WORKING DRAFT

10-166

1st June 2010 14:42

This is an internal working document of J3.

Contents

Foreword

1 ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and nongovernmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

2 International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

3 The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

4 Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

5 ISO/IEC TR 29113:2010(E) was prepared by Joint Technical Committee ISO/IEC/JTC1,*Information technology*, Subcommittee SC22, *Programming languages, their environments and system software interfaces.*

6 This technical report specifies an enhancement of the parallel processing facilities of the programming language Fortran. Fortran is specified by the International Standard ISO/IEC 1539-1:2010.

7 It is the intention of ISO/IEC JTC1/SC22/WG5 that the semantics and syntax specified by this technical report be included in the next revision of the Fortran International Standard without change unless experience in the implementation and use of this feature identifies errors that need to be corrected, or changes are needed to achieve proper integration, in which case every reasonable effort will be made to minimize the impact of such changes on existing implementations.

Introduction

Technical Report on Enhanced Parallel Computing Facilities

1 Fortran, as standardized by ISO/IEC 1539-1:2010, provides core facilities for parallel programming with coarrays. A Fortran program containing coarrays is interpreted as if it were replicated a fixed number of times and all copies were executed asynchronously. Each copy has its own set of data objects and is called an image.

2 ISO/IEC TR xxxxx extends these core facilities with

- features in support of teams of images collaborating independently of other images;
- collective intrinsic procedures, which are invoked on a team of images and act collaboratively;
- the image control statements NOTIFY and QUERY for more detailed control of the collaborative actions of the images; and
- input/output facilities for files connected on more than one image.

3 The facility specified in ISO/IEC TR xxxxx is a compatible extension of Fortran as standardized by ISO/IEC 1539-1:2010.

4 ISO/IEC TR xxxxx is organized in clauses that relate to clauses of ISO/IEC 1539-1:2010 with the same names, viz :

Overview	Clause 1
Execution control	Clause 8
Input/output statements	Clause 9
Procedures	Clause 12
Intrinsic procedures and modules	Clause 13

5 It also contains the following nonnormative material:

Extended notes	Annex C

Technical Report — Enhanced Parallel Computing Facilities —

1 Overview

1.1 Scope

1 ISO/IEC TR xxxxx specifies the form and establishes the interpretation of facilities that extend the Fortran language defined by ISO/IEC 1539-1:2010.

1.2 Normative references

1 The following referenced standard is indispensable for the application of this document.

2 ISO/IEC 1539-1:2010, *Information technology—Programming languages—Fortran*

1.3 Terms and definitions

1 For the purposes of this document, the following terms and definitions apply in addition to those defined in ISO/IEC 1539-1:2010.

1 **1.3.1**
collective subroutine
intrinsic subroutine that is invoked on a team of images to perform a calculation on those images and assign the value of the result on all of them (4.1.1)

1 **1.3.2**
connect team
team of images that can reference an external unit

1 **1.3.3**
team
set of images identified by a scalar data object of type IMAGE_TEAM (4.4.2)

1 **1.3.4**
team synchronization
synchronization of the images in a team (2.2)

1.4 Compatibility

1.4.1 New intrinsic procedures

1 ISO/IEC TR xxxxx defines intrinsic procedures in addition to those specified in ISO/IEC 1539-1:2010. Therefore, a Fortran program conforming to ISO/IEC 1539-1:2010 might have a different interpretation under ISO/IEC TR xxxxx if it invokes an external procedure having the same name as one of the new intrinsic procedures, unless that procedure is specified to have the EXTERNAL attribute.

1.4.2 Fortran 2008 compatibility

1 ISO/IEC TR xxxxx is an upwardly compatible extension to ISO/IEC 1539-1:2010.

2 Execution control

2.1 Image control statements

1 Each of the following is an additional image control statement:

- SYNC TEAM;
- NOTIFY;
- QUERY;
- OPEN for a file that is being opened on more than one image;
- CLOSE for a file that is open on more than one image;
- CALL for a collective subroutine (4.1.1) or FORM_TEAM (4.3.11) .

2 During an execution of a statement that invokes more than one procedure, more than one invocation might cause execution of a CLOSE statement for a file with a connect team of only one image.

2.1.1 Segments

1 If an image P writes a record during the execution of segment P_i to a file that is opened for direct access with a TEAM= specifier, no other image Q shall read or write the record during execution of a segment that is unordered with P_i. Furthermore, it shall not read the record in a segment that succeeds P_i unless

- after image P writes the record, it executes a FLUSH statement for the file during the execution of a segment P_k, where $k \geq i$, and
- before image Q reads the record, it executes a FLUSH for the file during the execution of a segment Q_j that succeeds P_k.

NOTE 2.1

The incorrect sequencing of image control statements can suspend execution indefinitely. For example, one image might be executing a blocking QUERY for which an image in its image set never executes the corresponding NOTIFY.

