MPI for Python

Release 3.1.3

Lisandro Dalcin

Nov 25, 2021

Contents

Abstract

This document describes the *MPI for Python* package. *MPI for Python* provides Python bindings for the *Message Passing Interface* (MPI) standard, allowing Python applications to exploit multiple processors on workstations, clusters and supercomputers.

This package builds on the MPI specification and provides an object oriented interface resembling the MPI-2 C++ bindings. It supports point-to-point (sends, receives) and collective (broadcasts, scatters, gathers) communication of any *picklable* Python object, as well as efficient communication of Python objects exposing the Python buffer interface (e.g. NumPy arrays and builtin bytes/array/memoryview objects).

1 Introduction

Over the last years, high performance computing has become an affordable resource to many more researchers in the scientific community than ever before. The conjunction of quality open source software and commodity hardware strongly influenced the now widespread popularity of Beowulf class clusters and cluster of workstations.

Among many parallel computational models, message-passing has proven to be an effective one. This paradigm is specially suited for (but not limited to) distributed memory architectures and is used in today's most demanding scientific and engineering application related to modeling, simulation, design, and signal processing. However, portable message-passing parallel programming used to be a nightmare in the past because of the many incompatible options developers were faced to. Fortunately, this situation definitely changed after the MPI Forum released its standard specification.

High performance computing is traditionally associated with software development using compiled languages. However, in typical applications programs, only a small part of the code is time-critical enough to require the efficiency of compiled languages. The rest of the code is generally related to memory management, error handling, input/output, and user interaction, and those are usually the most error prone and time-consuming lines of code to write and debug in the whole development process. Interpreted high-level languages can be really advantageous for this kind of tasks.

For implementing general-purpose numerical computations, MATLAB[1] is the dominant interpreted programming language. In the open source side, Octave and Scilab are well known, freely distributed software packages providing compatibility with the MATLAB language. In this work, we present MPI for Python, a new package enabling applications to exploit multiple processors using standard MPI "look and feel" in Python scripts.

1.1 What is MPI?

MPI, [mpi-using] [mpi-ref] the *Message Passing Interface*, is a standardized and portable message-passing system designed to function on a wide variety of parallel computers. The standard defines the syntax and semantics of library routines and allows users to write portable programs in the main scientific programming languages (Fortran, C, or C++).

Since its release, the MPI specification [mpi-std1] [mpi-std2] has become the leading standard for message-passing libraries for parallel computers. Implementations are available from vendors of high-performance computers and from well known open source projects like MPICH [mpi-mpich] and Open MPI [mpi-openmpi].

1.2 What is Python?

Python is a modern, easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming with dynamic typing and dynamic binding. It supports modules and packages, which encourages program modularity and code reuse. Python's elegant syntax, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. It is easily extended with new functions and data types implemented in C or C++. Python is also suitable as an extension language for customizable applications.

Python is an ideal candidate for writing the higher-level parts of large-scale scientific applications [Hinsen97] and driving simulations in parallel architectures [Beazley97] like clusters of PC's or SMP's. Python codes are quickly developed, easily maintained, and can achieve a high degree of integration with other libraries written in compiled languages.

[1] MATLAB is a registered trademark of The MathWorks, Inc.

1.3 Related Projects

As this work started and evolved, some ideas were borrowed from well known MPI and Python related open source projects from the Internet.

- OOMPI
  - It has no relation with Python, but is an excellent object oriented approach to MPI.
  - It is a C++ class library specification layered on top of the C bindings that encapsulates MPI into a functional class hierarchy.
  - It provides a flexible and intuitive interface by adding some abstractions, like *Ports* and *Messages*, which enrich and simplify the syntax.
- Pypar
  - Its interface is rather minimal. There is no support for communicators or process topologies.
  - It does not require the Python interpreter to be modified or recompiled, but does not permit interactive parallel runs.
  - General (*picklable*) Python objects of any type can be communicated. There is good support for numeric arrays, practically full MPI bandwidth can be achieved.
- pyMPI
  - It rebuilds the Python interpreter providing a built-in module for message passing. It does permit interactive parallel runs, which are useful for learning and debugging.
  - It provides an interface suitable for basic parallel programing. There is not full support for defining new communicators or process topologies.
  - General (picklable) Python objects can be messaged between processors. There is not support for numeric arrays.
- Scientific Python
  - It provides a collection of Python modules that are useful for scientific computing.
  - There is an interface to MPI and BSP (*Bulk Synchronous Parallel programming*).
  - The interface is simple but incomplete and does not resemble the MPI specification. There is support for numeric arrays.

Additionally, we would like to mention some available tools for scientific computing and software development with Python.

