Subject: Draft edits for Coroutines
From: Van Snyder
Reference: 04-380r2

1 Introduction

Assuming Coroutines get onto the J3 work plan, the reason for this paper is to get a running start on the edits.

2 Edits

Edits refer to 04-007. Page and line numbers are displayed in the margin. Absent other instructions, a page and line number or line number range implies all of the indicated text is to be replaced by associated text, while a page and line number followed by + (-) indicates that associated text is to be inserted after (before) the indicated line. Remarks are noted in the margin, or appear between [and] in the text.

---

or *resume-stmt* 11:28+

---

or *suspend-stmt* 11:31+

[Editor: Replace "When ... point." by the following paragraph:] 15:6-7

If execution of an instance of a procedure defined by a coroutine subprogram is suspended by execution of a SUSPEND statement (12.5.2.5$\frac{1}{2}$), and execution of that instance is subsequently resumed by execution of a RESUME statement (12.4.1), execution continues with the first executable construct following the SUSPEND statement that suspended execution of that instance. Otherwise, when any procedure is invoked, execution begins with the first executable construct appearing after the invoked entry point.

[Editor: Insert ", SUSPEND" after "RETURN".] 15:9

If *expr* is a derived-type object that has procedure instance associations (16.4.3$\frac{1}{2}$), *variable* acquires the same associations. 142:3+ New ¶

If *proc-target* is a procedure pointer that has a procedure instance association (16.4.3$\frac{1}{2}$), *proc-pointer-object* acquires the same association. 142:8+ New ¶

[Editor: Add a comma at the end of each item in the list.] 168:12-19

Execution of a SUSPEND statement (12.5.2.5$\frac{1}{2}$) within the range of the DO construct inactivates the DO construct but does not necessarily terminate the loop. The DO construct can be reactivated by execution of a RESUME statement (12.7.4) that resumes the procedure instance in which the loop is executing. 168:23+

[Editor: Replace "statement or an" by ", SUSPEND, or".] 249:16-17

If the main program is defined by a Fortran main program program unit, an instance of the main program is created when execution of the program begins. 250:1+ New ¶

[Editor: Replace "or subroutine" by ", subroutine, or coroutine".] 256:17

$3\frac{1}{2}$ The procedure is a coroutine, 258:4+

R1218$\frac{1}{2}$ *resume-stmt* **is** RESUME *procedure-designator* 266:19+

C1222$\frac{1}{2}$ (R1218$\frac{1}{2}$) The *procedure-designator* shall designate a coroutine subprogram.

When an instance of a procedure (12.5.2.3) defined by a coroutine subprogram is created by execution of a CALL statement, a procedure instance association (16.4.3$\frac{1}{2}$) is established (16.4.5). 268:0+ New ¶

[Editor: Insert a new subclause and renumber subsequent ones — TEX-o-matic:]

12.4.0$\frac{1}{2}$ RESUME statement

Execution of a RESUME statement resumes execution of an instance of the procedure defined by the coroutine specified by the *procedure-designator*. If the *procedure-designator* is a procedure name that is not a procedure pointer the resumed instance is the one that is procedure instance associated (16.4.3$\frac{1}{2}$) with the currently-executing instance of the scoping unit containing the RESUME statement. If the *procedure-designator* is a procedure pointer the resumed instance is the one that is procedure instance associated with the procedure pointer. If the *procedure-designator* is *data-ref*%*binding-name* the resumed instance is the one that is procedure instance associated with the *data-ref*. The resumed instance shall exist, and the necessary procedure instance association shall not have been terminated (16.4.5).

If we allow internal coroutines to be actual arguments or procedure pointer targets, we will need the following paragraph: If the subprogram designated by the *procedure-designator* is defined within a scoping unit, the same instance of that scoping unit shall exist as existed at the instant the SUSPEND statement was executed.	*J3 remark*

NOTE 12.19$\frac{2}{3}$

Execution of a RESUME statement does not invoke a procedure. Entities that normally become undefined or undergo default initialization when a procedure is invoked by a CALL statement do not become undefined or undergo default initialization when a RESUME statement is executed. Automatic entities are not created anew. Rather, the data entities of the resumed instance, and its argument associations, continue to exist.