2.2 SYNC TEAM statement

1
R201 *sync-team-stmt* **is** SYNC TEAM (*image-team* [, *sync-stat-list*])
R202 *image-team* **is** *scalar-variable*
C201 The *image-team* shall be a scalar variable of type IMAGE_TEAM from the intrinsic module ISO_FORTRAN_ENV.

2 Execution of a SYNC TEAM statement performs a team synchronization, which is a synchronization of the images in a team. The team is specified by the value of *image-team* and shall include the executing image. All images of the team shall execute a SYNC TEAM statement with a value of *image-team* that was constructed by corresponding invocations of the intrinsic subroutine FORM_TEAM for the team. They do not commence executing subsequent statements until all images in the team have executed a SYNC TEAM statement for the team an equal number of times since FORM_TEAM was invoked for the team. If images M and T are any two members of the team, the segments that execute before the statement on image M precede the segments that execute after the statement on image T.

3 Execution of an OPEN with a TEAM= specifier, a CLOSE for a unit whose connect team consists of more than one image, or a CALL for a collective subroutine is interpreted as if an execution of a SYNC TEAM statement for the

team occurred at the beginning and end of execution of the statement. If the statement contains an *image-team*, it specifies the team and shall satisfy the conditions required of an *image-team* in a SYNC TEAM statement; otherwise, the team is the connect team for the unit in a CLOSE statement or the set of all images for a CALL to a collective subroutine.

NOTE 2.2

Execution of the intrinsic subroutine FORM_TEAM also performs a team synchronization.

NOTE 2.3

In this example the images are divided into two teams, one for an ocean calculation and one for an atmosphere calculation.

```
USE, INTRINSIC :: ISO_FORTRAN_ENV
TYPE(IMAGE_TEAM) :: TEAM
INTEGER :: N2, STEP, NSTEPS
LOGICAL :: OCEAN

N2 = NUM_IMAGES()/2
OCEAN = (THIS_IMAGE()<=N2)
IF (OCEAN) THEN
   CALL FORM_TEAM (TEAM, [ (I, I=1,N2) ] )
ELSE
   CALL FORM_TEAM (TEAM, [ (I, I=N2+1,NUM_IMAGES()) ] )
END IF
   : ! Initial calculation
SYNC ALL
DO STEP = 1, NSTEPS
   IF (OCEAN) THEN
      DO
          : ! Ocean calculation
         SYNC TEAM (TEAM)
         IF ( ... ) EXIT ! Ready to swap data
      END DO
   ELSE
      DO
          : ! Atmosphere calculation
         SYNC TEAM (TEAM)
         IF ( ... ) EXIT ! Ready to swap data
      END DO
   END IF
   SYNC ALL
     :  ! Swap data
END DO
```

In the inner loops, each set of images first works entirely with its own data and each image synchronizes with the rest of its team. The number of synchronizations for the ocean team might differ from the number for the atmosphere team. The `SYNC ALL` that follows is needed to ensure that both teams have done their calculations before data are swapped.

2.3 NOTIFY and QUERY statements

R203 *notify-stmt* **is** NOTIFY (*image-set* [, *sync-stat-list*])
R204 *query-stmt* **is** QUERY (*image-set* [, *query-spec-list*])
1 R205 *query-spec* **is** READY = *scalar-logical-variable*
or *sync-stat*

C202 (R204) No specifier shall appear more than once in a given *query-spec-list*.

2 Execution on image M of a NOTIFY statement with a different image T in its *image-set* increments by 1 a record of the number of times, $N_{M \rightarrow T}$, image M executed such a NOTIFY statement.

3 A QUERY statement is blocking if and only if it has no READY= specifier. A QUERY statement is satisfied on completion of its execution if and only if it is a blocking QUERY statement or it set the variable specified by its READY= specifier to true.

4 Let $Q_{M \leftarrow T}$ denote the number of times image M has completed the execution of a satisfied QUERY statement with a different image T in its image set. Completion of execution on image M of a blocking QUERY statement is delayed until, for each different T in its image set, $N_{T \rightarrow M} > Q_{M \leftarrow T}$.

5 Execution of a non-blocking QUERY statement on image M causes the *scalar-logical-variable* of its READY= specifier to be assigned the value false if, for a different image T in the image set, $N_{T \rightarrow M} \leq Q_{M \leftarrow T}$; otherwise, true is assigned.

6 A NOTIFY statement execution on image T and a satisfied QUERY statement execution on image M correspond if and only if

- the NOTIFY statement's image set includes image M,
- the QUERY statement's image set includes image T, and
- after execution of both statements has completed, $N_{T \rightarrow M} = Q_{M \leftarrow T}$.

7 Segments on an image executed before the execution of a NOTIFY statement precede the segments on other images that follow execution of its corresponding QUERY statements.