- NumPy is a package that provides array manipulation and computational capabilities similar to those found in IDL, MATLAB, or Octave. Using NumPy, it is possible to write many efficient numerical data processing applications directly in Python without using any C, C++ or Fortran code.
- SciPy is an open source library of scientific tools for Python, gathering a variety of high level science and engineering modules together as a single package. It includes modules for graphics and plotting, optimization, integration, special functions, signal and image processing, genetic algorithms, ODE solvers, and others.
- Cython is a language that makes writing C extensions for the Python language as easy as Python itself. The Cython language is very close to the Python language, but Cython additionally supports calling C functions and declaring C types on variables and class attributes. This allows the compiler to generate very efficient C code from Cython code. This makes Cython the ideal language for wrapping for external C libraries, and for fast C modules that speed up the execution of Python code.

- SWIG is a software development tool that connects programs written in C and C++ with a variety of high-level programming languages like Perl, Tcl/Tk, Ruby and Python. Issuing header files to SWIG is the simplest approach to interfacing C/C++ libraries from a Python module.

2 Overview

MPI for Python provides an object oriented approach to message passing which grounds on the standard MPI-2 C++ bindings. The interface was designed with focus in translating MPI syntax and semantics of standard MPI-2 bindings for C++ to Python. Any user of the standard C/C++ MPI bindings should be able to use this module without need of learning a new interface.

2.1 Communicating Python Objects and Array Data

The Python standard library supports different mechanisms for data persistence. Many of them rely on disk storage, but *pickling* and *marshaling* can also work with memory buffers.

The `pickle` modules provide user-extensible facilities to serialize general Python objects using ASCII or binary formats. The `marshal` module provides facilities to serialize built-in Python objects using a binary format specific to Python, but independent of machine architecture issues.

MPI for Python can communicate any built-in or user-defined Python object taking advantage of the features provided by the `pickle` module. These facilities will be routinely used to build binary representations of objects to communicate (at sending processes), and restoring them back (at receiving processes).

Although simple and general, the serialization approach (i.e., *pickling* and *unpickling*) previously discussed imposes important overheads in memory as well as processor usage, especially in the scenario of objects with large memory footprints being communicated. Pickling general Python objects, ranging from primitive or container built-in types to user-defined classes, necessarily requires computer resources. Processing is also needed for dispatching the appropriate serialization method (that depends on the type of the object) and doing the actual packing. Additional memory is always needed, and if its total amount is not known *a priori*, many reallocations can occur. Indeed, in the case of large numeric arrays, this is certainly unacceptable and precludes communication of objects occupying half or more of the available memory resources.

MPI for Python supports direct communication of any object exporting the single-segment buffer interface. This interface is a standard Python mechanism provided by some types (e.g., strings and numeric arrays), allowing access in the C side to a contiguous memory buffer (i.e., address and length) containing the relevant data. This feature, in conjunction with the capability of constructing user-defined MPI datatypes describing complicated memory layouts, enables the implementation of many algorithms involving multidimensional numeric arrays (e.g., image processing, fast Fourier transforms, finite difference schemes on structured Cartesian grids) directly in Python, with negligible overhead, and almost as fast as compiled Fortran, C, or C++ codes.

2.2 Communicators

In *MPI for Python*, `Comm` is the base class of communicators. The `Intracomm` and `Intercomm` classes are sublcasses of the `Comm` class. The `Comm.Is_inter` method (and `Comm.Is_intra`, provided for convenience but not part of the MPI specification) is defined for communicator objects and can be used to determine the particular communicator class.

The two predefined intracommunicator instances are available: `COMM_SELF` and `COMM_WORLD`. From them, new communicators can be created as needed.

The number of processes in a communicator and the calling process rank can be respectively obtained with methods `Comm.Get_size` and `Comm.Get_rank`. The associated process group can be retrieved from a communicator by calling the `Comm.Get_group` method, which returns an instance of the `Group` class. Set operations with `Group` objects like

like `Group.Union`, `Group.Intersection` and `Group.Difference` are fully supported, as well as the creation of new communicators from these groups using `Comm.Create` and `Comm.Create_group`.

New communicator instances can be obtained with the `Comm.Clone`, `Comm.Dup` and `Comm.Split` methods, as well methods `Intracomm.Create_intercomm` and `Intercomm.Merge`.

Virtual topologies (`Cartcomm`, `Graphcomm` and `Distgraphcomm` classes, which are specializations of the `Intracomm` class) are fully supported. New instances can be obtained from intracommunicator instances with factory methods `Intracomm.Create_cart` and `Intracomm.Create_graph`.

2.3 Point-to-Point Communications

Point to point communication is a fundamental capability of message passing systems. This mechanism enables the transmission of data between a pair of processes, one side sending, the other receiving.

MPI provides a set of *send* and *receive* functions allowing the communication of *typed* data with an associated *tag*. The type information enables the conversion of data representation from one architecture to another in the case of heterogeneous computing environments; additionally, it allows the representation of non-contiguous data layouts and user-defined datatypes, thus avoiding the overhead of (otherwise unavoidable) packing/unpacking operations. The tag information allows selectivity of messages at the receiving end.