[Editor: After "*prefix*" insert "[COROUTINE]".] 282:8

C1247$\frac{1}{2}$ (R1232) If COROUTINE appears, *prefix* shall not include ELEMENTAL. 282:10+

Does PURE make sense for coroutines? See C1268, since the instance hangs around as though every data entity had the SAVE attribute.	*J3 question*

If COROUTINE appears, the subprogram is a **coroutine subprogram** (12.7.4). 282:29+

[Editor: Add a new sentence at the end of the paragraph:] 282:32

The instance is destroyed when execution of the subprogram or statement function is complete.

NOTE 12.39$\frac{1}{2}$

Suspending execution of an instance (12.5.2.5$\frac{1}{2}$) does not destroy it.

[Editor: Replace "and an" by ", an", "and local" by ", an independent set of local" and "objects" by "objects, and an independent set of procedure instance associations (16.4.3$\frac{1}{2}$)".] 282:33-34

12.5.2.5$\frac{1}{2}$ SUSPEND statement 284:34+

Execution of a SUSPEND statement suspends execution of the instance of the subprogram in which it appears. Execution continues at the executable construct following the CALL statement that initiated execution of that instance, or the RESUME statement that resumed execution of that instance, whichever was most recently executed. Execution of the suspended instance may be resumed at the executable construct following the SUSPEND statement by execution of a RESUME statement (12.4.0$\frac{1}{2}$).

R1237$\frac{1}{2}$ *suspend-stmt* **is** SUSPEND

C1259$\frac{1}{3}$ (R1237$\frac{1}{2}$) A SUSPEND statement is allowed only in the scoping unit of a coroutine subprogram.

NOTE 12.40$\frac{2}{3}$

Execution of a SUSPEND statement does not terminate execution of an instance of a procedure. Entities that normally become undefined when a RETURN or END statement is executed do not become undefined when a SUSPEND statement is executed. The effect is as if a nonrecursive coroutine contained a SAVE statement without a *saved-entity-list*, or as if each instance of a

NOTE 12.40$\frac{2}{3}$ (cont.)

recursive coroutine were defined by a separate subprogram that contained a SAVE statement without a *saved-entity-list*. Additionaly, argument associations are retained.

12.7.4 Coroutines 289:13+

A **coroutine subprogram** is a subroutine in which COROUTINE appears in its SUBROUTINE statement.

If execution of an instance of a procedure defined by a coroutine subprogram is suspended by execution of a SUSPEND statement (12.5.2.5$\frac{1}{2}$), and that instance is subsequently resumed by executing a RESUME statement (12.4.1), execution continues with the first executable construct following the SUSPEND statement that suspended execution of that instance of the coroutine. When a procedure defined by a coroutine subprogram is invoked by a CALL statement, execution begins with the first executable construct of the subprogram.

[Editor: Insert "procedure instance association," before "or".] 410:16

[Editor: Insert a new subclause and renumber subsequent ones — TEX-o-matic.] 418:7+

16.4.3$\frac{1}{2}$ Procedure instance association

Procedure instance association is the association between an instance of a coroutine subprogram and

(1) An instance of a scoping unit that contains a CALL statement in which the procedure designator is a procedure name that is not a procedure pointer and designates a coroutine subprogram; at any instant an instance of a scoping unit might have as many prodedure instance associations as the number of CALL statements within it in which the designator is a procedure name that is not a procedure pointer and designates a coroutine,

(2) A procedure pointer that designates a coroutine subprogram; at any instant a procedure pointer can have only one procedure instance association, or

(3) the *data-ref* in a *data-ref*%*binding-name* that designates a coroutine subprogram; at any instant a derived-type object might have as many procedure instance associations as the number of coroutines bound to its type.

A procedure instance association is established by 419:20+

(1) Execution of a CALL statement in which the *procedure-designator* is a procedure name that designates a coroutine subprogram and is not a procedure pointer; the association is between the currently-executing instance of the scoping unit containing the CALL statement and the instance of the procedure created by execution of the CALL statement,

(2) Execution of a CALL statement in which the *procedure-designator* is a procedure pointer that designates a coroutine subprogram; the association is between the instance of the procedure pointer designated by the procedure designator at the instant the CALL statement is executed and the instance of the procedure created by execution of the CALL statement,

(3) Execution of a CALL statement in which the *procedure-designator* is a *data-ref*%*binding-name* that designates a coroutine subprogram; the association is between the instance of the *data-ref* designated by the procedure designator at the instant the CALL statement is executed and the instance of the procedure created by execution of the CALL statement,

(4) Execution of a pointer assignment (7.4.2) in which the *proc-target* is a procedure pointer that has the procedure instance association; the *proc-pointer-object* acquires the same association, or

(5) Execution of an intrinsic assignment (7.4.1.2) in which the *expr* or a subobject of it is a derived-type object that has the procedure instance association; the *variable* or the subobject of it that corresponds to the subobject of *expr* that has the association acquires the same association.

