the public review period.

X3J3 Response. Assigned to the General Concepts subgroup.

Resolution L13 - Name-directed I/O (cf. Halifax 22)
individual vote: 23-6-6 country vote: 5-0-4

That WG5 withdraw its Halifax resolution 22 (1986).

X3J3 Response. X3J3 notes and appreciates this resolution.

Resolution L14 - Language and Style
individual vote: 35-0-0 country vote: 9-0-0

That WG5 appreciate that X3J3 has made significant improvements in the readability of S8, and particularly wishes to thank Lloyd Campbell and Walt Brainerd for their efforts.

WG5 suggest to X3J3 that the index be improved and that more examples be added.

X3J3 Response. Assigned to the Editorial subgroup.

Resolution L15 - Section Notes
individual vote: 27-4-4 country vote: 9-0-0

That WG5 requests X3J3 to introduce some reference mechanism between the text of the standard and the section notes.

X3J3 Response. Assigned to the Editorial subgroup.

Resolution L16 - Revision Indication
individual vote: 25-5-5 country vote: 9-0-0

That WG5 suggests to X3J3 that each succeeding internal draft of Fortran 8x have some indication of changes and deletions with respect to the previous draft.

X3J3 Response. Assigned to the Editorial subgroup.

Resolution L17 - Program Size and Complexity
individual vote: 20-9-6 country vote: 7-1-1

That WG5 requests that X3J3 investigate the possibility that a standard conforming processor should be capable of detecting and reporting violation of its processor dependent limits on program size and complexity.

X3J3 Response. Assigned to the General Concepts subgroup.

Resolution L18 - Usage of Interfaces
individual vote: 33-0-2 country vote: 9-0-0

That WG5 requests that X3J3 add more examples to clarify definition and usage of the concept of interfaces.

X3J3 Response. Assigned to the Procedures and Program Units subgroup. The section describing procedure interface blocks is being rewritten to make it clearer, and more examples illustrating the use of interface blocks will be added.

Resolution L19 - Multiple Character Sets
individual vote: 24-1-10 country vote: 7-0-2

That WG5 recommends that X3J3 in cooperation with the Japanese member body add a facility to the Fortran language to manipulate as data within a single program unit more than one character set, with very different numbers of characters in each set, so as to allow for the use within Fortran of natural languages such as Chinese or Kangi. Further WG5 recommends that such facility accommodate mixtures of characters from different character sets in input and output.

X3J3 Response. Assigned to Jim Matheny.
Tutorials on this topic were presented and discussed at X3J3 meetings #104 (Seattle, May 1987) and #105 (Liverpool, August 1987). A proposal to extend the CHARACTER data type with a KIND= type parameter is scheduled for discussion at X3J3 meeting #106 (Ft. Lauderdale, November 1987).

Resolution L20 - Referral to SC22 of Processing Ideographic Languages
individual vote: 34-0-0 country vote: 9-0-0

That WG5 requests its convenor to report to SC22 the concerns of WG5 that projects in the program of work of SC22 and related committees allow for the processing of ideographic languages such as Chinese and Kangi in a consistent and efficient manner.

X3J3 Response. X3J3 concurs with this resolution, and notes that it requires no action on X3J3's part.

Resolution L21 - Use of National Characters
individual vote: 31-0-4 country vote: 9-0-0

That WG5 expresses to X3J3 it concern about the negative effect on the production of standard-conforming processors if characters in the national use positions in ISO 646, such as square brackets, are required in the Fortran 8x character set.

X3J3 Response. Assigned to the General Concepts subgroup.

Resolution L22 - Bit Data Type
individual vote: 22-3-10 country vote: 7-0-2

That WG5 believes that there is a significant unsatisfied demand for a bit data type facility in Fortran and that the need for such a facility will tend to increase during the lifetime of Fortran 8x. It, therefore, recommends that X3J3 review its earlier decision to remove BIT from 8x.

X3J3 Response. Brian Smith and Jerry Wagener assigned to investigate providing bit functionality with the logical data type, and prepare a report for the Data Concepts subgroup. A tutorial on parameterizing the logical data type to provide the Appendix F bit functionality was presented and discussed at X3J3 meeting #105 (Liverpool, August 1987). This issue is scheduled for further discussion at X3J3 meeting #106 (Ft. Lauderdale, November 1987).

Resolution L23 - Passed-On Precision
individual vote: 25-2-8 country vote: 8-0-1

The WG5 draw the attention of X3J3 to the concerns of the German member body (DIN) about passed-on precision contained in paper N245.

X3J3 Response. Assigned to Brian Smith.

Resolution L24 - Range and Set Range
individual vote: 12-10-12 country vote: 4-3-2

That WG5 recommends to X3J3 that the RANGE and SET RANGE facilities be deleted from the language.

X3J3 Response. Assigned to the Data Concepts subgroup.

---

end 106.JLW-1

(62)

106.JLW-2
page 1 & 9

Amoco Production Company
4502 East 41st Street
Post Office Box 3385
Tulsa, Oklahoma 74102
Research Center

87251ART0127 JLW078
September 8, 1987

Mr. Paul Sinclair
Ryan-McFarland Corporation
609 Deep Valley Drive
Rolling Hills Estates, CA 90274

Fortran 8x Review

Dear Paul:

Thank you very much for your letter of July 24, and the very thorough and competent review of S8. You raise some excellent points in your review, and I suggest you submit this review as a formal comment during the public review period. I will be sending out information on the public review, and how to submit comments, when that information is available.

In the meantime I will send your comments to the X3J3 editor, to each of the technical subgroup heads (as your comments deal with technical issues of concern to each subgroup, as well as editorial matters), and to be included in the X3J3 distribution.

As you are interested in becoming an X3J3 member, I am including herewith a copy of the membership policies and procedures and a meeting schedule for the next year. The membership policies and procedures describe what the membership requirements are and how one applies for X3J3 membership.

Thanks again for your thorough review.

