TECHNICAL SPECIFICATION

ISO/IEC TS 18508

First edition
2015-12-01

Information technology — Additional Parallel Features in Fortran

Technologies de l'information — Caractéristiques parallèles supplémentaires en Fortran

Reference number
ISO/IEC TS 18508:2015(E)

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

Contents

1 Scope . . . 1

2 Normative references . . . 3

3 Terms and definitions . . . 5

4 Compatibility . . . 7
4.1 New intrinsic procedures . . . 7
4.2 Fortran 2008 compatibility . . . 7

5 Teams of images . . . 9
5.1 Team concepts . . . 9
5.2 TEAM_TYPE . . . 9
5.3 CHANGE TEAM construct . . . 9
5.4 Image selectors . . . 11
5.5 FORM TEAM statement . . . 12
5.6 SYNC TEAM statement . . . 13

6 Failed images . . . 15
6.1 Introduction . . . 15
6.2 FAIL IMAGE statement . . . 15
6.3 CRITICAL construct . . . 16
6.4 STAT_FAILED_IMAGE . . . 16

7 Events . . . 17
7.1 Introduction . . . 17
7.2 EVENT_TYPE . . . 17
7.3 EVENT POST statement . . . 18
7.4 EVENT WAIT statement . . . 18

8 Intrinsic procedures . . . 19
8.1 General . . . 19
8.2 Atomic subroutines . . . 19
8.3 Collective subroutines . . . 20
8.4 New intrinsic procedures . . . 21
8.4.1 ATOMIC_ADD (ATOM, VALUE [, STAT]) . . . 21
8.4.2 ATOMIC_AND (ATOM, VALUE [, STAT]) . . . 21
8.4.3 ATOMIC_CAS (ATOM, OLD, COMPARE, NEW [, STAT]) . . . 22
8.4.4 ATOMIC_FETCH_ADD (ATOM, VALUE, OLD [, STAT]) . . . 22
8.4.5 ATOMIC_FETCH_AND (ATOM, VALUE, OLD [, STAT]) . . . 22
8.4.6 ATOMIC_FETCH_OR (ATOM, VALUE, OLD [, STAT]) . . . 23
8.4.7 ATOMIC_FETCH_XOR (ATOM, VALUE, OLD [, STAT]) . . . 23
8.4.8 ATOMIC_OR (ATOM, VALUE [, STAT]) . . . 24
8.4.9 ATOMIC_XOR (ATOM, VALUE [, STAT]) . . . 24
8.4.10 CO_BROADCAST (A, SOURCE_IMAGE [, STAT, ERRMSG]) . . . 24
8.4.11 CO_MAX (A [, RESULT_IMAGE, STAT, ERRMSG]) . . . 25
8.4.12 CO_MIN (A [, RESULT_IMAGE, STAT, ERRMSG]) . . . 25
8.4.13 CO_REDUCE (A, OPERATOR [, RESULT_IMAGE, STAT, ERRMSG]) . . . 26

8.4.14 CO_SUM (A [, RESULT_IMAGE, STAT, ERRMSG]) . . . 27
8.4.15 EVENT_QUERY (EVENT, COUNT [, STAT]) . . . 27
8.4.16 FAILED_IMAGES ([TEAM, KIND]) . . . 28
8.4.17 GET_TEAM ([LEVEL]) . . . 28
8.4.18 IMAGE_STATUS (IMAGE, [TEAM]) . . . 29
8.4.19 STOPPED_IMAGES ([TEAM, KIND]) . . . 29
8.4.20 TEAM_NUMBER ([TEAM]) . . . 30
8.5 Modified intrinsic procedures . . . 31
8.5.1 ATOMIC_DEFINE and ATOMIC_REF . . . 31
8.5.2 IMAGE_INDEX . . . 31
8.5.3 MOVE_ALLOC . . . 31
8.5.4 NUM_IMAGES . . . 32
8.5.5 THIS_IMAGE . . . 32

9 Required editorial changes to ISO/IEC 1539-1:2010(E) . . . 33
9.1 General . . . 33
9.2 Edits to Introduction . . . 33
9.3 Edits to clause 1 . . . 33
9.4 Edits to clause 2 . . . 34
9.5 Edits to clause 4 . . . 35
9.6 Edits to clause 6 . . . 35
9.7 Edits to clause 8 . . . 36
9.8 Edits to clause 9 . . . 39
9.9 Edits to clause 13 . . . 39
9.10 Edits to clause 16 . . . 43
9.11 Edits to annex A . . . 44
9.12 Edits to annex C . . . 44

Annex A (informative) Extended notes . . . 45
A.1 Clause 5 notes . . . 45
A.1.1 Example using three teams . . . 45
A.1.2 Accessing coarrays in sibling teams . . . 45
A.1.3 Reducing the codimension of a coarray . . . 46
A.2 Clause 6 notes . . . 47
A.2.1 Example involving failed images . . . 47
A.3 Clause 7 notes . . . 49
A.3.1 EVENT_QUERY example . . . 49
A.3.2 EVENT_QUERY example that tolerates image failure . . . 50
A.3.3 EVENTS example . . . 52
A.4 Clause 8 notes . . . 53
A.4.1 Collective subroutine examples . . . 53
A.4.2 Atomic memory consistency . . . 53

Foreword

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.

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

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.

In other circumstances, particularly when there is an urgent market requirement for such documents, the joint technical committee may decide to publish an ISO/IEC Technical Specification (ISO/IEC TS), which represents an agreement between the members of the joint technical committee and is accepted for publication if it is approved by 2/3 of the members of the committee casting a vote.

An ISO/IEC TS is reviewed after three years in order to decide whether it will be confirmed for a further three years, revised to become an International Standard, or withdrawn. If the ISO/IEC TS is confirmed, it is reviewed again after a further three years, at which time it must either be transformed into an International Standard or be withdrawn.

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.

ISO/IEC TS 18508:2015 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC22, *Programming languages, their environments and system software interfaces*.

Introduction

The system for parallel programming in Fortran, as standardized by ISO/IEC 1539-1:2010, defines simple syntax for access to data on another image of a program, synchronization statements for controlling the ordering of execution segments between images, and collective allocation and deallocation of memory on all images.

The existing system for parallel programming does not provide for an environment where a subset of the images can easily work on part of an application while not affecting other images in the program. This complicates development of independent parts of an application by separate teams of programmers. The existing system does not provide a mechanism for a processor to identify what images have failed during execution of a program. This adversely affects the resilience of programs executing on large systems. The synchronization primitives available in the existing system do not provide a convenient mechanism for ordering execution segments on different images without requiring that those images arrive at a synchronization point before either is allowed to proceed. This introduces unnecessary inefficiency into programs. Finally, the existing system does not provide intrinsic procedures for commonly used collective and atomic memory operations. Intrinsic procedures for these operations can be highly optimized for the target computational system, providing significantly improved program performance.

This Technical Specification extends the facilites of Fortran for parallel programming to provide for grouping the images of a program into nonoverlapping teams that can more effectively execute independently parts of a larger problem, for the processor to indicate which images have failed during execution and allow continued execution of the program on the remaining images, for a system of events that can be used for fine grain ordering of execution segments, and for collective and atomic memory operation subroutines that can provide better performance for specific operations involving more than one image.

The facility specified in this Technical Specification is a compatible extension of Fortran as standardized by ISO/IEC 1539-1:2010, ISO/IEC 1539-1:2010/Cor 1:2012, and ISO/IEC 1539-1:2010/Cor 2:2013.

It is the intention of ISO/IEC JTC 1/SC22 that the semantics and syntax specified by this Technical Specification be included in the next revision of ISO/IEC 1539-1 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.

This Technical Specification is organized in 8 clauses:

Scope	Clause 1
Normative references	Clause 2
Terms and definitions	Clause 3
Compatibility	Clause 4
Teams of images	Clause 5
Failed images	Clause 6
Events	Clause 7
Intrinsic procedures	Clause 8
Required editorial changes to ISO/IEC 1539-1:2010(E)	Clause 9

It also contains the following nonnormative material:

Extended notes	Annex A

1 Scope

This Technical Specification specifies the form and establishes the interpretation of facilities that extend the Fortran language defined by ISO/IEC 1539-1:2010, ISO/IEC 1539-1:2010/Cor 1:2012, and ISO/IEC 1539-1:2010/Cor 2:2013. The purpose of this Technical Specification is to promote portability, reliability, maintainability, and efficient execution of parallel programs written in Fortran, for use on a variety of computing systems.

This Technical Specification does not specify formal data consistency model. Developing the formal data consistency model is left until the integration of these facilities into ISO/IEC 1539-1.

(Blank page)

2 Normative references

The following referenced standards are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

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

ISO/IEC 1539-1:2010/Cor 1:2012, *Information technology—Programming languages—Fortran—Part 1:Base language TECHNICAL CORRIGENDUM 1*

ISO/IEC 1539-1:2010/Cor 2:2013, *Information technology—Programming languages—Fortran—Part 1:Base language TECHNICAL CORRIGENDUM 2*

(Blank page)

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 1539-1:2010 and the following apply. The intrinsic module ISO_FORTRAN_ENV is extended by this Technical Specification.

3.1
active image
image that has not failed or initiated termination

3.2
asynchronous progress
ability of an image to reference or define a coarray on another image without waiting for that image to execute any particular statement or class of statement

3.3
established coarray
⟨in a team⟩ coarray that is accessible using a coindexed designator (5.1)

3.4
team
set of images that can readily execute independently of other images (5.1)

3.4.1
current team
⟨of an image⟩ team specified by the most recently executed CHANGE TEAM statement of an active CHANGE TEAM construct, or initial team if no CHANGE TEAM construct is active (5.1)

3.4.2
initial team
team, consisting of all images, that began execution of the program (5.1)

3.4.3
parent team
⟨of a team⟩ current team during the execution of the CHANGE TEAM statement that established the team (5.1)

3.4.4
team number
integer value identifying a team within its parent team (5.1)

3.5
failed image
image that has ceased participating in program execution but has not initiated termination (6.1)

3.6
stopped image
image that has initiated normal termination

3.7
event variable
scalar variable of type EVENT_TYPE (7.2) in the intrinsic module ISO_FORTRAN_ENV

3.8
team variable
scalar variable of type TEAM_TYPE (5.2) in the intrinsic module ISO_FORTRAN_ENV

(Blank page)

4 Compatibility

4.1 New intrinsic procedures

This Technical Specification 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 this Technical Specification 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.

4.2 Fortran 2008 compatibility

This Technical Specification specifies an upwardly compatible extension to ISO/IEC 1539-1:2010, as modified by ISO/IEC 1539-1:2010/Cor 1:2012 and ISO/IEC 1539-1:2010/Cor 2:2013.

(Blank page)

5 Teams of images

5.1 Team concepts

A team of images is a set of images that can readily execute independently of other images. Syntax and semantics of *image-selector* (R624 in ISO/IEC 1539-1:2010) are extended to determine how cosubscripts are mapped to image indices for sibling or ancestor team references. Initially, the current team consists of all images and this is known as the initial team. Except for the initial team, every team has a unique parent team. A team is divided into new teams by executing a FORM TEAM statement. Each new team is identified by an integer value known as its team number. Information about the team to which the current image belongs can be determined by the processor from the collective value of the team variables on the images of the team.

During execution, each image has a current team, which is only changed by execution of CHANGE TEAM and END TEAM statements. Executing a CHANGE TEAM statement changes the current team to the team specified by the *team-variable*, and execution of the corresponding END TEAM statement restores the current team back to that immediately prior to execution of the CHANGE TEAM statement.

A nonallocatable coarray that is neither a dummy argument, host associated with a dummy argument, declared as a local variable of a subprogram, nor declared in a BLOCK construct is established in the initial team. An allocated allocatable coarray is established in the team in which it was allocated. An unallocated allocatable coarray is not established. A coarray that is an associating entity in a *coarray-association* of a CHANGE TEAM statement is established in the team of its CHANGE TEAM construct. A nonallocatable coarray that is a dummy argument or host associated with a dummy argument is established in the team in which the procedure was invoked. A nonallocatable coarray that is a local variable of a subprogram or host associated with a local variable of a subprogram is established in the team in which the procedure was invoked. A nonallocatable coarray declared in a BLOCK construct is established in the team in which the BLOCK statement was executed. A coarray dummy argument is not established in any ancestor team even if the corresponding actual argument is established in one or more of them.

5.2 TEAM_TYPE

TEAM_TYPE is a derived type with private components. It is an extensible type with no type parameters. Each component is fully default-initialized. A scalar variable of this type describes a team. TEAM_TYPE is defined in the intrinsic module ISO_FORTRAN_ENV.

A scalar variable of type TEAM_TYPE is a team variable. The default initial value of a team variable shall not represent any valid team.

5.3 CHANGE TEAM construct

The CHANGE TEAM construct changes the current team to which the executing image belongs.

