8th Jan. 1997

Individual comments concerning the misc. requirements
===

These comments were returned together with the votes concerning the priority of miscellaneous requirements for Fortran2000.

They are listed in the following (arbitrary) sequence:

1. C. Dedo
2. W. Clodius
3. C. Weber
4. M. Cohen
5. S. Whitlock
6. M. Hennecke
7. M. Zuern
8. D. Schmitt
9. P. Lignelet
10. S. Morgan
11. J. Reid
12. D. Muxworthy
13. French member body

=========================

1. Comments by Craig Dedo

=========================

*** Omit / Low / Medium / High ***

The notations "WG5 #" and "CW #" refer respectively to the numbers given each item in the WG5 Repository and Christian Weber's categorization of 16 October 1996 (X3J3 / 96-168).

I am not including comment on items which are already accepted as X3J3 Requirements or Minor Technical Enhancements (MTEs).

```
WG5 #    CW # Requirement Title
-----    ----    -----------
 7       2.1     Block comments
```
NO - Users can get this functionality by using the proposed Conditional Compilation facility.

```
92       2.3     Reserved Words
```
NO - One of the strengths of Fortran is the lack of reserved words.

```
         2.4     IMPLICIT NONE by Default
```
Low - Most compilers already have a compiler switch that does this.

77 3.4 Special Character Designations (as in C)
NO - Fortran should strenuously AVOID introducing C language constructs. Programmers can already get such functionality by defining CHARACTER constants for these control characters.

3.5 Definable evaluation sequence for Logical Operators
NO

21 4.1 BIT Data Type
HIGH - A wide variety of applications can benefit from this feature. We MAY already have a champion for this effort.

37 4.3 Unsigned INTEGER
HIGH - Not only is C Interoperability the issue, some types of data uses naturally use unsigned rather than signed integers. This feature has been in most other languages for years, so implementation should not be difficult.
I originally proposed this feature; If approved, I would volunteer to be the champion, if I could get some assistance from other members of the Data Subgroup.

5* 5.1 Exception Handling
HIGH - this capability would be a MAJOR improvement for Fortran. If we can find a champion, we should definitely pursue this feature. We may need to revise the feature somewhat from its original concept in order to accommodate the concerns of those who believe that it will slow down execution even if exception handling is not used.
I prefer a version of John Reid's work.

25 5.2 Extend EXIT
Medium - This probably could be an MTE. It certainly provides a natural extension of the existing language at very small cost.

33 6.1 Nesting Internal Procedures
NO - I really do not see very much value, especially in light of the effort which would be required.

42 6.2 Allow Internal Procedures as Actual Arguments

HIGH - Malcolm Cohen's arguments convinced me that this would be a valuable feature for Fortran 2000.

100 6.3 COPY_IN Intent Attribute
NO - Would this be equivalent to pass-by-value? If so, will not the C Interoperability TR already provide this functionality?

6.4 Interface to Internal & Module Procedures
HIGH

6.5 Initialization of Programs in Modules
Low

26 7.2 Non-rectangular sub-arrays
NO - I really do not see very much value, especially in light of the effort which would be required.

48 8.1 Variable Format Expressions (VFE)
YES - This is a popular extension. Those compilers which have it implement it in (almost) exactly the same fashion, so there already is a well-established informal specification. This feature would offer application programmers the opportunity to dynamically vary format specifications in an easier, more straightforward, and less error-prone fashion than is currently available with character run-time formats.

If approved, I would volunteer to be the champion of this feature. I have already written a paper (X3J3/96-071) on this feature. The paper proposes to ratify existing practice. If people are willing to simply ratify existing practice, the amount of work could be quite small (possibly an MTE), rather than the medium amount of work suggested by Christian Weber.

I do not believe that an intrinsic procedure such as the TO_CHAR procedure mentioned previously, would offer the functionality that users expect.

63* 9.1 STREAM I/O
HIGH - This is a very valuable addition to the Fortran I/O capabilities. It offers the application programmer the ability to read large, unstructured files and incoming data from a variety of physical devices.

65 9.2 Non-Advancing I/O with List-Directed I/O
HIGH - Christian Weber's comments say it all. An obvious correction to an unnecessary irregularity.

68 9.3 Any Kind of Integer for I/O Specifiers
HIGH - This is an unnecessary irregularity. It is my idea and the amount of work is quite small, so I would volunteer to be the champion. I believe that the need among application programmers is already much greater than what most others realize.

76 9.5 Default I/O Mode
NO - I really do not see the need.

79 9.6 TAB character in Fortran source
Medium - I originally proposed this item. It could be an MTE. I am willing to be the champion.

93 9.7 READ_EOR, etc. in INQUIRE
94 9.8 IS_EOR & IS_EOF in INQUIRE, READ, & WRITE
Low - Both of these could offer some useful functionality.

9.9 DEFAULT READ & DEFAULT WRITE
NO

9.10 Asynchronous I/O
HIGH - X3J3 has already approved a technical specification.

20 10.1 Command Line Arguments and Environmental Variables
HIGH - This is a major weakness of Fortran right now. This is shown by the numerous and wholly incompatible vendor extensions in order to provide this capability.
I wrote a proposal similar to the one which David Matoon submitted. I would be willing to be the champion of this feature.

47 10.2 POSIX Interface to Fortran 90
HIGH - POSIX provides some major capabilities, especially with respect to operating system interfaces. Thus, a POSIX interface can be a major benefit to the application programmer.

In discussing this feature with others, I believe that it is possible to develop this feature in a piecemeal fashion; a POSIX interface is not an all-or-nothing proposition. It is my understanding that the POSIX interface is defined as a series of functions and/or subroutines rather than a wholistic, indivisible specification.

I would be willing to be the champion of this feature and would attempt a piecemeal approach.

86 10.3 Operating System Support (SYSTEM Command or Subroutine)
HIGH - This is another feature which has been implemented by many different vendors in very different fashions.
I wrote a proposal on this feature. I would be willing to be the champion.

99 10.4 Handling of Error Messages
NO

102 10.5 Primitive Graphic Support
NO - While I am sympathetic to the proposer's needs, I believe that it is about 20 years too late for this to succeed. I do not believe that there really is much of a market for this feature. This is due to the enormous popularity of existing windowing APIs, which can provide not only the functionality of this feature, but also much more. While the various windowing API interfaces are incompatible with each other, it is possible for a private developer to create cross-platform GUIs which have the same

API, e.g., Interacter.