Sincerely,

Jerrold L. Wagener
Research Director

JLW:sd

Enclosures

cc: J. C. Adams
C. D. Burch
L. W. Campbell
R. A. Hendrickson
K. W. Hirchert
J. H. Matheny
R. R. Ragan

Ryan-McFarland Corporation
609 Deep Valley Drive
Rolling Hills Estates,
California 90274
(213) 541-4828
TWX 910-344-6353
Telex: 294253

July 24, 1987

Mr. Jerrold L. Wagener
AMOCO PRODUCTION RESEARCH CENTER
P.O. Box 3385
Tulsa, Oklahoma 74102

Dear Jerry:

Thank you for the latest copy of the X3J3 standard document S8.104. Attached are some comments on this version.

I hope that you will be able to answer some of my questions and will find some of my suggestions useful.

I am currently an observer, but I am interested in becoming a member of the committee. I would appreciate any information on how this is done and the responsibilities of membership.

Sincerely,

RYAN-McFARLAND CORPORATION

Paul Sinclair

Paul Sinclair
Senior Technical Staff

PS/cv

Attachment

July 24th, 1987

Fortran 8X Comments

The following are comments on Draft S8, Version 104. References to the document are in the form: s.p:l or s.p:l-l where s is the section number (for example, 3.2.3). p is the page number (for example, 4-5), and l is a line number. Material copied from the draft is in quotes. Items in apostrophes are suggested rewordings.

1. The following two statements appear contradictory:

   1.4,1-1:35-36,

   > "The optional output forms produced by a processor, which are not under the control of a program, are an example of an exception."

   1.3.2,1-1:19, 1.3.2,1-1:23-24,

   > "1.2.2.Exclusions. This standard does not specify:
   >
   > (3) the method of transcription of programs or their input or data to or from a storage medium."

2. The terminology is not consistent when discussing obsolescent: redundant vs. better methods, removing vs. deleting. I recommend using the same terminology in all cases.

   1.6,1-4:32-33,

   > "Those in the second category, obsolescent features, are considered to have been redundant in ANSI X3.9-1978, but are still used frequently."

   1.6.2,1-4:41-42,

   > "1.6.2. Nature of Obsolescent Features.
   >
   > (1) Better methods existed in ANSI X3.9-1978."

   Add after 1.6.2,1-4:40 the following

   > '(3) It is recommended that future Fortran standards committees only consider deleting language features that appear on the list of obsolescent features in the prior Fortran revision.'

   1.6.2,1-5:2 and 1.6.2,1-5:4-5,

   > "removing" should be changed to 'deleting'.

1.6.2,1-5:6,

add to end of sentence 'in Appendix B of this revision.'

1.6.2,1-5:7-9 should be changed to

'(6) A standard-conforming processor must continue to support an obsolescent feature until it is moved to the deleted list.'

3. Why is the description of the Deprecated Features in B.3,B-2:43-47 and B.3,B-3:1-13 not covered in 1.6,1-5:28? How can the standard refer to something that is not part of the standard? Contents of section B.3 should be moved to section 1.6.3 and Section B.3 should be replaced with wording similar to section B.2. Having this information scattered is very confusing.

4. In section 1.5.3 references to syntatic classes refer to rule number except for array-spec (1.5.3,1-4:18). This is inconsistent. R513 should be specified for array-spec. Also, "e.g." and "for example" are both used (should use one or the other).

5. Wording of sentences starting at 2.2,2-3:49:

"An internal subprogram is a subprogram that is contained within a main program or another subprogram. A module subprogram is a subprogram that is contained in a module but is not an internal subprogram."

should be changed to:

'An internal subprogram is a subprogram that is contained within a main program or external subprogram. A module subprogram is a subprogram that is contained within a module.'

Current wording seems left over from when nesting of internal procedures was going to be allowed.

6. According to 2.1.4.2,2-7:30-32, assignment with type agreement, use of procedure arguments and function results, inquiry functions for parameter values, and input/output are intrinsic operations for a derived type.

According to 4.1.3,4-2:5-6, "For derived types, the only intrinsic operation is assignment. All other operations must be defined."

According to 7.1.7.7,7-13:7-8, "Two expressions of derived type are equivalent if their values are equal for all possible values of their primaries". This seems to imply an intrinsic comparison for equality.

According to 4.4.4,4-9:12-15, "4.4.4 Derived-Type Operations and Assignment. Any operations on derived-type entities and nonintrinsic assignment for derived-type entities must be defined explicitly by operator function or assignment subroutines. Such definitions are described in Section 12. Arguments and function values may be of any derived or intrinsic type."

The above statements seem contradictory. What are the intrinsic operations?

7. 3.3.1.3,3-5:16-17, "A statement must not contain more than 2640 characters." How are characters counted? 3.3.1.1,3-5:3-4 says "Lines containing only blanks or blank equivalents are ignored and may appear anywhere in a program unit." Does that mean the characters in those lines should not be counted even if the line appears between a line and its continuation? Is the & counted? (Says "& is not part of the statement".) I think some more explanation is needed here.

Also, it is not clear what should be done with the following:

```
A="string characters.... &
  ! a comment
   & rest of string ..."
```

Is this an error because the "next line" after the continued line does not have a &?

Also, it is not clear what should be done with the following:

```
B=3.4    &
  &
  + 6
```

Does the & in the second line mean both continuation and continued, or just continuation, or just continued?

8. In 4.1.3.2,4-3:15, change "effective decimal range" to 'effective decimal exponent range.'

9. According to 4.4.1.2,4-8:38-41, "Two data entities have the same type if they are declared with reference to the same derived-type definition; conversely, two entities are of different type if they reference different

derived-type definitions, even if the two derivec type have identical components declared in the sare order."

Do two entities have the same type if they reference the same type definition but with different values fcr the type parameters? If so, how is assignment done?

Need more explanation on intrinsic assignment for derived types.

10. In 6.1.1,6-2:7-8, allowing end to be less than start fcr a substring to mean 0 length string seems to imply a runtime check where in FORTRAN 77 none was needec. This enhancment decreases performance in a FORTRAN 77 feature. This is not acceptable.