R501 *change-team-construct* **is** *change-team-stmt*
block
end-change-team-stmt

R502 *change-team-stmt* **is** [*team-construct-name*:] CHANGE TEAM (*team-variable* ■
■ [, *coarray-association-list*] [, *sync-stat-list*])

R503 *coarray-association* **is** *codimension-decl* => *coselector-name*

R504 *end-change-team-stmt* **is** END TEAM [(*sync-stat-list*)] [*team-construct-name*]

R505 *team-variable* **is** *scalar-variable*

C501 (R501) A branch within a CHANGE TEAM construct shall not have a branch target that is outside the construct.

C502 (R501) A RETURN statement shall not appear within a CHANGE TEAM construct.

C503 (R501) An *exit-stmt* or *cycle-stmt* within a CHANGE TEAM construct shall not belong to an outer construct.

C504 (R501) If the *change-team-stmt* of a *change-team-construct* specifies a *team-construct-name*, the corresponding *end-change-team-stmt* shall specify the same *team-construct-name*. If the *change-team-stmt* of a *change-team-construct* does not specify a *team-construct-name*, the corresponding *end-change-team-stmt* shall not specify a *team-construct-name*.

C505 (R503) The *coarray-name* in the *codimension-decl* shall not be the same as any *coselector-name* in the *change-team-stmt* or the same as a *coarray-name* in another *codimension-decl* in the *change-team-stmt*.

C506 (R505) A *team-variable* shall be of type TEAM_TYPE (5.2).

C507 (R502) No *coselector-name* shall appear more than once in a *change-team-stmt*.

C508 (R503) A *coselector-name* shall be the name of an accessible coarray.

The values of the *team-variables* on the active images of the team shall be those of team variables defined by execution of the same FORM TEAM statement on all active images of the team or be the values of team variables for the initial team. The current team for the statements of the CHANGE TEAM *block* is the team specified by the value of the *team-variable*. The current team is not changed by a redefinition of the team variable during execution of the CHANGE TEAM construct. A CHANGE TEAM construct completes execution by executing its END TEAM statement.

A coselector name identifies a coarray. The coarray shall be established when the CHANGE TEAM statement begins execution. A *codimension-decl* in a *coarray-association* associates a coarray with an established coarray during the execution of the block. This coarray is an associating entity (8.1.3.2, 8.1.3.3, 16.5.1.6 of ISO/IEC 1539-1:2010). Its name is an associate name that has the scope of the construct. It has the declared type, dynamic type, type parameters, rank, and bounds of the established coarray. Its corank and cobounds are those specified in the *codimension-decl*.

Within a CHANGE TEAM construct, a coarray that is not an associating entity has the corank and cobounds that it had when it was established.

An allocatable coarray that was allocated when execution of a CHANGE TEAM construct began shall not be deallocated during execution of the construct. An allocatable coarray that is allocated when execution of a CHANGE TEAM construct completes is deallocated if it was not allocated when execution of the construct began.

The CHANGE TEAM and END TEAM statements are image control statements. All active images of the current team shall execute the same CHANGE TEAM statement. When a CHANGE TEAM statement is executed, there is an implicit synchronization of all active images of the team containing the executing image that is identified by *team-variable*. On each active image of the team, execution of the segment following the statement is delayed until all other active images of the team have executed the same statement the same number of times since execution last began in the team that was current before execution of the CHANGE TEAM statement. When a CHANGE TEAM construct completes execution, there is an implicit synchronization of all active images in the current team. On each active image of the team, execution of the segment following the END TEAM statement is delayed until all other active images of the team have executed the same construct the same number of times since execution last began in the team that was current before execution of the corresponding CHANGE TEAM

statement.

NOTE 5.1

Deallocation of an allocatable coarray that was not allocated at the beginning of a CHANGE TEAM construct, but is allocated at the end of execution of the construct, occurs even for allocatable coarrays with the SAVE attribute.

5.4 Image selectors

The syntax rule R624 *image-selector* in subclause 6.6 of ISO/IEC 1539-1:2010 is replaced by:

R624 *image-selector* **is** *lbracket cosubscript-list* ■
■ [, *team-identifier*] [, STAT = *stat-variable*] *rbracket*

R624a *team-identifier* **is** TEAM_NUMBER = *scalar-int-expr*
or TEAM = *team-variable*

C509 (R624) *stat-variable* shall not be a coindexed object.

If TEAM= appears in a coarray designator, *team-variable* shall be defined with a value that represents the current or an ancestor team. The coarray shall be established in that team or an ancestor of that team and the cosubscripts determine the image index in that team.

If TEAM_NUMBER = appears in a coarray designator and the current team is not the initial team, the *scalar-int-expr* shall be defined with the value of a team number for one of the teams that were formed by execution of the FORM TEAM statement for the current team. The coarray shall be established in an ancestor of the current team and the cosubscripts determine the image index in the team identified by TEAM_NUMBER. If TEAM_NUMBER = appears in a coarray designator and the current team is the initial team, the value of *scalar-int-expr* is ignored.

Execution of a statement containing an *image-selector* with a STAT= specifier causes the *stat-variable* to become defined. If the designator is part of an operand that is evaluated, and the object designated is on a failed image, the *stat-variable* is defined with the value STAT_FAILED_IMAGE in the intrinsic module ISO_FORTRAN_ENV; otherwise, it is defined with the value zero.

The denotation of a *stat-variable* in an *image-selector* shall not depend on the evaluation of any entity in the same statement. The value of an expression shall not depend on the value of any *stat-variable* that appears in the same statement. The value of a *stat-variable* in an *image-selector* shall not be affected by the execution of any part of the statement, other than by whether the image specified by the *image-selector* has failed.

NOTE 5.2

The image selector in `b[i]` identifies the current team. The image selector in `b[i,team_number=1]` identifies a sibling team. The image selector in `b[i,TEAM=ancestor]` identifies the team `ancestor`.

NOTE 5.3

The expression `x[i,stat=is(f(n))]`, where `is` is an array and `f(n)` is a function that returns an integer value is an example of the denotation of a *stat-variable* in an *image-selector* that depends on the evaluation of an entity in the same statement.

NOTE 5.4

The use of a STAT=specifier in an image selector allows a test to be made for a failed image in a reference where the use of a processor-dependent result could cause error termination or an incorrect execution path. Where there is no such possibility, it may be preferable to rely on the STAT=specifier in the next image control statement.

NOTE 5.5

In the following code, the vector *a* of length N*P is distributed over P images. Each image has an array A(0:N+1) holding its own values of *a* and halo values from its two neighbors. The images are divided into two teams that execute independently but periodically exchange halo data. Before the data exchange, all images (of the initial team) must be synchronized and for the data exchange the coindices of the initial team are needed.

```
USE, INTRINSIC :: ISO_FORTRAN_ENV, ONLY: TEAM_TYPE
TYPE(TEAM_TYPE) :: INITIAL, BLOCK
REAL :: A(0:N+1)[*]
INTEGER :: ME, P2
INITIAL = GET_TEAM()
ME = THIS_IMAGE()
P2 = NUM_IMAGES()/2
FORM TEAM(1+(ME-1)/P2,BLOCK)
CHANGE TEAM(BLOCK,B[*]=>A)
   DO
      ! Iterate within team
        :
      ! Halo exchange across team boundary
      SYNC TEAM(INITIAL)
      IF(ME==P2  ) B(N+1) = A(1)[ME+1,TEAM=INITIAL]
      IF(ME==P2+1)   B(0) = A(N)[ME-1,TEAM=INITIAL]
      SYNC TEAM(INITIAL)
   END DO
END TEAM
```

5.5 FORM TEAM statement

R506 *form-team-stmt* **is** FORM TEAM (*team-number*, *team-variable* ■
■ [, *form-team-spec-list*])

R507 *team-number* **is** *scalar-int-expr*

R508 *form-team-spec* **is** NEW_INDEX = *scalar-int-expr*
or *sync-stat*

C510 (R506) No specifier shall appear more than once in a *form-team-spec-list*.

The FORM TEAM statement creates one or more teams from the active images of the current team. The value of *team-number* shall be positive and identifies the new team to which the executing image will belong. The team number of each new team within the current team is the *team-number* value of the FORM TEAM statements executed by its images. Successful execution of a FORM TEAM statement assigns the *team-variable* on each participating image a value that specifies the new team that the image will belong to.

The value of the *scalar-int-expr* in a NEW_INDEX= specifier specifies the image index that the executing image will have in the team specified by *team-number*. It shall be positive and less than or equal to the number of images in the team. Each image with the same value for *team-number* shall have a different value for the NEW_INDEX= specifier. If the NEW_INDEX= specifier does not appear, the image index that the executing image will have in the team specified by *team-number* is a processor-dependent value that shall be positive, unique within the team, and not greater than the number of images in the team.

The FORM TEAM statement is an image control statement. If the FORM TEAM statement is executed on one image, the same statement shall be executed on all active images of the current team. When a FORM TEAM statement is executed, there is an implicit synchronization of all active images in the current team. On these images, execution of the segment following the statement is delayed until all other active images in the current

team have executed the same statement the same number of times since execution last began in this team. If an error condition other than detection of a failed image occurs, the team variable becomes undefined.

NOTE 5.6

Executing the statement

```
FORM TEAM ( 2-MOD(ME,2), ODD_EVEN )
```

with `ME` an integer with value THIS_IMAGE() and `ODD_EVEN` of type TEAM_TYPE, divides the current team into two teams according to whether the image index is even or odd.

NOTE 5.7

When executing on P^2 images with corresponding coarrays on each image representing parts of a larger array spread over a P by P square, the following code establishes teams for the rows with image indices equal to the column indices.

```
USE, INTRINSIC :: ISO_FORTRAN_ENV
TYPE(TEAM_TYPE) :: ROW
REAL :: A[P,*]
INTEGER :: ME(2)
ME(:) = THIS_IMAGE(A)
FORM TEAM(ME(1),ROW,NEW_INDEX=ME(2))
```

5.6 SYNC TEAM statement

R509 *sync-team-stmt* **is** SYNC TEAM (*team-variable* [, *sync-stat-list*])

The SYNC TEAM statement is an image control statement. The values of the *team-variable*s on the active images of the team shall be those defined by execution of the same FORM TEAM statement on all active images of the team or shall be the values of team variables for the initial team. Execution of a SYNC TEAM statement performs a synchronization of the executing image with each of the other active images of the team specified by *team-variable*. Execution on an image, M, of the segment following the SYNC TEAM statement is delayed until each active other image of the specified team has executed a SYNC TEAM statement specifying the same team as many times as has image M since execution last began in this team. The segments that executed before the SYNC TEAM statement on an image precede the segments that execute after the SYNC TEAM statement on another image.

NOTE 5.8

A SYNC TEAM statement performs a synchronization of images of a particular team whereas a SYNC ALL statement performs a synchronization of all images of the current team.

(Blank page)

6 Failed images

6.1 Introduction

A failed image is one that has ceased participating in program execution but has not initiated termination. A failed image remains failed for the remainder of the program execution. The conditions that cause an image to fail are processor dependent.

Defining a coindexed object on a failed image has no effect other than defining the *stat-variable*, if one appears, with the value STAT_FAILED_IMAGE (6.4). The value of an expression that includes a reference to a coindexed object on a failed image is processor dependent. Execution continues after such a reference.

When an image fails during the execution of a segment, a data object on a nonfailed image becomes undefined if it might be defined or undefined by execution of a statement of the segment other than an invocation of an atomic subroutine.

If the *lock-variable* in a LOCK statement was locked by a failed image and was not unlocked by that image, it becomes unlocked.

The CRITICAL statement described in subclause 8.1.5 of ISO/IEC 1539-1:2010 is extended as described in 6.3.

NOTE 6.1

A failed image is usually associated with a hardware failure of a cpu, memory system, or interconnection network. A failure that occurs while a coindexed reference or definition, or collective action, is in progress might leave variables on other images that would be defined by that action in an undefined state. Similarly, failure while using a file might leave that file in an undefined state.

NOTE 6.2

Continued execution after the failure of image 1 in the initial team might be difficult because of the lost connection to standard input. However, the likelihood of a given image failing is small. With a large number of images, the likelihood of some image other than image 1 in the initial team failing is significant and it is for this circumstance that STAT_FAILED_IMAGE is designed.

NOTE 6.3

In addition to detecting that an image has failed by having the variable in a STAT=specifier or a STAT argument of a call to a collective or atomic subroutine assigned the value STAT_FAILED_IMAGE, an image can get the indices of failed images in a specified team by invoking the intrinsic function FAILED_IMAGES.

6.2 FAIL IMAGE statement

R601 *fail-image-stmt* **is** FAIL IMAGE

Execution of a FAIL IMAGE statement causes the executing image to behave as if it has failed. Neither normal nor error termination is initiated, but no further statements are executed by that image.