Blocking Communications

MPI provides basic send and receive functions that are *blocking*. These functions block the caller until the data buffers involved in the communication can be safely reused by the application program.

In *MPI for Python*, the `Comm.Send`, `Comm.Recv` and `Comm.Sendrecv` methods of communicator objects provide support for blocking point-to-point communications within `Intracomm` and `Intercomm` instances. These methods can communicate memory buffers. The variants `Comm.send`, `Comm.recv` and `Comm.sendrecv` can communicate general Python objects.

Nonblocking Communications

On many systems, performance can be significantly increased by overlapping communication and computation. This is particularly true on systems where communication can be executed autonomously by an intelligent, dedicated communication controller.

MPI provides *nonblocking* send and receive functions. They allow the possible overlap of communication and computation. Non-blocking communication always come in two parts: posting functions, which begin the requested operation; and test-for-completion functions, which allow to discover whether the requested operation has completed.

In *MPI for Python*, the `Comm.Isend` and `Comm.Irecv` methods initiate send and receive operations, respectively. These methods return a `Request` instance, uniquely identifying the started operation. Its completion can be managed using the `Request.Test`, `Request.Wait` and `Request.Cancel` methods. The management of `Request` objects and associated memory buffers involved in communication requires a careful, rather low-level coordination. Users must ensure that objects exposing their memory buffers are not accessed at the Python level while they are involved in nonblocking message-passing operations.

Persistent Communications

Often a communication with the same argument list is repeatedly executed within an inner loop. In such cases, communication can be further optimized by using persistent communication, a particular case of nonblocking communication allowing the reduction of the overhead between processes and communication controllers. Furthermore , this kind of optimization can also alleviate the extra call overheads associated to interpreted, dynamic languages like Python.

In *MPI for Python*, the `Comm.Send_init` and `Comm.Recv_init` methods create persistent requests for a send and receive operation, respectively. These methods return an instance of the `Prequest` class, a subclass of the `Request` class. The actual communication can be effectively started using the `Prequest.Start` method, and its completion can be managed as previously described.

2.4 Collective Communications

Collective communications allow the transmittal of data between multiple processes of a group simultaneously. The syntax and semantics of collective functions is consistent with point-to-point communication. Collective functions communicate *typed* data, but messages are not paired with an associated *tag*; selectivity of messages is implied in the calling order. Additionally, collective functions come in blocking versions only.

The more commonly used collective communication operations are the following.

- Barrier synchronization across all group members.
- Global communication functions
  - Broadcast data from one member to all members of a group.
  - Gather data from all members to one member of a group.
  - Scatter data from one member to all members of a group.
- Global reduction operations such as sum, maximum, minimum, etc.

In *MPI for Python*, the `Comm.Bcast`, `Comm.Scatter`, `Comm.Gather`, `Comm.Allgather`, `Comm.Alltoall` methods provide support for collective communications of memory buffers. The lower-case variants `Comm.bcast`, `Comm.scatter`, `Comm.gather`, `Comm.allgather` and `Comm.alltoall` can communicate general Python objects. The vector variants (which can communicate different amounts of data to each process) `Comm.Scatterv`, `Comm.Gatherv`, `Comm.Allgatherv`, `Comm.Alltoallv` and `Comm.Alltoallw` are also supported, they can only communicate objects exposing memory buffers.

Global reducion operations on memory buffers are accessible through the `Comm.Reduce`, `Comm.Reduce_scatter`, `Comm.Allreduce`, `Intracomm.Scan` and `Intracomm.Exscan` methods. The lower-case variants `Comm.reduce`, `Comm.allreduce`, `Intracomm.scan` and `Intracomm.exscan` can communicate general Python objects; however, the actual required reduction computations are performed sequentially at some process. All the predefined (i.e., `SUM`, `PROD`, `MAX`, etc.) reduction operations can be applied.

2.5 Support for GPU-aware MPI

Several MPI implementations, including Open MPI and MVAPICH, support passing GPU pointers to MPI calls to avoid explict data movement between the host and the device. On the Python side, GPU arrays have been implemented by many libraries that need GPU computation, such as CuPy, Numba, PyTorch, and PyArrow. In order to increase library interoperability, two kinds of zero-copy data exchange protocols are defined and agreed upon: DLPack and CUDA Array Interface. For example, a CuPy array can be passed to a Numba CUDA-jit kernel.

MPI for Python provides an experimental support for GPU-aware MPI. This feature requires:

1. mpi4py is built against a GPU-aware MPI library.
2. The Python GPU arrays are compliant with either of the protocols.