NOTE 2.4

The NOTIFY and QUERY statements can be used to order statement executions between a producer and consumer image.

```
INTEGER,PARAMETER :: PRODUCER = 1, CONSUMER = 2
INTEGER :: VALUE[*]
LOGICAL :: READY

SELECT CASE (THIS_IMAGE())
CASE (PRODUCER)
   VALUE[CONSUMER] = 3
   NOTIFY (CONSUMER)
CASE (CONSUMER)
   WaitLoop: DO
      QUERY (PRODUCER,READY=READY)
      IF (READY) EXIT WaitLoop
      ! Statements neither referencing VALUE[CONSUMER], nor causing it to
      ! become defined or undefined
   END DO WaitLoop
   ! references to VALUE
CASE DEFAULT
   ! Statements neither referencing VALUE[CONSUMER], nor causing it to
   ! become defined or undefined
```

NOTE 2.4 (cont.)

```
END SELECT
```

Unlike SYNC IMAGES statements, the number of notifications and corresponding queries may be unequal. A program can complete with an excess number of notifies.

NOTE 2.5

NOTIFY/QUERY pairs can be used in place of SYNC ALL and SYNC IMAGES to achieve better load balancing and allow one image to proceed with calculations while another image is catching up. For example,

```
IF (THIS_IMAGE()==1) THEN
   DO I=1,100
      ...       ! Primary processing of column I
      NOTIFY(2) ! Done with column I
   END DO
   SYNC IMAGES(2)
ELSE IF (THIS_IMAGE()==2) THEN
   DO I=1,100
      QUERY(1) ! Wait until image 1 is done with column I
      ...      ! Secondary processing of column I
   END DO
   SYNC IMAGES(1)
END IF
```

3 Input/output statements

3.1 OPEN statement

3.1.1 TEAM= specifier in the OPEN statement

1 The OPEN statement has the additional specifier TEAM= *image-team*. The *image-team* specifies the connect team for the unit. If there is no TEAM= specifier, the connect team consists of only the executing image.

2 A named file that is opened with the TEAM= specifier is opened using the same name on each image of the team.

3 If the file is already connected on the image and the previous connect team has more than one image, the new connect team shall be the same.

4 Each record shall be read or written by a single image. The processor shall ensure that once an image commences transferring the data of a record to the file, no other image transfers data to the file until the whole record has been transferred.

5 If no error occurs during the execution of the OPEN statement with a NEWUNIT= specifier, the variable is defined with a processor determined NEWUNIT value that is the same on all images in the connect team.

6 All images in the connect team shall execute the same OPEN statement with identical values for the *connect-spec*s, except for ERR=, IOMSG=, IOSTAT=, NEWUNIT=, and TEAM=. There is an implicit team synchronization (2.2).

7 If the OPEN statement has a STATUS= specifier with the value SCRATCH, the processor shall connect the same file to the unit on all images in the connect team.

8 If the connect team contains more than one image, the OPEN statement shall

- specify direct access or
- specify sequential access and have an ACTION= specifier that evaluates to WRITE.

NOTE 3.1

Writing to a sequential file from more than one image without using synchronization is permitted, but is only useful for situations in which the ordering of records is unimportant. An example is for diagnostic output that is labeled with the image index.

9 A unit that is neither connected nor preconnected has an empty connect team.

10 The units identified by the values OUTPUT_UNIT and ERROR_UNIT in the intrinsic module ISO_FORTRAN_ENV are preconnected on all images. The unit identified by the value INPUT_UNIT in the intrinsic module ISO_FORTRAN_ENV is preconnected on image 1 and is not preconnected on other images. All other preconnected units have a connect team consisting of the executing image.

3.2 CLOSE statement

1 If an image executes a CLOSE statement, all images in the connect team of the unit specified shall execute a CLOSE statement for the unit with the same disposition. There is an implicit team synchronization associated with the execution of a CLOSE statement for a unit with a connect team that has more than one image (2.2).

2 During the completion step of termination of execution of a program, all units that are connected are closed.

NOTE 3.2

The effect is as though a CLOSE statement without a STATUS= specifier were executed on each connected unit, but without team synchronization for units with a connect team of more than one image.

3.3 File positioning statements

1 A unit whose connect team has more than one image shall not be referred to by a BACKSPACE, ENDFILE, or REWIND statement.

3.4 FLUSH statement

1 The FLUSH statement has the additional specifier TEAM= *image-team*.

2 Execution of a FLUSH statement causes data written to an external unit to be made available to other images of the unit's connect team which execute a FLUSH statement in a subsequent segment for that unit.

3.5 File inquiry statement

3.5.1 NEXTREC= specifier in the INQUIRE statement

1 The *scalar-int-variable* in the NEXTREC= specifier is assigned the value $n + 1$, where n is the record number of the last record read from or written to the connection for direct access by the executing image.

3.5.2 TEAM= specifier in the INQUIRE statement

1 The INQUIRE statement has the additional specifier TEAM= *image-team*.

2 The *image-team* in the TEAM= specifier is assigned the value of the connect team if the file or unit is connected; otherwise it is assigned a value that identifies an empty image set.

NOTE 3.3

The indices of the images in a team may be obtained by using TEAM_IMAGES (4.3.12).

3.6 Error conditions and the ERR= specifier

1 If an error condition occurs during execution of an OPEN or CLOSE statement on any of the images in the connect team, an error condition occurs on all images in the connect team.

