To: WG5, X3J3
From: Michael Hennecke
Subject: Response to X3J3's TR-C Liaison Report 96-106R1
References: ISO/IEC JTC1/SC22/WG5 N1178 (X3J3/96-069),
X3J3/96-106R1, X3J3/96-39.

Document X3J3/96-106R1 is the X3J3 Liaison Report on Interoperability of Fortran and C, based on the draft Technical Report WG5/N1178. I appreciate X3J3's review of this project. However, I have some concerns about the shifts on X3J3's opinion on basic functionalities of the TR, which occurred between the Liaison Reports X3J3/96-39 and X3J3/96-106R1.

1) BIND / KIND versus MAP_TO interoperability

X3J3/96-39 starts with a statement that "We do not advocate the MAP_TO approach", and topic (7) says that "A new keyword ... should be allowed in a Fortran derived type to require the same layout as in a C struct". WG5/N1178 specifies that using the <bind-spec> in a derived type definition does exactly this, and specifies one-to-one mappings from basic C datatypes to intrinsic Fortran datatypes (of suitable KINDs) rather than requiring a type conversion at a call to procedures with an EXTRINSIC/BIND(C) interface. It also carefully applies some recursive specification of mappings from C to Fortran types, in order to model as closely as possible C's recursive declarator construction.
X3J3 recently changed its opinion on the general direction of inter-callability, now "X3J3 would prefer to see a MAP_TO attribute as sketched out in X3J3/95-295 (HPF Calling C Interoperability Proposal)" in topic (1) of X3J3/96-106R1.
I do not agree with X3J3's opinion that X3J3/95-295 and WG5/N1178 provide comparable functionality. In particular, I am not aware of a formal specification of "recursive MAP_TO" to enhance X3J3/95-295 to allow consistent handling of C structs. I do also not agree that the MAP_TO approach places more responsibility on the processor than WG5/N1178 does. Document WG5/N1185 (X3J3/96-119) will discuss this.

2) C extern variables

X3J3/96-39, topic (1) as well as a straw vote conducted recently required the inclusion of a mechanism to access C data objects with external linkage from Fortran. WG5/N1178 contains an outline of such a mechanism, using MODULE variables. However, X3J3/96-106R1 does not take notice of this feature, and also does not discuss how access to global C data objects should be handled within a MAP_TO approach to interoperability.

3) C character strings

X3J3/96-39, topic (2) proposes "Introducing a C-style null-terminated character string", noting that this needs dynamic memory and some set of operations for these strings. WG5/N1178 contains the specification of derived type names for C strings and operations on such objects, without specifiying the actual implementation of these types (the most natural model implementation would be using a POINTER component). The processor is responsible for the mapping to C, this is easily possible by using the defined derived type names and relying on the fact that their internal representation is invisible to the user.
But X3J3/96-106R1, topic (4) now proposes to use a Fortran CHARACTER(1) array, which is not a varying length character string, to do the mapping from C to Fortran. This seems to be difficult both because it does not consider the fact that C strings have no maximum length, and because additional functionality is needed to instruct the processor to map such Fortran variables to C correctly.

These changes in X3J3's view of interoperability are quite drastic.
The WG5 strategic plan (WG5/N1151) requires that the contents of the TR must be technically acceptable to the primary development body prior to WG5

approval; according to X3J3/96-106R1 this is not the case for WG5/N1178.
It is therefore very important that the technical issues raised in
X3J3/96-106R1 are discussed within WG5 and X3J3, and that a direction for
interoperability with C which is acceptable to WG5 and X3J3 is specified
at the WG5 Dresden meeting.

For this reason, I would like to invite all members of WG5 and X3J3 to
carefully review the relevant documents as well as the email discussion
in sc22wg5-interop@ncsa.uiuc.edu (which is partially archived at
http://www.uni-karlsruhe.de/~SC22WG5/TR-C/Email/), and send their opinions
either to that list or the full WG5 reflector.