11. In 7.3.1,7-16:7-16, 7.3.2,7-16:17-29, and 7.5.1.6,7-20:24-36, an ambiguous situation seems to arise in determining which function subprogram should be used in defining the operation. What happens when x1 and x2 match effective shape d1 and d2 of a qualifying function subprogram and for another qualifying functicn subprogram x1 and x2 have same effective shape and d1 and d2 are scalar? Which function subprogram is used? How is this ambiquity resolved?

What is the difference between "same" and "match" for effective shapes? If they are equivalent terminology, then use one or the other (but not both).

12. In 7.5.2.1,7-21:3, why does "ELSEWHERE" not have a space in it like other double word keywords? I know it doesn't matter to the language but is seems to me standard should be consistent.

Change "ELSEWHERE" to 'ELSE WHERE.'

13. In 9.3.4,9-6:10-12, the implication is that to open a SCRATCH file on a connected unit to an existing file, an explicit CLOSE must occur to the unit prior to the OPEN. That is, there is no way to get an implicit close, when trying to open the SCRATCH file since SCRATCH can't have FILE= (and therefore connection stays the same) but then you can't have a STATUS which is different (9.3.4,9-6:19-21).

This doesn't seem very nice. When you have a FILE= the connection can be implicitly closed, but when trying to OPEN a SCRATCH file, there is no way to implicitly close the connection.

14. In 9.3.4.2,9-7:24-25, the sentence "Note that SCRATCH must not be specified with a named file" seems redundant since 9.3.4.2,9-7:1-2 already say this. Why coes this

need to be noted? If it needs to be noted, why coesn't line 1 need to be noted here also and why say "named file" (it should say 'FILE= specifier') to be consistent with line 2.

15. In 9.3.4.5,9-7:41-48, what happens when the RECL= specifier is omitted? Does it depend on whether an existing file has been opened or not? Is it processor-dependent in some case? This should be explained.

16. In 9.3.4.4,9-7:35, change "If the FORM=" to 'If this' to be consistent with the description of other specifiers (for example, 9.3.4.1,9-7:14).

17. In 9.3.4.6,9-8:5, change "If the BLANK=" to 'If this' to be consistent with the description of other specifiers (for example, 9.3.4.1,9-7:14).

18. In 9.3.4.8,9-8:18, add after last sentence:

    'For an existing file, the specified action must be included in the set of allowed actions for the file. For a new file, the processor creates the file with a set of allowed actions that includes the specified action.'

    This is stated for other specifiers. To be consistent, it should be stated here.

19. In 9.3.5,9-9:3, there is only one such specifier. Should change "A specifier that requires a scalar-char-expr" to 'The STATUS specifier' and make other appropriate changes throughout the rest of the paragraph.

20. In 9.6.1,9-19:26, change "An INQUIRE" to 'An inquire by file or inquire by unit form of the INQUIRE' since an inquire by output list must not have a FILE= or a UNIT= specifier.

21. See 9.6.1.20,9-21:39-44. IOLENGTH specifier is not listed in 9.6.1,9-19:4:25 but section 9.6.1.20 is under 9.6.1. This seems organizationally incorrect. I think 9.6.1 should list IOLENGTH as an inquire-spec and a constraint should be added saying IOLENGTH may only be used in an inquire by output list and must be used in an inquire by output list and is the only inquire specifier used in an inquire by output list. In 9.6.1,9-19:3, remove "either of the inquire by file or inquire by unit forms of." Change 9.6,9-18:41 to 'or INQUIRE (inquire-spec) output-item-list' and add constraint that this inquire-spec must be IOLENGTH=.

22. See 9.6.1.13,9-21:2. What is "maximal" for a sequential file? The largest current record? The largest record allowed for the file? Processor dependent? Need more definition here.

23. Change 12.5.2.2,12-9:22-23 and 12.5.2.3,12-10:30-31 to use same wording as in 11.1,11-1:17-18. That is, change "if ..." to 'if the ...', "...agree with ..." to '... be identical to ...', and "... on ... " to '... specified in ...'. When different phrasing is used it is, you aren't sure if something different is being said and it takes a while to determine if that is so.

24. See 13.1,13-1:13-16: Specific names are deprecated (see B.3.2,B-5:18) since they are supposedly redundant but how is the functionality of passing an intrinsic as an argument accomplished in Fortran 8x?

25. See 13.12.73,13-38:39-44 and 13.12.73,13-39:1-23: What is the initial seed for RANDOM? If processor-dependent, should say so. If RANDOMSEED must be called to set seed before calling RANDOM, should say so.

    Under RANDOM, a reference should be made to the seed set by RANDOMSEED. It should be clearly explained that the value returned by RANDOM changes from one call tc the next starting from the seed. That is, the value returned is the next value generated by the pseudo-random number generator that is mentioned under RANDOMSEED.

26. See 13.12.74,13-39:18. Should this processor-dependent value always be the same for all calls with no arguments or should (can) this value change (for example, based cn system clock)?

27. See B.2,B-1:10-. The rationale for some items (alternate return and ASSIGN and ASSIGNED GO TO) is not given in terms of redundacy in ANSI X3.9-1978. Instead, these features are shown as being redundant because of features of 8x (this is not a valid rationale according to the rules given in defining obsolescent features). For example, the statement of rationale for alternate returns should be changed to 'the better method was a computed GO TO in ANSI X3.9-1978 but since this feature is now deprecated, it is recommended that SELECT/CASE construct be used instead of alternate returns.'

28. Change B.3,B-3:3-13 to

    '(3) As deprecated features fall into disuse, it is recommended that future Fortran standards committees move these features from the deprecated list to the obsolescent list.

(4) It is recommended that future standards committees do not consider moving language features defined in this revision to the obsolescent list in a succeeding Fortran revision that do not appear on the list of deprecated features in Appendix B of this revision.

(5) A standard-conforming processor must continue to support a deprecated feature until it is moved to the deleted list after being moved to the obsolescent list. Therefore, at least three revisions must occur, the use of the feature must be insignificant, and the feature must be redundant before a feature is deleted and is no longer required to be supported by a standard-conforming processor.'

29. See C.9,C-7:40-41. RECL= is defined for both sequential and direct files (as the "maximal record length of the file"). Also it is defined if unconnected and inquire is by file for an existing file (see 9.6.1.3,9-21:1-5). Table C.1 should be corrected.