NOTE 6.4

The FAIL IMAGE statement allows a program to test a recovery algorithm without experiencing an actual failure.

NOTE 6.4 (cont.)

On a processor that does not have the ability to detect that an image has failed, execution of a FAIL IMAGE statement might provide a simulated failure environment that provides debug information.

In a piece of code that executes about once a second, invoking this subroutine on an image

```
SUBROUTINE FAIL
   REAL :: X
   CALL RANDOM_NUMBER(X)
   IF (X<0.001) FAIL IMAGE
END SUBROUTINE FAIL
```

will cause that image to have an independent 1/1000 chance of failure every second if the random number generators on different images are independent.

6.3 CRITICAL construct

The syntax rule R811 in subclause 8.1.5 of ISO/IEC 1539-1:2010 is replaced by:

R811 *critical-stmt* **is** [*critical-construct-name* :] CRITICAL [(*sync-stat-list*)]

If an image fails during the execution of a CRITICAL construct, the execution of the construct is regarded by other images as complete.

If the processor has the ability to detect that an image has failed, when an image completes execution of a CRITICAL statement that has a STAT= specifier and the previous image to have entered the construct failed while executing it, the specified variable becomes defined with the value STAT_FAILED_IMAGE from the intrinsic module ISO_FORTRAN_ENV. Otherwise, when an image completes execution of a CRITICAL statement that has a STAT= specifier, the specified variable becomes defined with the value zero. When an image completes execution of a CRITICAL statement that has an ERRMSG= specifier, if it assigns a nonzero value to the status variable of a STAT= specifier or would have done so if a STAT= specifier had appeared, the processor shall assign an explanatory message to the specified variable as if by intrinsic assignment; otherwise, the value of the specified variable shall not be changed.

6.4 STAT_FAILED_IMAGE

If the processor has the ability to detect that an image has failed, the value of the default integer scalar constant STAT_FAILED_IMAGE is positive; otherwise, the value of STAT_FAILED_IMAGE is negative. If the processor has the ability to detect that an image involved in execution of an image control statement, a reference to a coindexed object, or a collective or atomic subroutine has failed and does so, and no error condition other than a failed image is detected, the value of STAT_FAILED_IMAGE is assigned to the variable specified in a STAT=specifier in an execution of an image control statement or a reference to a coindexed object, or the STAT argument in an invocation of a collective or atomic procedure. If more than one nonzero status value is valid for the execution of a statement, the status variable is defined with a value other than STAT_FAILED_IMAGE. If the STAT= specifier of an execution of a CHANGE TEAM, END TEAM, FORM TEAM, SYNC ALL, SYNC IMAGES, or SYNC TEAM statement is assigned the value STAT_FAILED_IMAGE, the intended action shall have taken place for all active images involved.

STAT_FAILED_IMAGE is defined in the intrinsic module ISO_FORTRAN_ENV. The values of the named constants IOSTAT_INQUIRE_INTERNAL_UNIT, STAT_FAILED_IMAGE, STAT_LOCKED, STAT_LOCKED_OTHER_IMAGE, STAT_STOPPED_IMAGE, and STAT_UNLOCKED in that module shall be distinct.

7 Events

7.1 Introduction

An image can post an event to notify another image that it can proceed to work on tasks that use common resources. An image can wait on events posted by other images and can query if images have posted events.

Segments ordered by EVENT POST and EVENT WAIT are included in the partial order specified in subclause 2.3.5 of ISO/IEC 1539-1:2010.

7.2 EVENT_TYPE

EVENT_TYPE is a derived type with private components. It is an extensible type with no type parameters. Each nonallocatable component is fully default-initialized. EVENT_TYPE is defined in the intrinsic module ISO_FORTRAN_ENV.

A scalar variable of type EVENT_TYPE is an event variable. The value of an event variable includes its event count, which is updated by execution of a sequence of EVENT POST or EVENT WAIT statements. The effect of each change is as if the atomic subroutine ATOMIC_ADD were executed with a variable that stores the event count as its ATOM argument. A coarray that is of type EVENT_TYPE may be referenced or defined during the execution of a segment that is unordered relative to the execution of another segment in which that coarray of type EVENT_TYPE is defined. The event count is of type integer with kind ATOMIC_INT_KIND, where ATOMIC_INT_KIND is a named constant defined in the intrinsic module ISO_FORTRAN_ENV. The initial value of the event count of an event variable is zero.

C701 A named variable of type EVENT_TYPE shall be a coarray. A named variable with a noncoarray subcomponent of type EVENT_TYPE shall be a coarray.

C702 An event variable shall not appear in a variable definition context except as the *event-variable* in an EVENT POST or EVENT WAIT statement, as an *allocate-object* in an ALLOCATE statement without a SOURCE= *alloc-opt*, as an *allocate-object* in a DEALLOCATE statement, or as an actual argument in a reference to a procedure with an explicit interface if the corresponding dummy argument has INTENT (INOUT).

C703 A variable with a nonpointer subobject of type EVENT_TYPE shall not appear in a variable definition context except as an *allocate-object* in an ALLOCATE statement without a SOURCE= *alloc-opt*, as an *allocate-object* in a DEALLOCATE statement, or as an actual argument in a reference to a procedure with an explicit interface if the corresponding dummy argument has INTENT (INOUT).

NOTE 7.1

The restrictions against changing an event variable except via EVENT POST and EVENT WAIT statements ensure the integrity of its value and facilitate efficient implementation, particularly when special synchronization is needed for correct event handling.

NOTE 7.2

The updates of atomic variables are coherent but not necessarily consistent, so a processor might have to use extra synchronization to obtain the consistency required for the segments ordered by EVENT POST and EVENT WAIT statements.

7.3 EVENT POST statement

The EVENT POST statement provides a way to post an event. It is an image control statement.

R701 *event-post-stmt* **is** EVENT POST (*event-variable* [, *sync-stat-list*])

R702 *event-variable* **is** *scalar-variable*

C704 (R702) An *event-variable* shall be of type EVENT_TYPE (7.2).

Successful execution of an EVENT POST statement atomically increments the count of the event variable by 1. If an error condition occurs during execution of an EVENT POST statement, the value of the count of the event variable is processor dependent.

The completion of an EVENT POST statement does not depend on the execution of a corresponding EVENT WAIT statement.

7.4 EVENT WAIT statement

The EVENT WAIT statement provides a way to wait until events are posted. It is an image control statement.

R703 *event-wait-stmt* **is** EVENT WAIT (*event-variable* [, *wait-spec-list*])

R704 *wait-spec* **is** *until-spec*
or *sync-stat*

R705 *until-spec* **is** UNTIL_COUNT = *scalar-int-expr*

C705 (R703) An *event-variable* in an *event-wait-stmt* shall not be coindexed.

C706 (R703) An *until-spec* shall not appear more than once in an *event-wait-stmt*.

Execution of an EVENT WAIT statement causes the following sequence of actions:

(1) the threshold value is set to *scalar-int-expr* if the UNTIL_COUNT specifier appears and *scalar-int-expr* has a positive value, and to 1 otherwise,
(2) the executing image waits until the count of the event variable is greater than or equal to its threshold value or an error condition occurs, and
(3) if no error condition occurs, the count of the event variable is atomically decremented by its threshold value.

If an error occurs during execution of an EVENT WAIT statement, the value of the count of its event variable is processor dependent.

An EVENT POST statement execution is initially unsatisfied. Successful execution of an EVENT WAIT statement with a threshold of k satisfies the first k unsatisfied EVENT POST statement executions for that event variable. This EVENT WAIT statement execution causes the segment following the EVENT WAIT statement execution to succeed the segments preceding those k EVENT POST statement executions.

NOTE 7.3

An unreasonably long wait on an EVENT WAIT statement in the presence of failed images may be because it was intended that the event be posted by an image that has failed.

NOTE 7.4

Event variables are restricted so that EVENT WAIT statements can only wait on an event variable on the executing image. This enables more efficient implementation of this concept.

8 Intrinsic procedures

8.1 General

Detailed specifications of the generic intrinsic procedures ATOMIC_ADD, ATOMIC_AND, ATOMIC_CAS, ATOMIC_FETCH_ADD, ATOMIC_FETCH_AND, ATOMIC_FETCH_OR, ATOMIC_FETCH_XOR, ATOMIC_OR, ATOMIC_XOR, CO_BROADCAST, CO_MAX, CO_MIN, CO_REDUCE, CO_SUM, EVENT_QUERY, FAILED_IMAGES, GET_TEAM, IMAGE_STATUS, STOPPED_IMAGES, and TEAM_NUMBER are provided in 8.4. The types and type parameters of the arguments to these intrinsic procedures are determined by these specifications. The "Argument" paragraphs specify requirements on the actual arguments of the procedures. All of these intrinsic procedures are pure.

The intrinsic procedures ATOMIC_DEFINE, ATOMIC_REF, IMAGE_INDEX, MOVE_ALLOC, NUM_IMAGES, and THIS_IMAGE described in clause 13 of ISO/IEC 1539-1:2010, as modified by ISO/IEC 1539-1:2010/Cor 1:2012, are extended as described in 8.5.

8.2 Atomic subroutines

An atomic subroutine is an intrinsic subroutine that performs an action on its ATOM argument or the count of its EVENT argument atomically. For any two executions of atomic subroutines in unordered segments by different images on the same atomic object, the effect is as if one of the executions is performed before the other in a single segment on a separate image, without access to the object in either execution interleaving with access to the object in the other. Which is executed first is indeterminate. The sequence of atomic actions within ordered segments is specified in 2.3.5 of ISO/IEC 1539-1:2010. If two variables are updated by atomic actions in segments P_1 and P_2, and the changes to them are observed by atomic accesses from a segment Q which is unordered relative to either P_1 or P_2, the changes need not be observed in segment Q in the same order as they are made in segments P_1 and P_2, even if segments P_1 and P_2 are ordered.

Atomic operations shall make asynchronous progress (3.2). If the atomic variable X[A] is defined by an atomic subroutine on image B, image C repeatedly references X[A] by an atomic subroutine in an unordered segment, and no other image defines X[A], image C will eventually receive the value defined by image B, even if none of the images A, B, or C execute an image control statement until after X[A] is defined by image B and that value is received by image C.

For invocation of an atomic subroutine with an argument OLD, the determination of the value to be assigned to OLD is part of the atomic operation even though the assignment of that value to OLD is not. For invocation of an atomic subroutine, evaluation of an INTENT(IN) argument is not part of the atomic action.

If the STAT argument is present in an invocation of an atomic subroutine and no error condition occurs, the argument is assigned the value zero.

If the STAT argument is present in an invocation of an atomic subroutine other than ATOMIC_REF and an error condition occurs, any ATOM or OLD argument becomes undefined. The STAT argument is assigned the value STAT_FAILED_IMAGE if a coindexed ATOM or EVENT argument is determined to be located on a failed image; otherwise, the STAT argument is assigned a processor-dependent positive value that is different from STAT_FAILED_IMAGE.

If a condition occurs that would assign a nonzero value to a STAT argument but the STAT argument is not present, error termination is initiated.

NOTE 8.1

The properties of atomic subroutines support their use for designing customized synchronization mechanisms. The programmer needs to account for all possible orderings of sequences of atomic subroutine executions that can arise as a consequence of the above rules; the orderings can turn out to be different on different images even in the same program run.

8.3 Collective subroutines

A collective subroutine is one that is invoked on each active image of the current team to perform a calculation on those images and that assigns the computed value on one or all of them. If it is invoked by one image, it shall be invoked by the same statement on all active images of the current team in execution segments that are not ordered with respect to each other. From the beginning to the end of execution as the current team, the sequence of invocations of collective subroutines shall be the same on all active images of the current team. A call to a collective subroutine shall appear only in a context that allows an image control statement.

If the A argument to a collective subroutine is a whole coarray the corresponding ultimate arguments on all active images of the current team shall be corresponding coarrays as described in 2.4.7 of ISO/IEC 1539-1:2010.

Collective subroutines have the optional arguments STAT and ERRMSG. If the STAT argument is present in the invocation on one image it shall be present on the corresponding invocations on all active images of the current team.

If the STAT argument is present in an invocation of a collective subroutine and its execution is successful, the argument is assigned the value zero.

If the STAT argument is present in an invocation of a collective subroutine and an error condition occurs, the argument is assigned a nonzero value and the A argument becomes undefined. If execution involves synchronization with an image that has initiated normal termination, the argument is assigned the value of STAT_STOPPED_IMAGE in the intrinsic module ISO_FORTRAN_ENV; otherwise, if all images of the current team are active, the argument is assigned a processor-dependent positive value that is different from the value of STAT_STOPPED_IMAGE or STAT_FAILED_IMAGE in the intrinsic module ISO_FORTRAN_ENV. Otherwise, if an image of the current team has been detected as failed, the argument is assigned the value of the constant STAT_FAILED_IMAGE.