I will explain my personal preferences for BIND/KIND and against MAP_TO
in a separate document, WG5/N1185 (X3J3/96-119). Some more technical
comments on X3J3/96-106R1 are added below, feedback would be helpful.

> 1) ... MAP_TO ...

This topic will be discussed in document WG5/N1185 (X3J3/96-119).
I do not think that the suggestion (i) to use kind type parameters to
specify the mapping (rather than C datatypes) is useful with a MAP_TO
approach.

> 2) X3J3 would prefer to see the BIND attribute be renamed as EXTRINSIC,

This should only be done if the syntax and semantics of BIND and EXTRINSIC
are very similar, which is (currently) not the case. If it would be,
I have no strong preference for BIND over EXTRINSIC for procedures.
I would not like it very much to have EXTRINSIC in TYPEs, but again this
is not very important. A global change may be done at any time.

>   simply because there exist a number of Fortran implementations which
>   use the EXTRINSIC keyword for inter-language calling issues, as
>   specified by HPF.
                                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Which are these Fortran implementations? I only know the following
commercial HPF implementations, which to my knowledge all don't support
EXTRINSIC(C) procedures:

 - The Portland Group pgHPF 2.02, released Apr-96
 - Pacific Sierra VAST-HPF 3.4F beta, unreleased Apr-96
 - IBM XL Fortran 1.0, released Apr-96
 - Applied Parallel Research xhpf 2.0, released Jan-95 (old version)
 - The Numerical Algorithm Group f90 v2.2, released Apr-96

If somebody knows about an existing implementation that has EXTRINSIC(C),
I would be very interested to learn more about it.

> 3) It is not clear from section 3.1 of the draft TR, whether an external
>   procedure with an implicit interface may be specified in the BIND
>   statement. X3J3 would prefer that BIND be permitted only for
>   procedures with explicit interfaces.

This was the intention in N1178, and motivated the two constraints at
N1178, pages 23+ (edit for page 48, subclause 5.1 and edit for page 207,
subclause 12.5.2.2). I realize that there are some holes, e.g.
  INTEGER(c_int_ki), EXTERNAL, BIND(C,"foo") :: FOO
in the <specification-part> of a module. I'll fix this.

>   In addition, it is not clear whether the BIND statement can be
>   specified for the result variable of a procedure whose interface is
>   defined by an interface body.

N1178, page 8, after Editor's Note 4, says:
"The <bind-spec> may appear ... as a <prefix-spec> or <attr-spec> within
 an interface block for an external procedure.
 ...
 The <bind-attr> for external procedures ... may alternatively be

specified by a <<BIND statement>>."

This allows BIND as an <attr-spec> in a <type-decl-stmt>, and also as a <bind-stmt>. All of them only in an interface block because of the constraints above. The wording of the quoted paragraph may be improved to make the situation clearer. However, Note 3.14 on page 19 of N1178 shows the three possibilities that were intended.

> X3J3 would prefer that this not be
> allowed, and that, for procedures, the BIND attribute be specified only
> on the <function-stmt> or the <subroutine-stmt>. X3J3 considers BIND
> in this situation to be a property of the procedure as whole, analogous
> to PURE and RECURSIVE, rather than a property of the result variable.

I also consider BIND to be a "property" of the procedure as a whole.
IF BIND is also considered to be an "attribute" of the procedure, then I would prefer to keep the BIND statement to be consistent with section 5.2 of IS-1539 which provides a statement form for all "attributes".
If BIND is not considered to be an "attribute", then I agree that the <bind-stmt> for procedures could be dropped from the TR. However, it will very likely be considered to be an "attribute" of a MODULE variable which is used to bind to a C extern data object...

> 4) X3J3 believes that rather than pursuing a solution for C character
> strings involving a new derived type definition, the TR should specify
> a module which provides a set of procedures for conversion between
> Fortran character strings and character arrays with a character length
> parameter of one and a kind parameter of C_CHAR_KC. These character
> arrays would correspond to C character strings.

They won't, because actual args always have a fixed DIMENSION. See below.

