SUMMARY OF CHANGES TO FORTRAN DRAFT

In response to the first public review comments on the draft proposed Fortran standard (X3J3/S8.104), a number of changes were made to the document resulting in a revised draft standard (X3J3/S8.112).

Numerous editorial changes proposed by X3J3 in their final ballot commentary were considered in addition to the public review commentary. These changes corrected many errors and resulted in a document that is editorially sound. Improvements were made to the technical content of the draft as well.

A table of items summarizes the technical changes made based on the public review. A rationale is submitted for making these changes. Two more general responses from the public review letters are included in the final section of this report. These two responses were approved formally by X3J3 at their last meeting.

TABLE OF CHANGES MADE TO FORTRAN DRAFT

GENERAL CONCEPTS

The concept of deprecation was removed. The concept of obsolescence was retained.
INCLUDE was added.
Blanks were made significant in free form source.

PROCEDURES AND PROGRAM UNITS

User-defined elemental functions were removed.
Interface blocks were changed to accomodate user-defined generic specs
(permitting overloaded procedure names).
The syntax for internal and module procedures were made the same as for
external procedures.

DATA CONCEPTS

Pointers were added.
RANGE/SET RANGE and ALIAS/IDENTIFY were removed.
All Intrinsic Types are parameterized.
The MIL-STD 1753 bit intrinsic functions were added.
The new form of the DATA statement was removed.
Derived types (and structures) were allowed in COMMON and EQUIVALENCE.
Parameterized derived type was removed.
Array objects of zero-size may be declared.
Array constructors use implied-DO syntax;(/,/) pairs instead of square brackets.
Vector valued subscripts were restored from an Appendix in S8.104.
Binary, octal and hexadecimal constants were added.

CONTROL STRUCTURES

DO TIMES was replaced by DO WHILE.
Overlapping CASE selector ranges were prohibited.
The case action was well defined even in the absence of DEFAULT.

INPUT OUTPUT

Partial record (stream) I/O was added.
Binary, octal and hexadecimal edit descriptors were added.
The VALUES= I/O specifier was removed.

RATIONALE FOR MAJOR CHANGES TO DRAFT FORTRAN STANDARD

(Jeanne Adams, Chair, and Jerrold Wagener, Vice-chair)

GENERAL CONCEPTS

1. Deprecated Features Removed

The Deprecated Features have been removed entirely. Storage association, including COMMON and EQUIVALENCE, had been deprecated (removal possible in 20+ years). The public review comments contained significant opposition to this, so the concept of deprecated features was removed. Obsolescent features remain, however. These are features that could be removed in the next revision of standard Fortran, in 10+ years or so. There are only a few, rather minor, obsolescent features, such as arithmetic IF, double precision DO indices, etc. (They're all identified in S8.112).

2. INCLUDE Added

Probably the single strongest request from the public review comments was to add INCLUDE to Fortran 8x. Though modules provide most of the functionality of INCLUDE, INCLUDE is so strongly entrenched as common practice that many want it standardized. Thus, INCLUDE was added (and modules were kept).

3. Blank Significance

Blanks are significant in free form source. This decision was not made lightly, and a necessary consequence is that Fortran must support two incompatible source forms from now on. Strong public support for significant blanks, and the existence of significant blanks in virtually all other programming languages, make this decision desirable. To maintain compatibility with Fortran 77 codes, Fortran 8X will continue to support the traditional fixed form, with its special formats. Blanks remain insignificant characters in fixed form. It seems impossible to combine the fixed and free forms in a way that is compatible with Fortran 77; therefore Fortran will have two source forms for the forseeable future.

PROCEDURES AND PROGRAM UNITS

1. Remove User-defined Elemental Functions

One of features that was removed to reduce the complexity of the draft Fortran Standard was user-defined elemental functions.

2. Change Interface Blocks

The interface facility was unduly restricted in S8.104. The feature in S8.112 is generalized so that all procedures (both external and module) may have operator and generic properties. The feature was simplified in response to the comments received on complexity.

3. Internal and Module Procedures

Prior to the public review, procedures defined in modules could be given generic names and could be associated with user-defined operators. These properties did not extend to external procedures. (External procedures are Fortran 77 procedures; module procedures were (almost) the same, except their definitions were contained in MODULE program units rather than external to all other program units of the application.) In essence, the public review comments requested that either (1) module procedures and external procedures be exactly the same, or (2) get rid of module procedures. The committee chose option (1), removing all operator syntax from module procedure definitions and requiring a unique name for all module procedures in a given module. Operator and generic properties are given to module and external procedures alike via a generalized form of the procedure interface block.