If a condition occurs that would assign a nonzero value to a STAT argument but the STAT argument is not present, error termination is initiated.

If an ERRMSG argument is present in an invocation of a collective subroutine and an error condition occurs during its execution, the processor shall assign an explanatory message to the argument. If no error condition occurs, the processor shall not change the value of the argument.

NOTE 8.2

The argument A becomes undefined if an error condition occurs during execution of a collective subroutine because it is intended that implementations be able to use A as scratch space.

NOTE 8.3

All of the collective subroutines have an argument A with INTENT(INOUT) that holds the original data on entry and the result on return. If it is desired to retain the original data, this is readily obtained by making a copy before entry. Here is an example:

```
REDUCTION = ORIGINAL
CALL CO_MIN(REDUCTION)
```

NOTE 8.4

There is no separate synchronization at the beginning and end of an invocation of a collective subroutine, which allows overlap with other actions. However, each collective subroutine involves transfer of data between images. A transfer from an image cannot occur before the collective subroutine has been invoked on that image. The rules of Fortran do not allow the value of an associated argument such as A to be changed except via the argument. This includes action taken by another image that has not started its execution of the collective subroutine or has finished it. This restriction has the effect of a partial synchronization of invocations of a collective subroutine.

If the actual argument is a coarray, the implementation might be able to optimize the execution of the collective, for example by avoiding copying of data.

8.4 New intrinsic procedures

8.4.1 ATOMIC_ADD (ATOM, VALUE [, STAT])

Description. Atomic add operation.

Class. Atomic subroutine.

Arguments.

ATOM shall be a scalar coarray or coindexed object of type integer with kind ATOMIC_INT_KIND, where ATOMIC_INT_KIND is a named constant in the intrinsic module ISO_FORTRAN_ENV. It is an INTENT (INOUT) argument. ATOM becomes defined with the value of ATOM + INT(VALUE, ATOMIC_INT_KIND).

VALUE shall be an integer scalar. It is an INTENT (IN) argument.

STAT (optional) shall be a noncoindexed integer scalar with a decimal exponent range of at least four. It is an INTENT(OUT) argument.

Example.

CALL ATOMIC_ADD(I[3], 42) causes the value of I on image 3 to become defined with the value 46 if the value of I[3] was 4 when the atomic operation executed.

8.4.2 ATOMIC_AND (ATOM, VALUE [, STAT])

Description. Atomic bitwise AND operation.

Class. Atomic subroutine.

Arguments.

ATOM shall be a scalar coarray or coindexed object of type integer with kind ATOMIC_INT_KIND, where ATOMIC_INT_KIND is a named constant in the intrinsic module ISO_FORTRAN_ENV. It is an INTENT (INOUT) argument. ATOM becomes defined with the value IAND (ATOM, INT(VALUE, ATOMIC_INT_KIND)).

VALUE shall be an integer scalar. It is an INTENT(IN) argument.

STAT (optional) shall be a noncoindexed integer scalar with a decimal exponent range of at least four. It is an INTENT(OUT) argument.

Example. CALL ATOMIC_AND (I[3], 6) causes I on image 3 to become defined with the value 4 if the value of I[3] was 5 when the atomic operation executed.

8.4.3 ATOMIC_CAS (ATOM, OLD, COMPARE, NEW [, STAT])

Description. Atomic compare and swap.

Class. Atomic subroutine.

Arguments.

ATOM shall be a scalar coarray or coindexed object of type integer with kind ATOMIC_INT_KIND or of type logical with kind ATOMIC_LOGICAL_KIND, where ATOMIC_INT_KIND and ATOMIC_LOGICAL_KIND are named constants in the intrinsic module ISO_FORTRAN_ENV. It is an INTENT (INOUT) argument. If the value of ATOM is equal to the value of COMPARE, ATOM becomes defined with the value of INT (NEW, ATOMIC_INT_KIND) if it is of type integer, and with the value of NEW if it is of type logical. If the value of ATOM is not equal to the value of COMPARE, the value of ATOM is not changed.

OLD shall be a scalar of the same type and kind as ATOM. It is an INTENT (OUT) argument. It is defined with the value of ATOM that was used for performing the atomic operation.

COMPARE shall be a scalar of the same type and kind as ATOM. It is an INTENT(IN) argument.

NEW shall be a scalar of the same type as ATOM. It is an INTENT(IN) argument.

STAT (optional) shall be a noncoindexed integer scalar with a decimal exponent range of at least four. It is an INTENT(OUT) argument.

Example. CALL ATOMIC_CAS(I[3], OLD, Z, 1) causes I on image 3 to become defined with the value 1 if its value is that of Z, and OLD to be defined with the value of I on image 3 that was used for performing the atomic operation.

8.4.4 ATOMIC_FETCH_ADD (ATOM, VALUE, OLD [, STAT])

Description. Atomic fetch and add operation.

Class. Atomic subroutine.

Arguments.

ATOM shall be a scalar coarray or coindexed object of type integer with kind ATOMIC_INT_KIND, where ATOMIC_INT_KIND is a named constant in the intrinsic module ISO_FORTRAN_ENV. It is an INTENT (INOUT) argument. ATOM becomes defined with the value of ATOM + INT(VALUE, ATOMIC_INT_KIND).

VALUE shall be an integer scalar. It is an INTENT (IN) argument.

OLD shall be a scalar of the same type and kind as ATOM. It is an INTENT (OUT) argument. It is defined with the value of ATOM that was used for performing the atomic operation.

STAT (optional) shall be a noncoindexed integer scalar with a decimal exponent range of at least four. It is an INTENT(OUT) argument.

Example. CALL ATOMIC_FETCH_ADD(I[3], 7, OLD) causes I on image 3 to become defined with the value 12 and the value of OLD on the image executing the statement to be defined with the value 5 if the value of I[3] was 5 when the atomic operation executed.

8.4.5 ATOMIC_FETCH_AND (ATOM, VALUE, OLD [, STAT])

Description. Atomic fetch and bitwise AND operation.

Class. Atomic subroutine.

Arguments.

ATOM shall be a scalar coarray or coindexed object of type integer with kind ATOMIC_INT_KIND, where ATOMIC_INT_KIND is a named constant in the intrinsic module ISO_FORTRAN_ENV. It is an IN-

TENT (INOUT) argument. ATOM becomes defined with the value of IAND(ATOM, INT(VALUE, ATOMIC_INT_KIND)).

VALUE shall be an integer scalar. It is an INTENT (IN) argument.

OLD shall be a scalar of the same type and kind as ATOM. It is an INTENT (OUT) argument. It is defined with the value of ATOM that was used for performing the atomic operation.

STAT (optional) shall be a noncoindexed integer scalar with a decimal exponent range of at least four. It is an INTENT(OUT) argument.

Example. CALL ATOMIC_FETCH_AND (I[3], 6, IOLD) causes I on image 3 to become defined with the value 4 and the value of IOLD on the image executing the statement to be defined with the value 5 if the value of I[3] was 5 when the atomic operation executed.

8.4.6 ATOMIC_FETCH_OR (ATOM, VALUE, OLD [, STAT])

Description. Atomic fetch and bitwise OR operation.

Class. Atomic subroutine.

Arguments.

ATOM shall be a scalar coarray or coindexed object of type integer with kind ATOMIC_INT_KIND, where ATOMIC_INT_KIND is a named constant in the intrinsic module ISO_FORTRAN_ENV. It is an INTENT (INOUT) argument. ATOM becomes defined with the value of IOR(ATOM, INT(VALUE, ATOMIC_INT_KIND)).

VALUE shall be an integer scalar. It is an INTENT (IN) argument.

OLD shall be a scalar of the same type and kind as ATOM. It is an INTENT (OUT) argument. It is defined with the value of ATOM that was used for performing the atomic operation.

STAT (optional) shall be a noncoindexed integer scalar with a decimal exponent range of at least four. It is an INTENT(OUT) argument.

Example. CALL ATOMIC_FETCH_OR (I[3], 1, IOLD) causes I on image 3 to become defined with the value 3 and the value of IOLD on the image executing the statement to be defined with the value 2 if the value of I[3] was 2 when the atomic operation executed.

8.4.7 ATOMIC_FETCH_XOR (ATOM, VALUE, OLD [, STAT])

Description. Atomic fetch and bitwise exclusive OR operation.

Class. Atomic subroutine.

Arguments.

ATOM shall be a scalar coarray or coindexed object of type integer with kind ATOMIC_INT_KIND, where ATOMIC_INT_KIND is a named constant in the intrinsic module ISO_FORTRAN_ENV. It is an INTENT (INOUT) argument. ATOM becomes defined with the value of IEOR(ATOM, INT(VALUE, ATOMIC_INT_KIND)).

VALUE shall be an integer scalar. It is an INTENT (IN) argument.

OLD shall be a scalar of the same type and kind as ATOM. It is an INTENT (OUT) argument. It is defined with the value of ATOM that was used for performing the atomic operation.

STAT (optional) shall be a noncoindexed integer scalar with a decimal exponent range of at least four. It is an INTENT(OUT) argument.

Example. CALL ATOMIC_FETCH_XOR (I[3], 1, IOLD) causes I on image 3 to become defined with the value 2 and the value of IOLD on the image executing the statement to be defined with the value 3 if the value of I[3] was 3 when the atomic operation executed.

8.4.8 ATOMIC_OR (ATOM, VALUE [, STAT])

Description. Atomic bitwise OR operation.

Class. Atomic subroutine.

Arguments.

ATOM shall be a scalar coarray or coindexed object of type integer with kind ATOMIC_INT_KIND, where ATOMIC_INT_KIND is a named constant in the intrinsic module ISO_FORTRAN_ENV. It is an INTENT (INOUT) argument. ATOM becomes defined with the value IOR (ATOM, INT(VALUE, ATOMIC_INT_KIND)).

VALUE shall be an integer scalar. It is an INTENT (IN) argument.

STAT (optional) shall be a noncoindexed integer scalar with a decimal exponent range of at least four. It is an INTENT(OUT) argument.

Example. CALL ATOMIC_OR (I[3], 1) causes I on image 3 to become defined with the value 3 if the value of I[3] was 2 when the atomic operation executed.

8.4.9 ATOMIC_XOR (ATOM, VALUE [, STAT])

Description. Atomic bitwise exclusive OR operation.

Class. Atomic subroutine.

Arguments.

ATOM shall be a scalar coarray or coindexed object of type integer with kind ATOMIC_INT_KIND, where ATOMIC_INT_KIND is a named constant in the intrinsic module ISO_FORTRAN_ENV. It is an INTENT (INOUT) argument. ATOM becomes defined with the value IEOR (ATOM, INT(VALUE, ATOMIC_INT_KIND)).

VALUE shall be an integer scalar. It is an INTENT (IN) argument.

STAT (optional) shall be a noncoindexed integer scalar with a decimal exponent range of at least four. It is an INTENT(OUT) argument.

Example. CALL ATOMIC_XOR (I[3], 1) causes I on image 3 to become defined with the value 2 if the value of I[3] was 3 when the atomic operation executed.

8.4.10 CO_BROADCAST (A, SOURCE_IMAGE [, STAT, ERRMSG])

Description. Copy a value to all images of the current team.

Class. Collective subroutine.

Arguments.

A shall have the same dynamic type and type parameter values on all images in the current team. It shall not be a coindexed object. It is an INTENT(INOUT) argument. If it is an array, it shall have the same shape on all images in the current team. A becomes defined, as if by intrinsic assignment, on all images in the current team with the value of A on image SOURCE_IMAGE.

SOURCE_IMAGE shall be an integer scalar. It is an INTENT(IN) argument. It shall be the image index of an image in the current team and have the same value on all images in the current team.

STAT (optional) shall be a noncoindexed integer scalar with a decimal exponent range of at least four. It is an INTENT(OUT) argument.

ERRMSG (optional) shall be a noncoindexed default character scalar. It is an INTENT(INOUT) argument.

The effect of the presence of the optional arguments STAT and ERRMSG is described in 8.3.

Example. If A is the array [1, 5, 3] on image one, after execution of CALL CO_BROADCAST(A,1) the value of A on all images of the current team is [1, 5, 3].

8.4.11 CO_MAX (A [, RESULT_IMAGE, STAT, ERRMSG])

Description. Compute elemental maximum value on the current team of images.

Class. Collective subroutine.

Arguments.

A shall be of type integer, real, or character. It shall have the same type and type parameters on all images in the current team. It shall not be a coindexed object. It is an INTENT(INOUT) argument. If it is a scalar, the computed value is equal to the maximum value of A on all images in the current team. If it is an array it shall have the same shape on all images of the current team and each element of the computed value is equal to the maximum value of all corresponding elements of A on the images in the current team.

RESULT_IMAGE (optional) shall be an integer scalar. It is an INTENT(IN) argument. If it is present, it shall be present on all images in the current team, have the same value on all images in the current team, and that value shall be the image index of an image in the current team.