A procedure instance association between an entity and a procedure instance is terminated by

(1) Execution of a RETURN or END statement within the instance associated with the entity,
(2) Establishment of a procedure instance association between
  (a) An instance of the same coroutine and the entity, if the entity is not a procedure pointer, or
  (b) An instance of any coroutine and the entity, if the entity is a procedure pointer,
(3) Execution of a RETURN or END statement within the entity, if the entity is a procedure,
(4) An event that causes an entity that is a procedure pointer to become disassociated (16.4.2.1.2), to become undefined (16.4.2.1.3), to have its association status changed in another manner (16.4.2.1.4), or to be pointer assigned (7.4.2) other than by a pointer assignment that copies the same association,
(5) An event that causes an entity that is a *data-ref* to become defined (16.5.5) other than by an intrinsic assignment that copies the same association, or
(6) An event that causes an entity that is a *data-ref* to become undefined (16.5.6).

NOTE 16.16$\frac{1}{2}$

In intrinsic assignment of derived-type objects, pointer components, including procedure pointer components, are assigned as if by execution of a pointer assignment statement.

C.9.4$\frac{1}{2}$ Coroutines (12.7.4)

480:38+

The most common use, but not the exclusive use, of coroutines is to provide access from "library" procedures to user-provided code by way of reverse communication. Here is an example of a zero-finder that can access user-provided code either by forward or reverse communication.

```
COROUTINE SUBROUTINE ZERO_FINDER ( X1, X, F, TOL, STATUS, FUNC )
  ! ASSUME F IS MONOTONE BETWEEN X1 AND X.  F(X1)*F(X) SHALL BE <= 0.
  REAL, INTENT(INOUT) :: X1, X, F ! ARGUMENTS, FUNCTION VALUE
  REAL, INTENT(IN) :: TOL         ! TOLERANCE
  INTEGER, INTENT(OUT) :: STATUS  ! 0 => EVALUATE F(X)
                                  ! 1 => DONE, |F(X)| <= TOL
                                  ! -1 => F(X1)*F(X) INITIALLY > 0
  INTERFACE
    REAL FUNCTION FUNC ( X )
      REAL, INTENT(IN) :: X
    END FUNCTION FUNC
  END INTERFACE
  OPTIONAL :: FUNC

  REAL :: XSAVE, F1

  IF ( PRESENT(FUNC) ) THEN
    F1 = FUNC(X1)
    F = FUNC(X)
  ELSE
    XSAVE = X
    STATUS = 0
    X = X1
    SUSPEND
    F1 = F
    STATUS = 0
```

```
    X = XSAVE
    SUSPEND
  END IF
  IF ( F1 > 0.0 .AND. F > 0.0 .OR. F1 < 0.0 .AND. F < 0.0 ) THEN
    STATUS = -1
    RETURN
  END IF
  DO
    IF ( ABS(F) <= TOL ) THEN
      STATUS = 1
      RETURN
    END IF
    X = 0.5 * ( X + X1 ) ! VERY NAIVE
    IF ( PRESENT(FUNC) ) THEN
      F = FUNC(X)
    ELSE
      STATUS = 0
      SUSPEND ! GET A NEW F
    END IF
  END DO
END SUBROUTINE ZERO_FINDER

PROGRAM FIND_ZERO
  ! COMPUTE SQRT(2) USING THE ZERO_FINDER COROUTINE
  REAL :: X1=1.0, X=2.0, F, TOL=1.0E-6
  INTEGER :: STATUS
  CALL ZERO_FINDER ( X1, X, F, TOL, STATUS )
  DO
    SELECT CASE ( STATUS )
    CASE ( -1: )
      PRINT *, 'OOPS, ZERO NOT BRACKETED'
      STOP
    CASE ( 0 )
      F = 2 - X**2
      RESUME ZERO_FINDER
    CASE ( 1 )
      PRINT *, 'SQRT(2) = ', X
      EXIT
    END SELECT
  END DO
END PROGRAM FIND_ZERO
```