Parameter transmission to non-Simula procedures
==========

The S-port FEC will now recognize the following identifiers as KIND
in an external procedure declaration:

LIBRARY - will follow the S-port conventions, except no context
parameter is generated

C - will follow the SDG proposed C-interface with extensions.

Pascal - will follow the SDG proposed C-interface with extensions.

When compiling an actual call on a C-procedure, FEC will reverse the
parameter list. Note that this means that, within such a C-call, the
parameters are evaluated FROM RIGHT TO LEFT. Apart from that, no
distinction is made between C and Pascal procedures.

1. Remarks on the SDG-proposal.

The "address of the first attribute" of the SDG-proposal is taken to
mean "the address of the first attribute of the formal qualifying
class, if such an attribute exists".

Probably, a reference qualified by a class with hidden protected
attributes should not constitute a legal parameter by name. It could
ve argued that by value transmission is allowed as long as only visible
attributes are copied. In the interest of efficiency (both from the
compiler viewpoint, and at run time), this implementation do not
bother about such niceties. If anything is to be worked into the
proposal, I suggest that such a parameter qualification is illegal.

A TEXT array transmitted by value is copied "in toto", i.e. the
elements of the copy will be pointers to copies of the referred texts.
No such copying is attempted for REF arrays, i.e. the copy elements
point back to (the first attribute of) SIMULA objects (in SIMULA space).

Default mode is not allowed for C-procedures; FEC will generate a
mode specification (NAME for arrays and reference types, VALUE for
simple types) and issue a warning.

FEC will allow a SIMULA procedure identification as actual parameter for
a formal procedure parameter, if the procedure is a proper procedure
(i.e. has no type) without parameters. In addition, the identification
of any none-SIMULA procedure (with/without type and/or parameters)
may occur as actual parameter.

2. S-code generated for C-procedure calls.

a) by name:

type on S-stack before ASSPAR

simple GADDR of attribute
ref GADDR of first attribute of formal qualification
GNONE if no such attribute, or actual parameter is NONE
(note: only static none-checking, the value of the
actual parameter is NOT checked)
text GADDR of first character of text parameter
GNONE if the value of the actual parameter is notext

array GADDR of first array element (in allocation order)
       GNONE for dummy arrays
procedure RADDR (body tag). If the actual is a non-SIMULA procedure,
       the tag is the body tag of the external routine. Otherwise,
       a complete profile and routine body is generated, enclosed
       within BSEG-ESEG, and the routine parameter is the body tag
       of this "thunk".

b) by value:

simple Value of parameter
ref OADDR of copy of the attribute part of the referred object.
       ONONE if no attributes, or actual value is NONE.
text OADDR of copy of referred (sub)text value, NUL-terminated.
         (note: NEVER ONONE)
array OADDR of copy of array elements (see below)

The computation of the address for by-name transmission of an array
is performed by the run time system, as are all value copying
of none-simple parameters. REF/TEXT by name computation is performed
inline. The run time support routines are in the (new) RTS module CINT
(for C-interface).
Copying of a TEXT array is recursive over all referenced texts.

The severe restrictions on SIMULA procedures to be used as
"call back" from C (or other languages) are caused by a wish
to make this transmission less of a burden on the compiler. In
addition, it is not clear how a return value should be handled
within the framework of the S-code.
Parameter and value transfer may be done through the use of
variables (objects) non-local to the procedure. Note also that
the procedure may well be an attribute of some class.

THIS FEATURE IS INHERENTLY UNSAFE. Users should be warned about
the use of e.g. the sequencing procedures directly or indirectly
from such call-back procedures, since it is assumed that CURINS
IS UNCHANGED DURING ANY EXECUTION OF NON-SIMULA PROCEDURES.

On the PC-implementation, use of call-back procedures invalidates the
statement break/single stepping - which may generate spurious
interrupts because of the way we handle the hardware stack.

The code generated for a procedure parameter P may be described as

```
BSEG routine Pcall; exit returnAddress;
   begin curins.lsc    := returnAddress;
      returnAddress :=
         A_CPRO( "P's static encloser relative to curins",
               !NOTE: curins at time of call of the C-procedure;
               "P's prototype" );
   end
ESEG
   PUSHV  Pcall.bodytag ASSPAR
```

The RTS routine A_CPRO is included in "cint". It will generate the
activation record for the P incarnation, link it to it's static encloser
and to curins, and return the start address of P. From the viewpoint of
SIMULA (disregarding any side effects from the C-call), the effect will
be as if the entire C-call was replaced by a call on P. It is easy to
see that if P calls the same or another C-routine, which e.g. calls P
back VIA THE PREVIOUSLY TRANSMITTED ADDRESS, anything may happen to the
consistency of the SIMULA system!!!! - and probably to the C system as

well...

3. Extensions of the compiled language.

3.1. Records.

Controlling directive: %RECORD.
(%SPORT does not control this one, since it introduces a new keyword RECORD).

In order to handle structures, a record concept is introduced. It is mainly intended for descriptive purposes, i.e. no such quantity can be materialised within a "pure" Simula program. The record is at the moment so ill defined that I hesitate to give any kind of definition! Here's what will be treated correctly at the moment (bearing in mind that the compiler treatment of this facility is still shaky - we will not publicise it until the whole thing is stabilised):

```
record-declaration
      =  [ record-prefix ]  RECORD  record-identifier ;
         BEGIN  { record-variable-declaration } END

record-variable-declaration
      =  ( value-type | object-reference-type )
         identifier  [ "(" rep-count ")" ]
```

I plan to extend this concept (and to formalise it) to the full capability of S-code, by emulating to a large extent the record definition of Simuletta.

Attributes declared as REF(record-identifier) will get the new type "pointer". Within pure Simula, no records can be generated and such attributes are therefore unassignable (except for ... :- NONE). Non-Simula procedures may, however, be declared as being of this type, and the procedure return value may be assigned to such attributes. (external C ... is ref(some record) procedure malloc(...)...; will work beautifully).

This concept slips thru garbage collection like you would expect, and allows SIMOB (with the proper declarations) to display (change) values of C structures passed back to Simula, and to follow pointers.

3.2 Macros.

Controlling directive: %SPORT

The macro facility has been incorporated in the production compiler. Further details follows!