55 11.1 Regularize RANDOM_NUMBER Functionality
Low

80 11.4 Size Intrinsic for Derived Types
HIGH - C has this, why not Fortran?

81 11.5 Sort Intrinsic
NO - There are many different sorting algorithms. This functionality is much better provided by commercial or public libraries or by operating system utilities.

82 11.6 FFT Intrinsic
NO - This is a sophisticated numeric function. It is much better left to the high powered commercial or public math and engineering libraries.

90 11.7 Rounding and Bit Handling Functions
91 11.8 PATTERN=
97 11.9 POSITION and LOCATION
98 11.10 SCRIPT and SCALAR
No - I really do not see the need. Perhaps if all of these were very small work items, I would favor them.

13.1 Subset Language
NO - Creating various subsets would only complicate the standards making process. A lot of time and effort would be spent on this rather than going into the creation of needed new features.

13.2 Review of Implementation defined & undefined
Medium

Sincerely,
Craig T. Dedo Internet: Craig.Dedo@mixcom.com
Elmbrook Computer Services Voice Phone: (414) 783-5869
17130 W. Burleigh Place
Brookfield, WI 53005 Disclaimer: These opinions are mine alone.
USA They do NOT represent any organization.

"They that can give up essential liberty to obtain a little temporary
safety deserve neither liberty nor safety." -- Benjamin Franklin (1759)

========================
2. Comments by W. Clodius
========================

2.4   new   IMPLICIT NONE by default   small   L

Large incompatibility with heritage code makes this a difficult sell, although it is tempting. If this is to be done at all it should be done in stages. Create an "IMPLICIT ALL" construct to indicate code where IMPLICIT is should be used. Require compilers to have an option to turn off IMPLICIT typing (don't most compilers provide this already?). In a subsequent standard deprecate IMPLICIT typing.

3.5   new   Definable evaluation sequence for logical small   M
          operations

There are higher priorities for me, but most of the part time programmers of my acquaintence appear to assume short circuiting, and at least untill last year, a widely accepted authority on F90 (Michael Olagnon) was under the mistaken belief that Fortran required short circuiting. The problem is not that the language does not provide short-circuiting, but that a machine dependent exception can be thrown due to the execution of code that need not be evaluated to determine the result of a logical operation, and the processor does not suppress the exception upon noting that it need not be generated. This can occur for array out of bounds, divide by zero, integer overflow, and many other intrinsic operations. I now believe that an operator

would imply the wrong semantics, and believe that something like

PROVIDED (...) THEN IF (...) ...

would be preferable.

4.1   21   Bit Data Type, String   large   M

I think this is a usefull capability for the language, but with all the other tasks addressing this in full may be difficult. Instead I would encourage someone to attempt to implement this as a module using the planned capabilities of F2000, and identify any problems in using the language for such an implementation so that these limitations can be removed.

4.2   34   Varying length character with declared   large   O
          maximum

There are too many other irons in the fire to require the inclusion of a third character capability in the language.

5.2   25   Extend the semantics of the EXIT statement small   O/L

The little I have seen of this proposal suggests that the additional functionality it provides is small, and there is the devil to pay in the details.

5.3 new Real arguments in CASE/FORALL/WHERE small O/L

I mostly put this on the table to challenge people's beliefs, it did that for sure. I have other higher priorities that I do not want to be delayed by the controversy this would engender. I would like to note that there is very little research of any kind as to the relative safety/unsafety of any constructs in any language, and most arguments on such topics must rely on plausibility arguments, whose underlying assumptions are rarely examined.

There is no doubt that it is easy to use these constructs in erroneous ways and that people would use these constructs in such ways. My point is that there is no evidence that even if these constructs are not provided there is no evidence that people will not implement the equivalent of these constructs in order to commit the same errors. There is some anectodal evidence to the contrary. I remember a European teacher and F90 advocate noting that when he

forbade his students from using REAL DO loop indices, they all implemented the equivalent errors using DO WHILE codes. He blamed this on the existence of DO WHILE. However, the conceptiual and implementation differences between
DO WHILE(...) and DO ... IF (.NOT. ...) EXIT are so small, that I believe that if he had forbidden DO WHILE, they would have made the same mistakes using DO ... IF (.NOT. ...) EXIT ....

The problem is not that REAL DO indices make it easy to make a mistake, the problem is that the REAL type has significantly more complexity and gotcha's than other common types. It takes work and time to recognize those subtleties. Too many students are unwilling to invest the effort to identify in advance potential problems with this data type. This effort is enormous relative to the effort of figuring out how to code the equivalent of a REAL DO index by using DO WHILE(...) or DO ... IF (.NOT. ...) EXIT ... . They have already learned how to implement alternative data flow structures in their previous work.

10. Access to Features of the Environment

These capabilities are almost all system dependent. There may come a time when Mr. Gates makes system dependency unimportant, but that time has not yet
arrived. There may be a few features that are so universal that Fortran could justify including them, but they are at best a small subset of one or two of these proposals.

11.   New / better Intrinsic functions

I have a list of functions that would be usefull to include in F2000, but none of these proposals intersect that list. I might suggest some of the core procedures of the Numerical Recipes F90 text, and many of the HPF procedures but not these.

13.1 new   Creation of a subset language   large   L

Too controversial and too large. I might like to revisit it when F2000 is almost completed, but there are far higher priorities at the moment.

13.2 new   Review implementation-defined / undefined   medium   O
        constructs

My idea was too undefined. Some of my highest priority items, e.g., internal procedures as arguments are already on the list.

Hope this is helpful

Bill Clodius

=========================
3. Comments by Chr. Weber
=========================

4.1 Bit Data Type: I am strictly opposed against a completely new data type BIT, but some SELECT_LOGICAL_KIND function together with some concatenation operation for arrays and some conversion intrinsics would be a good idea.

4.4 Specifying Default Precisions: I think that the needs behind this requirement can adequately handled by some compiler option where the user is responsible himself that the option is used in all compilation units of a full program (else there will be linkage problems)

5.1 Exception Handling: I believe that the lack of any exception handling facility which allows to control a whole code area together with all subroutine calls is the most important missing feature of Fortran, reducing considerably its competitiveness against other languages such as C or C++.

5.4 Parallelized SELECT CASE:
I think that parallelization issues should not be treated one at a time, but there should be one full concept (message passing? Data

parallelization as in HPF?) before individual steps are done.