Why were module procedures retained now that they're the same as external procedures? There appear to be three reasons, a software engineering reason, a reliability reason, and an efficiency reason. The first is that since modules are program units for grouping related data that is used globally in the application, it is highly desirable to be able to package with those data objects the procedures whose primary (or only) purpose is to provide various operations on that data. The second reason is that the interface information for a module procedure does not have to be duplicated in a procedure interface block as it does for an external procedure; as duplication is always error-prone, module procedures therefore provide increased reliability over external procedures when explicit procedure interfaces are required. And the third reason is that module procedures enhance the processor's ability to perform the optimization of inline expansion.

DATA CONCEPTS

1. Pointers Added

Another very strong request from the public review comments was for pointers. Pointers had not been added previously, primarily because of concerns about their impact on efficiency. (One principle in developing Fortran 8x was to maintain Fortran's historic characteristic of high run-time efficiency). There were two central issues in the ensuing discussion on pointers. One had to do with declaring pointees (pointer targets) as well as pointers, as a means not to preclude certain optimizations because of the presence of pointers, and hence help preserve efficiency. The second issue was whether pointers should be provided via a data type attribute (presumably these would be "safe" pointers, while providing adequate capability for dynamic objects) or via a general pointer data type. These were tough decisions, but in the end a pointer facility involving POINTER and TARGET attributes was adopted.

2. RANGE and IDENTIFY Removed.

These array features allowed an alias to be given to a portion of a parent array, and the shape of this alias could change dynamically. This introduced complexities/confusion involving "declared bounds" and "effective bounds" concepts that permeated the language. There was significant objection to this in the public review comments, and RANGE and IDENTIFY were therefore removed. The functionality that they provided has been at

least partially restored by pointers; array can be declared with the pointer attribute and subsequently refer to a dynamically varying array section.

The pointer facility provides all of the removed RANGE functionality (without the disadvantages of RANGE) and most of the IDENTIFY functionality (all except for skewed sections). These features were removed to simplify the language.

3. Parameterized Intrinsic Types

All of the intrinsic data types (integer, real, complex, logical, character) have been parameterized in a consistent way, with the KIND parameter. The functionality provided is that of, for example, REAL*8 and INTEGER*2, but with slightly different syntax. For example, REAL*8 may be written as REAL(8) or REAL(KIND=8), and similarly with the other data types. The KIND numbers (e.g., 8) are not specified, but the KIND intrinsic function provides a measure of portability. For example KIND(1.0D0) returns the double precision (REAL*8) KIND value for that processor; to continue this example, if DP were an integer symbolic constant given this value, then REAL(DP) would be a portable way of specifying REAL*8.

This change was prompted by significant public review protest to the "selectable precision" capability originally provided. With selectable precision, REAL(8) meant 8 decimal digits of precision, and the processor had to provide the smallest numeric storage unit available having at least REAL(9); even if they both produced the same kind of numeric object, it could not be legally associated across procedure boundaries. The KIND parameterization resolves this and other problems with selectable precision and it is more like current common practice. However, a measure of the functionality of selectable precision, which is important for some numerically sensitive algorithms, is retained in the SELECTED_REAL_KIND intrinsic function. This function takes an argument that specifies the desired number of decimal digits of precision and returns the appropriate corresponding KIND value for that processor. For example,

```
INTEGER, PARAMETER :: LP = SELECTED_REAL_KIND(20)
```

would establish LP (which could stand for "long precision") with the appropriate KIND value for objects with 20 decimal digits of precision. Then REAL(LP) could be used to declare objects of that KIND (if the processor supports them - otherwise an error condition exists).

The KIND parameter is similarly used for complex, integer, and logical declarations. This provides the functionality of current common practice (e.g., INTEGER(2), LOGICAL(1)), as requested in many of the public review comments.

Another reason for the consistent application of KIND parameterization to intrinsic types was the significant call in the public review comments largely from international sources, especially Japan, but also from U.S. vendors, to support additional character sets such as Kanji. (This is sometimes called multi-byte character support, since Kanji has about 10,000 characters and Hanzi (Chinese) has about 7,000 characters. Most Fortran 77 implementations of CHARACTER use one byte per character, which is adequate for the

ASCII or EBCDIC character sets, but not for character sets with more than 256 characters.) With parameterized character, things like CHARACTER(KANJI) (where KANJI has been parameterized to the appropriate integer value for that processor) can be used to specify objects of the desired character set.

