D R A F T

Document for a Standard Message-Passing Interface

Message Passing Interface Forum

December 11, 2019
This work was supported in part by NSF and ARPA under NSF contract CDA-9115428 and Esprit under project HPC Standards (21111).

This is the result of a LaTeX run of a draft of a single chapter of the MPIF Final Report document.

Chapter 9

The Info Object

Many of the routines in MPI take an argument info. info is an opaque object with a handle of type MPI_Info in C and Fortran with the `mpi_f08` module, and `INTEGER` in Fortran with the `mpi` module or the include file `mpif.h`. It stores an unordered set of (key,value) pairs (both key and value are strings). A key can have only one value. MPI reserves several keys and requires that if an implementation uses a reserved key, it must provide the specified functionality. An implementation is not required to support these keys and may support any others not reserved by MPI. Some info hints allow the MPI library to restrict its support for certain operations in order to improve performance or resource utilization. If an application provides such an info hint, it must be compatible with any changes in the behavior of the MPI library that are allowed by the info hint.

An implementation must support info objects as caches for arbitrary (key,value) pairs, regardless of whether it recognizes the key. Each function that takes hints in the form of an MPI_Info must be prepared to ignore any key it does not recognize. This description of info objects does not attempt to define how a particular function should react if it recognizes a key but not the associated value. MPI_INFO_GET_NKEYS, MPI_INFO_GET_NTHKEY, MPI_INFO_GET_VALUELEN, MPI_INFO_GET, and MPI_INFO_GET_STRING must retain all (key,value) pairs so that layered functionality can also use the Info object.

Keys have an implementation-defined maximum length of MPI_MAX_INFO_KEY, which is at least 32 and at most 255. Values have an implementation-defined maximum length of MPI_MAX_INFO_VAL. In Fortran, leading and trailing spaces are stripped from both. Returned values will never be larger than these maximum lengths. Both key and value are case sensitive.

> *Rationale.* Keys have a maximum length because the set of known keys will always be finite and known to the implementation and because there is no reason for keys to be complex. The small maximum size allows applications to declare keys of size MPI_MAX_INFO_KEY. The limitation on value sizes is so that an implementation is not forced to deal with arbitrarily long strings. (*End of rationale.*)
>
> *Advice to users.* MPI_MAX_INFO_VAL might be very large, so it might not be wise to declare a string of that size. (*End of advice to users.*)

When info is used as an argument to a nonblocking routine, it is parsed before that routine returns, so that it may be modified or freed immediately after return.

When the descriptions refer to a key or value as being a boolean, an integer, or a list, they mean the string representation of these types. An implementation may define its own

rules for how info value strings are converted to other types, but to ensure portability, every implementation must support the following representations. Valid values for a boolean must include the strings "true" and "false" (all lowercase). For integers, valid values must include string representations of decimal values of integers that are within the range of a standard integer type in the program. (However it is possible that not every integer is a valid value for a given key.) On positive numbers, + signs are optional. No space may appear between a + or − sign and the leading digit of a number. For comma separated lists, the string must contain valid elements separated by commas. Leading and trailing spaces are stripped automatically from the types of info values described above and for each element of a comma separated list. These rules apply to all info values of these types. Implementations are free to specify a different interpretation for values of other info keys.

MPI_INFO_CREATE(info)

OUT	info	info object created (handle)

```
int MPI_Info_create(MPI_Info *info)

MPI_Info_create(info, ierror)
    TYPE(MPI_Info), INTENT(OUT) ::  info
    INTEGER, OPTIONAL, INTENT(OUT) ::  ierror

MPI_INFO_CREATE(INFO, IERROR)
    INTEGER INFO, IERROR
```

MPI_INFO_CREATE creates a new info object. The newly created object contains no key/value pairs.

MPI_INFO_SET(info, key, value)

INOUT	info	info object (handle)
IN	key	key (string)
IN	value	value (string)

```
int MPI_Info_set(MPI_Info info, const char *key, const char *value)

MPI_Info_set(info, key, value, ierror)
    TYPE(MPI_Info), INTENT(IN) ::  info
    CHARACTER(LEN=*), INTENT(IN) ::  key, value
    INTEGER, OPTIONAL, INTENT(OUT) ::  ierror

MPI_INFO_SET(INFO, KEY, VALUE, IERROR)
    INTEGER INFO, IERROR
    CHARACTER*(*) KEY, VALUE
```

MPI_INFO_SET adds the (key,value) pair to info, and overrides the value if a value for the same key was previously set. key and value are null-terminated strings in C. In Fortran, leading and trailing spaces in key and value are stripped. If either key or value are larger than the allowed maximums, the errors MPI_ERR_INFO_KEY or MPI_ERR_INFO_VALUE are

raised, respectively.