6.2 Allow internal procedures as actual arguments:
I think that this feature will cause unduly much implementation complications.

6.5 Initialization programs in modules:
This feature would really add a new quality to the Fortran module concept: implementors of a module would not have to take care any more at each module function that the module might still have to be initialized: statements like
IF (INIT .EQ. .FALSE.) CALL MODULE_INIT

would not be needed any more, gaining quite a bit of performance for each module function.

8.1 Variable Repeat Specifiers in Formats:
I would be in favor of a simple conversion function TO_CHAR(int), but of nothing more.

9.2 Non-advancing I/0 combined with free format:
This extremely strange irregularity (rather a mistake) should be removed.

9.9 New keywords DEFAULT_READ and DEFAULT_WRITE in INQUIRE statement:
This could create an incompatibilty on certain processors since the default units need not necessarily be assigned to a number as well.

10.2 POSIX Binding to Fortran 90:
Already covered by the C interoperability.

11.4 Intrinsic 'size' function for derived types:
It should be decided by the C interoperability TR if this function is needed; I would not like it separately from this TR.

Merry Christmas and a Happy New Year to everybody

Regards

Christian

==

Christian Weber
Siemens Nixdorf
BS2000 SA
Otto-Hahn-Ring 6
81739 Munich / Germany

e-mail: christian.weber@s31.mch1.x400scn.sni.de
Tel.:   +49 89 636 49240
Fax: +49 89 636 41776
===

============================
4. Comments by Malcolm Cohen
============================

>2. General Syntax Enhancements
>2.1 Block Comments (Repository-entry 7)

Omit.
 - The proposal in the repository introduces a gratuitous incompatibility with
   Fortran 90/95.
 - The question of whether such block comments are nested is problematic; the
   proposal in the repository assumes not, which makes one of its proposed
   uses (commenting-out sections of code) highly error-prone.
 - Putting "!" at the beginning of each line in a section of code is very easy
   with most editors.

>2.2 Lower Case Syntax Elements (67)

J3/M7: Low priority.

There does not seem to be any problem with this (I am unaware of any recent
machine which does not provide lowercase) but nor does it seem to gain (I am
unaware of any Fortran 90 implementations which reject lowercase).

>2.3 Reserved words (92)

Omit.
 - this is a gratuitous incompatibility with Fortran 90/95.
 - I do not see this providing any use for programmers, it just decreases the
   size of the namespace.
 - I do not see how this helps the implementor

>3. Constants and Expressions
>3.1 Extend Initialization of COMPLEX Variables (66)

J3/M17: Low priority(allow CMPLX)/Omit(change complex literal constant defn).
 - this would be more useful if it were to extend initialisation by expanding
   the set of intrinsics like CMPLX that are allowed in initialisation
   expressions.
 - the current prohibition of CMPLX(a,b) in an initialisation expression does
   not lose any functionality: the user can write a+b*(0,1): it is just a

niggling inconvenience.
- currently complex literal constants are indeed literal constants. Allowing PARAMETER names or constant expressions in a "complex literal constant" means that it is not a literal constant any longer, just an implicit reference to the complex constructor function CMPLX!

>3.2 Permit BOZ constants in the TRANSFER function (69)

J3/M8: Low priority.
- These should be allowed everywhere, not just in TRANSFER.

>3.3 Allow MERGE in constant expressions (71)

J3/M9: Medium priority.
- This is genuinely useful and avoids having to do strange things with "DIM".
- Must be allowed in initialisation expressions as well...

>3.4 New Special Character designations (77)

Omit.
- these control characters are not necessarily present on all systems
- there is potential for confusion with \n and Fortran end-of-record markers;
  if \n is defined to be a "proper character", we would then require it to be allowed within a Fortran record - ruining those systems which use \n as an end-of-record marker
  or if \n is defined to create a new record (in sequential output only?), so is forbidden to appear inside a record, we are imposing additional overhead onto non-Unix systems to make them look like Unix.

This proposal would fit better as part of some sort of Posix facility.

>4. Data Types
>4.1 Bit Data Type, String (21)

Medium priority.
- we should do something about this long-standing requirement.
- bit string data type seems to be what the users want, though many would accept messing about with LOGICAL arrays.

>4.2 Varying length character with declared maximum (34)

Omit.
- parameterised derived types seem to provide the ability to write ones own version of this facility with fair efficiency.

>4.3 Unsigned INTEGER Data Type (37)

Low priority.

- useful for interfacing to C and other system/library functions
- occasionally (rarely?) useful by itself
- seems to be a lot of work to add this, since it affects many parts of the standard (mixed-mode arithmetic makes things complicated). Is it worth it?

>4.4 Specifying Default Precisions (49)

J3/M12: Omit.

- the proposal in the repository allows real and double precision to be the same; this has negative effects not only on storage association but also on mixed-mode arithmetic.
- this seems to be a lot of work for the potential gain

>5. Control Flow Constructs
>5.1 Exception Handling (5,5a,5b,5c)

Omit.
- this provides too little functionality for too great a cost

>5.2 Extend the semantics of the EXIT statement (25)

Omit.
- if EXIT could exit from IF/SELECT, this would change the semantics of existing code (or we have an inconsistency in that to exit from an IF one would have to specify the construct name).

>6. Procedures
>6.1 Nesting of internal procedures (33)

Low priority.

>6.2 Allow internal procedures as actual arguments (42)

High priority.
- allows function arguments to be thread-safe
- removes the current need for global variables or other hacks when passing user functions to routines for integration/differentiation/interpolation etc.

>Since internal procedures have access to the variables in all
>the surrounding scoping units, it has to be defined how this
>access works (and has to be implemented) from an internal
>procedure which is called via an external procedure.

No, the standard does not have to explain how to implement this.

It does have to explain the semantics; fortunately these are obvious for Fortran 95: an internal procedure invoked via the dummy argument mechanism

has access to the local variables of the instantiation of the host procedure
which passed it in the first place. Not much of a problem.

In Fortran 2000, assuming we have procedure pointers/variables, one has the
additional caveat that a procedure pointer/variable eventually associated with
an internal procedure becomes undefined once the host procedure exits.

>(Such problems are present in the PASCAL language currently and
>have been solved there, but it was complicated for users as
>well as implementors).

Such "problems" (or as some would have it, "features") have been present in
computer languages for more than 3 decades.

How can you say it was complicated for the users when it is impossible
otherwise?