> For example, a generic procedure named CSTRING might handle the
> conversion from a Fortran character value to a C null-terminated
> string.
>
```
>   PROGRAM P
>     USE ISO_C_STRINGS
>     INTERFACE
>       BIND(C, 'THING') SUBROUTINE SUB(STR)
>         USE ISO_C_KINDS
>         CHARACTER(LEN=1, KIND=C_CHAR_KC) :: STR(*)
>       END SUBROUTINE SUB
>     END INTERFACE
>
>     CALL SUB(CSTRING('HELLO, FORTRAN!'))
>   END PROGRAM P
```

From the example I deduce that CSTRING is a function.
How can this approach possibly handle INTENT(OUT) arguments of SUB?

If an INTENT(OUT) char[] argument of a C procedure is not converted at the CALL statement as shown above, how does Fortran know the size of the actual CHARACTER(1) array argument needed to hold the result?

Another advantage of defining a derived type and a limited number of operations on it is that the Fortran program cannot pass a "corrupted" C string, like a string that does not contain an ASCII NUL character: the type's components are PRIVATE, and the operations automatically handle the NUL-termination of C and truncation/padding of Fortran.

> 5) X3J3 concurs with the need for a type alias statement to handle C
> typedefs. However, it has some reservations about the possibility for
> interaction between the proposed type alias statement and the
> Parameterised Derived Types proposal, and possibly with any Object
> Oriented Fortran features which might be a part of Fortran 2000.

There is clearly a possiblity of interference with these two streams.
However, the proposed <type-alias-stmt> provides only slightly more functionality than a <rename> of derived types at a USE statement in

the current standard. Since the latter has to be dealt with anyway
in PDT or OOF, I expect the impact of a <type-alias-stmt> to be small.

> 6) Rather than creating a new <lang-keyword> to handle stdargs, X3J3
>   would
>   prefer that additional optional "linkage" specifiers be permitted on
>   the BIND specifier, for example, BIND(C, 'FOO', STDARG). If, at some
>   point in time, it is decided that it is desirable to support
>   interoperability between Fortran and some other language which
>   supports
>   several combinations of linkage, it is easier to permit the various
>   combinations to be individually specified, than it is to create
>   distinct keywords for each combination.

Both ways are possible and quickly interchangeable in the TR text.
In my opinion, it is easier the way N1178 does it because there is no
guarantee that some other language also uses STDARG to designate some
alternative binding. So it is likely that a new keyword needs to be
created for the proposed linkage specifier (instead of for LANG=),
which saves nothing. For example, for <lang-keyword>=K&R_C this would
read VARARGS (which is close), almost every other language is likely
to use a completely different terminology. Having a separate specifier
would span a tensor product with only very few valid combinations of
<lang-keyword> and <linkage-keyword>.

> 7) X3J3 feels that there must be some specification of the fact that the
>   dummy arguments specified in the interface body for a procedure with
>   the BIND specifier must match the arguments of the associated C
>   function. That is, that the order of the arguments is the same in the
>   two.

N1178, page 18, 1st par of 3.3.1 says:
"The <interface-body> that specifies a Fortran interface to a C procedure
 shall specify dummy arguments that correspond by position with the
 arguments of the C procedure, ..."

>   In some common existing implementations of interoperability, the
>   orders of the two are reversed.

Which are these implementations?

> 9) X3J3 feels that, in Section 3.3.1 of the draft TR, function results
>   of C procedures must be prohibited from being:
>   (i) of type COMPLEX or LOGICAL;
>   (ii) arrays; or
>   (iii) characters of length other than one.

(i) and (iii) are ruled out by N1178, page 18, 2nd par. The phrase
"a C type for which this TR establishes a corresponding Fortran type"
is used in several places to allow concentrating all these details in
section 3.2 of the TR: No C type is mapped to COMLEX, LOGICAL, or
CHARACTER(n>1), so this implies X3J3's restrictions. Also note that
"In all other cases, ... the behavior is processor-dependent", which
at least allows support of extensions from C9X (e.g. A.2 of N1174).