3.7 IOSTAT= specifier

1 Execution of an input/output statement containing the IOSTAT= specifier causes the *scalar-int-variable* in the IOSTAT= specifier to become defined with the processor-dependent positive integer value of the constant STAT_STOPPED_IMAGE if the operation involves a team with more than one image and at least one of the images of the team initiates termination of execution.

4 Intrinsic procedures

4.1 Specification of the standard intrinsic procedures

4.1.1 Collective subroutine

1 A collective subroutine is one that is invoked on a team of images to perform a calculation on those images and which assigns the value of the result on all of them. If it is invoked by one image of a team, it shall be invoked by the same statement on all images of the team. There is an implicit team synchronization (2.2) at the beginning and end of the execution of a collective subroutine.

4.1.2 Arguments to collective subroutines

1 Each actual argument to a collective subroutine shall have the same bounds, cobounds, and type parameters as the corresponding actual argument on any other image of the team. Each actual argument corresponding to an INTENT (IN) argument of type IMAGE_TEAM shall have a value constructed by an invocation of FORM_TEAM for the team on that image.

2 On any two images of the team, the ultimate arguments for the first coarray dummy argument shall be corresponding coarrays as described in 2.4.7 of ISO/IEC 1539-1:2010, and the ultimate arguments of the RESULT dummy argument shall be corresponding coarrays.

4.2 Standard generic intrinsic procedures

1 In the Class column of Table 4.1,

C indicates that the procedure is a collective subroutine,

T indicates that the procedure in a transformational function.

Table 4.1: **Standard generic intrinsic procedure summary**

Procedure	Arguments	Class	Description
CO_ALL	(MASK, RESULT [, TEAM])	C	Determine whether all corresponding elements of MASK are true on a team of images.
CO_ANY	(MASK, RESULT [, TEAM])	C	Determine whether any corresponding element of MASK is true on a team of images.
CO_COUNT	(MASK, RESULT [, TEAM])	C	Count the numbers of true elements on a team of images.
CO_FINDLOC	(SOURCE, VALUE, RESULT, TEAM [, BACK]) or (SOURCE, VALUE, RESULT [, BACK])	C	Determine the image indeximage indices of the first or last image, in image index order, having a value that matches VALUE, on a team of images.
CO_MAXLOC	(SOURCE, RESULT [, TEAM])	C	Determine the image indices of the maximum values of the elements on a team of images.
CO_MAXVAL	(SOURCE, RESULT [, TEAM])	C	Determine the maximum values of the elements on a team of images.
CO_MINLOC	(SOURCE, RESULT [, TEAM])	C	Determine the image indices of the minimum values of the elements on a team of images.

Table 4.1: Standard generic intrinsic procedure summary (cont.)

Procedure	Arguments	Class	Description
CO_MINVAL	(SOURCE, RESULT [, TEAM])	C	Determine the minimum values of the elements on a team of images.
CO_PRODUCT	(SOURCE, RESULT [, TEAM])	C	Compute the products of elements on a team of images.
CO_SUM	(SOURCE, RESULT [, TEAM])	C	Sum elements on a team of images.
FORM_TEAM	(TEAM, IMAGES [, STAT, ERRMSG])	S	Form a team of images.
TEAM_IMAGES	(TEAM)	T	Rank one array of the indices of the images in a team.

4.3 Specifications of the standard intrinsic procedures

4.3.1 CO_ALL (MASK, RESULT [, TEAM])

1 **Description.** Determine whether all corresponding elements of MASK are true on a team of images.

2 **Class.** Collective subroutine.

3 **Arguments.**

MASK shall be a coarray of type logical. It may be a scalar or an array. It is an INTENT (IN) argument.

RESULT shall be a coarray of type logical and shall have the same shape as MASK. It is an INTENT (OUT) argument. If it is scalar, it is assigned the value true if the value of MASK is true on all the images of the team, and the value false otherwise. If it is an array, each element is assigned the value true if all corresponding elements of MASK are true on all the images of the team, and the value false otherwise.

TEAM (optional) shall be a scalar of type IMAGE_TEAM(4.4.2). It is an INTENT (IN) argument that specifies the team for which CO_ALL is performed. If TEAM is not present, the team consists of all images.

4 **Example.** If the number of images is two and MASK is the array [true, false, true] on one image and [true, true, true] on the other image, the value of RESULT after executing the statement CALL CO_ALL (MASK, RESULT) is [true, false, true].

4.3.2 CO_ANY (MASK, RESULT [, TEAM])

1 **Description.** Determine whether any corresponding element of MASK is true on a team of images.

2 **Class.** Collective subroutine.

3 **Arguments.**

MASK shall be a coarray of type logical. It may be a scalar or an array. It is an INTENT (IN) argument.

RESULT shall be a coarray of type logical and shall have the same shape as MASK. It is an INTENT (OUT) argument. If it is scalar it is assigned the value true if any value of MASK is true on any image of the team, and false otherwise. If it is an array, each element is assigned the value true if any of the corresponding elements of MASK are true on any image of the team, and false otherwise.