30. Why is there no type parameter for INTEGER type? What is offered to replace INTEGER*1, INTEGER*2, and INTEGER*4 that is consistent with what was done in providing the precision-selector to replace REAL*4 and REAL*8? Not being able to define the desired size (presumably in decimal digits) of an INTEGER seems to me to severely restrict portability.

63

106.JLW-3
page 1 of 5

Amoco Production Company
4502 East 41st Street
Post Office Box 3385
Tulsa, Oklahoma 74102
Research Center

87251ART0253 JLW082
September 8, 1987

Dr. William Van Snyder
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109

Comments on Fortran 8x

Dear Dr. Van Snyder:

Thank you for your letters dated August 3 and August 4 (two on August 4). I will send copies to Lloyd Campbell (the editor, for the clarification on structure arrays with array components), Rich Ragan (head of subgroup handling RANGE), Jim Matheny (head of subgroup handling CASE), and Dick Hendrickson (to be included in the X3J3 distribution).

The quick answer to your question on structure arrays with array components is that subobjects of these are not allowed. (They are in appendix F, with a description of what they really are.) You are, of course, right that the text should not be unclear/contradictory on this.

I have no quick answer for your RANGE comment, and must defer to Rich Ragan's subgroup. Likewise your comments on CASE must be considered by Jim Matheny's subgroup. Note, however, that the colon notation for a range is also used in array constructors, and this would not be as amenable to your change. Also, I believe the reason for disallowing REAL case values is similar to the objection to allowing REAL do-variables: round-off error could affect the result in unpredictable ways.

Thanks again for your comments.

Sincerely,

Jerrold L. Wagener
Research Director

JLW:sd

cc: J. C. Adams, w/attach.
L. W. Campbell, w/attach.
R. H. Hendrickson, w/attach.
J. H. Matheny, w/attach.
R. R. Ragan, w/attach.

521

Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109

August 3, 1987

Jerrold L.Wagener
AMOCO Production Company
Tulsa, OK 74102

Dear Mr. Wagener:

I have found some statements in the S8.104 document that appear to be contradictory. They should be clarified, or the implied restrictions eliminated.

The statement on page 6-2, lines 34-35 is unclear. What subobject results if both the parent structure and component are arrays? I could find no explicit restriction that prevents a parent structure and component to be arrays, but this statement seem to imply it. If this is indeed a restriction, the sentence on page 6-8, lines 25-28 and the example on page 6-10, line 14 contradict this restriction.

The statement on page 6-7, lines 24-25 that a SET RANGE statement cannot affect the actual argument bound to a formal argument introduces a bewildering special case. Prohibiting a SET RANGE statement from affecting the actual argument bound to a formal argument is equivalent to prohibiting an assignment to a structure component of a formal argument from affecting the corresponding structure component of the corresponding actual argument. The latter is clearly silly. The statement was added between S8.103 and S8.104, perhaps in response to Robert Allison's letter ballot. It would have been better to have solved the problem addressed in Allison's ballot by some other means.

Sincerely,

W. Van Snyder
Mail Stop 301-490

Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109

August 4, 1987

Jerrold L.Wagener
AMOCO Production Company
Tulsa, OK 74102

Dear Mr. Wagener:

I enclose this proposal separately from a related proposal on the CASE statement because the proposals are independent issues:

Replace lines 12 to 14 on page 8-4 by
{
The block following the CASE statement containing the matching selector is executed. If the last statement in a block is a CONTINUE statement that is not a *do-terminator*, the following block if any is executed. If the last statement in a block is not a CONTINUE statement, or is a CONTINUE statement that is a *do-terminator*, execution of the construct is completed.
}
This would provide the frequently useful semantics of the "C" case statement, but with control passing into the next block only by active selection instead of by default.

One might want to invent a new keyword such as PROCEED, but there seem to be too many keywords already.

Sincerely,

W. Van Snyder
Mail Stop 301-490

523

Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109

August 4, 1987

Jerrold L.Wagener
AMOCO Production Company
Tulsa, OK 74102

Dear Mr. Wagener:

As a consequence of the size of X3J3/S8.104, I have only now noticed the CASE statement is not usable for REAL *case-expr*. This might be repaired by the following small changes:

On page 8-3, replace lines 24-26 by
{

R812	*case-expr*	is *scalar-numeric-expr* or *scalar-char-expr*

}
Replace lines 30-36 by
{

R814	*case-value-range*	is *case-value* [*rel* * [*rel case-value*]] or * *rel case-value*
R814.5	*rel*	is < or .LT. or <= or .LE.
R815	*case-value*	is *scalar-numeric-const-expr* or *scalar-char-const-expr*

}
OR
{

R814	*case-value-range*	is *case-value* [*rel-op* * [*rel-op case-value*]] or * *rel-op case-value*

Constraint: If *case-value-range* is of the form
case-value rel-op * *rel-op case-value*, each instance of *rel-op* must be one of <, .LT., <= or .LE., or each instance of *rel-op* must be one of >, .GT., >= or .GE.

R815	*case-value*	is *scalar-numeric-const-expr* or *scalar-char-const-expr*

}
The constraint on R814 could instead be expressed by more syntax rules.

Allowing a LOGICAL *case-expr* provides no functionality or performance benefit as compared to an IF statement, so I have not included that possibility.

Replace from line 39 on page 8-3 to line 11 on page 8-4 by
{
8.1.3.2 Execution of a CASE Construct. The execution of the SELECT CASE statement causes the case expression to be evaluated. The resulting value is called the case discriminant and must match exactly one of the selectors of one of the CASE statements of the construct. If the case selector is a case value range list, the case discriminant matches the selector if it matches any of the

case value ranges in the list. A case discriminant with value *c* is defined to match a case value range in the following circumstances:

(1) If the case value range contains a single value *cv*, *c* matches the case value range if and only if *c* .EQ. *cv*.

(2) If the case value range is of the form $cv_1\ rel_1\ *\ rel_2\ cv_2$, *c* matches the case value range if and only if $cv_1\ rel_1\ c$ .AND. $c\ rel_2\ cv_2$ is true.

