Future Revision of Fortran

UK Input to Requirements Specification

1 General Position

The UK believes that the revision scheduled for publication in 1996 should be limited to the incorporation of the corrigenda material and other editorial improvements agreed before 1995, the movement of F90 obsolescent features into the obsolete category, the definition of a new set of obsolescent features, plus a very small number of extensions. This list of extensions should be agreed by WG5 no later than July 1994.

The UK believes that extensions considered for inclusion in F96 should be limited to those that serve a significant well-recognised need, are relatively cheap to implement (both in terms of changes to the standard and in changes to F90 compiling systems) and are relatively uncontroversial; accordingly, the UK proposes that the extensions discussed below be those included in F96.

The revision planned for 2001 could, and probably should, be a more comprehensive updating of the language and as such the requirements specification will need to be completed relatively sooner to allow the implementation body time to develop the necessary document changes. The UK also includes in this paper an initial list of major areas that should be considered for incorporation in F2001.

2 Extensions to be included in Fortran 96

The following extensions are those that the UK believes should be included in Fortran 96; they are expressed in the format specified in WG5-N870. As stated above, the UK does not believe that F96 should include major new functionalities; the list of extensions accepted for F96 should be restricted to enhancement or regularisation of the functionality already present in Fortran 90.

Number: **13**
Title: **FORALL statement**
Submitted by: UK Fortran Panel
Status: For consideration for 1995/96 Revision
References: High Performance Fortran Language Specification, Version 1.0, May 1993, section 4.1.
X3J3/S8.104, section F.2.3, June 1987.
Basic Functionality: Allows array assignments to be specified in terms of a set of array element and array section assignments, as in HPF.
Rationale: The UK believes it was a mistake that this statement was left out of F90, and would wish for it to be included in F96. Although there is some sympathy for the view that a FORALL construct as well is desirable, the consensus opinion of the UK is that only the statement form should be included at this stage. It is recommended that the proposals in this area of the HPF group be considered as the prime starting

point for the design. However, given the deficiencies of the current HPF draft, this should be a guide only.

Estimated impact: No effect on existing standard-conforming programs. Possible effect on HPFF programs. Small effect on processors.

Detailed specification: As in HPF

History: Submitted by UK Panel, June 1993.

Number: **14**

Title: **Kind Parameters for Derived Types**

Submitted by: UK Fortran Panel

Status: For consideration for 1995/96 Revision

References: X3J3/S8.104, June 1987.

Basic Functionality: Allows derived types to have Kind parameters.

Rationale: The lack of Parameterised Derived Types is a glaring anomaly in F90. All the intrinsic types are parameterised but the derived types are not. This is already greatly restricting the use of derived types in any context where selected kind intrinsic types are used.

A common requirement is for libraries of procedures that are generic over floating-point kinds. This is straightforward in F90 provided all arguments are of intrinsic type; the kind parameterisation enables this well. This is not the case if any argument of a derived type is required because the lack of parameterisation requires separately defined and named types for each relevant kind. Similarly the lack of parameterisation of derived types means that separate modules would need to be written for each character kind, with different type names etc., for a facility like the varying string module.

Very nearly complete proposals existed in F8x which were dropped as part of the political processes leading to the finally adopted version of the language. These could be used as an indication of the possibilities for developing an extension to meet this need.

Estimated Impact: No effect on existing standard-conforming programs. Medium impact on processors.

Detailed Specification: The UK believes that the Fortran 96 development body would wish to define the detailed specification but the UK is willing to provide a detailed specification if requested. The following serves to illustrate the requirement.

The basic idea behind the proposals involved in this area for F8x was that a type definition could include a set of essentially dummy parameters of type integer. These could be used in any specification expressions used to declare values such as lengths and bounds for components. When a structure of such a type was declared, actual values would have to be provided again by specification expressions for these parameters; hence determining the relevant values for the components. The complication is on how to deal with KIND type parameters where the values have to be statically determinable.

We are suggesting here that a restrictive but practical

solution to this problem is to allow only one kind type parameter for any derived type and that this be called KIND. This restricted parameterisation would nevertheless allow all but the most esoteric types to be created and problems to be programmed.
We could with this proposal define types

```
TYPE VECTOR3(KIND)
  REAL(KIND=KIND) :: x(3)
ENDTYPE VECTOR3
TYPE VECTOR4(KIND)
  REAL(KIND=KIND) :: x(4)
ENDTYPE VECTOR4
```

We could then declare variables

```
INTEGER,PARAMETER :: sp=KIND(0.0),dp=KIND(0.0D0)
type(VECTOR3(sp)) :: displ,vel,acceln
type(VECTOR4(dp)) :: spacetime,mmtmeng
```

where the variables for displacement, velocity and acceleration are declared as single precision three-vectors, and the spacetime and momentum-energy vectors are declared as double precision four-vectors.
As in the F8x proposal, each such type declaration implicitly creates a KIND inquiry function with argument of the type. This inquiry function returns the declared value for the KIND parameter of the argument. Because of the special nature of KIND, this is simply an overload of the existing generic function. The overloads will be resolved by the type of the argument. In structure declarations and elsewhere, the KIND parameter value must be determined by a static expression.