TEAM (optional) shall be a scalar of type IMAGE_TEAM(4.4.2). It is an INTENT (IN) argument that specifies the team for which CO_ANY is performed. If TEAM is not present, the team consists of all images.

4 **Example.** If the number of images is two and MASK is the array [true, false, false] on one image and [true, true, false] on the other image, the value of RESULT after executing the statement CALL CO_ANY (MASK,

RESULT) is [true, true, false].

4.3.3 CO_COUNT (MASK, RESULT [, TEAM])

1 **Description.** Count the numbers of true elements on a team of images.

2 **Class.** Collective subroutine.

3 **Arguments.**

MASK shall be a coarray of type logical. It may be a scalar or an array. It is an INTENT (IN) argument.

RESULT shall be a coarray of type integer and shall have the same shape as MASK. It is an INTENT (OUT) argument. If it is scalar, it is assigned a value equal to the number of images of the team for which MASK has the value true. If it is an array, each element is assigned a value equal to the number of corresponding elements of MASK on the images of the team that have the value true.

TEAM (optional) shall be a scalar of type IMAGE_TEAM(4.4.2). It is an INTENT (IN) argument that specifies the team for which CO_COUNT is performed. If TEAM is not present, the team consists of all images.

4 **Example.** If the number of images is two and MASK is the array [true, false, false] on one image and [true, true, false] on the other image, the value of RESULT after executing the statement
CALL CO_COUNT (MASK, RESULT) is [2, 1, 0].

4.3.4 CO_FINDLOC (SOURCE, VALUE, RESULT, TEAM [, BACK]) or CO_FINDLOC (SOURCE, VALUE, RESULT [, BACK])

1 **Description.** Determine the image indices of the first or last image, in image index order, having a value that matches VALUE, on a team of images.

2 **Class.** Collective subroutine.

3 **Arguments.**

SOURCE shall be a coarray of intrinsic type. It may be a scalar or an array. It is an INTENT (IN) argument.

VALUE shall be scalar and in type conformance with ARRAY, as specified in Table 7.3 of ISO/IEC 1539-1:2010 for relational intrinsic operations. It is an INTENT (IN) argument.

RESULT shall be a coarray of type integer and shall have the same shape as SOURCE. It is an INTENT (OUT) argument.

Case (i): RESULT is scalar. It is assigned the image index of an image of the team whose value of SOURCE matches VALUE, or zero if no such image exists.

Case (ii): RESULT is an array. Each element is assigned the image index of an image of the team whose corresponding element of SOURCE matches VALUE, or zero if no such image exists.

If both SOURCE and VALUE are of type logical, the comparison is performed using .EQV.; otherwise, the comparison is performed using == (.EQ.). If the value of the comparison is true, SOURCE or the element of SOURCE matches VALUE.

TEAM (optional) shall be a scalar of type IMAGE_TEAM(4.4.2). It is an INTENT (IN) argument that specifies the team for which CO_FINDLOC is performed. If TEAM does not appear, the team consists of all images.

BACK (optional) shall be a logical scalar. It is an INTENT (IN) argument.

If more than one image has a value that matches VALUE, and BACK is absent or present with the value false, the smallest such image index is assigned to RESULT. If BACK is present with the value true, the image whose index is assigned to RESULT is the largest such image index.

4 **Examples.** If the number of images is four and SOURCE is a scalar with the values 2, 4, 6, and 8 on the four different images, the value of RESULT after the statement CALL CO_FINDLOC (SOURCE, 6, RESULT) is 3

on all images.

5 If the number of images is two and SOURCE is the array [6, 5, 6] on the first image and [4, 6, 6] on the second image, the value of RESULT after the statement CALL CO_FINDLOC (SOURCE, 6, RESULT) is [1, 2, 1] and the value after the statement CALL CO_FINDLOC (SOURCE, 6, RESULT, .TRUE.) is [1, 2, 2].

4.3.5 CO_MAXLOC (SOURCE, RESULT [, TEAM])

1 **Description.** Determine the image indices of the maximum values of the elements on a team of images.

2 **Class.** Collective subroutine.

3 **Arguments.**

SOURCE shall be a coarray of type integer, real, or character. It may be a scalar or an array. It is an INTENT (IN) argument.

RESULT shall be a coarray of type integer and shall have the same shape as SOURCE. It is an INTENT (OUT) argument. If it is scalar, it is assigned a value equal to the image index of the maximum value of SOURCE on the images of the team; if more than one image has the maximum value, the smallest such image index is assigned. If RESULT is an array, each element of RESULT is assigned a value equal to the image index of the maximum value of all the corresponding elements of SOURCE on the images of the team; if more than one image has the maximum value, the smallest such image index is assigned.

If SOURCE is of type character, the result is the value that would be selected by application of intrinsic relational operators; that is, the collating sequence for characters with the kind type parameter of the argument is applied.

TEAM (optional) shall be a scalar of type IMAGE_TEAM(4.4.2). It is an INTENT (IN) argument that specifies the team for which CO_MAXLOC is performed. If TEAM is not present, the team consists of all images.