Anyway, if one is starting from scratch, implementation is trivial - a dummy
procedure contains the code address and the host frame pointer. As it happens,
this was the case on the old Apollo Domain workstations - which supported
Pascal as well as C and Fortran.

Even when not starting from scratch - i.e. dummy procedures just have a code
address - thunking does it (i.e. you write several instructions onto the
stack which load the host frame pointer with the right value and then jump to
the "real" address of the internal procedure, and pass the address of this
code block as the dummy argument). With some CPUs you need to flush the
instruction cache as well, so it is not as fast as passing an external
procedure, but it does the job without slowing down existing mechanisms, plus
it retains backwards binary compatibility.

>Amount of work for X3J3:
>large.

It does not look so large to me. Medium at the most.

>6.3 New INTENT attribute: COPY_IN (100)

Omit.
 - the proposer claims we need this because a dummy argument might be
modified
   other than via the dummy argument while one is within a call to that
   routine. Since this is illegal, that piece of rationale is incorrect.
 - It is already possible for the calling routine to make a copy of the
   value of the dummy argument. For arrays, this is probably no less
   efficient that having the compiler do it.

 - INTENT(IN) already provides the guarantee that the dummy argument is not

altered through the dummy argument name, and the standard already
prohibits
changes to actual arguments associated with a dummy argument other than
through the dummy argument name, so this facility is already present.
- if the user wants his compiler to *detect* illegal Fortran he should ask
his vendor (preferably with money).

>7. Executable Statements
>7.1 Controlling Pointer Bounds (2)

J3/M4. Low Priority.
- I think this is almost done anyway.

>7.2 Selecting subarrays of non-rectangular form (26)

Omit.
- unless it is limited to the case where the elements of each dimension of the
target happen to be evenly spaced (by array element number), this would
likely require all implementations to change how array pointers are
implemented.

>8. FORMAT processing
>8.1 Variable Repeat Specifiers in Formats (48)

Omit.
- Using internal writes to create a format-spec provides all this and more.

>9. I/O enhancements
>9.1 STREAM I/O, Binary stream I/O (63, 63a)

Low/medium priority.
(Cannot make up my mind; there are 2 different proposals here anyway)
- "FILESIZE" seems unnecessary; if the user wants to count he can.
- "CURRPOS" is confusing; if the user cannot BACKSPACE, how can the
position
be other than at the end of the file? If the intent is to count the number
of units read/written, let the user do it themselves. Don't require it for
everyone.

>9.2 Non-advancing I/0 combined with free format, Extend non-
>advancing I/O to List-Directed I/O (63b, 65)

Medium priority.
- this is a needless restriction
- why is recursive i/o not a requirement? Enough users want it.

>9.3 Any Kind of Integer for I/O Specifiers (68)

Low priority.
 - I do not see anywhere this is useful other than in REC=.
   Just doing it for REC= would be better than doing it everywhere.

>9.4 Named Scratch Files (73)

J3/M10. Low priority.

>9.5 Default I/O mode (76)

Low/Medium priority.
 - Seems easy to implement both in the standard and in the compilers

>9.6 Recognition of TAB characters in Fortran input (79)

Low priority (one TAB=one blank)/Omit (one TAB = variable blanks).
 - the 'tr' filter on Unix (which is where the C program is likely running)
   can be used to change tabs to blanks.

```
>Treat TAB characters like the corresponding string of blanks in
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```
Unfortunately this differs according to the output device on which one is printing the file, and with many devices it depends on where the tab stops have been set.

The best solution is to use 'tr' if you want one tab = one blank, use 'expand' if you want one tab = 1-7 blanks to next tab stop (every 8th char), or change the C program to output blanks!

>(formatted) Fortran input.

>9.7 New keywords READ_EOR, READ_EOF, WRITE_EOR (?), WRITE_EOF
>(?) in INQUIRE statements (93)

Medium priority.
 - Actually it would be better just to standardise the values returned in
   IOSTAT for END and EOR.

>9.8 New keywords IS_EOR and IS_EOF in INQUIRE, READ and WRITE
>statements (94)

Omit/low priority.
 - It would be better to do item 93 instead.

>9.9 New keywords DEFAULT_READ and DEFAULT_WRITE in INQUIRE
>statement (95)

Low priority.

- it would not seem impossible for those implementations which provide "unnumbered" default i/o units to add a couple of new "magic unit numbers" to help provide this facility. Or if that is deemed too much of a burden, the enquiries could return -1 to indicate the lack of such unit numbers.

>10. Access to Features of the Environment

All of the 10.x features could probably be done nicely as an intrinsic module (so we only hit the namespace once, even if we expand the number of facilities later).

>10.1 Command Line Arguments and Environmental Variables (20)

Low priority.
- genuinely useful but very system-specific
- perhaps environment variables should be part of some Posix package.

>10.2 POSIX Binding to Fortran 90 (47)

Medium priority.
- it would be good to come up with a minimal set of Posix facilities
- but full POSIX.1 binding is too much work for F2000

>10.3 Operation System Support (86)

Medium priority.
- most implementations already provide this in some form, there seems to be a fair amount of user demand for this (as one of the vendors who does NOT provide a "system" function, I can testify to the user demand).

>The requirement may be covered by 10.2 (see above).

No it is not, though in my view it could be packaged together with anything that comes out of 10.2 in an intrinsic module.

>10.4 Handling of error messages (99)

Low priority.
- these messages usually have some context (sometimes embedded in the message)
  such as filename, unit number, system error code, etc.
- POSIX handles this by standardising the error codes (actually the names of the error codes, not the values). We could similarly standardise some set of error code names (perhaps as PARAMETERs in an intrinsic module...)

>10.5 Primitive graphic facilities in Fortran (102)

Omit.

- Too system-specific. Too primitive. Too expensive.
- This is better done by an auxiliary standard, or as a binding to an
  existing standard (even an existing de facto standard).

>11. New / better Intrinsic functions
>11.1 Regularize RANDOM_SEED functionality (55)
>
>Subject:
>Provide some means to reset the random number generator in a
>non-repeatable way (and furthermore create a more comfortable
>interface for the repeatable reset).

Low priority for first:
- programs which require multiple independent pseudo-random number
streams
  should not use RANDOM_NUMBER. Commercial and public-domain
random-number
  generators are available for this purpose. (Note that this proposal does
  not provide acceptable multiple random number streams, it just gives it
  the appearance thereof).

Omit for second:
- Programs which require multiple independent random-number streams
should not
  be encouraged to use RANDOM_NUMBER.

It is my view that provision of more sophisticated random-number generation
facilities in the Fortran language is inappropriate. Unless the language also
standardises the RNG involved, it will mislead users with special needs into
thinking they can trust the implementation to provide streams with special
characteristics.

>11.2 Generic COUNT_RATE Argument for SYSTEM_CLOCK (61)

J3/M3. Low priority.
- Seems to be underway already, though I see no need for it.

>11.3 Extend MAX, MIN, etc. to CHARACTER Data Type (64)

J3/M5. Low priority.
- Seems to be underway already. Though it has limited usefulness it does

  give the language the appearance of more regularity, I suppose.

>11.4 Intrinsic 'size' function for derived types (80)

Omit (as far as the "misc" group is concerned).
- this belongs in the interoperability TR, if anywhere.

>11.5 Instrinsic 'sort' for arrays of intrinsic type (81)

Omit.
 - it is unrealistic to expect compiler writers to provide dozens of highly
   optimised sorting routines and to pick the correct one which matches the
   characteristics of the users data (indeed, often the program contains no
   information about key characteristics of the data, like whether it will
   fit in memory!).
 - Commercial and public domain sorting routines for particular kinds of data
   are already available. By chosing among them, the user will almost
   certainly do better than the implementor will by guessing.
 - too expensive for too little gain

>11.6 Intrinsic function 'fft' - Fast Fourier Transformation (82)

Omit.
 - too expensive for too little gain
 - this kind of function belongs in specialised libraries, not in the language

>11.7 Four new elemental intrinsic functions: TRUNCATE, ROUND,
>IBCHNG,ICOUNT (90)

Omit/Low Priority.
 - TRUNCATE,ROUND: Omit. Expensive to compute and encourage the
mistaken
     view of floating-point numbers as decimal floats instead of binary floats.
 - IBCHNG: Omit. Functionality already provided by Fortran 90:
    IBCHG(I,POS) = IEOR(I,IBSET(1,POS))
 - ICOUNT: Low Priority.
     We should probably call this "POPCNT" (it is in the HPF_LIBRARY module
of
     HPF; using the same name avoids confusion and does not cause a name clash
     with the HPF_LIBRARY since it does the same thing).

>11.8 PATTERN= in bit manipulation functions such as IBCLR,
>IBSET, IBCHNG (91)

Low priority.
 - this is unlikely to be more efficient as an intrinsic instead of the obvious
   DO loop, e.g.