See the *Tutorial* section for further information. We note that

- Whether or not a MPI call can work for GPU arrays depends on the underlying MPI implementation, not on mpi4py.
- This support is currently experimental and subject to change in the future.

2.6 Dynamic Process Management

In the context of the MPI-1 specification, a parallel application is static; that is, no processes can be added to or deleted from a running application after it has been started. Fortunately, this limitation was addressed in MPI-2. The new specification added a process management model providing a basic interface between an application and external resources and process managers.

This MPI-2 extension can be really useful, especially for sequential applications built on top of parallel modules, or parallel applications with a client/server model. The MPI-2 process model provides a mechanism to create new processes and establish communication between them and the existing MPI application. It also provides mechanisms to establish communication between two existing MPI applications, even when one did not *start* the other.

In *MPI for Python*, new independent process groups can be created by calling the `Intracomm.Spawn` method within an intracommunicator. This call returns a new intercommunicator (i.e., an `Intercomm` instance) at the parent process group. The child process group can retrieve the matching intercommunicator by calling the `Comm.Get_parent` class method. At each side, the new intercommunicator can be used to perform point to point and collective communications between the parent and child groups of processes.

Alternatively, disjoint groups of processes can establish communication using a client/server approach. Any server application must first call the `Open_port` function to open a *port* and the `Publish_name` function to publish a provided *service*, and next call the `Intracomm.Accept` method. Any client applications can first find a published *service* by calling the `Lookup_name` function, which returns the *port* where a server can be contacted; and next call the `Intracomm.Connect` method. Both `Intracomm.Accept` and `Intracomm.Connect` methods return an `Intercomm` instance. When connection between client/server processes is no longer needed, all of them must cooperatively call the `Comm.Disconnect` method. Additionally, server applications should release resources by calling the `Unpublish_name` and `Close_port` functions.

2.7 One-Sided Communications

One-sided communications (also called *Remote Memory Access*, *RMA*) supplements the traditional two-sided, send/receive based MPI communication model with a one-sided, put/get based interface. One-sided communication that can take advantage of the capabilities of highly specialized network hardware. Additionally, this extension lowers latency and software overhead in applications written using a shared-memory-like paradigm.

The MPI specification revolves around the use of objects called *windows*; they intuitively specify regions of a process's memory that have been made available for remote read and write operations. The published memory blocks can be accessed through three functions for put (remote send), get (remote write), and accumulate (remote update or reduction) data items. A much larger number of functions support different synchronization styles; the semantics of these synchronization operations are fairly complex.

In *MPI for Python*, one-sided operations are available by using instances of the `Win` class. New window objects are created by calling the `Win.Create` method at all processes within a communicator and specifying a memory buffer . When a window instance is no longer needed, the `Win.Free` method should be called.

The three one-sided MPI operations for remote write, read and reduction are available through calling the methods `Win.Put`, `Win.Get`, and `Win.Accumulate` respectively within a `Win` instance. These methods need an integer rank identifying the target process and an integer offset relative the base address of the remote memory block being accessed.

The one-sided operations read, write, and reduction are implicitly nonblocking, and must be synchronized by using two primary modes. Active target synchronization requires the origin process to call the `Win.Start` and `Win.Complete` methods at the origin process, and target process cooperates by calling the `Win.Post` and `Win.Wait` methods. There is also a collective variant provided by the `Win.Fence` method. Passive target synchronization is more lenient, only the origin process calls the `Win.Lock` and `Win.Unlock` methods. Locks are used to protect remote accesses to the locked remote window and to protect local load/store accesses to a locked local window.

2.8 Parallel Input/Output

The POSIX standard provides a model of a widely portable file system. However, the optimization needed for parallel input/output cannot be achieved with this generic interface. In order to ensure efficiency and scalability, the underlying parallel input/output system must provide a high-level interface supporting partitioning of file data among processes and a collective interface supporting complete transfers of global data structures between process memories and files. Additionally, further efficiencies can be gained via support for asynchronous input/output, strided accesses to data, and control over physical file layout on storage devices. This scenario motivated the inclusion in the MPI-2 standard of a custom interface in order to support more elaborated parallel input/output operations.

The MPI specification for parallel input/output revolves around the use objects called *files*. As defined by MPI, files are not just contiguous byte streams. Instead, they are regarded as ordered collections of *typed* data items. MPI supports sequential or random access to any integral set of these items. Furthermore, files are opened collectively by a group of processes.

The common patterns for accessing a shared file (broadcast, scatter, gather, reduction) is expressed by using user-defined datatypes. Compared to the communication patterns of point-to-point and collective communications, this approach has the advantage of added flexibility and expressiveness. Data access operations (read and write) are defined for different kinds of positioning (using explicit offsets, individual file pointers, and shared file pointers), coordination (non-collective and collective), and synchronism (blocking, nonblocking, and split collective with begin/end phases).

