

```

1 MPI_TYPE_UB( datatype, displacement)
2     IN          datatype          datatype (handle)
3
4     OUT         displacement      displacement of upper bound from origin, in bytes (in-
5                                     teger)
6

```

```

7 int MPI_Type_ub(MPI_Datatype datatype, MPI_Aint* displacement)
8

```

```

9 MPI_TYPE_UB( DATATYPE, DISPLACEMENT, IERROR)
10    INTEGER DATATYPE, DISPLACEMENT, IERROR
11

```

The following function is deprecated and is superseded by
 MPI_COMM_CREATE_KEYVAL in MPI-2.0. The language independent definition of the
 deprecated function is the same as *[of the new function, except of the function name]that of*
the new function, except for the function name and a different behavior in the C/Fortran
language interoperability, see Section 16.3.7 on page 523. The language bindings are modi-
 fied.

```

18 MPI_KEYVAL_CREATE(copy_fn, delete_fn, keyval, extra_state)
19
20     IN          copy_fn           Copy callback function for keyval
21
22     IN          delete_fn        Delete callback function for keyval
23
24     OUT         keyval           key value for future access (integer)
25
26     IN          extra_state      Extra state for callback functions
27
28
29 int MPI_Keyval_create(MPI_Copy_function *copy_fn, MPI_Delete_function
30                      *delete_fn, int *keyval, void* extra_state)
31
32 MPI_KEYVAL_CREATE(COPY_FN, DELETE_FN, KEYVAL, EXTRA_STATE, IERROR)
33    EXTERNAL COPY_FN, DELETE_FN
34    INTEGER KEYVAL, EXTRA_STATE, IERROR
35

```

The copy_fn function is invoked when a communicator is duplicated by
 MPI_COMM_DUP. copy_fn should be of type MPI_Copy_function, which is defined as follows:

```

36 typedef int MPI_Copy_function(MPI_Comm oldcomm, int keyval,
37                               void *extra_state, void *attribute_val_in,
38                               void *attribute_val_out, int *flag)
39

```

A Fortran declaration for such a function is as follows:

```

40 SUBROUTINE COPY_FUNCTION(OLDCOMM, KEYVAL, EXTRA_STATE, ATTRIBUTE_VAL_IN,
41                          ATTRIBUTE_VAL_OUT, FLAG, IERR)
42    INTEGER OLDCOMM, KEYVAL, EXTRA_STATE, ATTRIBUTE_VAL_IN,
43    ATTRIBUTE_VAL_OUT, IERR
44    LOGICAL FLAG
45

```

copy_fn may be specified as MPI_NULL_COPY_FN or MPI_DUP_FN from either C or
 FORTRAN; MPI_NULL_COPY_FN is a function that does nothing other than returning
 flag = 0 and MPI_SUCCESS. MPI_DUP_FN is a simple-minded copy function that sets flag =

function is called, using the right calling convention for the language of that function; and similarly, for the delete callback function.

Advice to implementors. This requires that attributes be tagged either as “C,” “C++” or “Fortran,” and that the language tag be checked in order to use the right calling convention for the callback function. (*End of advice to implementors.*)

The attribute manipulation functions described in Section 6.7 on page 240 define attributes arguments to be of type `void*` in C, and of type `INTEGER`, in Fortran. On some systems, `INTEGER`s will have 32 bits, while C/C++ pointers will have 64 bits. This is a problem if communicator attributes are used to move information from a Fortran caller to a C/C++ callee, or vice-versa.

MPI [will store]behaves as if it stores, internally, address sized attributes. If Fortran `INTEGER`s are smaller, then the Fortran function `MPI_ATTR_GET` will return the least significant part of the attribute word; the Fortran function `MPI_ATTR_PUT` will set the least significant part of the attribute word, which will be sign extended to the entire word. (These two functions may be invoked explicitly by user code, or implicitly, by attribute copying callback functions.)

As for addresses, new functions are provided that manipulate Fortran address sized attributes, and have the same functionality as the old functions in C/C++. These functions are described in Section 6.7, page 240. Users are encouraged to use these new functions.

MPI supports two types of attributes: address-valued (pointer) attributes, and integer valued attributes. C and C++ attribute functions put and get address valued attributes. Fortran attribute functions put and get integer valued attributes. When an integer valued attribute is accessed from C or C++, then `MPI_xxx_get_attr` will return the address of (a pointer to) the integer valued attribute, which is a pointer to `MPI_Aint` if the attribute was stored with Fortran `MPI_xxx_SET_ATTR`, and a pointer to `int` if it was stored with the deprecated Fortran `MPI_ATTR_PUT`. When an address valued attribute is accessed from Fortran, then `MPI_xxx_GET_ATTR` will convert the address into an integer and return the result of this conversion. This conversion is lossless if new style attribute functions are used, and an integer of kind `MPI_ADDRESS_KIND` is returned. The conversion may cause truncation if deprecated attribute functions are used. In C, the deprecated routines `MPI_Attr_put` and `MPI_Attr_get` behave identical to `MPI_Comm_set_attr` and `MPI_Comm_get_attr`.