STAT (optional) shall be a noncoindexed integer scalar with a decimal exponent range of at least four. It is an INTENT(OUT) argument.

ERRMSG (optional) shall be a noncoindexed default character scalar. It is an INTENT(INOUT) argument.

If RESULT_IMAGE is not present, the computed value is assigned to A on all images in the current team. If RESULT_IMAGE is present, the computed value is assigned to A on image RESULT_IMAGE and A on all other images in the current team becomes undefined.

The effect of the presence of the optional arguments STAT and ERRMSG is described in 8.3.

Example. If the number of images in the current team is two and A is the array [1, 5, 3] on one image and [4, 1, 6] on the other image, the value of A after executing the statement CALL CO_MAX(A) is [4, 5, 6] on both images.

8.4.12 CO_MIN (A [, RESULT_IMAGE, STAT, ERRMSG])

Description. Compute elemental minimum value on the current team of images.

Class. Collective subroutine.

Arguments.

A shall be of type integer, real, or character. It shall have the same type and type parameters on all images in the current team. It shall not be a coindexed object. It is an INTENT(INOUT) argument. If it is a scalar, the computed value is equal to the minimum value of A on all images in the current team. If it is an array it shall have the same shape on all images in the current team and each element of the computed value is equal to the minimum value of all corresponding elements of A on the images in the current team.

RESULT_IMAGE (optional) shall be an integer scalar. It is an INTENT(IN) argument. If it is present, it shall be present on all images in the current team, have the same value on all images in the current team, and that value shall be the image index of an image in the current team.

STAT (optional) shall be a noncoindexed integer scalar with a decimal exponent range of at least four. It is an INTENT(OUT) argument.

ERRMSG (optional) shall be a noncoindexed default character scalar. It is an INTENT(INOUT) argument.

If RESULT_IMAGE is not present, the computed value is assigned to A on all images in the current team. If RESULT_IMAGE is present, the computed value is assigned to A on image RESULT_IMAGE and A on all other

images in the current team becomes undefined.

The effect of the presence of the optional arguments STAT and ERRMSG is described in 8.3.

Example. If the number of images in the current team is two and A is the array [1, 5, 3] on one image and [4, 1, 6] on the other image, the value of A after executing the statement CALL CO_MIN(A) is [1, 1, 3] on both images.

8.4.13 CO_REDUCE (A, OPERATOR [, RESULT_IMAGE, STAT, ERRMSG])

Description. General reduction of elements on the current team of images.

Class. Collective subroutine.

Arguments.

A shall not be polymorphic. It shall have the same type and type parameters on all images in the current team. It shall not be a coindexed object. It is an INTENT(INOUT) argument. If A is a scalar, the computed value is the result of the reduction operation of applying OPERATOR to the values of A on all images in the current team. If A is an array it shall have the same shape on all images in the current team and each element of the computed value is equal to the result of the reduction operation of applying OPERATOR to all corresponding elements of A on all images in the current team.

OPERATOR shall be a pure function with two scalar, nonallocatable, nonpointer, nonoptional arguments of the same type and type parameters as A. Its result shall have the same type and type parameters as A. The arguments and result shall not be polymorphic. If one argument has the ASYNCHRONOUS, TARGET, or VALUE attribute, both shall have the attribute. OPERATOR shall implement a mathematically commutative and associative operation. OPERATOR shall be the same function on all images in the current team.

RESULT_IMAGE (optional) shall be an integer scalar. It is an INTENT(IN) argument. If it is present, it shall be present on all images in the current team, have the same value on all images in the current team, and that value shall be the image index of an image in the current team.

STAT (optional) shall be a noncoindexed integer scalar with a decimal exponent range of at least four. It is an INTENT(OUT) argument.

ERRMSG (optional) shall be a noncoindexed default character scalar. It is an INTENT(INOUT) argument.

If RESULT_IMAGE is not present, the computed value is assigned to A as if by intrinsic assignment on all images in the current team. If RESULT_IMAGE is present, the computed value is assigned to A as if by intrinsic assignment on image RESULT_IMAGE and A on all other images in the current team becomes undefined.

The computed value of a reduction operation over a set of values is the result of an iterative process. Each iteration involves the evaluation of `OPERATOR`(x,y) for x and y in the set, the removal of x and y from the set, and the addition of the value of `OPERATOR`(x,y) to the set. The process terminates when the set has only one element which is the value of the reduction.

The effect of the presence of the optional arguments STAT and ERRMSG is described in 8.3.

Example. The subroutine below demonstrates how to use CO_REDUCE to create a collective counterpart to the intrinsic function ALL:

```
SUBROUTINE co_all(boolean)
  LOGICAL, INTENT(INOUT) :: boolean
  CALL CO_REDUCE(boolean,both)
CONTAINS
  PURE FUNCTION both(lhs,rhs) RESULT(lhs_and_rhs)
    LOGICAL, INTENT(IN) :: lhs,rhs
    LOGICAL :: lhs_and_rhs
```



```
    lhs_and_rhs = lhs .AND. rhs
  END FUNCTION both
END SUBROUTINE co_all
```

8.4.14 CO_SUM (A [, RESULT_IMAGE, STAT, ERRMSG])

Description. Sum elements on the current team of images.

Class. Collective subroutine.

Arguments.

A shall be of numeric type. It shall have the same type and type parameters on all images in the current team. It shall not be a coindexed object. It is an INTENT(INOUT) argument. If it is a scalar, the computed value is equal to a processor-dependent approximation to the sum of the values of A on all images in the current team. If it is an array, it shall have the same shape on all images in the current team and each element of the computed value is equal to a processor-dependent approximation to the sum of all corresponding elements of A on the images in the current team. If RESULT_IMAGE is present, the computed value may depend on the image that RESULT_IMAGE specifies; otherwise, the computed value is the same on all images in the current team.

RESULT_IMAGE (optional) shall be an integer scalar. It is an INTENT(IN) argument. If it is present, it shall be present on all images in the current team, have the same value on all images in the current team, and that value shall be the image index of an image in the current team.

STAT (optional) shall be a noncoindexed integer scalar with a decimal exponent range of at least four. It is an INTENT(OUT) argument.

ERRMSG (optional) shall be a noncoindexed default character scalar. It is an INTENT(INOUT) argument.

If RESULT_IMAGE is not present, the computed value is assigned to A on all images in the current team. If RESULT_IMAGE is present, the computed value is assigned to A on image RESULT_IMAGE and A on all other images in the current team becomes undefined.

The effect of the presence of the optional arguments STAT and ERRMSG is described in 8.3.

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

8.4.15 EVENT_QUERY (EVENT, COUNT [, STAT])

Description. Query the count of an event variable.

Class. Atomic subroutine.

Arguments.

EVENT shall be a scalar of type EVENT_TYPE defined in the intrinsic module ISO_FORTRAN_ENV. It shall not be coindexed. It is an INTENT(IN) argument.

COUNT shall be an integer scalar with a decimal range no smaller than that of default integer. It is an INTENT(OUT) argument. If no error condition occurs, COUNT is assigned the value of the count of EVENT. Otherwise, it is assigned the value -1.

STAT (optional) shall be a noncoindexed integer scalar with a decimal exponent range of at least four. It is an INTENT(OUT) argument.

Example. If EVENT is an event variable for which there have been no successful posts or waits in preceding segments, and for which there are no posts or waits in an unordered segment, after the invocation

```
CALL EVENT_QUERY ( EVENT, COUNT )
```

the integer variable COUNT has the value 0. If there have been 10 successful posts to EVENT and 2 successful waits without an UNTIL_COUNT specification in preceding segments, and for which there are no posts or waits in an unordered segment, after the invocation

```
CALL EVENT_QUERY ( EVENT, COUNT )
```

COUNT has the value 8.

NOTE 8.5

Execution of EVENT_QUERY does not imply any synchronization.

8.4.16 FAILED_IMAGES ([TEAM, KIND])

Description. Indices of failed images.

Class. Transformational function.

Arguments.

TEAM (optional) shall be a scalar of type TEAM_TYPE defined in the intrinsic module ISO_FORTRAN_ENV. If TEAM is present its value shall represent the current or an ancestor team, and it specifies the team; otherwise, the team specified is the current team.

KIND (optional) shall be a scalar integer constant expression. Its value shall be the value of a kind type parameter for type INTEGER. The decimal range for integers of this kind shall be at least as large as for default integer.

Result Characteristics. Integer. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise, the kind type parameter is that of default integer type. The result is an array of rank one whose size is equal to the number of images in the specified team that are known by the invoking image to have failed.

Result Value. The elements of the result are the values of the image indices of the known failed images in the specified team, in numerically increasing order. If the executing image has previously executed an image control statement whose STAT= specifier assigned the value STAT_FAILED_IMAGE from the intrinsic module ISO_FORTRAN_ENV, or invoked a collective subroutine whose STAT argument was set to STAT_FAILED_IMAGE, and has not meanwhile entered or left a CHANGE TEAM construct, at least one image in the set of images participating in that image control statement or collective invocation shall be known to have failed.

Examples. If image 3 is the only image in the current team that is known by the invoking image to have failed, FAILED_IMAGES() has the value [3]. If there are no images in the current team that are known by the invoking image to have failed, FAILED_IMAGES() is a zero-sized array.

8.4.17 GET_TEAM ([LEVEL])

Description. Team value.

Class. Transformational function.

Argument. LEVEL (optional) shall be a scalar integer whose value shall be equal to one of the named constants INITIAL_TEAM, PARENT_TEAM, or CURRENT_TEAM defined in the intrinsic module ISO_FORTRAN_ENV.

Result Characteristics. Scalar of type TEAM_TYPE defined in the intrinsic module ISO_FORTRAN_ENV.

Result Value. The result is the value of a team variable for the current team if LEVEL is not present, LEVEL is present with the value CURRENT_TEAM, or the current team is the initial team. Otherwise, the result is the value of a team variable for the parent team if LEVEL is present with the value PARENT_TEAM, and for the initial team if LEVEL is present with the value INITIAL_TEAM.

Examples.

```
USE,INTRINSIC :: ISO_FORTRAN_ENV, ONLY: TEAM_TYPE
TYPE(TEAM_TYPE) :: WORLD_TEAM, TEAM2

! Define a team variable representing the initial team
WORLD_TEAM = GET_TEAM()
END

SUBROUTINE TT (A)
USE,INTRINSIC :: ISO_FORTRAN_ENV, ONLY: TEAM_TYPE
REAL A[*]
TYPE(TEAM_TYPE) :: NEW_TEAM, PARENT_TEAM

... ! Form NEW_TEAM

PARENT_TEAM = GET_TEAM()

CHANGE TEAM(NEW_TEAM)

   ! Reference image 1 in parent's team
   A [1,TEAM=PARENT_TEAM] = 4.2

   ! Reference image 1 in current team
   A [1] = 9.0
END TEAM
END SUBROUTINE TT
```

8.4.18 IMAGE_STATUS (IMAGE, [TEAM])

Description. Status of images.

Class. Elemental function.

Arguments.

IMAGE shall be of type integer.

TEAM (optional) shall be a scalar of type TEAM_TYPE defined in the intrinsic module ISO_FORTRAN_ENV. If TEAM is present its value shall represent the current or an ancestor team, and it specifies the team; otherwise, the team specified is the current team.

Result Characteristics. Default integer.

Result Value. The result value is STAT_FAILED_IMAGE if the specified image has failed, STAT_STOPPED_IMAGE if that image has initiated normal termination, a positive processor-dependent value different from STAT_FAILED_IMAGE or STAT_STOPPED_IMAGE if some other error has occurred for that image, and zero otherwise.

Example. If image 3 of the current team has failed, IMAGE_STATUS (3) has the value STAT_FAILED_IMAGE.

8.4.19 STOPPED_IMAGES ([TEAM, KIND])

Description. Indices of stopped images.

Class. Transformational function.

Arguments.

TEAM (optional) shall be a scalar of type TEAM_TYPE defined in the intrinsic module ISO_FORTRAN_ENV. If TEAM is present its value shall represent the current or an ancestor team, and it specifies the team; otherwise, the team specified is the current team.

KIND (optional) shall be a scalar integer constant expression. Its value shall be the value of a kind type parameter for type INTEGER. The decimal range for integers of this kind shall be at least as large as for default integer.

Result Characteristics. Integer. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise, the kind type parameter is that of default integer type. The result is an array of rank one whose size is equal to the number of images in the specified team that have initiated normal termination.

Result Value. The elements of the result are the values of the indices of the images that have initiated normal termination in the specified team, in numerically increasing order. If the executing image has previously executed an image control statement whose STAT= specifier assigned the value STAT_STOPPED_IMAGE from the intrinsic module ISO_FORTRAN_ENV or invoked a collective subroutine whose STAT argument was set to STAT_STOPPED_IMAGE, and has not meanwhile entered or left a CHANGE TEAM construct, at least one of the images participating in that image control statement or collective invocation shall have initiated normal termination.