In *MPI for Python*, all MPI input/output operations are performed through instances of the `File` class. File handles are obtained by calling the `File.Open` method at all processes within a communicator and providing a file name and the intended access mode. After use, they must be closed by calling the `File.Close` method. Files even can be deleted by calling method `File.Delete`.

After creation, files are typically associated with a per-process *view*. The view defines the current set of data visible and accessible from an open file as an ordered set of elementary datatypes. This data layout can be set and queried with the `File.Set_view` and `File.Get_view` methods respectively.

Actual input/output operations are achieved by many methods combining read and write calls with different behavior regarding positioning, coordination, and synchronism. Summing up, *MPI for Python* provides the thirty (30) methods defined in MPI-2 for reading from or writing to files using explicit offsets or file pointers (individual or shared), in blocking or nonblocking and collective or noncollective versions.

2.9 Environmental Management

Initialization and Exit

Module functions `Init` or `Init_thread` and `Finalize` provide MPI initialization and finalization respectively. Module functions `Is_initialized` and `Is_finalized` provide the respective tests for initialization and finalization.

Note: `MPI_Init()` or `MPI_Init_thread()` is actually called when you import the `MPI` module from the `mpi4py` package, but only if MPI is not already initialized. In such case, calling `Init` or `Init_thread` from Python is expected to generate an MPI error, and in turn an exception will be raised.

Note: `MPI_Finalize()` is registered (by using Python C/API function `Py_AtExit()`) for being automatically called when Python processes exit, but only if `mpi4py` actually initialized MPI. Therefore, there is no need to call `Finalize` from Python to ensure MPI finalization.

Implementation Information

- The MPI version number can be retrieved from module function *Get_version*. It returns a two-integer tuple `(version, subversion)`.
- The *Get_processor_name* function can be used to access the processor name.
- The values of predefined attributes attached to the world communicator can be obtained by calling the *Comm.Get_attr* method within the *COMM_WORLD* instance.

Timers

MPI timer functionalities are available through the *Wtime* and *Wtick* functions.

Error Handling

In order facilitate handle sharing with other Python modules interfacing MPI-based parallel libraries, the predefined MPI error handlers *ERRORS_RETURN* and *ERRORS_ARE_FATAL* can be assigned to and retrieved from communicators using methods *Comm.Set_errhandler* and *Comm.Get_errhandler*, and similarly for windows and files.

When the predefined error handler *ERRORS_RETURN* is set, errors returned from MPI calls within Python code will raise an instance of the exception class *Exception*, which is a subclass of the standard Python exception `RuntimeError`.

Note: After import, mpi4py overrides the default MPI rules governing inheritance of error handlers. The *ERRORS_RETURN* error handler is set in the predefined *COMM_SELF* and *COMM_WORLD* communicators, as well as any new *Comm*, *Win*, or *File* instance created through mpi4py. If you ever pass such handles to C/C++/Fortran library code, it is recommended to set the *ERRORS_ARE_FATAL* error handler on them to ensure MPI errors do not pass silently.

Warning: Importing with `from mpi4py.MPI import *` will cause a name clashing with the standard Python `Exception` base class.

3 Tutorial

Warning: Under construction. Contributions very welcome!

Tip: Rolf Rabenseifner at HLRS developed a comprehensive MPI-3.1/4.0 course with slides and a large set of exercises including solutions. This material is available online for self-study. The slides and exercises show the C, Fortran, and Python (mpi4py) interfaces. For performance reasons, most Python exercises use NumPy arrays and communication routines involving buffer-like objects.

Tip: Victor Eijkhout at TACC authored the book *Parallel Programming for Science and Engineering*. This book is available online in PDF and HTML formats. The book covers parallel programming with MPI and OpenMP in C/C++ and Fortran, and MPI in Python using mpi4py.

MPI for Python supports convenient, *pickle*-based communication of generic Python object as well as fast, near C-speed, direct array data communication of buffer-provider objects (e.g., NumPy arrays).

- Communication of generic Python objects

  You have to use methods with **all-lowercase** names, like `Comm.send`, `Comm.recv`, `Comm.bcast`, `Comm.scatter`, `Comm.gather` . An object to be sent is passed as a parameter to the communication call, and the received object is simply the return value.

  The `Comm.isend` and `Comm.irecv` methods return `Request` instances; completion of these methods can be managed using the `Request.test` and `Request.wait` methods.

  The `Comm.recv` and `Comm.irecv` methods may be passed a buffer object that can be repeatedly used to receive messages avoiding internal memory allocation. This buffer must be sufficiently large to accommodate the transmitted messages; hence, any buffer passed to `Comm.recv` or `Comm.irecv` must be at least as long as the *pickled* data transmitted to the receiver.

  Collective calls like `Comm.scatter`, `Comm.gather`, `Comm.allgather`, `Comm.alltoall` expect a single value or a sequence of `Comm.size` elements at the root or all process. They return a single value, a list of `Comm.size` elements, or `None`.