Example 16.17 A. Setting an attribute value in C

```
int set_val = 3;
struct foo set_struct;

/* Set a value that is a pointer to an int */

MPI_Comm_set_attr(MPI_COMM_WORLD, keyval1, &set_val);
/* Set a value that is a pointer to a struct */
MPI_Comm_set_attr(MPI_COMM_WORLD, keyval2, &set_struct);
/* Set an integer value */
MPI_Comm_set_attr(MPI_COMM_WORLD, keyval3, (void *) 17);
```

B. Reading the attribute value in C

```

int flag, *get_val;
struct foo *get_struct;

/* Upon successful return, get_val == &set_val
   (and therefore *get_val == 3) */
MPI_Comm_get_attr(MPI_COMM_WORLD, keyval1, &get_val, &flag);
/* Upon successful return, get_struct == &set_struct */
MPI_Comm_get_attr(MPI_COMM_WORLD, keyval2, &get_struct, &flag);
/* Upon successful return, get_val == (void*) 17 */
/*      i.e., (MPI_Aint) get_val == 17 */
MPI_Comm_get_attr(MPI_COMM_WORLD, keyval3, &get_val, &flag);

```

C. Reading the attribute value with (deprecated) Fortran MPI-1 calls

```

LOGICAL FLAG
INTEGER IERR, GET_VAL, GET_STRUCT

! Upon successful return, GET_VAL == &set_val, possibly truncated
CALL MPI_ATTR_GET(MPI_COMM_WORLD, KEYVAL1, GET_VAL, FLAG, IERR)
! Upon successful return, GET_STRUCT == &set_struct, possibly truncated
CALL MPI_ATTR_GET(MPI_COMM_WORLD, KEYVAL2, GET_STRUCT, FLAG, IERR)
! Upon successful return, GET_VAL == 17
CALL MPI_ATTR_GET(MPI_COMM_WORLD, KEYVAL3, GET_VAL, FLAG, IERR)

```

D. Reading the attribute value with Fortran MPI-2 calls

```

LOGICAL FLAG
INTEGER IERR
INTEGER (KIND=MPI_ADDRESS_KIND) GET_VAL, GET_STRUCT

! Upon successful return, GET_VAL == &set_val
CALL MPI_COMM_GET_ATTR(MPI_COMM_WORLD, KEYVAL1, GET_VAL, FLAG, IERR)
! Upon successful return, GET_STRUCT == &set_struct
CALL MPI_COMM_GET_ATTR(MPI_COMM_WORLD, KEYVAL2, GET_STRUCT, FLAG, IERR)
! Upon successful return, GET_VAL == 17
CALL MPI_COMM_GET_ATTR(MPI_COMM_WORLD, KEYVAL3, GET_VAL, FLAG, IERR)

```

Example 16.18 A. Setting an attribute value with the (deprecated) Fortran MPI-1 call

```

INTEGER IERR, VAL = 7
CALL MPI_ATTR_PUT(MPI_COMM_WORLD, KEYVAL, VAL, IERR)

```

B. Reading the attribute value in C

```

int flag;
int *value;

/* Upon successful return, value points to internal MPI storage and
   *value == (int) 7 */
MPI_Comm_get_attr(MPI_COMM_WORLD, keyval, &value, &flag);

```

C. Reading the attribute value with (deprecated) Fortran MPI-1 calls

```

1  LOGICAL FLAG
2
3  INTEGER IERR, VALUE
4
5
6  ! Upon successful return, VALUE == 7
7  CALL MPI_ATTR_GET(MPI_COMM_WORLD, KEYVAL, VALUE, FLAG, IERR)
8

```

D. Reading the attribute value with Fortran MPI-2 calls

```

10 LOGICAL FLAG
11 INTEGER IERR
12 INTEGER (KIND=MPI_ADDRESS_KIND) VALUE
13
14 ! Upon successful return, VALUE == 7 (sign extended)
15 CALL MPI_COMM_GET_ATTR(MPI_COMM_WORLD, KEYVAL, VALUE, FLAG, IERR)
16