4 **Example.** If the number of images is two and SOURCE is the array [1, 5, 6] on one image and [4, 1, 6] on the other image, the value of RESULT after executing the statement
CALL CO_MAXLOC (SOURCE, RESULT) is [2, 1, 1].

4.3.6 CO_MAXVAL (SOURCE, RESULT [, TEAM])

1 **Description.** Determine the maximum values of the elements on a team of images.

2 **Class.** Collective subroutine.

3 **Arguments.**

SOURCE shall be a coarray of type integer, real, or character. It may be a scalar or an array. It is an INTENT (IN) argument.

RESULT shall be a coarray of the same type, type parameters and shape as SOURCE. It is an INTENT (OUT) argument. If it is scalar, it is assigned a value equal to the maximum value of SOURCE on all the images of the team. If it is an array, each element is assigned a value equal to the maximum value of all the corresponding elements of SOURCE on all the images of the team.

If SOURCE is of type character, the result is the value that would be selected by application of intrinsic relational operators; that is, the collating sequence for characters with the kind type parameter of the argument is applied.

TEAM (optional) shall be a scalar of type IMAGE_TEAM(4.4.2). It is an INTENT (IN) argument that specifies the team for which CO_MAXVAL is performed. If TEAM is not present, the team consists of all images.

4 **Example.** If the number of images is two and SOURCE is the array [1, 5, 3] on one image and [4, 1, 6] on the other image, the value of RESULT after executing the statement

CALL CO_MAXVAL (SOURCE, RESULT) is [4, 5, 6].

4.3.7 CO_MINLOC (SOURCE, RESULT [, TEAM])

1 **Description.** Determine the image indices of the minimum values of the elements on a team of images.

2 **Class.** Collective subroutine.

3 **Arguments.**

SOURCE shall be a coarray of type integer, real, or character. It may be a scalar or an array. It is an INTENT (IN) argument.

RESULT shall be a coarray of type integer and shall have the same shape as SOURCE. It is an INTENT (OUT) argument. If it is scalar, it is assigned a value equal to the image index of the minimum value of SOURCE on all the images of the team; if more than one image has the minimum value, the smallest such image index is assigned. If it is an array, each element is assigned a value equal to the image index of the minimum value of all the corresponding elements of SOURCE on the images of the team; if more than one image has the minimum value, the smallest such image index is assigned.

If SOURCE is of type character, the result is the value that would be selected by application of intrinsic relational operators; that is, the collating sequence for characters with the kind type parameter of the argument is applied.

TEAM (optional) shall be a scalar of type IMAGE_TEAM(4.4.2). It is an INTENT (IN) argument that specifies the team for which CO_MINLOC is performed. If TEAM is not present, the team consists of all images.

4 **Example.** If the number of images is two and SOURCE is the array [1, 5, 6] on one image and [4, 1, 6] on the other image, the value of RESULT after executing the statement
CALL CO_MINLOC (ARRAY, RESULT) is [1, 2, 1].

4.3.8 CO_MINVAL (SOURCE, RESULT [, TEAM])

1 **Description.** Determine the minimum values of the elements on a team of images.

2 **Class.** Collective subroutine.

3 **Arguments.**

SOURCE shall be a coarray of type integer, real, or character. It may be a scalar or an array. It is an INTENT (IN) argument.

RESULT shall be a coarray of the same type, type parameters, and shape as SOURCE. It is an INTENT (OUT) argument. If it is scalar it is assigned a value equal to the minimum value of SOURCE on all the images of the team. If it is an array, each element is assigned a value equal to the minimum value of all the corresponding elements of SOURCE on all the images of the team.

If SOURCE is of type character, the result is the value that would be selected by application of intrinsic relational operators; that is, the collating sequence for characters with the kind type parameter of the argument is applied.

TEAM (optional) shall be a scalar of type IMAGE_TEAM(4.4.2). It is an INTENT (IN) argument that specifies the team for which CO_MINVAL is performed. If TEAM is not present, the team consists of all images.

4 **Example.** If the number of images is two and SOURCE is the array [1, 5, 3] on one image and [4, 1, 6] on the other image, the value of RESULT after executing the statement
CALL CO_MINVAL (SOURCE, RESULT) is [1, 1, 3].

4.3.9 CO_PRODUCT (SOURCE, RESULT [, TEAM])

1 **Description.** Compute the products of elements on a team of images.

2 **Class.** Collective subroutine.

3 **Arguments.**

SOURCE shall be a coarray of numeric type. It may be a scalar or an array. It is an INTENT (IN) argument.

RESULT shall be a coarray of the same type, type parameters, and shape as SOURCE. It is an INTENT (OUT) argument. If it is scalar, it is assigned a value equal to a processor-dependent and image-dependent approximation to the product of the values of SOURCE on all the images of the team. If it is an array, each element is assigned a value equal to a processor-dependent and image-dependent approximation to the product of all the corresponding elements of SOURCE on the images of the team.