MPI_INFO_DELETE(info, key)

INOUT	info	info object (handle)
IN	key	key (string)

```
int MPI_Info_delete(MPI_Info info, const char *key)

MPI_Info_delete(info, key, ierror)
    TYPE(MPI_Info), INTENT(IN) ::  info
    CHARACTER(LEN=*), INTENT(IN) ::  key
    INTEGER, OPTIONAL, INTENT(OUT) ::  ierror

MPI_INFO_DELETE(INFO, KEY, IERROR)
    INTEGER INFO, IERROR
    CHARACTER*(*) KEY
```

MPI_INFO_DELETE deletes a (key,value) pair from info. If key is not defined in info, the call raises an error of class MPI_ERR_INFO_NOKEY.

MPI_INFO_GET(info, key, valuelen, value, flag)

IN	info	info object (handle)
IN	key	key (string)
IN	valuelen	length of value arg (integer)
OUT	value	value (string)
OUT	flag	true if key defined, false if not (boolean)

```
int MPI_Info_get(MPI_Info info, const char *key, int valuelen, char *value,
            int *flag)

MPI_Info_get(info, key, valuelen, value, flag, ierror)
    TYPE(MPI_Info), INTENT(IN) ::  info
    CHARACTER(LEN=*), INTENT(IN) ::  key
    INTEGER, INTENT(IN) ::  valuelen
    CHARACTER(LEN=valuelen), INTENT(OUT) ::  value
    LOGICAL, INTENT(OUT) ::  flag
    INTEGER, OPTIONAL, INTENT(OUT) ::  ierror

MPI_INFO_GET(INFO, KEY, VALUELEN, VALUE, FLAG, IERROR)
    INTEGER INFO, VALUELEN, IERROR
    CHARACTER*(*) KEY, VALUE
    LOGICAL FLAG
```

This function retrieves the value associated with key in a previous call to MPI_INFO_SET. If such a key exists, it sets flag to true and returns the value in value, otherwise it sets flag to false and leaves value unchanged. valuelen is the number of characters

available in value. If it is less than the actual size of the value, the value is truncated. In C, valuelen should be one less than the amount of allocated space to allow for the null terminator.

If key is larger than MPI_MAX_INFO_KEY, the call is erroneous.

MPI_INFO_GET_VALUELEN(info, key, valuelen, flag)

IN	info	info object (handle)
IN	key	key (string)
OUT	valuelen	length of value arg (integer)
OUT	flag	true if key defined, false if not (boolean)

```
int MPI_Info_get_valuelen(MPI_Info info, const char *key, int *valuelen,
              int *flag)

MPI_Info_get_valuelen(info, key, valuelen, flag, ierror)
    TYPE(MPI_Info), INTENT(IN) ::  info
    CHARACTER(LEN=*), INTENT(IN) ::  key
    INTEGER, INTENT(OUT) ::  valuelen
    LOGICAL, INTENT(OUT) ::  flag
    INTEGER, OPTIONAL, INTENT(OUT) ::  ierror

MPI_INFO_GET_VALUELEN(INFO, KEY, VALUELEN, FLAG, IERROR)
    INTEGER INFO, VALUELEN, IERROR
    LOGICAL FLAG
    CHARACTER*(*) KEY
```

Retrieves the length of the value associated with key. If key is defined, valuelen is set to the length of its associated value and flag is set to true. If key is not defined, valuelen is not touched and flag is set to false. The length returned in C does not include the end-of-string character.

If key is larger than MPI_MAX_INFO_KEY, the call is erroneous.

MPI_INFO_GET_STRING(info, key, buflen, value, flag)

IN	info	info object (handle)
IN	key	key (string)
INOUT	buflen	length of value buffer (integer)
OUT	value	value (string)
OUT	flag	true if key defined, false if not (boolean)

```
int MPI_Info_get_string(MPI_Info info, const char *key, int *buflen,
              char *value, int *flag)

MPI_Info_get_string(info, key, buflen, value, flag, ierror)
    TYPE(MPI_Info), INTENT(IN) ::  info
    CHARACTER(LEN=*), INTENT(IN) ::  key
```

```
    INTEGER, INTENT(INOUT) ::  buflen
    CHARACTER(LEN=valuelen), INTENT(OUT) ::  value
    LOGICAL, INTENT(OUT) ::  flag
    INTEGER, OPTIONAL, INTENT(OUT) ::  ierror

MPI_INFO_GET_STRING(INFO, KEY, BUFLEN, VALUE, FLAG, IERROR)
    INTEGER INFO, BUFLEN, IERROR
    CHARACTER*(*) KEY, VALUE
    LOGICAL FLAG
```