Examples. If image 3 is the only image in the current team that has initiated normal termination, STOPPED_IMAGES() has the value [3]. If there are no images in the current team that have initiated normal termination, STOPPED_IMAGES() is a zero-sized array.

8.4.20 TEAM_NUMBER ([TEAM])

Description. Team number.

Class. Transformational function.

Argument. TEAM (optional) shall be a scalar of type TEAM_TYPE defined in the intrinsic module ISO_FORTRAN_ENV. If TEAM is present its value shall represent the current or an ancestor team, and it specifies the team; otherwise, the team specified is the current team.

Result Characteristics. Default integer scalar.

Result Value. If the specified team is the initial team, the result is -1; otherwise, the result value is the team number identifying the team of the invoking image within the specified team.

Example. The following code illustrates the use of TEAM_NUMBER to control which code is executed.

```
TYPE(TEAM_TYPE) :: ODD_EVEN
    :
ME = THIS_IMAGE()
FORM TEAM ( 2-MOD(ME,2), ODD_EVEN )
CHANGE TEAM (ODD_EVEN)
  SELECT CASE (TEAM_NUMBER())
  CASE (1)
     : ! Code for images with odd image indices in parent team
  CASE (2)
     :  ! Code for images with even image indices in parent team
  END SELECT
END TEAM
```

8.5 Modified intrinsic procedures

8.5.1 ATOMIC_DEFINE and ATOMIC_REF

The intrinsic subroutines ATOMIC_DEFINE and ATOMIC_REF in ISO/IEC 1539-1:2010 are modified to take account of the possibility that an ATOM argument is located on a failed image and to add the optional argument STAT.

The STAT argument shall be a noncoindexed scalar of type integer with a decimal range of at least four. It is an INTENT(OUT) argument.

8.5.2 IMAGE_INDEX

The intrinsic function IMAGE_INDEX in ISO/IEC 1539-1:2010 is modified by adding two additional versions that specify the team with the argument TEAM or the argument TEAM_NUMBER, and a modified result if either of these versions is invoked.

The TEAM argument shall be a scalar of type TEAM_TYPE defined in the intrinsic module ISO_FORTRAN_ENV. Its value shall represent the current or an ancestor team.

The TEAM_NUMBER argument shall be a positive integer scalar. If the current team is the initial team, its value is ignored. Otherwise, its value shall be that of a team number for a team that was formed by execution of a FORM TEAM statement for the current team.

The result is the image index in the specified team.

8.5.3 MOVE_ALLOC

The intrinsic subroutine MOVE_ALLOC in ISO/IEC 1539-1:2010, as modified by ISO/IEC 1539-1:2010/Cor 2:2013, is extended to take account of the possibility of failed images and to add two optional arguments, STAT and ERRMSG, and modified semantics of execution.

The STAT argument shall be a noncoindexed integer scalar with a decimal exponent range of at least four. It is an INTENT(OUT) argument.

The ERRMSG argument shall be a noncoindexed default character scalar. It is an INTENT(INOUT) argument.

If the execution is successful

(1) The allocation status of TO becomes unallocated if FROM is unallocated on entry to MOVE_ALLOC. Otherwise, TO becomes allocated with dynamic type, type parameters, array bounds, array cobounds, and value identical to those that FROM had on entry to MOVE_ALLOC.
(2) If TO has the TARGET attribute, any pointer associated with FROM on entry to MOVE_ALLOC becomes correspondingly associated with TO. If TO does not have the TARGET attribute, the pointer association status of any pointer associated with FROM on entry becomes undefined.
(3) The allocation status of FROM becomes unallocated.

When a reference to MOVE_ALLOC is executed for which the FROM argument is a coarray, there is an implicit synchronization of all active images of the current team. On each active image, execution of the segment (8.5.2 of ISO/IEC 1539-1:2010) following the CALL statement is delayed until all other active images of the current team have executed the same statement the same number of times since execution last began in this team.

If the STAT argument appears and execution is successful on all images, the argument is assigned the value zero; if a failed image is detected and execution is otherwise successful, the STAT= specifier is assigned the value STAT_FAILED_IMAGE in the intrinsic module ISO_FORTRAN_ENV.

If the STAT argument appears and an error condition occurs, the argument is assigned the value STAT_STOPPED_IMAGE in the intrinsic module ISO_FORTRAN_ENV if the reason is that a successful execution would

have involved an interaction with an image that has initiated termination; otherwise, the value is a processor-dependent positive value that is different from the value of STAT_STOPPED_IMAGE or STAT_FAILED_IMAGE.

If the STAT argument does not appear and an error condition occurs or an image involved in execution of the statement has failed, error termination is initiated.

If the ERRMSG argument is present and an error condition occurs, the processor shall assign an explanatory message to the argument. If no error condition occurs, the processor shall not change the value of the argument.

8.5.4 NUM_IMAGES

The intrinsic function NUM_IMAGES in ISO/IEC 1539-1:2010 is modified by adding two versions that specify the team with the argument TEAM or the argument TEAM_NUMBER, and a modified result if either of these versions is invoked.

The TEAM argument shall be a scalar of type TEAM_TYPE defined in the intrinsic module ISO_FORTRAN_ENV. Its value shall represent the current or an ancestor team.

The TEAM_NUMBER argument shall be a positive integer scalar. If the current team is the initial team, its value is ignored. Otherwise, its value shall be that of a team number for a team that was formed by execution of a FORM TEAM statement for the current team.

The result is the number of images in the specified team.

8.5.5 THIS_IMAGE

The intrinsic function THIS_IMAGE in ISO/IEC 1539-1:2010, as modified by ISO/IEC 1539-1:2010/Cor 1:2012, is modified by adding an optional argument TEAM and a modified result if TEAM is present.

The TEAM argument shall be a scalar of type TEAM_TYPE defined in the intrinsic module ISO_FORTRAN_ENV. Its value shall represent the current or an ancestor team. If TEAM is present, the result is the image index that the invoking image has in the team specified by the value of TEAM; otherwise, the result value is the image index of the invoking image in the current team.

9 Required editorial changes to ISO/IEC 1539-1:2010(E)

9.1 General

The following editorial changes, if implemented, would provide the facilities described in foregoing clauses of this Technical Specification. Descriptions of how and where to place the new material are enclosed in braces {}. Edits to different places within the same clause are separated by horizontal lines.

In the edits, except as specified otherwise by the editorial instructions, underwave (underwave) and strike-out (~~strike-out~~) are used to indicate insertion and deletion of text.

9.2 Edits to Introduction

{In paragraph 1 of the Introduction}

After "informally known as Fortran 2008, plus the facilities defined in ISO/IEC TS 29113:2012" add "and ISO/IEC TS 18508:2015".

{In the Introduction and after the paragraph added by ISO/IEC TS 29113:2012, insert new paragraph}

• Features previously described in ISO/IEC TS 18508:2015:

Optional STAT= and ERRMSG= specifiers are added to the CRITICAL statement. Optional arguments are added to the intrinsic procedures ATOMIC_DEFINE, ATOMIC_REF, IMAGE_INDEX, MOVE_ALLOC, NUM_IMAGES, and THIS_IMAGE. Extensions of image selector syntax permit designation of coarrays across team boundaries and optional STAT= specifiers. The STOPPED_IMAGES intrinsic procedure provides the indices of stopped images. The teams facility uses the CHANGE TEAM construct, TEAM_TYPE derived type, FORM TEAM and SYNC TEAM statements, and the GET_TEAM and TEAM_NUMBER intrinsic procedures, to provide a capability for a subset of the images of the program to act as if it consists of all images for the purposes of image index values, coarray allocations, and synchronization. The collective subroutines CO_BROADCAST, CO_MAX, CO_MIN, CO_REDUCE, and CO_SUM perform computations based on values on all active images of the current team, offering the possibility of efficient execution of reduction operations. The additional atomic subroutines ATOMIC_ADD, ATOMIC_AND, ATOMIC_CAS, ATOMIC_FETCH_ADD, ATOMIC_FETCH_AND, ATOMIC_FETCH_OR, ATOMIC_FETCH_XOR, ATOMIC_OR, and ATOMIC_XOR perform integer addition, compare and swap, and bitwise computations. The event facility uses the EVENT POST and EVENT WAIT statements, the EVENT_TYPE derived type, and the EVENT_QUERY intrinsic procedure to allow one-sided ordering of execution segments. The FAIL TEAM statement, the FAILED_IMAGES and IMAGE_STATUS intrinsic procedures, and the defined constant STAT_FAILED_IMAGE provide support for continued execution after one or more images have failed.

9.3 Edits to clause 1

{In 1.3 Terms and definitions, insert new terms as follows}

1.3.8a
asynchronous progress
ability of an image to reference or define a coarray on another image without waiting for that image to execute any particular statement or class of statement

1.3.68a
established coarray

⟨in a team⟩ coarray that is accessible using a coindexed designator (8.1.4a)

1.3.83a
active image
image that has not failed or initiated termination (2.3.6)

1.3.83b
failed image
image that has ceased participating in program execution but has not initiated termination (2.3.6)

1.3.83d
stopped image
image that has initiated normal termination (2.3.6)

1.3.145a
team
set of images that can readily execute independently of other images (2.3.4)

1.3.145a.1
current team
⟨of an image⟩ team specified by the most recently executed CHANGE TEAM statement of an active CHANGE TEAM construct, or initial team if no CHANGE TEAM construct is active (2.3.4)

1.3.145a.2
initial team
team, consisting of all images, that began execution of the program (2.3.4)

1.3.145a.3
parent team
⟨of a team⟩ current team during the execution of the CHANGE TEAM statement that established the team (2.3.4)

1.3.145a.4
team number
integer value identifying a team within its parent team (2.3.4)

1.3.154.1-
event variable
scalar variable of type EVENT_TYPE (13.8.2.8a) from the intrinsic module ISO_FORTRAN_ENV

1.3.154.3
team variable
scalar variable of type TEAM_TYPE (13.8.2.26) from the intrinsic module ISO_FORTRAN_ENV

9.4 Edits to clause 2

{In 2.1 High level syntax, Add new construct and statements into the syntax list as follows: In R213 *executable-construct* insert alphabetically "*change-team-construct*"; in R214 *action-stmt* insert alphabetically "*event-post-stmt*", "*event-wait-stmt*", "*fail-image-stmt*", "*form-team-stmt*", and "*sync-team-stmt*".

{In 2.3.4 Program execution, after the first paragraph, insert 5.1, paragraphs 1 and 2, of this Technical Specification with the following changes: In the first paragraph delete "in ISO/IEC 1539-1:2010" following "R624" and insert "(8.5.2c)" following "FORM TEAM statement". In the second paragraph insert "(8.1.4a)" following "CHANGE TEAM construct". }

{After 2.3.5 Execution sequence, insert a new subclause "2.3.6 Image execution states" consisting of five paragraphs, with the third, fourth, and fifth paragraphs consisting of the first three paragraphs of 6.1 Introduction

and Note 6.1 of this Technical Specification after changing "(6.4)" in the second paragraph to "13.8.2.21b", and the first and second paragraphs as follows.}

A stopped image is an image that has initiated termination of execution.

An active image is an image that is not a failed image or a stopped image.

{In 2.4.7 Coarray, after the first paragraph, insert 5.1 paragraph 3 of this Technical Specification.}

{In 2.4.7 Coarray, edit the second paragraph as follows.}

For each coarray on an image in a team, there is a corresponding coarray with the same type, type parameters, and bounds on every other image in that team.

{In 2.4.7 Coarray, edit the first sentence of the third paragraph as follows.}

The set of corresponding coarrays on all images in a team is arranged in a rectangular pattern.

9.5 Edits to clause 4

{In 4.5.2.1 Syntax, edit constraint C433 as follows}

C433 (R425) If EXTENDS appears and the type being defined has ~~an ultimate~~ a component of type EVENT_TYPE or LOCK_TYPE from the intrinsic module ISO_FORTRAN_ENV at any level of nonpointer component selection, its parent type shall have ~~an ultimate~~ a component at some level of nonpointer component selection of type EVENT_TYPE or LOCK_TYPE, respectively.

{In 4.5.6.2 The finalization process, add to the end of NOTE 4.48}

in the current team

9.6 Edits to clause 6

{In 6.6 Image selectors, replace R624 with}

R624 *image-selector* **is** *lbracket cosubscript-list* ■
■ [, *team-identifier*] [, STAT = *stat-variable*] *rbracket*

R624a *team-identifier* **is** TEAM_NUMBER = *scalar-int-expr*
or TEAM = *team-variable*

C627a (R624) *stat-variable* shall not be a coindexed object.