(3) If the case value range is of the form $cv_1\ rel\ *$, *c* matches the case value range if and only if $cv_1\ rel\ c$ is true.

(4) If the case value range is of the form $*\ rel\ cv_2$, *c* matches the case value range if and only if $c\ rel\ cv_2$ is true.

(5) If *c* matches no other case selector and a CASE DEFAULT selector is present, *c* matches the CASE DEFAULT selector.

(6) If *c* matches no other case selector and no CASE DEFAULT selector is present, an error is signalled and program execution terminates.

}

On page 8-4, line 16, replace "index" by "discriminant".

On page 8-4 replace lines 20-30 by

{

8.1.3.3 Examples of CASE Constructs. INTEGER and REAL signum functions:

```
INTEGER FUNCTION SIGNUM (N)
INTEGER N
SELECT CASE (N)
CASE (* < 0)
        SIGNUM = -1
CASE (0)
        SIGNUM = 0
CASE (0 < *)
        SIGNUM = 1
END CASE
END FUNCTION SIGNUM
```

```
INTEGER FUNCTION SIGNUM (X)
REAL X
SELECT CASE (X)
CASE (* < 0.0)
        SIGNUM = -1
CASE (* == 0.0)
        SIGNUM = 0
CASE (* > 0.0)
        SIGNUM = 1
END CASE
END FUNCTION SIGNUM
```

}

Delete lines 1-20 on page 8-5.

I don't know if there are other examples using the CASE construct and the colon notation for a range. If so, they would need to be changed.

This change would allow the CASE construct completely to subsume the functionality of the arithmetic IF with no loss of efficiency.

Sincerely,

W. Van Snyder
Mail Stop 301-490

64

TO: X3J3

FROM: Jerry Wagener

SUBJECT: Embedded SQL in Fortran

X3H2 has requested that X3J3 collectively comment on the comments submitted by X3J3 members on the Embedded SQL ballot. As I understand it, X3H2 would appreciate a collective response from X3J3 as well as the individual comments.

The X3J3 individual ballots are in 105.JCA-18, item 18 (pages 221-238) in the Liverpool premeeting distribution. Attached herewith are the X3H2 draft responses to the X3J3 ballot comments. Copies of the original ballot comments receiving responses are also attached.

Agenda time has been reserved on Monday morning at Ft. Lauderdale to discuss this X3H2 request.

X3H2 SQL2 RESPONSE TO BALLOT

Document Number: X3H2-87-168

Date: September 29, 1987

Author: Alan R. Hirsch
Amoco Corporation

Subject: Responses to Comments on X3H2-87-79
Embedded SQL Ballot from X3J3 FORTRAN

References

1. Letter Ballots from X3J3; Jeanne Adams, Chair; 25 June 1987; X3H2-87-167.

2. (draft proposed) Database Language Embedded SQL; March 1987; X3H2-87-79.

One ballot voted YES with a comment attached, one ballot ABSTAINING with a comment attached, and three ballots voted NO, WITH REASONS were received from the X3J3 Chair, Jeanne Adams in response to X3H2-87-79, Embedded SQL as coordinating liaison for the FORTRAN language. Herewith are proposed responses to the comments that were distributed to the members of the X3H2 committee as document X3H2-87-167.

Valerie G. Bowe (UNISYS)

Significant blanks in <embedded SQL statement>s are a matter of style in the evolution of the SQL database language. SQL has been accepted as a standard language for defining and manipulating databases consisting of tables that generally conform to the relational data model. Though SQL was originally developed as a research project by IBM, it has evolved and been adopted by many other vendors of database products. The style of SQL conformed, wherever possible, to a "natural" language consisting of English words strung together in phrases and clauses separated by commas. English syntax uses blanks as word separators and this style has been carried forward in the SQL formal definition.

FORTRAN has chosen a syntax where word-like tokens, e.g. statement identifiers, variable names, etc., are generally alternated with special characters, e.g. parentheses, brackets, etc., or numbers. Some exceptions exist to this general rule: The do-stmt permits the label to be omitted if the do-construct is terminated by a end-do-stmt. The syntax scanner must scan ahead in the do-stmt until it finds a comma or a left-parenthesis before it can distinguish the do-stmt from an assignment-stmt. SQL syntax, with significant blanks, is constructed to limit the look-ahead to one token.

Embedded SQL statements are introduced by "EXEC SQL". Converting <embedded SQL statement>s to the corresponding SQL module language is commonly performed by a pre-processor that scans the entire program for the "EXEC

SQL" prefix, extracts the SQL statement that follows from the program, and either deletes the entire SQL statement or replaces it with a call-stmt. This modified program is then passed to the FORTRAN compiler. Thus, the FORTRAN compiler need never know about the SQL language syntax.

SQL was designed as a readable language that requires little added documentation to convey the intent of the statements to other people. Natural language was used as a model to achieve this goal. People using SQL are accustomed to reading natural language with blanks used to separate words in a phrase.

Identifiers in SQL may be up to 18 characters in length. The <underscore> character is used to separate words in an identifier instead of a blank to improve readability.

The X3H2 committee finds that significant blanks are a basic part of the SQL language that cannot be removed without major change to the SQL language. Moreover, SQL statements embedded in FORTRAN host programs are introduced by EXEC SQL thus sufficiently isolating them from the rest of the program. We will retain significant blanks in <embedded SQL FORTRAN program>s unless we receive a concrete proposal from X3J3 that removes their need.

Kurt W. Hirchert

The members of X3H2 have considered your comments and respond as follows:

TRANSLATION OF EMBEDDED SQL STATEMENT

It is common practice among vendors of database management systems that use SQL to provide a preprocessor to scan a program for <embedded SQL statement>s. The preprocessor would scan for the EXEC SQL trigger, extract the following SQL statement or declaration section, and perform the actions specified in syntax rule 8 of section 9.1 <embedded SQL host program>. The resulting program is required to be valid FORTRAN program by syntax rule 10 in section 9.1. The FORTRAN processor would never see the <embedded SQL statement>s.