The KIND parameterization of the intrinsic data types addresses in a conceptually consistent way the problems with selectable precision, the accommodation of common practice, and the need to support large character sets.

4. MIL-STD 1753 Bit Intrinsic Functions Added

The public review comments strongly requested that Fortran 8x provide bit-level capabilities. This surprised no one, as there have been several attempts in the past to add a bit data type. The problem has always been that there are different groups of users that require different kinds of bit data type. For example, bit arrays are appropriate for some, whereas variable-length bit strings are required by others. There has never been a bit data type proposal presented that adequately satisfies this diverse set of bit processing needs, and the public review comments, while underscoring the need for such, did not help in identifying what that facility might be. Therefore, essentially to provide something as a stop-gap measure until a better solution is devised, the MIL-STD bit intrinsic functions were added to the Fortran 8x intrinsic functions. These functions provide access to the bits of integer data objects.

5. New Form of DATA Statement Removed

This was removed in response to the general request to reduce the complexity of the language.

6. Structures Allowed in COMMON and EQUIVALENCE

The two things consistently most requested in Fortran have been array operations and data structures, which have been important parts of Fortran 8x almost from the beginning. However, structured objects, which can contain any mix of component types (real and character, for example), were not permitted in COMMON blocks. The rationale for this restriction is the Fortran 77 prohibition on having numeric and character objects in the same COMMON block. It was thought that not allowing data structures in COMMON was not a serious restriction, since they could be made global by placing them in MODULE program units. However, the public review comments strongly requested that data structures (and all other data objects) be allowed in COMMON blocks. Some Fortran 77 restrictions on COMMON blocks have therefore been removed to allow structures in COMMON, though there are still some rules on how things can be placed in a given COMMON block so that a measure of portability is retained. Any mix of data objects can still be placed in MODULES, which is the simplest way to provide global access to data safely and portably. But MODULE packaging of global data objects does not allow the storage association manipulation of internal structure that COMMON provides, and the public review comments requested that capability.

In order to make storage association in Fortran compatible with the new data types, structures were allowed in EQUIVALENCE as well

7. Parameters Removed from Derived Type

The committee decided to remove parameterization of user-defined types to reduce the complexity of Fortran 8x in response to the simplicity issue in the public review comments. The possibility of adding material to the Journal of Development on this topic may be considered at a later time, if the second public review indicates the need. The ISO community supports inclusion of this feature.

8. Declaration of Zero-sized objects was allowed.

Allowing zero-sized arrays regularizes the syntax, and eliminates a restriction. It also simplifies the description of arrays in the draft standard.

9. Syntax of Array Constructors Changed

The syntax was modified to be more consistent with other conventions in the language. The square brackets were removed. There was strong ISO support for their removal.

10. Vector-valued Subscripts Restored

Vector-valued subscripts had been removed before the public review because they were the only form of array section for which the elements are not necessarily regularly spaced within an array object. Vector-valued sections were, however, the only means of providing gather-scatter functionality, and the public review comments indicated that such functionality was needed. (Also, hardware support for vector-valued subscripts has become more widespread in the last couple of years.) Vector-valued subscripts were therefore reinstated.

11. Binary, Octal, Hexadecimal Constants Added

There were many requests in the public review comments for this feature. They are available in current compilers and represent common practice.

CONTROL STRUCTURES

1. DO WHILE Added, DO TIMES Removed

At public request, a WHILE option was added to the DO construct. With the addition of the bit intrinsics, INCLUDE, and DO WHILE, all of the MIL-STD 1753 extensions to Fortran 77 have been included in Fortran 8x.

In response to the general criticism that the draft was too complex, and because DO WHILE was added, DO TIMES was removed

2. Overlapping CASE Ranges Disallowed

The CASE construct is processed in parallel, it allows processors to branch immediately to the selected case. The public review comments requested this prohibition.

3. CASE Action has been Well-defined (even in absence of DEFAULT)

The CASE construct has been modified so that, if no match is found and there is no CASE DEFAULT, execution continues with the statement following the CASE construct. This was considered preferable to terminating on an error condition.