```
     IBCLR(IVAL,PATTERN=A)
   vs.
    DO I=1,SIZE(A) ; IVAL = IBCLR(IVAL,A(I)) ; ENDDO
```

 - there is an improvement in clarity of the code, but I am surprised that
   people wanting this do not instead want bitwise reduction functions like
   HPF's IALL/IANY/IPARITY (array reduction of IAND/IOR/IEOR respectively).
   Since IBCLR(IVAL,PATTERN=A) = IALL(IBCLR(IVAL,A)), I would suggest that
   adding these reduction functions would satisfy the requirement.

>11.9 New transformational functions: POSITION and LOCATION (97)

Omit.
 - these seem very specialised. I cannot imagine many people using them.

>11.10 New functions to handle arrays: SCRIPT and SCALAR (98)

Omit.
 - SCRIPT functionality already provided (it is just syntactic sugar)
 - SCALAR encourages the view of multi-dimensional arrays as single-dim
arrays.
   This seems likely to cause slowdowns in code which might later be
   parallelised.

>12. Relaxation of Restrictions
>12.1 Remove the restriction on the maximum rank of arrays,
>Greater than 7 Array Dimensions (24, 24a)

J3/M14: Low Priority.
 - it is probably about time to increase the limit past 7, due to increasing
   memory sizes.
 - as long as it is not increased too much!

>12.2 Remove limitation on statement length (50)

Omit/Low priority.
 - the statement length limit is useful for implementors and users alike.
   It can be increased but should not be removed.

****
*

Comments on Additional Suggestions:

0.
 - These have not been processed by the US member body so have no official
   standing; in particular there is no agreement that there is real user
   demand for these features
 - It behooves us to consider that the features brought forward by the US and

   other member bodies at the San Diego and Dresden meetings are of higher
   priority than these.
 - We already have more proposals to consider, and more of them have good
   ideas, than we can possibly implement in F2000. To accept one of the
   unofficial proposals we must be clear that not only is it worthwhile, but
   that it is noticeably more worthwhile than the official proposal whose
   rejection it causes (due to time constraints).

1. Allow interfaces to internal and module procedures

Omit.
 - the "well known compiler writer" (whoever it is) complaining on
   comp.lang.fortran is complaining about a problem already solved by
   all Fortran 90 vendors.

2. Review implementation defined and undefined constructs

Since this is not a concrete proposal there is nothing to agree/disagree to.

3. Include a parallelised SELECT CASE

Omit.
 - functionality already present in Fortran 95 by using an ELEMENTAL procedure
   containing the SELECT CASE.

4. Allow programs in modules

Hmm, "Allow an automatic initialisation subprogram for a module" is what I think is being suggested here. Something like

***Begin specification

```
MODULE module
  INITIALISED BY :: init
  ...
CONTAINS
  SUBROUTINE init ! Must be a subroutine with no arguments
    ...
  END SUBROUTINE
END
```

And "init" is automatically called at the beginning of the execution of the first subprogram which USEs (directly or indirectly) the module. (Note that BLOCK DATA USEs will not count.)

***End specification

Omit/Low priority
 - this is just the same as adding "LOGICAL :: module_inited = .FALSE." to the
   module, and putting "IF (.NOT.module_inited) CALL module_init" in every
   subprogram that USEs the module.

5. Floating-point CASE/FORALL/WHERE

Omit.
 - This proposal encourages bad code without making it any easier to write

good code.

6. Conditional (i.e. non-commutative) AND/OR

Omit/Low priority.
 - the proposer admits that these are merely syntactic sugar for nested IF.
 - I do not accept the assertion that most experienced Fortran programmers
   think that .AND. et al are "short-circuit". (Anyway, "real" Fortran
   programmers run without subscript checks!).

7. Develop a subset

Omit.
 - No, No, a thousand times No.
 - I have never heard a user complain that we provided too many features.
 - "a representative of Meisner"??????
 - Still no.
 - Have we not yet learned after all this time that subsets are disasters?

--
...........................Malcolm Cohen, NAG Ltd., Oxford, U.K.
(malcolm@nag.co.uk)

==========================
5. Comments by S. Whitlock
==========================

o I strongly oppose opening up the repository for new items. I believe this exercise was to review the existing items in the WG5 repository and decide if any should be included. My votes to "Omit" reflect that.

o In table 1, I marked but did not rank the "minor technical enhancements" that J3 has already accepted and is working on as well as the firm requirments that WG5 has accepted and we're working on. Those items

cannot be removed from the F2000 requirements by this vote and I would strongly oppose such. If WG5 wants to tell J3 to not work on a minor technical enhancement, there should be a specific vote.

o I believe that J3 already has too much work to do and want to urge both WG5 and J3 to be conservative about F2000 requirements. We've had plenty of chances and plenty of votes and yet, we keep voting on the same items repeatedly. This wastes time and energy. Whatever the Feb-1997 meeting decides, I am concerned that it will be "too much, too late", as F90 was.

/Stan

==========================
6. Comments by M. Hennecke
==========================

A vote of "-" means no opinion (not read / understood / both).