History: Submitted by UK Panel, June 1993.

Number: **1b**
Title: **Initial status of Pointers**
Submitted by: UK Fortran Panel
Status: For consideration for 1995/96 Revision
References:
Basic Functionality: Allows specification of a defined initial status for pointers and pointer components. For dynamic objects, such as unsaved local variables, allocatable arrays, and allocated pointer targets, this must include every instance created at run time.
Rationale: It should be possible for the F96 programmer to specify that pointers, and pointer components, are to be created with a defined initial status. The lack of this facility has and will continue to result in code which is unnecessarily inefficient; see for example the varying string module which is by no means abnormal in its design.
A possible syntax and semantics providing this capability is given here.
Estimated Impact: Allows significantly more efficient execution of programs. No effect on existing programs. Small effect on compilers.
Detailed Specification: This section contains a possible syntax and accompanying semantics that could be used to allow a user

to specify a defined initial state for pointer variables and pointer components. It is a minimalist suggestion due originally to Jerry Wagener. It merely allows the qualifier NULLIFY to be specified along with the POINTER attribute, for example,

```
REAL,POINTER(NULLIFY) :: pra(:,:)
```

The semantics proposed is that such a pointer would initially be disassociated. For example, the varying-string module could use the type:

```
TYPE VARYING_STRING
  CHARACTER,POINTER(NULLIFY) :: chars(:)
ENDTYPE VARYING_STRING
```

The semantics now are that whenever an object of type VARYING_STRING is created, the pointer component will be created in the disassociated state. It could be regarded as having zero length and the assignment procedure could safely check for an already allocated variable and deallocate it. The declaration

```
type(VARYING_STRING) :: page(66)
```

would create an array of zero-length strings. This would greatly reduce the degree of memory leakage in the example implementation.

This proposal is minimalist in that it permits only setting the initial state of a pointer to be disassociated. It does not allow the pointer to be initially associated with an existing target, nor are any non-pointer components specifiable as being initially defined.

If this approach is agreed by WG5, the UK is willing to provide a detailed specification.

History: Submitted by UK Panel, June 1993.

Number: **16**
Title: **ALLOCATABLE Components**
Submitted by: UK Fortran Panel
Status: For consideration for 1995/96 Revision
References:
Basic Functionality: Allows components of variables of derived type to be allocated.
Rationale: It would be highly desirable for components of a derived type to be specified as ALLOCATABLE. This would allow for very much more efficient implementation of applications using dynamic sized but otherwise simple structures. For example, if the varying-string module were based on the type

```
TYPE string
  CHARACTER,ALLOCATABLE :: chars(:)
ENDTYPE string
```

it is likely that implementations based on straightforward compilation of the published module would be much more efficient.

Estimated Impact: Allows notably more efficient execution of programs. No effect on existing programs. Small effect on compilers.

Detailed Specification:
History: Submitted by UK Panel, June 1993.

Number: **3a**
Title: **Features to be declared obsolescent in F96**
Submitted by: UK Fortran Panel
Status: For consideration for 1995/96 Revision
References: none
Basic Functionality: Definition of the following features of Fortran 90 to be obsolescent:

- Computed GOTO
- Alternate ENTRY
- Statement Functions
- DATA statements among the executables
- Assumed character length functions
- Old fixed form source
- Assumed size arrays
- Pointers in storage associated contexts
- The .EQ., .LT., etc. forms for the relational operators

Rationale: The UK believes that the features specified in the list are essentially redundant and their use in new code is generally undesirable. It would wish to signal that should they fall into disuse, as we would hope, then they will be candidates for removal in a future revision of the Fortran language. There would be majority support in the UK for declaring all storage association based facilities obsolescent at this stage, particularly given the inherent inefficiency of the mechanism in any distributed memory MIMD or SIMD multi-processor environment, but the UK refrains from proposing this at this point on the grounds that this would almost certainly be controversial.
Estimated Impact: No effect on standard-conforming programs until publication of Fortran 2001. Small effect (detection of new obsolescent features) on compilers.
Detailed Specification: Extension to Appendix B2 and corresponding changes to fonts in the body of the standard.
History: Submitted by UK Panel, June 1993.

3 Features to be considered for F2001

The following features form the initial set of proposals from the UK for consideration for incorporation in Fortran 2001. They would provide much needed extension and regular completion of the language design directions successfully initiated in F90, but are considered too large for incorporation in F96.

Number: **5c**
Title: **Exception Handling**
Submitted by: UK Fortran Panel
Status: For consideration for 2000/2001 revision
References: X3J3/S8.104, section F.4, June 1987.