INPUT/OUTPUT

1. Partial Record I/O Added

The international community provided most of the public comment on character processing, such as described above for Kanji support. Another request from this community was for varying character capability. It was finally agreed that the best way to provide this was as a varying character module, but that some sort of stream I/O (e.g., GETCHAR/PUTCHAR) was needed to do that. As a stream I/O facility would have much wider use than just this application it was decided to add a stream I/O capability to Fortran 8x. Initially considered were GETCHAR and PUTCHAR intrinsic functions. However, the final decision was to add versions of the READ and WRITE statements that work on only part of the current I/O record - hence, the term partial record I/O.

2. Binary, Octal and Hexadecimal Edit Descriptors Added

In response to a large number of requests in the public review commentary, this feature was added. It is available in many current Fortran compilers. It was also a feature supported by the ISO Fortran community.

3. VALUES= I/O Specifier Removed

The VALUES= specifier in a cilist was removed because it can be impossible to determine where an error occurred in an input stream. NULLS= is provided for the NAMELIST feature.

TWO GENERAL RESPONSES FROM X3J3 TO PUBLIC COMMENTERS

There are two responses prepared by the General Concepts subgroup of X3J3 that address the rationale for the changes in a more general way. The first begins by addressing the concept of deprecation, the second examines the issue of the size and complexity of the language.

Both responses were approved by X3J3 at their last meeting. The public comment responses have been approved, and the letters to commenters are in preparation.

RESPONSE TO THE ISSUE OF DEPRECATION

There was considerable reaction to the concept of deprecation in the public comments. Although many approved of the concept of language evolution and felt that features should eventually be removed from the language, the clear majority of the comments about language evolution were opposed to large-scale removal of features, particularly storage association, from Fortran. There was also considerable opposition within X3J3 towards the language evolution model. As a result of the public comment evaluation we have removed the concept of deprecation from the draft of Fortran 8X. In addition, we have further integrated some of the new features with the related old features; for example, structures and objects of nondefault kinds can be in common and can be equivalenced. Although we cannot control the actions of future standardization committees we believe that, as a practical matter, this means storage association will never be removed from Fortran.

We have not removed the concept of language evolution nor have we made any changes to the current obsolescent features list. We believe that provision must be maintained for removing features from the language that prove to be seldom used, unsafe or non-portable. An example from FORTRAN 66 would be the Hollerith data type. Although it was widely implemented, it was not standardized in FORTRAN 77 because the implementations differed significantly. A current example would be DO loops with REAL index variables. They are on the obsolescent features list because they are not portable; the actual number of iterations may vary among machines with different round off characteristics. With the rules and procedures that X3J3 follows the only features that could be considered for removal from the next revision of Fortran are the features in the obsolescent features list; the clear intent of the current committee members is that these features not be removed unless they have truly fallen into disuse.

We realize that adding new features without removing old ones makes the language both larger and more redundant. We believe that the redundant new features make codes easier to write and maintain and that, just as current programmers generally use IF-THEN-ELSE blocks rather than 3 branch IFs, most programmers will quickly adopt the new redundant syntax. New features such as MODULE/USE are admittedly "large", but they have considerable public support; there does not seem to be a practical way to remove equally "large" current features to maintain the language size. Although there is sharp disagreement within the committee, the majority has basically accepted the increase in size of the language and feels that the benefits are worth the added complexity.

Since FORTRAN 77 is a subset of Fortran 8X we believe there should be no need to extensively modify codes and little or no conversion costs associated with moving a standard conforming code to the new 8X compilers. Fortran 8X has added a large number of new intrinsic functions. In moving a code to an 8X compiler it will be necessary to check for conflicts between user supplied external functions and the new intrinsics. The conflicts can be resolved by adding an EXTERNAL statement. Most, but not all, of the new function names are longer than 6 characters and should not conflict with existing standard conforming codes.

RESPONSE TO THE ISSUES OF SIZE AND COMPLEXITY

The public expressed considerable concern about the size and complexity of Fortran 8X. Although many felt that the language was neither too big nor too complex the clear majority of people commenting on size or complexity disagreed. Related comments observed that this was a new language with little relationship to traditional FORTRAN; that the committee did not follow its charter to "standardize existing practice"; that the size and complexity would make it difficult to learn to use the language. Others felt that compilers would be inefficient, late in arriving and be riddled with errors. Some suggested that the language should be split into two languages, either by defining one or more subsets of Fortran 8X or by defining a new language with most or all of the obsolescent, deprecated, or redundant features of FORTRAN 77 removed. A significant minority of the committee members echoed some or all of these sentiments. There was often support for some of the features, or sets of features, in Fortran 8X, usually coupled with strong opposition to other features in the language.