  Note: *MPI for Python* uses the **highest** protocol version available in the Python runtime (see the `HIGHEST_PROTOCOL` constant in the `pickle` module). The default protocol can be changed at import time by setting the `MPI4PY_PICKLE_PROTOCOL` environment variable, or at runtime by assigning a different value to the `PROTOCOL` attribute of the `pickle` object within the `MPI` module.

- Communication of buffer-like objects

  You have to use method names starting with an **upper-case** letter, like `Comm.Send`, `Comm.Recv`, `Comm.Bcast`, `Comm.Scatter`, `Comm.Gather`.

  In general, buffer arguments to these calls must be explicitly specified by using a 2/3-list/tuple like `[data, MPI.DOUBLE]`, or `[data, count, MPI.DOUBLE]` (the former one uses the byte-size of `data` and the extent of the MPI datatype to define `count`).

  For vector collectives communication operations like `Comm.Scatterv` and `Comm.Gatherv`, buffer arguments are specified as `[data, count, displ, datatype]`, where `count` and `displ` are sequences of integral values.

  Automatic MPI datatype discovery for NumPy/GPU arrays and PEP-3118 buffers is supported, but limited to basic C types (all C/C99-native signed/unsigned integral types and single/double precision real/complex floating types) and availability of matching datatypes in the underlying MPI implementation. In this case, the buffer-provider object can be passed directly as a buffer argument, the count and MPI datatype will be inferred.

  If mpi4py is built against a GPU-aware MPI implementation, GPU arrays can be passed to upper-case methods as long as they have either the `__dlpack__` and `__dlpack_device__` methods or the `__cuda_array_interface__` attribute that are compliant with the respective standard specifications. Moreover, only C-contiguous or Fortran-contiguous GPU arrays are supported. It is important to note that GPU buffers must be fully ready before any MPI routines operate on them to avoid race conditions. This can be ensured by using the synchronization API of your array library. mpi4py does not have access to any GPU-specific functionality and thus cannot perform this operation automatically for users.

3.1 Running Python scripts with MPI

Most MPI programs can be run with the command **mpiexec**. In practice, running Python programs looks like:

```
$ mpiexec -n 4 python script.py
```

to run the program with 4 processors.

3.2 Point-to-Point Communication

- Python objects (`pickle` under the hood):

```python
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    data = {'a': 7, 'b': 3.14}
    comm.send(data, dest=1, tag=11)
elif rank == 1:
    data = comm.recv(source=0, tag=11)
```

- Python objects with non-blocking communication:

```python
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    data = {'a': 7, 'b': 3.14}
    req = comm.isend(data, dest=1, tag=11)
    req.wait()
elif rank == 1:
    req = comm.irecv(source=0, tag=11)
    data = req.wait()
```

- NumPy arrays (the fast way!):

```python
from mpi4py import MPI
import numpy

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

# passing MPI datatypes explicitly
if rank == 0:
    data = numpy.arange(1000, dtype='i')
    comm.Send([data, MPI.INT], dest=1, tag=77)
elif rank == 1:
    data = numpy.empty(1000, dtype='i')
    comm.Recv([data, MPI.INT], source=0, tag=77)
```

(continues on next page)

(continued from previous page)

```python
# automatic MPI datatype discovery
if rank == 0:
    data = numpy.arange(100, dtype=numpy.float64)
    comm.Send(data, dest=1, tag=13)
elif rank == 1:
    data = numpy.empty(100, dtype=numpy.float64)
    comm.Recv(data, source=0, tag=13)
```

3.3 Collective Communication

- Broadcasting a Python dictionary:

```python
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    data = {'key1' : [7, 2.72, 2+3j],
            'key2' : ( 'abc', 'xyz')}
else:
    data = None
data = comm.bcast(data, root=0)
```

- Scattering Python objects:

```python
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

if rank == 0:
    data = [(i+1)**2 for i in range(size)]
else:
    data = None
data = comm.scatter(data, root=0)
assert data == (rank+1)**2
```

- Gathering Python objects:

```python
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

data = (rank+1)**2
data = comm.gather(data, root=0)
if rank == 0:
```

(continues on next page)

(continued from previous page)

```
    for i in range(size):
        assert data[i] == (i+1)**2
else:
    assert data is None
```

- Broadcasting a NumPy array:

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    data = np.arange(100, dtype='i')
else:
    data = np.empty(100, dtype='i')
comm.Bcast(data, root=0)
for i in range(100):
    assert data[i] == i
```

- Scattering NumPy arrays:

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

sendbuf = None
if rank == 0:
    sendbuf = np.empty([size, 100], dtype='i')
    sendbuf.T[:,:] = range(size)
recvbuf = np.empty(100, dtype='i')
comm.Scatter(sendbuf, recvbuf, root=0)
assert np.allclose(recvbuf, rank)
```