If this model is implemented, then an <embedded SQL statement> could be the action-stmt of an if-stmt or the do-termination.

If an <embedded exception declaration> appears in the program and it specifies a <go to> <exception action>, then the goto-stmt must be valid at every place an <embedded SQL statement> occurs following the <embedded exception declaration>. An implementation may choose to generate an if-construct following the call-stmt that is generated for the <SQL statement>. This may lead to an invalid FORTRAN program.

PROPOSAL

Add a syntax rule 4 in section 9.1 <embedded SQL host program> and renumber syntax rules 4 through 10 to be 5 through 11 as follows:

> An <embedded SQL statement> in an <embedded SQL FORTRAN program> shall not appear as the action statement of an IF statement nor as the terminating statement of a DO construct.

This proposal could be restricted to the case when an <embedded exception declaration> specifies an <exception action> of <go to> but if an <embedded exception declaration> is added to the <embedded SQL FORTRAN program> later on, it may invalidate some <embedded SQL statement>s that were correctly placed before.

FORTRAN VARIABLE DEFINITION

The <FORTRAN variable definition> applies to host variables that appear within the procedure argument list defined in the SQL module that is generated from the SQL statement. These variables must appear in the parameter list of that procedure. In this case, they are only dummy arguments. The variable definition need only specify the FORTRAN type attribute so that SQL will know how to use or store the data they represent.

The <FORTRAN variable definition> is retained in the derived host program. The variables may be defined with other attributes so long as these other attributes permit the variable name to be used in the argument list on the call-stmt that invokes the SQL procedure.

COMPILATION UNIT

A compilation unit includes all program units that are passed to a language processor at one time.

The statement label appearing in a <go to> <target> must be contained in the same subroutine, function, or main program as the <embedded exception declaration> is contained. See syntax rule 1.d in section 9.2, <embedded exception declaration> for the scope of the <go to> <target>.

STATEMENT NUMBERS

We propose the following change:

PROPOSAL

In syntax rule 2 of section 9.6 <embedded SQL FORTRAN program>:

delete the second sentence,

remove the word "Otherwise, " from the beginning of the third sentence, and

change the first sentence to read:

> An embedded SQL statement> may be specified wherever an executable FORTRAN statement may be specified except as the action statement of an IF statement or as the termination statement of a DO construct.

CONSOLIDATED DOCUMENT

We understand your concerns. Procedurally, however, it was felt that a single addendum could be handled more expeditiously than six similar addenda. X3H2 will be responsible for interpreting the addendum and will tailor the interpretations to the host language(s) to which the question applies.

USE OF EXEC AS A KEYWORD

EXEC was chosen as a generic escape mechanism to identify sections of code embedded in a host program that did not obey the syntax of the language of that host program. SQL further identifies the specific "foreign" language that is embedded in the host program.

Removing the single escape word EXEC would cause the name of each "foreign" language to possibly become a reserved word in the host language. Looking to the future when other embedded languages may be defined, using EXEC as the trigger minimizes the impact to the host language.

PLACE OF SPECIFICATION AND EXECUTABLE STATEMENTS

Following your suggestion, we propose the following:

PROPOSAL

Insert a new syntax rule 4 in section 9.1 <embedded SQL host program> and renumber syntax rules 4 through 10 to be 5 through 11:

> <host variable definition>s shall appear in the <embedded SQL host program> only in such places as the host language syntax permits. <SQL statement>s shall appear in the <embedded SQL host program> only in such places as the host language syntax permits for executable statements.

STATEMENT LABELS

We believe that the statement in syntax rule 1.d. of section 9.2 <embedded exception declaration> is clear enough to establish the intent.

CHARACTER DATA TYPES

Clause 8 refers to the syntax rules for <procedure> in SQL and the cases refer to data type declarations for SQL variables in SQL procedures, not for FORTRAN variables in FORTRAN programs. CHARACTER(1) is valid in SQL variable definitions.

VALID FORTRAN NAMES

Though we agree with your point about insignificant blanks, a <FORTRAN host identifier> is intended to be string of characters excluding blanks to which all occurrences of the same string of characters in the same order but including blanks map. Therefore, any occurrence of a FORTRAN host variable in an EXEC SQL block must be written without interspersed blanks.

ARRAYS AND COMPLEX AND LOGICAL TYPES

X3H2 recognizes the need for SQL to support arrays and complex and logical data types. If the support is strong enough, they will incorporated into the next version of the SQL database language.

EMBEDDED EXCEPTION DECLARATION OPTIONS

The X3H2 committee hopes that host programming languages would evolve to incorporate adequate exception handling mechanisms that could, in general, replace the <embedded exception declaration> in the future. At this time, the choice was made to support the simplest form of branching.

LENGTH OF VARIABLE NAMES

X3H2 agrees that the possibility exists that an <embedded SQL statement> could be correctly interpreted as a FORTRAN statement in the FORTRAN 8x proposed standard.

PROPOSAL

Add a sentence at the end of syntax rule 1 of section 9.6 <embedded SQL FORTRAN program>:

> If a statement can be interpreted as either a FORTRAN statement or either an <embedded SQL statement>, an <embedded SQL begin declare>, or an <embedded SQL end declare>, it shall be interpreted as an <embedded SQL statement>, <embedded SQL begin declare>, or <embedded SQL end declare>, respectively.

ERROR IN C LANGUAGE TYPES

There is an error in clause 8, case b, item 5.

PROPOSAL

In syntax rule 8b5 in clause 8 Data Type Binding, change the reference to "C double" to "C float".

Richard R. Ragan

X3H2 appreciates your comments and responds as follows:

END-EXEC KEYWORD

This keyword is expressly allowed only for COBOL programs. In a FORTRAN program, no statement terminator will be required since FORTRAN uses the end of line as a statement terminator. Please see syntax rule 2 in section 9.1 <embedded SQL host program>. Sentence 2 states:

> An <embedded SQL statement>, <embedded SQL begin declare>, or <embedded SQL end declare> contained in an <embedded SQL FORTRAN program> shall not contain a <SQL terminator>.