```
> 3.2   69    Permit BOZ constants in the TRANSFER       small   L
>            function
```

Although I would vote NO on the heading of this req., some way
to specify binary / octal / hex values outside of DATA is necessary.
Imagine setting UNIX mode bits :-)

```
> 4.1   21    Bit Data Type, String                  large   M
```

```
> 6.3   100   New INTENT attribute: COPY_IN            medium  M
```

Although COPY_IN is not important to me, the NO_COPY or VOLATILE
variants ARE and should be dealt with in F2000. I don't like any
of the current spect, though.

```
> 6.4   new   Interfaces to internal & module procedures small   M
```

This is a vital cross-checking option between interface blocks
and procedure definitions, which e.g. C allows.

```
> 10.2  47    POSIX Binding to Fortran 90              large   O
```

This is a BIG resource problem.
Also addressed by C Interoperability.

```
> 10.5  102   Primitive graphic facilities in Fortran    large   O
```

This is a BIG resource problem.
Partly addressed by C Interoperability.

```
> 11.1  55    Regularize RANDOM_SEED functionality       small   H
```

I have commented more than once. The reset option of
RANDOM_SEED is of no use as it stands (I do not speak of
quality, this is just a user interface specification).

```
> 11.4  80    Intrinsic 'size' function for derived      small   L
>          types
```

NO to extending SIZE (what would happen for arrays of derived type?). YES for a new function, e.g. BYTESIZE, also for intrinsic types. Might be used for RECL specifications...

Christian, you've done a great job.

Thanks,
Michael

```
=================================================================
======
  Michael Hennecke      http://www.uni-karlsruhe.de/~Michael.Hennecke/
 -----------------------------------------------------------------------
  University of Karlsruhe         RFC822: hennecke@rz.uni-karlsruhe.de
  Computing Center (G20.21 R210)             No longer on BITNET :-(
  Zirkel 2  *  P.O. Box 69 80              Phone: +49 721  608-4862
  D-76128  Karlsruhe                         Fax: +49 721  32550

=================================================================
======
```

=======================

7. Comments by M. Zuern
=======================

4.1   Bit Data Type, String
It is not quiet clear to me how this requirement will effect performance.

4.3   Unsigned INTEGER Data Type
As interoperability is one of the most important things for future Fortran standards, I believe this requirement is quiet important.

6.3   New INTENT attribute: COPY_IN
Also very important is a possibility to specify NO_COPY_IN for array arguments. At the moment with developing subroutines taking arrays as arguments it must always be assumed that array sections might be passed and therefore a copy of the argument is required. This e.g. is one of the major problems for the

Fortran 90 interface for MPI !
MPI works with references of objects and therefore provides
a procedure to determine the address of an object. If an array
'a' is passed to a subroutine 'sub1' and a copy of the argument
is made, another process which wants to work on 'a' via its
reference will not get the original reference of 'a' but of its
copy within the subroutine 'sub1' if it is called in some
nested context within 'sub1'.

6.5 Initialization programs in modules
This one sounds very much like one of the constructor functionalities
to me. This is going to be discussed in /data in context with
destructors.

7.2 Selecting subarrays of non-rectangular form
As F90 array sections already are already quiet performance
decreasing I wonder where this feature would lead us to if
implemented ...

9.2 Non-advancing I/0 combined with free format
This requirement allows combination between non advancing I/O
and free I/O format to be able to read ascii files in free
I/O format. Output in software systems written to files is often
orgaized as a number of data, one or multiple per line. The exact
format cannot be known in advance, neither the data per line.
Files produced in the above way (e.g. by C programs) very often
have to be read by other programs.

9.6 Recognition of TAB characters in Fortran input

As interoperability is one of the most important things
for future Fortran standards, I believe this requirement
is quiet important.
TAB characters turn up very often in C output and UNIX
tools like make, sort, paste. Not recognizing TABs means
not being able to use those UNIX tools and output produced
by C programs.

--
==

Manuela Zuern e-mail: zuern@rus.uni-stuttgart.de

Numerical Methods for Supercomputers
Computing Center, University of Stuttgart
Allmandring 30 Phone ++49-711-685-2511
70550 Stuttgart FAX ++49-711-678-7626

Germany

==

========================
8. Comments by D. Schmitt
========================

These are my totally PERSONAL votes.

I have put a '?' in the first list whenever I thought that I had too few information.

I'd like to make a short explanation about my understanding of low/med/hi priority:

L: Not really necessary fo F2k, but if you have some spare time you are free to do it. Also champions could 'take care' of those reqs. since most of them are 'small'. The 'large' ones I think are not that important to be handled immediately.

M: Could be in F2k.

H: Should definetly be in F2k because I think that they have a strong impact on userfriendlyness while staying almost totally in the 'small'.

To the different proposals:

```
4.2 (34) Varying length character with declared maximum (large) L
5.1 (5,5a,5b,5c) Exception Handling (large)              L
10.3 (86) Operation System Support (small)               L/M
10.4 (99) Handling of error messages (medium)            L
```

Proposals like these would be nice but I don't think they are worth the effort to press them into the F2k standard.

```
5.4 (new) Parallelized SELECT CASE (medium)              L/M
```

Would be a good addition for multiprocessor systems (?) but is not necessary on single-proc-machines ... I don't know if there are so many multi-proc-comps out there.