{In 6.6 Image selectors, edit the last sentence of the second paragraph as follows.}

An image selector shall specify an image index value that is not greater than the number of images in the team specified by a *team-identifier* if it appears, or in the current team otherwise.

{In 6.6 Image selectors, after paragraph 2 insert the four paragraphs following C509 in 5.4 of this Technical Specification with the following change: following "FORM TEAM statement" insert "(8.5.2c)" }

{In 6.7.1.2, Execution of an ALLOCATE statement, edit paragraphs 3 and 4 as follows}

If an *allocation* specifies a coarray, its dynamic type and the values of corresponding type parameters shall be the same on every active image in the current team. The values of corresponding bounds and corresponding cobounds shall be the same on ~~every image~~ these images. If the coarray is a dummy argument, its ultimate

argument (12.5.2.3) shall be the same coarray on ~~every image~~ these images.

When an ALLOCATE statement is executed for which an *allocate-object* is a coarray, there is an implicit synchronization of all active images in the current team. On ~~each image~~ these images, execution of the segment (8.5.2) following the statement is delayed until all other active images in the current team have executed the same statement the same number of times since execution last began in this team.

{In 6.7.3.2, Deallocation of allocatable variables, edit paragraphs 11 and 12 as follows}

When a DEALLOCATE statement is executed for which an *allocate-object* is a coarray, there is an implicit synchronization of all active images in the current team. On ~~each image~~ these images, execution of the segment (8.5.2) following the statement is delayed until all other active images in the current team have executed the same statement the same number of times since execution last began in this team. If the coarray is a dummy argument, its ultimate argument (12.5.2.3) shall be the same coarray on ~~every image~~ these images.

There is also an implicit synchronization of all active images in the current team in association with the deallocation of a coarray or coarray subcomponent caused by the execution of a RETURN or END statement or the termination of a BLOCK construct.

{In 6.7.4 STAT=specifier, edit paragraph 2 as follows}

If the STAT= specifier appears, successful execution of the ALLOCATE or DEALLOCATE statement on all images in the current team causes the *stat-variable* to become defined with the value zero; otherwise, if a failed image is detected and execution is otherwise successful, the STAT= specifier is assigned the value of the named constant STAT_FAILED_IMAGE in the intrinsic module ISO_FORTRAN_ENV (13.8.2).

{In 6.7.4 STAT= specifier, para 3, replace the text to the bullet list with}

If the STAT= specifier appears in an ALLOCATE or DEALLOCATE statement with a coarray *allocate-object* and an error condition occurs, the specified variable is assigned a positive value. The value shall be that of the named constant STAT_STOPPED_IMAGE in the intrinsic module ISO_FORTRAN_ENV if the reason is that a successful execution would have involved an interaction with an image that has initiated termination; otherwise, the value is a processor-dependent positive value that is different from the values of the named constants STAT_STOPPED_IMAGE or STAT_FAILED_IMAGE in the intrinsic module ISO_FORTRAN_ENV. In all of these cases, each *allocate-object* has a processor-dependent allocation status:

{At the end of 6.7.4 STAT= specifier, append the following new paragraph}

If the STAT argument does not appear and an error condition occurs or an image involved in execution of the statement has failed, error termination is initiated.

9.7 Edits to clause 8

{In 8.1.1 General, paragraph 1, following the BLOCK construct entry in the list of constructs insert}

• CHANGE TEAM construct;

{Following 8.1.4 BLOCK construct insert 5.3 CHANGE TEAM construct from this Technical Specification as 8.1.4a, with rule, constraint, and Note numbers modified, the reference "(5.2)" in C506 changed to "(13.8.2.26)", and in the third paragraph following C508, delete "of ISO/IEC 1539-1:2010". }

{In 8.1.5 CRITICAL construct: In para 1, line 1, after "one image" add "in the current team". In para 1, at the end of R811, add "[*sync-stat-list*]". In para 2, line 1, at the end of the sentence, add "or the executing image fails". After para 2 add the paragraph following R811 in 6.3 CRITICAL construct of this Technical Specification. In para 3, line 1, after "other image" add "in the current team". After para 3, add a new para: "The effect of a STAT= or an ERRMSG= specifier in a CRITICAL statement is explained in 8.5.7.".}

{Following 8.4 STOP and ERROR STOP statements, insert 6.2 FAIL IMAGE statement from this Technical Specification as 8.4a, with rule and Note numbers modified.}

{In 8.5.1 Image control statements, paragraph 2, insert extra bullet points following the CRITICAL and END CRITICAL line}

- CHANGE TEAM and END TEAM;
- EVENT POST and EVENT WAIT;
- FORM TEAM;
- SYNC TEAM;

{In 8.5.1 Image control statements, edit paragraph 3 as follows}

All image control statements except CRITICAL, END CRITICAL, EVENT POST, EVENT WAIT, FORM TEAM, LOCK, and UNLOCK include the effect of executing a SYNC MEMORY statement (8.5.5).

{In 8.5.2 Segments, after the first sentence of paragraph 3, insert the following }

A coarray that is of type EVENT_TYPE (13.8.2.8a) may be referenced or defined during the execution of a segment that is unordered relative to the execution of another segment in which that coarray of type EVENT_TYPE is defined.

{In 8.5.2 Segments, edit the first sentence of NOTE 8.34 as follows}

The model upon which the interpretation of a program is based is that there is a permanent memory location for each coarray and that all images on which it is established can access it.

{Following 8.5.2 Segments insert 7.3 EVENT POST statement from this Technical Specification as 8.5.2a, with rule and constraint numbers modified, and change the "(7.2)" in C704 to "(13.8.2.8a)". }

{Following 8.5.2 Segments insert 7.4 EVENT WAIT statement from this Technical Specification as 8.5.2b, with rule and constraint numbers modified.}

{Following 8.5.2 Segments insert 5.5 FORM TEAM statement from this Technical Specification as 8.5.2c, with rule and Note numbers modified.}

{In 8.5.3 SYNC ALL statement, edit paragraph 2 as follows}

Execution of a SYNC ALL statement performs a synchronization of all active images in the current team. Execution on an image M, of the segment following the SYNC ALL statement is delayed until each other active image in the current team has executed a SYNC ALL statement as many times as has image M since execution last began in this team. The segments that executed before the SYNC ALL statement on an image precede the segments that execute after the SYNC ALL statement on another image.

{In 8.5.4 SYNC IMAGES, edit paragraphs 1 through 4 as follows}

If *image-set* is an array expression, the value of each element shall be positive and not greater than the number of images in the current team, and there shall be no repeated values.

If *image-set* is a scalar expression, its value shall be positive and not greater than the number of images in the current team.

An *image-set* that is an asterisk specifies all images in the current team.

Execution of a SYNC IMAGES statement performs a synchronization of the image with each of the other active

images in the *image-set*. Executions of SYNC IMAGES statements on images M and T correspond if the number of times image M has executed a SYNC IMAGES statement in the current team with T in its image set since execution last began in this team is the same as the number of times image T has executed a SYNC IMAGES statement in the current team with M in its image set since execution last began in this team. The segments that executed before the SYNC IMAGES statement on either image precede the segments that execute after the corresponding SYNC IMAGES statement on the other image.

{Following 8.5.5 SYNC MEMORY statement, insert 5.6 SYNC TEAM statement from this Technical Specification as 8.5.5a, with the rule number modified.}

{In 8.5.6 LOCK and UNLOCK statements: in para 1, change "image and" to "image, that image has not failed, and the lock variable"; in para 4, add new second sentence "If an image fails after locking and before unlocking a lock variable, the variable becomes unlocked."}

{In 8.5.7 STAT= and ERRMSG= specifiers in image control statements replace paragraphs 1 and 2 by}

If the STAT= specifier appears in a SYNC MEMORY statement and its execution is successful, the specified variable is assigned the value zero. If the STAT= specifier appears in a CHANGE TEAM, END TEAM, EVENT POST, EVENT WAIT, FORM TEAM, LOCK, SYNC ALL, SYNC IMAGES, SYNC TEAM, or UNLOCK statement and its execution is successful on the involved images, the specified variable is assigned the value zero; otherwise, if a failed image is detected and execution is otherwise successful, the STAT= specifier is assigned the value of the named constant STAT_FAILED_IMAGE in the intrinsic module ISO_FORTRAN_ENV (13.8.2).

If the STAT= specifier appears in a CHANGE TEAM, END TEAM, EVENT POST, EVENT WAIT, FORM TEAM, LOCK, SYNC ALL, SYNC IMAGES, SYNC MEMORY, SYNC TEAM, or UNLOCK statement and an error condition occurs, the specified variable is assigned a positive value. The value shall be the named constant STAT_STOPPED_IMAGE in the intrinsic module ISO_FORTRAN_ENV if the reason is that a successful execution would have involved an interaction with an image that has initiated termination; otherwise, the value is a processor-dependent positive value that is different from the values of the named constants STAT_STOPPED_IMAGE or STAT_FAILED_IMAGE in the intrinsic module ISO_FORTRAN_ENV.

The set of images involved in execution of an END TEAM, FORM TEAM, or SYNC ALL statement is that of the current team. The set of images involved in execution of a CHANGE TEAM or SYNC TEAM statement is that of the team specified by the value of the specified *team-variable*. The set of images involved in execution of a SYNC IMAGES statement is the union of its *image-set* and the executing image. The images involved in execution of a LOCK or UNLOCK statement are the ones on which the referenced lock variable is located and the executing image. The images involved in execution of an EVENT POST statement are the ones on which the referenced event variable is located and the executing image.

After execution of an image control statement with a STAT= specifier that is not a SYNC MEMORY statement, all failed images involved in the statement shall be known by the executing image to have failed.

If the STAT= specifier appears in a CHANGE TEAM, END TEAM, SYNC ALL, SYNC IMAGES, or SYNC TEAM statement and an error condition occurs, the effect is the same as that of executing the SYNC MEMORY statement, except for defining the STAT= variable.

{In 8.5.7 STAT= and ERRMSG= specifiers in image control statements replace paragraphs 4 and 5 by the following paragraphs, then add the final paragraph of 6.3 CRITICAL construct of this Technical Specification.}

If the STAT= specifier does not appear in a CHANGE TEAM, END TEAM, EVENT POST, EVENT WAIT, FORM TEAM, LOCK, SYNC ALL, SYNC IMAGES, SYNC MEMORY, SYNC TEAM, or UNLOCK statement and its execution is not successful or an image involved in execution of the statement has failed, error termination is initiated.

If an ERRMSG= specifier appears in a CHANGE TEAM, END TEAM, EVENT POST, EVENT WAIT, FORM TEAM, LOCK, SYNC ALL, SYNC IMAGES, SYNC MEMORY, SYNC TEAM, or UNLOCK statement and its execution is not successful, the processor shall assign an explanatory message to the specified variable. If the

execution is successful, the processor shall not change the value of the variable.

9.8 Edits to clause 9

{In 9.5.1, Referring to a file, edit the first sentence of paragraph 4 as follows}

In a READ statement, an *io-unit* that is an asterisk identifies an external unit that is preconnected for sequential formatted input on image 1 in the initial team only (9.6.4.3).

9.9 Edits to clause 13

{In 13.1 Classes of intrinsic procedures, edit paragraph 1 as follows}

Intrinsic procedures are divided into ~~seven~~ eight classes: inquiry functions, elemental functions, transformational functions, elemental subroutines, pure subroutines, atomic subroutines, collective subroutines, and (impure) subroutines.

{In 13.1 Classes of intrinsic procedures, replace paragraph 3 by paragraphs 1 through 4 and NOTES 8.1 and 8.2 of 8.2 Atomic subroutines of this Technical Specification, with these changes: Delete "of ISO/IEC 1539-1:2010" and renumber the NOTES.}

{In 13.1 Classes of intrinsic procedures, insert the contents of 8.3 Collective subroutines of this Technical Specification after paragraph 3 and Note 13.1, with these changes: In paragraph 2 of 8.3, delete "of ISO/IEC 1539-1:2010"; In paragraph 5 of 8.3, add "(13.8.2)" after the first "ISO_FORTRAN_ENV".}

{In 13.5 Standard generic intrinsic procedures, paragraph 2 after the line "A indicates ... atomic subroutine" insert a new line}

C indicates that the procedure is a collective subroutine.