SIGNIFICANT BLANKS

---- Copy of response to V G Bowe, above ---

John K. Reid

X3H2 appreciates your comments and responds as follows:

USE OF EXEC SQL TRIGGER IN FORTRAN 8X

--- Copy of Response on Length of Variable Names to K W Hirchert, above

OTHER COMMENTS

Our reponses to your other comments are:

1. Proposal:

   Add a clause to the last sentence of section 4.2, item 1 so that it reads:

   > Such a hybrid application program is defined...separate SQL <module>, and that each <embedded SQL begin declare> and each <embedded SQL end declare> have been removed.

2. Proposal:

   In syntax rule 10a of section 9.1 <embedded SQL host program>, replace the last two words, "this document", with "Database Language SQL".

3. The compilation unit can contain more than one FORTRAN program unit. For example, the compilation unit could contain a main program, some subroutines, and some functions, each of which is a program unit. In this context, the generic word "program" seems to cover this case.

4. Proposal:

   In the format for <embedded SQL FORTRAN program> in section 9.6, change the phrase "See the Syntax Rules." to "See the Syntax Rules for this Section.".

5. Proposal:

   Delete the second sentence of syntax rule 2 in section 9.6 <embedded SQL FORTRAN program> and remove the word "Otherwise, " from the beginning of the third sentence.

6. Statement labels are indeed allowed on <embedded SQL statement>s because there is no specific prohibition on statement labels. To specify the statement label in the Format of section 9.6 would essentially duplicate the description contained in the FORTRAN Language Standard. X3H2 prefers to leave it out.

7. Proposal:

   In syntax rule 5 of section 9.6 <embedded SQL FORTRAN program>, change the phrase "a host variable" to "one or more host variables".

Brian T. Smith (Argonne National Laboratory)

X3H2 appreciates your comments and responds as follows:

1. Proposal:

   In the Function description of section 9.2 <embedded exception declaration>, change the word "whebn" to "when".

2. Proposal:

   In syntax rule 4 of section 9.6 <embedded SQL FORTRAN program>, change the first sentence to read:

   A <FORTRAN host identifier> is any valid FORTRAN variable name with all blank characters removed.

87272ART0172

Unisys votes yes on the X3H2 document, Database Language Embedded SQL, with one reservation.

We are concerned by the statement on page 24 of the document that "blanks are significant in <embedded SQL statement>s" of <embedded SQL FORTRAN program>s. This is a major inconsistency within the FORTRAN language. A FORTRAN programmer, who uses the insignificant blanks feature of FORTRAN, would find it more that a little troublesome to remember that blanks are now sometimes significant. Also, FORTRAN implementors will now have the burden of writing processors which must deal with both insignificant and significant blanks.

Bowe

Hirchert

For the most part, my review of this document was limited to those sections pertaining to Fortran, as I do not have the expertise to adequately review the sections pertaining to the other languages.

My negative vote stems from the fact that the description in this document is insufficiently precise to serve as a specification for a product. In particular, I found the following "holes":

- It is unclear whether an embedded SQL statement is to be allowed as the object of a logical IF statement. If the translation of the SQL statement would be multiple Fortran statements, it would also be necessary to convert the logical IF into a block IF. Also recognizing an embedded SQL statement, in which blanks are significant, as the terminal part of a Fortran statement, in which blanks are not significant, may be too much of a burden on the translation processor.

- I believe it is necessary to specify that an embedded statement not be the terminal statement of a DO loop. If the translation of an embedded SQL statement would be multiple statements, there is no placement of the label on the translation that would be correct both for transfers of control and termination of the loop.

- Fortran does not have a concept of declaring a variable. Instead, names are specified to have particular attributes, such as type. The attribute of being a (scalar) variable is deduced from the absence of specifications or usage contrary to that hypothesis. Thus, it is necessary to specify what specifications or usage of a name are inconsistent with its being specified in an embedded SQL declaration. For example, the document should probably prohibit appearance in other type specifications, appearance as an array declarator (in a DIMENSION or COMMON statement), appearance in an EXTERNAL statement, appearance in a PARAMETER statement, definition as a statement function, and reference as a function. On the other hand, it could permit appearance in a COMMON statement, appearance in a SAVE statement (including the SAVE of a common block in which it appears), appearance in a DATA statement, and appearance in the dummy argument list of the host program unit.

- The document uses the term "compilation unit". This term is not defined here or in the Fortran standard and is ambiguous in current usage. Do you mean the minimum unit that can be separately compiled (i.e., a program unit) or that part of a program which is compiled at one time (possibly several program units)? This question has relevance in determining the effective range of an embedded exception declaration.

- The statements concerning which embedded SQL statements can have statement numbers make no sense. Note that in the Fortran standard all statements can have statement numbers. However, the statement numbers on some statements may not be referenced.

In addition to the above questions, I have a number of concerns that would not be sufficient to cause me to vote negatively on this document, but which I would like to see addressed:

- I object to the procedures that lead to X3J3 members being forced to vote as individuals on this document. At not time were we ever asked whether we, as individuals, were willing to accept this burden. Moreover, the question we have been asked is unreasonable. Few, if any, of us have the expertise to vote on this

document as a whole. The best we can be expected to do is review its implications on Fortran.

- The editorial approach in this document is poorly suited for its intended audience. Most users of embedded SQL and implementors of the translators for embedded SQL are likely to be interested only in a single host language. This intermingling of rules for 6 different languages only serves to confuse things. Ideally, the rules for the different languages should be described in separate documents or at least in separate sections of this document. As an aid to those few people who are interested in multiple host languages, identical language should be used to describe concepts and rules common to all of the languages and a mechanism such as revision bars should be used to distinguish text specific to a particular lanugage from text common to all languages.

  At the very least, syntax which is different in the different host language should be provided separately for each language in question rather than providing a description of the union of all such syntaxes!

- I have misgivings about the use of the word EXEC as the initial keyword on all embedded SQL statements. This is preempting a "useful" keyword and may eventually interfere with extension of the host language standards or extension of this standard to a host language already using the keyword (or reserved word) EXEC. In addition, EXEC is a misleading keyword when applied to declarations. I suggest the SQL EXEC would be preferable to EXEC SQL for executable statements and the SQL BEGIN DECLARE SECTION would be preferable to EXEC SQL BEGIN DECLARE SECTION for declarations. Similarly, SQL WHENEVER may be preferable to EXEC SQL WHENEVER.