- Gathering NumPy arrays:

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

sendbuf = np.zeros(100, dtype='i') + rank
recvbuf = None
if rank == 0:
    recvbuf = np.empty([size, 100], dtype='i')
comm.Gather(sendbuf, recvbuf, root=0)
if rank == 0:
```

(continues on next page)

(continued from previous page)

```
    for i in range(size):
        assert np.allclose(recvbuf[i,:], i)
```

- Parallel matrix-vector product:

```
from mpi4py import MPI
import numpy

def matvec(comm, A, x):
    m = A.shape[0] # local rows
    p = comm.Get_size()
    xg = numpy.zeros(m*p, dtype='d')
    comm.Allgather([x,  MPI.DOUBLE],
                   [xg, MPI.DOUBLE])
    y = numpy.dot(A, xg)
    return y
```

3.4 MPI-IO

- Collective I/O with NumPy arrays:

```
from mpi4py import MPI
import numpy as np

amode = MPI.MODE_WRONLY|MPI.MODE_CREATE
comm = MPI.COMM_WORLD
fh = MPI.File.Open(comm, "./datafile.contig", amode)

buffer = np.empty(10, dtype=np.int)
buffer[:] = comm.Get_rank()

offset = comm.Get_rank()*buffer.nbytes
fh.Write_at_all(offset, buffer)

fh.Close()
```

- Non-contiguous Collective I/O with NumPy arrays and datatypes:

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

amode = MPI.MODE_WRONLY|MPI.MODE_CREATE
fh = MPI.File.Open(comm, "./datafile.noncontig", amode)

item_count = 10

buffer = np.empty(item_count, dtype='i')
```

(continues on next page)

(continued from previous page)

```
buffer[:] = rank

filetype = MPI.INT.Create_vector(item_count, 1, size)
filetype.Commit()

displacement = MPI.INT.Get_size()*rank
fh.Set_view(displacement, filetype=filetype)

fh.Write_all(buffer)
filetype.Free()
fh.Close()
```

3.5 Dynamic Process Management

- Compute Pi - Master (or parent, or client) side:

```
#!/usr/bin/env python
from mpi4py import MPI
import numpy
import sys

comm = MPI.COMM_SELF.Spawn(sys.executable,
                           args=['cpi.py'],
                           maxprocs=5)

N = numpy.array(100, 'i')
comm.Bcast([N, MPI.INT], root=MPI.ROOT)
PI = numpy.array(0.0, 'd')
comm.Reduce(None, [PI, MPI.DOUBLE],
            op=MPI.SUM, root=MPI.ROOT)
print(PI)

comm.Disconnect()
```

- Compute Pi - Worker (or child, or server) side:

```
#!/usr/bin/env python
from mpi4py import MPI
import numpy

comm = MPI.Comm.Get_parent()
size = comm.Get_size()
rank = comm.Get_rank()

N = numpy.array(0, dtype='i')
comm.Bcast([N, MPI.INT], root=0)
h = 1.0 / N; s = 0.0
for i in range(rank, N, size):
    x = h * (i + 0.5)
    s += 4.0 / (1.0 + x**2)
PI = numpy.array(s * h, dtype='d')
```

(continues on next page)

(continued from previous page)

```
comm.Reduce([PI, MPI.DOUBLE], None,
            op=MPI.SUM, root=0)

comm.Disconnect()
```

3.6 CUDA-aware MPI + Python GPU arrays

- Reduce-to-all CuPy arrays:

```
from mpi4py import MPI
import cupy as cp

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

sendbuf = cp.arange(10, dtype='i')
recvbuf = cp.empty_like(sendbuf)
assert hasattr(sendbuf, '__cuda_array_interface__')
assert hasattr(recvbuf, '__cuda_array_interface__')
cp.cuda.get_current_stream().synchronize()
comm.Allreduce(sendbuf, recvbuf)

assert cp.allclose(recvbuf, sendbuf*size)
```

3.7 One-Sided Communications

- Read from (write to) the entire RMA window:

```
import numpy as np
from mpi4py import MPI
from mpi4py.util import dtlib

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

datatype = MPI.FLOAT
np_dtype = dtlib.to_numpy_dtype(datatype)
itemsize = datatype.Get_size()

N = 10
win_size = N * itemsize if rank == 0 else 0
win = MPI.Win.Allocate(win_size, comm=comm)

buf = np.empty(N, dtype=np_dtype)
if rank == 0:
    buf.fill(42)
    win.Lock(rank=0)
    win.Put(buf, target_rank=0)
    win.Unlock(rank=0)
```

(continues on next page)

(continued from previous page)

```
    comm.Barrier()
else:
    comm.Barrier()
    win.Lock(rank=0)
    win.Get(buf, target_rank=0)
    win.Unlock(rank=0)
    assert np.all(buf == 42)
```