Basic Functionality: Allows specification of action to be taken at an exception.
Rationale: There is a clear need to include facilities in the language to handle exceptions. There should be both a standard mechanism for handling exceptions and a set of defined standard exception conditions, e.g. divide by zero.
Estimated impact: No effect on existing standard-conforming programs. Significant effect on processors.
Detailed specification: The UK is willing to produce a detailed specification if requested by WG5. It should be a significant simplification of the proposal in X3J3/S8.104, section F.4.
History: Submitted by UK Panel, June 1993.

Number: **17**
Title: **Input/Output for Derived Types**
Submitted by: UK Fortran Panel
Status: For consideration for 2000/2001 revision
References: X3J3/S8.104, Section F.4, June 1987
Basic Functionality: Extends intrinsic input/output syntax to cover derived types.
Rationale: The lack of facilities to extend the intrinsic I/O syntax and semantics to cover derived type objects consistently is a major constraint on both proper data-abstraction/semantic-extension capabilities and will similarly constrain any attempts to extend these ideas further into "Object Oriented" programming.
Estimated impact: No effect on existing standard-conforming programs. Medium effect on processors.
Detailed specification: The UK is willing to produce a detailed specification if requested by WG5.
History: Submitted by UK Panel, June 1993.

Number: **23**
Title: **Multi-threaded execution facilities**
Submitted by: UK Fortran Panel
Status: For consideration for 2000/2001 revision
References:
Basic Functionality: Allows program execution to proceed using multiple execution threads.
Rationale: This facility is needed to make better use of current and likely future architectures. This should be implicit rather than explicit in much the same way as "parallel" execution of array operations is possible without being mandated. The language should provide facilities for whole blocks of code that can be executed out of sequence or in any sequence.
Estimated impact: No effect on existing standard-conforming programs. Medium effect on processors.
Detailed specification: The HPF proposals should be studied and used as a guide in production of a detailed specification.
History Submitted by UK Panel, June 1993.

Number: **18**
Title: **Object-oriented facilities**
Submitted by: UK Fortran Panel
Status: For consideration for 2000/2001 revision
References:
Basic Functionality: Allows the object-oriented model to be realised in Fortran.
Rationale: The F2001 language must provide support for and interfacing capability to object oriented programming. Much of the data-abstraction infra-structure included in F90 is already OO but there are other features of the OOP paradigm that need to be considered, if for no other reason that F2001 programs will have to be able to call/invoke facilities from libraries built in other languages to exploit OO techniques. One specific defect which hampers both the semantic extension and object oriented paradigms stems from the inability for the user of a type extension module to declare named constants of a type whose structure is private. This is essential to for true OOP. This problem arises because of the classification of expressions and the restrictions placed on what may appear in initialisation expressions. The F90 classification of expressions is overly restrictive, complicated and irregular. We have general expressions which are employed in the general execution of the program. These can use any data-object and procedure that is accessible and defined at the point of execution. However we also have constant expressions, initialisation, expressions, restricted expression, specification expressions and KIND expressions. These have highly involved overlapping definitions relating to the restrictions that apply and the contexts in which they must be used. These should be extensively revised and restrictions applied only where absolutely necessary. Since we require that the KIND of an object be determined at compilation it is essential that expressions determining KIND values, wherever they might occur, can be evaluated statically. PARAMETER values, except where they are used to determine KIND values, do not strictly need to be known at compile time. They, along with the initial values for variables, need be known only at load time or even as late as the first invocation of the procedure. Specification expressions where array bounds and character lengths are determined do not have to be evaluated, and frequently cannot even now be evaluated, until run time. This analysis leads to a simpler and less restrictive classification of expressions:

- KIND or static expressions that involve primaries that are literal constants, symbolic constants whose values are determined by static expressions, possibly involving intrinsic operators, and a restricted set of intrinsic functions;
- Initialisation expressions that involve primaries that are static expressions, accessible explicit procedures that are proper functions; and

- Specification expressions that involve any data-object or proper function that is accessible and defined at the time of invocation of the program or procedure.

We need to define "proper function". This would need to be a function whose value was uniquely determined by its arguments and which had no side effects. (The HPF concept of the PURE procedure is relevant here). This relaxation has vast benefits in allowing regularisation of treatment of derived type and intrinsic type objects. For instance, it allows the override definition of assignment to be used to initialise structure variables where this is necessary. It would also with the changes above allow the construction of facilities for a user to declare structured symbolic constants when USEing a type with its structure declared PRIVATE.
These are necessary steps towards full data-abstraction and OO facilities but require small upward compatible changes to the existing language. There would be other more significant changes and additions to fully support a Fortran flavour of the OO paradigm.

Estimated impact: No effect on existing standard-conforming programs.
Medium effect on processors.
Detailed specification: The UK panel is willing to provide detailed specifications if requested by WG5.
History: Submitted by UK Panel, June 1993.

David Muxworthy, for BSI Fortran Panel
June 1993