```

Example 16.19 A. Setting an attribute value via a Fortran MPI-2 call

```

18
19 INTEGER IERR
20 INTEGER(KIND=MPI_ADDRESS_KIND) VALUE1 = 42
21 INTEGER(KIND=MPI_ADDRESS_KIND) VALUE2 = pow(2, 40)
22
23 CALL MPI_COMM_SET_ATTR(MPI_COMM_WORLD, KEYVAL1, VALUE1, IERR)
24 CALL MPI_COMM_SET_ATTR(MPI_COMM_WORLD, KEYVAL2, VALUE2, IERR)
25

```

B. Reading the attribute value in C

```

27 int flag;
28 MPI_Aint *value1, *value2;
29
30
31 /* Upon successful return, value1 points to internal MPI storage and
32    *value1 == 42 */
33 MPI_Comm_get_attr(MPI_COMM_WORLD, keyval1, &value1, &flag);
34 /* Upon successful return, value2 points to internal MPI storage and
35    *value2 == 2^40 */
36 MPI_Comm_get_attr(MPI_COMM_WORLD, keyval2, &value2, &flag);
37

```

C. Reading the attribute value with (deprecated) Fortran MPI-1 calls

```

39 LOGICAL FLAG
40 INTEGER IERR, VALUE1, VALUE2
41
42 ! Upon successful return, VALUE1 == 42
43 CALL MPI_ATTR_GET(MPI_COMM_WORLD, KEYVAL1, VALUE1, FLAG, IERR)
44 ! Upon successful return, VALUE2 == 2^40, or 0 if truncation
45 ! needed (i.e., the least significant part of the attribute word)
46 CALL MPI_ATTR_GET(MPI_COMM_WORLD, KEYVAL2, VALUE2, FLAG, IERR)
47

```

D. Reading the attribute value with Fortran MPI-2 calls

```

LOGICAL FLAG
INTEGER IERR
INTEGER (KIND=MPI_ADDRESS_KIND) VALUE1, VALUE2

! Upon successful return, VALUE1 == 42
CALL MPI_COMM_GET_ATTR(MPI_COMM_WORLD, KEYVAL1, VALUE1, FLAG, IERR)
! Upon successful return, VALUE2 == 2^40
CALL MPI_COMM_GET_ATTR(MPI_COMM_WORLD, KEYVAL2, VALUE2, FLAG, IERR)

```

The predefined MPI attributes can be integer valued or address valued. Predefined integer valued attributes, such as MPI_TAG_UB, behave as if they were put by a [Fortran call]call to the deprecated Fortran routine MPI_ATTR_PUT, i.e., in Fortran, MPI_COMM_GET_ATTR(MPI_COMM_WORLD, MPI_TAG_UB, val, flag, ierr) will return in val the upper bound for tag value; in C, MPI_Comm_get_attr(MPI_COMM_WORLD, MPI_TAG_UB, &p, &flag) will return in p a pointer to an int containing the upper bound for tag value.

Address valued predefined attributes, such as MPI_WIN_BASE behave as if they were put by a C call, i.e., in Fortran, MPI_WIN_GET_ATTR(win, MPI_WIN_BASE, val, flag, ierror) will return in val the base address of the window, converted to an integer. In C, MPI_Win_get_attr(win, MPI_WIN_BASE, &p, &flag) will return in p a pointer to the window base, cast to (void *).

Rationale. The design is consistent with the behavior specified for predefined attributes, and ensures that no information is lost when attributes are passed from language to language. Because the language interoperability for predefined attributes was defined based on MPI_ATTR_PUT, this definition is kept for compatibility reasons although the routine itself is now deprecated. (*End of rationale.*)

Advice to implementors. Implementations should tag attributes either as [address attributes or as integer attributes, according to whether they were set in C or in Fortran. Thus, the right choice can be made when the attribute is retrieved.](1) address attributes, (2) as INTEGER(KIND=MPI_ADDRESS_KIND) attributes or (3) as INTEGER attributes, according to whether they were set in (1) C (with MPI_Attr_put or MPI_Xxx_set_attr), (2) in Fortran with MPI_XXX_SET_ATTR or (3) with the deprecated Fortran routine MPI_ATTR_PUT. (*End of advice to implementors.*)

16.3.8 Extra State

Extra-state should not be modified by the copy or delete callback functions. (This is obvious from the C binding, but not obvious from the Fortran binding). However, these functions may update state that is indirectly accessed via extra-state. E.g., in C, extra-state can be a pointer to a data structure that is modified by the copy or callback functions; in Fortran, extra-state can be an index into an entry in a COMMON array that is modified by the copy or callback functions. In a multithreaded environment, users should be aware that distinct threads may invoke the same callback function concurrently: if this function modifies state associated with extra-state, then mutual exclusion code must be used to protect updates and accesses to the shared state.