- Although it may be implicit in your description of equivalent modules and host language programs, I would appreciate an explicit statement that embedded SQL declare sections may appear only where the equivalent host language declarations may appear and that executable embedded SQL statements may appear only where executable host language statements may appear.

- Failure to use appropriate host language terminology and syntax can be misleading. For example, the syntax for a Fortran statement label (not statement number) is similar to, but not identical to, the syntax for an unsigned integer constant, so that an embedded exception declaration really ought to contain the former rather than the latter.

- In clause 8, case d, item 3, reference is made to a data type CHARACTER(L). Note that Fortran has only CHARACTER*(L) or CHARACTER*_l_ (where _l_ is an unsigned integer constant).

- The statement that any valid Fortran name is a <FORTRAN host identifier> could be misinterpreted. Note that since blanks are not significant in Fortran, a name such as N CASES is valid in Fortran. I assume that a name with embedded blanks would not be valid in SQL.

- It is unfortunate that this standard does not support Fortran arrays or the Fortran types LOGICAL and COMPLEX.

- The allowed options on embedded exception declarations are inadequate. In Fortran, I can imagine wishing to use an assigned GOTO or computed GOTO. In

Ada, I would strongly desire the ability to use the language's exception handling mechanisms. It might be desirable to add a third form that would bracket arbitrary host language statements as means of providing a general mechanism.

- Although the current Fortran standard allows only 6 character variable names, there is no immediate ambiguity, but if longer variables names are permitted (as in the proposed revision of the Fortran standard and many implementations of the existing standard) the possibility exists of ordinary Fortran statements beginning with EXEC SQL. (Since we were not provided with the syntax of SQL statements, I can't discount the possibility that there may be statements that would be both valid embedded SQL statements and valid Fortran statements.) It seems incredibly short-sighted not to address this issue.

- I hope that every C reviewer has pointed out the error in saying that SQL REAL should correspond to C double. Since you say elsewhere that SQL REAL corresponds to C float, it is unlikely that anyone would be mislead, but an error as serious as this really ought to be fixed before this document is released to the public.

X3 Subgroup Letter Ballot

(20) 22

Accredited Standards Committee
X3, Information Processing Systems*

Doc. No. X3J3-87-104/JCA-15

Date: May 1, 1987
Project:
Ballot Period: May 10, - June 20, 1987

Subject: Database Language Embedded SQL

X3J3 is coordinating liaison to X3H2

Ballot Closes: NOON

FOR ACTION

Ref. Document: X3H2-87-32
X3H2-87-79

Statement:

X3H2 has requested that X3J3 ballot on the above referenced document as coordinating liaision to X3H2.

Question:

Do you approve X3H2-87-79 (draft proposed) Database Language Embedded SQL, March 1987, for submission to X3 for further processing as an American National Standard?

_____ Yes

X No, with reasons

1) END-EXEC is not consistent with FORTRAN Keywords (hyphen not allowed).

2) Blanks are not significant in FORTRAN but are significant in embedded SQL. This will confuse FORTRAN users.

Mail to: Jeanne Adams
NCAR/SCD
PO Box 3000
Boulder, CO 80307-3000

NAME Richard Ragan
(PLEASE PRINT)

SIGNATURE Richard R. Ragan

DATE: 6/12/87

*Operating under the procedures of The American National Standards Institute.

NOTE: If you find that you cannot vote YES and wish to be recorded as NOT VOTING or voting NO, please state this and explain the reasons for your position in the space above or on a separate sheet.

X3 Secretariat: Computer and Business Equipment Manufacturers Association, 311 First Street, NW, Suite 500, Washington, DC 20001-2178

Tel: 202/737-8888
Fax: 202/638-4922

Reasons for No vote

John Reid

An embedded SQL FORTRAN program consists of a mixture of Fortran statements and embedded SQL statements, which I take to include <embedded SQL begin declare> and <embedded SQL end declare>. It is essential that the SQL processor can distinguish between Fortran statements and embedded SQL statements. This is achieved by starting each embedded SQL statement with EXEC SQL. This is adequate for Fortran 77 because no keyword commences like this and identifiers may have at most 6 characters.

The reason for my NO vote is that I cannot be sure that this will be adequate for Fortran 8x. The current draft maintains the interpretation of blanks as insignificant and allows identifiers of up to 31 characters. The simplest fix is to state in rule 1 on page 24 that if a statement can be interpreted as either a Fortran statement or an embedded SQL statement, it is interpreted as an embedded SQL statement.

I also have the following comments:

1. In the last sentence of page 4, you need to state that each <embedded SQL begin declare> and each <embedded SQL end declare> is removed.
2. On page 15, line 4, replace 'this document' by 'Database Language SQL'.
3. On page 15, line 11, replace 'program' by 'program unit'.
4. On page 24, line 6, replace 'Syntax Rules' by 'Syntax Rules of this Section'.
5. On page 24, syntax rule 2, replace each 'statement number' by 'statement label', which is the correct Fortran term.
6. Page 24, syntax rule 2: The BNF does not appear to allow for an <embedded SQL statement> to have a label.
7. On page 25, line 1, replace 'a host variable' by 'one or more host variables'.

Comment: I am unqualified to comment on this draft standard X3H2-87-79, Database language Embedded SQL, March 1987, and do not have the time to become sufficiently knowledgable of the Database SQL [reference 6]. My vote of 'Yes' or 'No' would be misleading, and therefore I refuse to vote.

I have however two comments to make:

1. Page 16, line 3, change 'whebn' to 'when'.

2. Page 24, points 3 and 4 are contradictory.
"A Fortran variable name is allowed to be X Y where there is a blank between X and Y (as blanks are insignificant in X 3.9-1978) yet according to point 3, blanks are significant in SQL so X Y is not the variable XY. How do you resolve this contradiction?

Bruce E Smith
5/14/87