At the same time there was considerable pressure in the comments and from the international counterpart of X3J3 to produce a standard in a timely manner and to not significantly reduce the scope of the standard.

We have made significant changes to the proposed standard in an attempt to reach a broader consensus within X3J3, the user community, and the international community. We have removed the RANGE and IDENTIFY statements, generic precision, and the concept of deprecation. We have simplified specified precision, generic functions and structure definitions. We have added an INCLUDE statement, POINTERs, the DO-WHILE loop, the military standard 1753 BIT manipulation functions, support for a logical data type which does not have to occupy a full word, support for additional character sets, and we have allowed structures and objects of nondefault kind to be in common and equivalence. FORTRAN 77 continues to be a complete subset of Fortran 8X.

We agree that the draft language is larger and more complex than FORTRAN 77. Although there is disagreement on this point within the committee, the majority believes that the added functionality is beneficial and the accompanying increases in size and complexity are not excessive. Most of the newly standardized features are based on extensions available in existing Fortran compilers. Others have existed for years in "small" languages such as C or Pascal. In some cases we have tried to standardize the intent and functionality of common practice rather than the exact syntax. A good example would be the specified precision feature. We did not attempt to standardize the "REAL*4" syntax since the implementation of this varies from machine to machine. Rather we standardized a parameterized data type. Individual processors can easily map their version of "REAL*4" onto the parameterized types as an extension.

From a programmers point of view most of the new features are reasonably independent of each other. Array processing, specified precision, and modules and derived data types can all be used independently of each other. Although some basic familiarity with the entire language will be desirable there is no need to be an expert in the parts that are not needed for a particular type of problem. In addition, since FORTRAN 77 is a subset of Fortran 8X there is no need to "unlearn" anything until new features are used as replacements for existing features. Nor is there any need to extensively modify existing standard conforming codes to run them with a Fortran 8X compiler. Because of the large number of new intrinsic functions added it might be necessary to add EXTERNAL statements to resolve name conflicts with existing external functions.

Some of the size of Fortran 8X is due to the addition of redundant features. We believe that, although they are redundant, the new control structures and new data definition statements will make it easier to write and maintain programs. The FORTRAN 77 IF-THEN-ELSE block is redundant with IF-GOTO and the three branched IF, yet most programmers prefer it since it adds clarity to the program flow. We predict that DO-ENDDO will quickly replace DO-CONTINUE as the looping structure.

Many of the new features are provided to eliminate one of FORTRAN's biggest weaknesses: the chance for error in the subroutine call process. Interface blocks, keyword arguments, modules, and new data attributes all allow, but do not require, adoption of a style that allows for compile time checking of most of the call interface.

Although clearly a matter of subjective judgment the committee majority does not believe the language will be too big for micro computers. Nor do we believe that the language will be too complex for the average scientist or engineer to use.

We are concerned about program efficiency. We have removed two statements often commented on as being inefficient: the RANGE and IDENTIFY statements. We don't believe that the other additions to the language are inherently inefficient. Many of the new array intrinsic functions perform complex operations and will execute "slowly". However, there does not seem to be any reason to believe they will execute any slower than the corresponding FORTRAN 77 code would. In fact, they might execute faster since the compiler can generate "optimal" code or call an efficient library routine without having to recognize a complex source code pattern. Some of the new features, particularly some of the array processing features, will require compiler optimization. however, we do not believe the optimizations are beyond the range of current compiler technology.

The committee is almost unanimously opposed to the concept of subsets. Experience with FORTRAN 77 indicated that subsets were almost never implemented; and when they were there was usually marketplace confusion about what exactly was implemented. Given the magnitude of the changes made to the standard there does not seem to be a practical way to define a subset. The majority of the committee also does not believe it would be practical or desirable to attempt to define a new "modern" Fortran with the deprecated or redundant features removed. There is simply too much investment in old codes to propose a wholesale replacement with a new language. Since FORTRAN 77 is a complete subset of Fortran 8X and since we have included many of the current extensions we believe there can be a gradual shift to the new features and still preserve the investment in old codes.