TEAM (optional) shall be a scalar of type IMAGE_TEAM(4.4.2). It is an INTENT (IN) argument that specifies the team for which CO_PRODUCT is performed. If TEAM is not present, the team consists of all images.

4 **Example.** If the number of images is two and SOURCE is the array [1, 5, 3] on one image and [4, 1, 6] on the other image, the value of RESULT after executing the statement
CALL CO_PRODUCT (SOURCE, RESULT) is [4, 5, 18].

4.3.10 CO_SUM (SOURCE, RESULT [, TEAM])

1 **Description.** Sum elements on a team of images.

2 **Class.** Collective subroutine.

3 **Arguments.**

SOURCE shall be a coarray of numeric type. It may be a scalar or an array. It is an INTENT (IN) argument.

RESULT shall be a coarray of the same type, type parameters, and shape as SOURCE. It is an INTENT (OUT) argument. If it is scalar, it is assigned a value equal to a processor-dependent and image-dependent approximation to the sum of the values of SOURCE on all the images of the team. If it is an array, each element is assigned a value equal to a processor-dependent and image-dependent approximation to the sum of all the corresponding elements of SOURCE on the images of the team.

TEAM (optional) shall be a scalar of type IMAGE_TEAM(4.4.2). It is an INTENT (IN) argument that specifies the team for which CO_SUM is performed. If TEAM is not present, the team consists of all images.

4 **Example.** If the number of images is two and SOURCE is the array [1, 5, 3] on one image and [4, 1, 6] on the other image, the value of RESULT after executing the statement
CALL CO_SUM (SOURCE, RESULT) is [5, 6, 9].

4.3.11 FORM_TEAM (TEAM, IMAGES [, STAT, ERRMSG])

1 **Description.** Form a team of images.

2 **Class.** Subroutine.

3 **Arguments.**

TEAM shall be a scalar of type IMAGE_TEAM(4.4.2). It is an INTENT (OUT) argument.

IMAGES shall be a rank-one integer array. It is an INTENT (IN) argument that specifies the image indices of the team members. An error condition occurs if

- IMAGES does not specify the same set of images on all images of the team,
- any element of IMAGES is not in the range 1, . . . , NUM_IMAGES ( ),
- any element of IMAGES has the same value as another element, or
- no element of IMAGES has the value THIS_IMAGE ( ).

STAT (optional) shall be a default integer scalar. It is an INTENT (OUT) argument. If no error occurs it is assigned the value zero. If any of the images of the team has initiated termination of execution it is assigned the value of the constant STAT_STOPPED_IMAGE in the intrinsic module ISO_FORTRAN_ENV. If any other error occurs, it is assigned a processor-dependent positive value different from STAT_STOPPED_IMAGE.

ERRMSG (optional) shall be a default character scalar. It is an INTENT (INOUT) argument. If an error condition occurs, it is assigned a processor-dependent explanatory message; otherwise, it is unchanged.

4 If FORM_TEAM is invoked by an image, an error condition occurs if it is not invoked by the same statement on all images specified by the IMAGES argument. If no error condition occurs, there is an implicit team synchronization after the team is formed.

5 If an error condition occurs and STAT is not present, error termination of execution is initiated.

6 **Example.** The following code fragment splits images into two groups and implicitly synchronizes each of the teams if there are two or more images. If there is only one image, that image becomes the only team member. The members of the team may be specified in a different order on different images.

7

```
USE, INTRINSIC :: ISO_FORTRAN_ENV, ONLY: IMAGE_TEAM
INTEGER :: I
TYPE(IMAGE_TEAM) :: TEAM
IF (THIS_IMAGE()<=NUM_IMAGES()/2) THEN
   CALL FORM_TEAM(TEAM, [(I,I=1,NUM_IMAGES()/2)])
ELSE
   CALL FORM_TEAM(TEAM, [(I,I=NUM_IMAGES()/2+1,NUM_IMAGES())])
END IF
```

4.3.12 TEAM_IMAGES (TEAM)

1 **Description.** Rank one array of the indices of the images in a team.

2 **Class.** Transformatiional function.

3 **Argument.** TEAM shall be a scalar of type IMAGE_TEAM(4.4.2).

4 **Result Characteristics.** The result is a default integer array of rank one and of size equal to the number of images in the team identified by TEAM.

5 **Result Value.** The result is a rank-one array whose element values are the indices, in increasing order, of the images in the team identified by TEAM.

6 **Examples.** If the value of TEAM was defined by the statement CALL FORM_TEAM (TEAM, [4, 2, 1]) then TEAM_IMAGES (TEAM) has the value [1, 2, 4]. For a value of TEAM that identifies an empty image set, the result is an array of size zero.

4.4 The ISO_FORTRAN_ENV intrinsic module

4.4.1 General

1 ISO/IEC TR xxxxx defines an additional object in the ISO_FORTRAN_ENV intrinsic module.

4.4.2 IMAGE_TEAM

1 A scalar object of type IMAGE_TEAM identifies a team of images. This type is extensible, has only private components, has pointer components but no allocatable components, has no type parameters, and has default initialization to a value that identifies an empty image set.