{In 13.5 Standard generic intrinsic procedures, Table 13.1, insert new entries into the table, alphabetically}

ATOMIC_ADD	(ATOM, VALUE [, STAT])	A	Atomic add operation.
ATOMIC_AND	(ATOM, VALUE [, STAT])	A	Atomic bitwise AND operation.
ATOMIC_CAS	(ATOM, OLD, COMPARE, NEW [, STAT])	A	Atomic compare and swap.
ATOMIC_FETCH_ADD	(ATOM, VALUE, OLD [,STAT])	A	Atomic fetch and add operation.
ATOMIC_FETCH_AND	(ATOM, VALUE, OLD [,STAT])	A	Atomic fetch and bitwise AND operation.
ATOMIC_FETCH_OR	(ATOM, VALUE, OLD [,STAT])	A	Atomic fetch and bitwise OR operation.
ATOMIC_FETCH_XOR	(ATOM, VALUE, OLD [,STAT])	A	Atomic fetch and bitwise exclusive OR operation.
ATOMIC_OR	(ATOM, VALUE [, STAT])	A	Atomic bitwise OR operation.
ATOMIC_XOR	(ATOM, VALUE [, STAT])	A	Atomic bitwise exclusive OR operation.
CO_BROADCAST	(A, SOURCE_IMAGE [, STAT, ERRMSG])	C	Copy a value to all images of the current team.
CO_MAX	(A [, RESULT_IMAGE, STAT, ERRMSG])	C	Compute maximum of elements across images.
CO_MIN	(A [, RESULT_IMAGE, STAT, ERRMSG])	C	Compute minimum of elements across images.
CO_REDUCE	(A, OPERATOR [, RESULT_IMAGE, STAT, ERRMSG])	C	General reduction of elements across images.

CO_SUM	(A [, RESULT_IMAGE, STAT, ERRMSG])	C	Sum elements across images.
EVENT_QUERY	(EVENT, COUNT [, STAT])	A	Count of an event variable.
FAILED_IMAGES	([TEAM, KIND])	T	Indices of failed images.
GET_TEAM	([LEVEL])	T	Team value.
IMAGE_STATUS	(IMAGE [, TEAM])	E	Status of images.
STOPPED_IMAGES	([TEAM, KIND])	T	Indices of stopped images.
TEAM_NUMBER	([TEAM])	T	Team number.

{In 13.5 Standard generic intrinsic procedures, Table 13.1, edit the entries for ATOMIC_DEFINE, ATOMIC_REF, IMAGE_INDEX, MOVE_ALLOC, NUM_IMAGES, and THIS_IMAGE, as modified by ISO/IEC 1539-1:2010/Cor 1:2012, as follows}

ATOMIC_DEFINE	(ATOM, VALUE [, STAT])	A	Define a variable atomically.
ATOMIC_REF	(VALUE, ATOM [, STAT])	A	Reference a variable atomically.
IMAGE_INDEX	(COARRAY, SUB) or (COARRAY, SUB, TEAM) or (COARRAY, SUB, TEAM_NUMBER)	I	Image index from cosubscripts.
MOVE_ALLOC	(FROM, TO [, STAT, ERRMSG])	PS	Move an allocation.
NUM_IMAGES	( ) or (TEAM) or (TEAM_NUMBER)	T	Number of images.
THIS_IMAGE	([TEAM])	T	Index of the invoking image.
THIS_IMAGE	(COARRAY [, TEAM]) or (COARRAY, DIM [, TEAM])	T	Cosubscript(s) for this image.

{In 13.5, Standard generic intrinsic procedures, paragraph 3, insert "in the initial team" after "image 1"}

{In 13.7 Specifications of the standard intrinsic procedures, insert subclauses 8.4.1 through 8.4.20 of this Technical Specification in order alphabetically, with subclause numbers adjusted accordingly.}

{In 13.7.20 ATOMIC_DEFINE, edit the subclause title as follows}

13.7.20 ATOMIC_DEFINE (ATOM, VALUE [,STAT])

{In 13.7.20 ATOMIC_DEFINE, add the argument description as follows}

STAT (optional) shall be a noncoindexed integer scalar with a decimal range of at least four. It is an INTENT(OUT) argument.

{In 13.7.21 ATOMIC_REF, edit the subclause title as follows}

13.7.21 ATOMIC_REF (VALUE, ATOM [,STAT])

{In 13.7.21 ATOMIC_REF, add the argument description and a paragraph as follows}

STAT (optional) shall be a noncoindexed integer scalar with a decimal range of at least four. It is an INTENT(OUT) argument.

If an error condition occurs, the VALUE argument becomes undefined.

{In 13.7.79 IMAGE_INDEX, edit the subclause title as follows}

13.7.79 IMAGE_INDEX (COARRAY, SUB) or IMAGE_INDEX (COARRAY, SUB, TEAM) or IMAGE_INDEX (COARRAY, SUB, TEAM_NUMBER)

{In 13.7.79 IMAGE_INDEX, edit the COARRAY argument description as follows}

COARRAY shall be a coarray of any type. If the function is invoked with a TEAM_NUMBER argument, it shall be established in an ancestor of the specified team. Otherwise, it shall be established in the specified team.

{In 13.7.79 IMAGE_INDEX, add the arguments descriptions as follows}

TEAM shall be a scalar of type TEAM_TYPE defined in the intrinsic module ISO_FORTRAN_ENV. Its value shall represent the current or an ancestor team.

TEAM_NUMBER shall be a positive integer scalar. If the current team is the initial team, its value is ignored. Otherwise, its value shall be that of a team number for a team that was formed by execution of a FORM TEAM statement for the current team.

If TEAM appears, it specifies the team. Otherwise, the team specified is the current team.

{In 13.7.79 IMAGE_INDEX, replace paragraph 5 with}

Result Value. If the value of SUB is a valid sequence of cosubscripts for COARRAY in the specified team, the result is the index of the corresponding image in that team. Otherwise, the result is zero.

{In 13.7.118 MOVE_ALLOC, edit the subclause title as follows}

13.7.118 MOVE_ALLOC (FROM, TO [, STAT, ERRMSG])

{In 13.7.118 MOVE_ALLOC, add the arguments descriptions as follows}

STAT (optional) shall be a noncoindexed default integer scalar. It is an INTENT(OUT) argument.

ERRMSG (optional) shall be a noncoindexed default character scalar. It is an INTENT(INOUT) argument.

{In 13.7.118 MOVE_ALLOC, replace paragraphs 4 through 6 and the paragraph that was added by ISO/IEC 1539-1:2010/Cor 2:2013 by paragraphs 4 through 8 of 8.5.3 of this Technical Specification, deleting "of ISO/IEC 1539-1:2010" in paragraph 5.}

{In 13.7.126 NUM_IMAGES, edit the subclause title as follows}

13.7.126 NUM_IMAGES ( ) or NUM_IMAGES (TEAM) or NUM_IMAGES (TEAM_NUMBER)

{In 13.7.126 NUM_IMAGES, replace paragraph 3 with}

Arguments.

TEAM shall be a scalar of type TEAM_TYPE defined in the intrinsic module ISO_FORTRAN_ENV. Its value shall represent the current or an ancestor team.

TEAM_NUMBER shall be a positive integer scalar. If the current team is the initial team, its value is ignored. Otherwise, its value shall be that of a team number for a team that was formed by execution of a FORM TEAM statement for the current team.

If TEAM appears, it specifies the team. Otherwise, the team specified is the current team.

{In 13.7.126 NUM_IMAGES, replace paragraph 5 with}

Result Value. The number of images in the team specified.

{In 13.7.165, as modified by ISO/IEC 1539-1:2010/Cor 1:2012, THIS_IMAGE ( ) or THIS_IMAGE (COARRAY) or THIS_IMAGE (COARRAY, DIM) edit the subclause title as follows }

13.7.165 THIS_IMAGE ([TEAM]) or THIS_IMAGE (COARRAY [, TEAM]) or THIS_IMAGE (COARRAY, DIM [, TEAM])

{In 13.7.165, as modified by ISO/IEC 1539-1:2010/Cor 1:2012, THIS_IMAGE ( ) or THIS_IMAGE (COARRAY) or THIS_IMAGE (COARRAY, DIM) insert a new argument at the end of paragraph 3 }

TEAM (optional) shall be a scalar of type TEAM_TYPE defined in the intrinsic module ISO_FORTRAN_ENV. Its value shall represent the current or an ancestor team. If COARRAY appears, it shall be established in the team specified by the value of TEAM.

{In 13.7.165, as modified by ISO/IEC 1539-1:2010/Cor 1:2012, THIS_IMAGE ( ) or THIS_IMAGE (COARRAY) or THIS_IMAGE (COARRAY, DIM) at the end of paragraph 5 add }

Case (iv): The result of THIS_IMAGE (TEAM) is a scalar with a value equal to the index of the invoking image in the team specified by the value of TEAM.

Case (v): The result of THIS_IMAGE (COARRAY, TEAM) is the sequence of cosubscript values for COARRAY that would specify the invoking image in the team specified by the value of TEAM.

Case (vi): The result of THIS_IMAGE (COARRAY, DIM, TEAM) is the value of cosubscript DIM in the sequence of cosubscript values for COARRAY that would specify the invoking image in the team specified by the value of TEAM.

{In 13.7.172 UCOBOUND, edit the Result Value as follows.}

The final upper cobound is the final cosubscript in the cosubscript list for the coarray that selects the image with index ~~NUM_IMAGES( )~~ equal to the number of images in the current team when the coarray was established.

{In 13.8.2 The ISO_FORTRAN_ENV intrinsic module, insert a new subclause}

13.8.2.7a CURRENT_TEAM

The value of the default integer scalar constant CURRENT_TEAM identifies the current team in an invocation of the function GET_TEAM.

{In 13.8.2 The ISO_FORTRAN_ENV intrinsic module, insert a new subclause 13.8.2.8a consisting of subclause 7.2 EVENT_TYPE of this Technical Specification, but omitting the final sentence of the first paragraph and the fourth sentence of the second paragraph.}

{In 13.8.2 The ISO_FORTRAN_ENV intrinsic module, insert a new subclause}

13.8.2.9a INITIAL_TEAM

The value of the default integer scalar constant INITIAL_TEAM identifies the initial team in an invocation of the function GET_TEAM.

{In 13.8.2 The ISO_FORTRAN_ENV intrinsic module, insert a new subclause}

13.8.2.19a PARENT_TEAM

The value of the default integer scalar constant PARENT_TEAM identifies the parent team in an invocation of the function GET_TEAM.

{In 13.8.2 The ISO_FORTRAN_ENV intrinsic module, insert a new subclause 13.8.2.21b consisting of subclause 6.4 STAT_FAILED_IMAGE of this Technical Specification, but omitting the second paragraph.}

{In 13.8.2.24 STAT_STOPPED_IMAGE, edit the paragraph as follows.}

The value of the default integer scalar constant STAT_STOPPED_IMAGE is assigned to the variable specified in a STAT= specifier (6.7.4, 8.5.7) or a STAT argument in a call to a collective subroutine if execution of the statement with that specifier or argument requires synchronization with an image that has initiated termination of execution. This value shall be positive ~~and different from the value of IOSTAT_INQUIRE_INTERNAL_UNIT~~.

{In 13.8.2.24 STAT_STOPPED_IMAGE, insert a new Note after paragraph 1}

In addition to detecting that an image has initiated normal termination by having the variable in a STAT=specifier or a STAT argument of a call to a collective subroutine assigned the value STAT_STOPPED_IMAGE, an image can get the indices of the images that have initiated normal termination in a specified team by invoking the intrinsic function STOPPED_IMAGES.

{In 13.8.2 The ISO_FORTRAN_ENV intrinsic module, append a new subclause 13.8.2.26 consisting of subclause 5.2 TEAM_TYPE of this Technical Specification, but omitting the final sentence of the first paragraph.}

{In 13.8.2 The ISO_FORTRAN_ENV intrinsic module, append a new subclause}

13.8.2.26a Uniqueness of values of named constants

The values of the named constants IOSTAT_INQUIRE_INTERNAL_UNIT, STAT_FAILED_IMAGE, STAT_LOCKED, STAT_LOCKED_OTHER_IMAGE STAT_STOPPED_IMAGE, and STAT_UNLOCKED shall be distinct.

9.10 Edits to clause 16

{In 16.4 Statement and construct entities, in paragraph 1, after "DO CONCURRENT" replace "or" with a comma; after "ASSOCIATE construct" insert ", or as a coarray specified by a *codimension-decl* in a CHANGE TEAM construct,"}

{In 16.4 Statement and construct entities, add the following new paragraph after paragraph 8}

The associate names of a CHANGE TEAM construct have the scope of the block. They have the declared type, dynamic type, type parameters, rank, bounds, and cobounds as specified in 8.1.4a.

{In 16.5.1.6 Construct association, append the following sentence to the paragraph 1}

Execution of a CHANGE TEAM statement establishes an association between each coselector and the corresponding associate name of the construct.

{In 16.6.5 Events that cause variables to become defined, add the following list item}

(33) Failure of the image that locked a lock variable before it has unlocked the variable causes the variable to become unlocked.

{In 16.6.6 Events that cause variables to become undefined, add the following list item}

(27) When an image fails during the execution of a segment, a data object on a non-failed image becomes undefined if it might be defined or undefined by execution of a statement of the segment other than an invocation of an atomic subroutine.

{In 16.6.7, Variable definition context, after item (13) insert a new list item}