```
10.5  102   Primitive graphic facilities in Fortran    large   L
```

Definetly YES somehow in the future of Fortran and also definetly NOT in F2k because there is so much that could go wrong.

12.1 ... Remove '7-dimension-restriction' (small) O/L

If you allocate 4GB for a single 7-dimensional array each dimension only may have 23 elements ... on the other hand I also could imagine very 'thin' arrays ...

Cheeers,

David Schmitt, AUSTRIA

PS: Feel free to redistribute any of my comments to the group.
    Feel also free to point out any logical errors or other
    'stupidities' I made.

THX DS

```
------------------------------------------------------------------------------
eMail: david@edvzbb2.ben-fh.tuwien.ac.at  <--- TRY THIS ONE FIRST!!
       dschmitt@verman.zserv.tuwien.ac.at
sMail: David Schmitt     <-\_____ This is purely coincidental,
       Schmidgasse 4/9  <-/       isn't it ??
       1080 Wien
       AUSTRIA
WWW  : http://verman.zserv.tuwien.ac.at/~dschmitt/
------------------------------------------------------------------------------
```

=========================
9. Comments by P. Lignelet
=========================

3.3 (MERGE) THE reason why this is up to now prohibited is that it ALWAYS include a LOGICAL arg, which is not CHAR or INT required for init. expr....

4.1 For new types, Isuggest a MODULE is the best way to add it to the language. It could even be then optional (good for compiler writers...)

4.2 : idem

6.2 I insist upon its being non recursive in a non recursive host, at least for F2000, te render its implementation easier (and be sure to have this feature included...)

10.2 : through a MODULE... (optional...)

10.3: USE the preceeding MODULE...

Various addings / limitations relaxations: WHERE TO STOP?

Yours sincerely,