NOTE 4.1

When values of type IMAGE_TEAM are constructed by calling the intrinsic subroutine FORM_TEAM on the images of a team, the processor may choose to store information in such values to speed later processing of team synchronizations and collective subroutine calls. This information is likely to vary between images. The standard treats the information as if held in pointer components in order that copying a value of type IMAGE_TEAM to another image causes its value on the other image to become undefined.

Annex A

(Informative)

Extended notes

A.1 Notes re Clause 2 of ISO/IEC 1539-1:2010

A.1.1 Normal and error termination of execution

1 This code fragment illustrates the use of STOP and ALL STOP in a climate model that uses two teams, one for the ocean and one for the atmosphere.

2 If something goes badly wrong in the atmosphere calculation, the whole model is invalid and a restart is impossible, so all images stop as soon as possible without trying to preserve any data.

3 If something goes slightly wrong with the atmosphere calculation, the images in the atmosphere team write their data to files and stop, but their data remain available to the ocean images which complete execution of the OCEAN subroutine. On return from the computation routines, if something went slightly wrong with the atmosphere calculation, the ocean images write data to files and stop, ready for a restart in a later run.

4

```
   USE,INTRINSIC :: ISO_FORTRAN_ENV
   TYPE(IMAGE_TEAM) :: OCEAN_TEAM, ATMOSPHERE_TEAM
   INTEGER :: I, SYNC_STAT
   :
! Form two teams
   CALL FORM_TEAM (OCEAN_TEAM, [I,I=1,NUM_IMAGES()/2])
   CALL FORM_TEAM (ATMOSPHERE_TEAM, [I,I=1+NUM_IMAGES()/2,NUM_IMAGES()])
   :
! Perform independent calculations
   IF (THIS_IMAGE() > NUM_IMAGES()/2) THEN
      CALL ATMOSPHERE (ATMOSPHERE_TEAM)
   ELSE
      CALL OCEAN (OCEAN_TEAM)
   END IF
! Wait for both teams to finish
   SYNC ALL (STAT=SYNC_STAT)
   IF (SYNC_STAT == SYNC_STOPPED_IMAGE) THEN
       : ! preserve data on file
      STOP
   END IF
   CALL EXCHANGE_DATA ! Exchange data between teams
   :
CONTAINS
   SUBROUTINE ATMOSPHERE (TEAM)
      TYPE(IMAGE_TEAM) :: TEAM
      : ! Perform atmosphere calculation.
      IF (...) THEN ! something has gone slightly wrong
         : ! preserve data on file
         STOP
      END IF
   :
      IF (...) ALL STOP  ! something has gone very badly wrong
      :
      SYNC TEAM (TEAM, STAT=SYNC_STAT))
      IF (SYNC_STAT == SYNC_STOPPED_IMAGE) THEN
         : ! remaining atmosphere images preserve data in a file
         STOP
      END IF
   END SUBROUTINE ATMOSPHERE
```

A.2 Notes re Clause 13 of ISO/IEC 1539-1:2010

A.2.1 Collective coarray subroutine variations

1 For a scalar coarray, an intrinsic collective subroutine applies an operation to the values of all the corresponding coarrays on a set of images and provides the result on all the images of the set in a scalar argument of INTENT (OUT). For a coarray that is an array, the operation is applied to each set of corresponding elements and the result is provided on all the images in an array of the shape of the coarray.

2 Simple routines can be written to also apply the operation to the elements of the coarray on an image. Various versions of a global sum can be programmed, for example:

3

```
MODULE global_sum_module
   INTRINSIC, PRIVATE :: CO_SUM, SIZE, SUM
CONTAINS
   REAL FUNCTION global_sum(array)
      REAL,INTENT(IN) :: array(:,:)[*]
      REAL,SAVE       :: temp[*]
      temp = SUM(array)             ! Sum on the executing image
      CALL CO_SUM(temp, global_sum)
   END FUNCTION global_sum

   REAL FUNCTION global_sum_mask(array, mask)
      REAL,INTENT(IN)    :: array(:,:)[*]
      LOGICAL,INTENT(IN) :: mask(:,:)
      REAL,SAVE          :: temp[*]
      temp = SUM(array, MASK=mask)  ! Sum on the executing image
      CALL CO_SUM(temp, global_sum_mask)
   END FUNCTION global_sum_mask

   FUNCTION global_sum_dim(array, dim)
      REAL, INTENT(IN)    :: array(:,:)[*]
      INTEGER, INTENT(IN) :: dim
      REAL, ALLOCATABLE   :: global_sum_dim(:)
      REAL, ALLOCATABLE   :: temp(:)[:]
      ALLOCATE (global_sum_dim(SIZE(array, 3-dim)))
      ALLOCATE (          temp(SIZE(array, 3-dim))[*])
      temp = SUM(array, dim)       ! Sum of the local part of the coarray.
      CALL CO_SUM(temp, global_sum_dim)
   END FUNCTION global_sum_dim
END MODULE global_sum_module
```