This function retrieves the value associated with key in a previous call to MPI_INFO_SET. If such a key exists, it sets flag to true and returns the value in value, otherwise it sets flag to false and leaves value unchanged. buflen on input is the size of the provided buffer, for the output of buflen it is the size of the buffer needed to store the value string. If the buflen passed into the function is less than the actual size needed to store the value string (including null terminator in C), the value is truncated. On return, the value of buflen will be set to the required buffer size to hold the value string. If buflen is set to 0, value is not changed. In C, buflen includes the required space for the null terminator. In C, this function returns a null terminated string in all cases where the buflen input value is greater than 0.

If key is larger than MPI_MAX_INFO_KEY, the call is erroneous.

Advice to users. The MPI_INFO_GET_STRING function can be used to obtain the size of the required buffer for a value string by setting the buflen to 0. The returned buflen can then be used to allocate memory before calling MPI_INFO_GET_STRING again to obtain the value string. (*End of advice to users.*)

MPI_INFO_GET_NKEYS(info, nkeys)

IN	info	info object (handle)
OUT	nkeys	number of defined keys (integer)

```
int MPI_Info_get_nkeys(MPI_Info info, int *nkeys)

MPI_Info_get_nkeys(info, nkeys, ierror)
    TYPE(MPI_Info), INTENT(IN) ::  info
    INTEGER, INTENT(OUT) ::  nkeys
    INTEGER, OPTIONAL, INTENT(OUT) ::  ierror

MPI_INFO_GET_NKEYS(INFO, NKEYS, IERROR)
    INTEGER INFO, NKEYS, IERROR
```

MPI_INFO_GET_NKEYS returns the number of currently defined keys in info.

MPI_INFO_GET_NTHKEY(info, n, key)

IN	info	info object (handle)
IN	n	key number (integer)
OUT	key	key (string)

```
int MPI_Info_get_nthkey(MPI_Info info, int n, char *key)

MPI_Info_get_nthkey(info, n, key, ierror)
    TYPE(MPI_Info), INTENT(IN) ::  info
    INTEGER, INTENT(IN) ::  n
    CHARACTER(LEN=*), INTENT(OUT) ::  key
    INTEGER, OPTIONAL, INTENT(OUT) ::  ierror

MPI_INFO_GET_NTHKEY(INFO, N, KEY, IERROR)
    INTEGER INFO, N, IERROR
    CHARACTER*(*) KEY
```

This function returns the nth defined key in info. Keys are numbered $0 \ldots N-1$ where N is the value returned by MPI_INFO_GET_NKEYS. All keys between 0 and $N-1$ are guaranteed to be defined. The number of a given key does not change as long as info is not modified with MPI_INFO_SET or MPI_INFO_DELETE.

MPI_INFO_DUP(info, newinfo)

IN	info	info object (handle)
OUT	newinfo	info object (handle)

```
int MPI_Info_dup(MPI_Info info, MPI_Info *newinfo)

MPI_Info_dup(info, newinfo, ierror)
    TYPE(MPI_Info), INTENT(IN) ::  info
    TYPE(MPI_Info), INTENT(OUT) ::  newinfo
    INTEGER, OPTIONAL, INTENT(OUT) ::  ierror

MPI_INFO_DUP(INFO, NEWINFO, IERROR)
    INTEGER INFO, NEWINFO, IERROR
```

MPI_INFO_DUP duplicates an existing info object, creating a new object, with the same (key,value) pairs and the same ordering of keys.

MPI_INFO_FREE(info)

INOUT	info	info object (handle)

```
int MPI_Info_free(MPI_Info *info)

MPI_Info_free(info, ierror)
    TYPE(MPI_Info), INTENT(INOUT) ::  info
    INTEGER, OPTIONAL, INTENT(OUT) ::  ierror
```

```
MPI_INFO_FREE(INFO, IERROR)
    INTEGER INFO, IERROR
```

This function frees info and sets it to MPI_INFO_NULL. The value of an info argument is interpreted each time the info is passed to a routine. Changes to an info after return from a routine do not affect that interpretation.

Index