```
 ---------------------------------------------------------
 *   Patrice LIGNELET           tel : 01 40 27 22 58   *
 *   lignelet@vcnam.cnam.fr                   23 83   *
 *                                           *
 *   Conservatoire National des Arts et Metiers         *
 *   Departement d'informatique   fax : 01 40 27 23 77   *
 *   292, rue Saint Martin                          (\
 *   75141 PARIS CEDEX 03                            ( \
 ============================================================) ) />
                                      / )  / / /))/
                                      \ \_/ / / / /
                                       \      /
                                        \_    /
                                         |   |
                                         |   |
                                         -----
                                         -----
```

========================
10. Comments by S. Morgan
========================

4.1 BIT Data Type

I think this is an important issue for F2000 but would like to see it

solved by a module interface approach. As someone pointed out on comp-fortran-90, the exercise of defining such an interface and implementing a sample code would be a good test of the (hopefully enhanced) data abstraction features of the language. Hence the H for this.

10.5 Primitive Graphics Facilities

I still haven't seen a convincing proposal for this. It is a very large and controversial issue and could take up a lot of valuable committee time with an almost certain negative outcome. So I've given it an O.

9.1 STREAM I/O, Binary Stream I/O

I have seen many local users, who want to process image data and data from various experimental kits, struggle in Fortran and then move to C to do this sort of thing. Definitely 3 points

9.10 Asynch I/O

Obviously important to HPC since HPF is 'doing it'.
--

Dr. J. S. Morgan   Tel. 0151-794-3746 (3719 for messages)
Computing Services Department   Fax. 0151-794-3759
The University of Liverpool   email. J.S.Morgan @ liverpool.ac.uk
Chadwick Building
Peach Street
Liverpool L69 7ZF

=========================

11. Comments by John Reid

=========================

1. I have been pretty "hard-nosed" because I think the language must not be allowed to grow much bigger for several reasons:
   a. It is very hard to construct reliable software in a language that progammers do not fully understand.
   b. We do not have the resources to make a big revision to the standard itself.
   c. Vendors must have some time and energy left for optimization.

2. I have given all my points to item 42 because this is the only one that I really care about. It provides a beautiful solution to a long-standing problem faced by those of us constructing numerical libraries and all the users of those libraries. The library code needs to call user's code that defines the problem (e.g. a

   function to be minimized). If the user can provide an internal procedure for this purpose, that procedure (when called by the library code) has access to the whole data environment of the code at the point of call of the library code. See the repository for further explanation.

   I emphasize that it is not just for the convenience of library writers that this is needed. It is to allow us to write code that is easy for our users to call.

   There some disagreement over how hard this is for vendors when the host is recursive. Please could we at least have it for non-recursive hosts?

   I disagree about it being much work for X3J3. We just have to remove a restriction from the standard.

===============================

12. Comments by David Muxworthy

================================

Congratulations on doing all that work! I am sorry I didn't submit a
vote in time. I would have voted in a very similar way to John Reid,
that is with a high number of 'O' votes.

I would certainly have put 'O' on bit data type, given the vast amount
of time X3J3 spent going round in circles on this topic during the
1980s, and on unsigned integers. I would also have voted 'O' on any
item where the functionality is already present in the language, like
variable repeat specifiers in format statements.

Still, we can discuss this further at WG5. Your group has made very
valuable input.
Happy new year,
David

==
13. Comments by the French member body
==

2. General Syntax Enhancements
2.1 Block Comments (Repository-entry 7)
Note:
This requirement might already been satisfied by the "condi

tional compilation facility" (which allows to blank out large
blocks of text - e.g. comments - from the compilation process).   -- YES.
answered to by COCO.

2.2 Lower Case Syntax Elements (67)   -- OF COURSE!

2.3 Reserved words (92)

Subject:
The use of reserved words should be forbidden for other than   -- OF
COURSE...
their dedicated purposes (e.g. as variable names).

3. Constants and Expressions

3.2 Permit BOZ constants in the TRANSFER function (69)
Problems:
I doubt that the use of BOZ constants can reasonably be limited
to a particular function (as e.g. TRANSFER); it might be
extended in general, though.

-- NO. USE VARIABLES.

3.3 Allow MERGE in constant expressions (71)   -- YES! 1 point...

I am not fully sure what exactly is missing in Fortran95 (MERGE
appears to be elemental and therefore is permitted already in
constant expressions - or not?).
-- IT CAN BE OF ANY type (not only Int or Char)...

3.4 New Special Character designations (77)
--NO. Not upward compatible wiith Fortran 90/95.

4. Data Types
4.1 Bit Data Type, String (21)
Problems:
There are doubts within the DIN working group that the BIT type
is really needed (alternatives: better bit intrinsics which
operate on INTEGER arrays, or some LOGICAL kind which assigns
one bit of storage in combination with some general "stringing"
facility).
-- WE ARE IN FAVOUR OF A MODULE for this new type, if introduced.

4.2 Varying length character with declared maximum (34)
Problems:
This data type is a kind of competitor (within Fortran) to the
string module.

The fixed maximum length (necessary for a very efficient
implementation) limits its usefulness.
-- THE FRENCH GROUP is giving 1 point for it.

4.3 Unsigned INTEGER Data Type (37) --NO.
Problems:
If C-Interoperability is the main issue, then simpler means
(MAP_TO approach, some conversion intrinsic) may solve the
problems. -- GOOD REMARK.

4.4 Specifying Default Precisions (49)

-- There is another solution: promoting automatically the constsant
-- according to the KIND of the receiver.

5. Control Flow Constructs
5.1 Exception Handling (5,5a,5b,5c)   -- 3 points...

5.2 Extend the semantics of the EXIT statement (25)
Subject:

Enhance EXIT such that any kind of executable block construct
can be left.
-- WE ARE VERY RESERVED ON THIS SUBJECT (rather against it).

6. Procedures
6.1 Nesting of internal procedures (33) -- YES.

6.2 Allow internal procedures as actual arguments (42)

-- YES (3 points).

-- For the first time, let's forbid any kind of recursivity
-- (for the host, as for the argument procedure).

6.3 New INTENT attribute: COPY_IN (100) -- OF COURSE...

7. Executable Statements
7.1 Controlling Pointer Bounds (2) -- (no idea on this...)

7.2 Selecting subarrays of non-rectangular form (26)
Subject:
Allow the use of implied-do control on the right hand side of a
pointer assignment to an array pointer.
-- SEEMS SOMEHOW COMPLICATED, for a doubtful result...

8. FORMAT processing
8.1 Variable Repeat Specifiers in Formats (48)
-- WHY NOT? (already exists in DEC VMS F77 compiler).

FMT='10X,'//TO_CHAR(M)//'(2X,''NO'',2X)' -- OH NO!...

9. I/O enhancements
9.1 STREAM I/O, Binary stream I/O (63, 63a)
-- IT DO SEEM USEFUL. Cf BUFFER IN/BUFFER OUT in french Fortran-IV C.I.I.

9.2 Non-advancing I/0 combined with free format, Extend non- -- YES.
advancing I/O to List-Directed I/O (63b, 65)

9.3 Any Kind of Integer for I/O Specifiers (68) -- NO!
-- PRAY RECALL that "default" int are stored in
-- one numerical word (as reals)...
-- We don't know of real being stored in 2 bytes...

9.4 Named Scratch Files (73)   -- NO.

9.5 Default I/O mode (76)
Subject:
Allow the default I/O mode for READ/WRITE statements to be
ADVANCE="NO" (depending on some new OPEN parameter).
                                                            -- YES.

9.6 Recognition of TAB characters in Fortran input (79)

-- The french group is against (I am personnally in favour...)

9.7 New keywords READ_EOR, READ_EOF, WRITE_EOR (?), WRITE_EOF   --
YES.
(?) in INQUIRE statements (93)

9.8 New keywords IS_EOR and IS_EOF in INQUIRE, READ and WRITE   --
YES.
statements (94)

9.9 New keywords DEFAULT_READ and DEFAULT_WRITE in INQUIRE   --
YES.
-- + OPEN ! -- statement (95)

10. Access to Features of the Environment
10.1 Command Line Arguments and Environmental Variables (20)   -- YES.
Amount of work for X3J3:
small.   -- IS IT SURE?... (strings are declared bounded
                          -- in Fortran; they are not "pointers to
                          -- array of char"...

10.2 POSIX Binding to Fortran 90 (47)
Problems:
The requirement might be covered by the standard C-interface of
POSIX in combination with the C interoperability.
-- IT IS WHERE A MODULE MUST be used!
-- (or, yes, through C...)

10.3 Operation System Support (86)
-- COULDN"T THERE BE a FORTRAN_SYSTEM Module,
-- containing a SYSTEM subroutine?

.4 Handling of error messages (99)   -- YES.

10.5 Primitive graphic facilities in Fortran (102)
-- WE ARE NOT in favour of introducing that in the
-- language, as the effect shall depend on the
-- technical environment.
-- There are standardised PHIGS interfaces to Fortran...

11. New / better Intrinsic functions
11.1 Regularize RANDOM_SEED functionality (55)
-- TO FURTHER STUDY...

11.2 Generic COUNT_RATE Argument for SYSTEM_CLOCK (61)
-- WHY NOT?

11.3 Extend MAX, MIN, etc. to CHARACTER Data Type (64)
-- NO!

11.4 Intrinsic 'size' function for derived types (80)
-- PERHAPS "TYPE_SIZE", or "BIT_SIZE"?

11.5 Instrinsic 'sort' for arrays of intrinsic type (81)
-- NO, as there are no "general" sort subroutine;
-- the "best" one depends on the pre-ordering of data
-- to sort.

11.6 Intrinsic function 'fft' - Fast Fourier Transformation
-- NO! Where to stop?...

11.7 Four new elemental intrinsic functions: TRUNCATE, ROUND,
IBCHNG,ICOUNT (90)   -- YES...

11.8 PATTERN= in bit manipulation functions such as IBCLR,
IBSET, IBCHNG (91)   -- YES...

11.9 New transformational functions: POSITION and LOCATION (97)
-- YES...

11.10 New functions to handle arrays: SCRIPT and SCALAR (98)
-- YES...

12. Relaxation of Restrictions
12.1 Remove the restriction on the maximum rank of arrays,
Greater than 7 Array Dimensions (24, 24a)
-- NO! (Where to stop?...)

12.2 Remove limitation on statement length (50) -- NO!

Sincerely yours,

```
 ---------------------------------------------------------
 *   Patrice LIGNELET           tel : 01 40 27 22 58   *
 *   lignelet@vcnam.cnam.fr                 23 83   *
 *                                           *
 *   Conservatoire National des Arts et Metiers          *
 *   Departement d'informatique   fax : 01 40 27 23 77   *
 *   292, rue Saint Martin                        (\
 *   75141 PARIS CEDEX 03                          ( \
 ============================================================) ) />
```