- Accessing a part of the RMA window using the `target` argument, which is defined as `(offset, count, datatype)`:

```
import numpy as np
from mpi4py import MPI
from mpi4py.util import dtlib

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

datatype = MPI.FLOAT
np_dtype = dtlib.to_numpy_dtype(datatype)
itemsize = datatype.Get_size()

N = comm.Get_size() + 1
win_size = N * itemsize if rank == 0 else 0
win = MPI.Win.Allocate(
    size=win_size,
    disp_unit=itemsize,
    comm=comm,
)
if rank == 0:
    mem = np.frombuffer(win, dtype=np_dtype)
    mem[:] = np.arange(len(mem), dtype=np_dtype)
comm.Barrier()

buf = np.zeros(3, dtype=np_dtype)
target = (rank, 2, datatype)
win.Lock(rank=0)
win.Get(buf, target_rank=0, target=target)
win.Unlock(rank=0)
assert np.all(buf == [rank, rank+1, 0])
```

3.8 Wrapping with SWIG

- C source:

```
/* file: helloworld.c */
void sayhello(MPI_Comm comm)
{
  int size, rank;
  MPI_Comm_size(comm, &size);
  MPI_Comm_rank(comm, &rank);
```

(continues on next page)

(continued from previous page)

```
  printf("Hello, World! "
         "I am process %d of %d.\n",
         rank, size);
}
```

- SWIG interface file:

```
// file: helloworld.i
%module helloworld
%{
#include <mpi.h>
#include "helloworld.c"
}%

%include mpi4py/mpi4py.i
%mpi4py_typemap(Comm, MPI_Comm);
void sayhello(MPI_Comm comm);
```

- Try it in the Python prompt:

```
>>> from mpi4py import MPI
>>> import helloworld
>>> helloworld.sayhello(MPI.COMM_WORLD)
Hello, World! I am process 0 of 1.
```

3.9 Wrapping with F2Py

- Fortran 90 source:

```
! file: helloworld.f90
subroutine sayhello(comm)
  use mpi
  implicit none
  integer :: comm, rank, size, ierr
  call MPI_Comm_size(comm, size, ierr)
  call MPI_Comm_rank(comm, rank, ierr)
  print *, 'Hello, World! I am process ',rank,' of ',size,'.'
end subroutine sayhello
```

- Compiling example using f2py

```
$ f2py -c --f90exec=mpif90 helloworld.f90 -m helloworld
```

- Try it in the Python prompt:

```
>>> from mpi4py import MPI
>>> import helloworld
>>> fcomm = MPI.COMM_WORLD.py2f()
>>> helloworld.sayhello(fcomm)
Hello, World! I am process 0 of 1.
```

4 mpi4py

This is the **MPI for Python** package.

The *Message Passing Interface* (MPI) is a standardized and portable message-passing system designed to function on a wide variety of parallel computers. The MPI standard defines the syntax and semantics of library routines and allows users to write portable programs in the main scientific programming languages (Fortran, C, or C++). Since its release, the MPI specification has become the leading standard for message-passing libraries for parallel computers.

MPI for Python provides MPI bindings for the Python programming language, allowing any Python program to exploit multiple processors. This package build on the MPI specification and provides an object oriented interface which closely follows MPI-2 C++ bindings.

4.1 Runtime configuration options

`mpi4py.rc`

This object has attributes exposing runtime configuration options that become effective at import time of the *MPI* module.

Attributes Summary

initialize	Automatic MPI initialization at import
threads	Request initialization with thread support
thread_level	Level of thread support to request
finalize	Automatic MPI finalization at exit
fast_reduce	Use tree-based reductions for objects
recv_mprobe	Use matched probes to receive objects
errors	Error handling policy

Attributes Documentation

`mpi4py.rc.initialize`

Automatic MPI initialization at import.

Type `bool`

Default `True`

See also:

MPI4PY_RC_INITIALIZE

`mpi4py.rc.threads`

Request initialization with thread support.

Type `bool`

Default `True`

See also:

MPI4PY_RC_THREADS

`mpi4py.rc.thread_level`

Level of thread support to request.

Type `str`

Default `"multiple"`

Choices `"multiple"`, `"serialized"`, `"funneled"`, `"single"`

See also:

MPI4PY_RC_THREAD_LEVEL

`mpi4py.rc.`**`finalize`**

Automatic MPI finalization at exit.

Type `None` or `bool`

Default `None`

See also:

MPI4PY_RC_FINALIZE

`mpi4py.rc.`**`fast_reduce`**

Use tree-based reductions for objects.

Type `bool`

Default `True`

See also:

MPI4PY_RC_FAST_REDUCE

`mpi4py.rc.`**`recv_mprobe`**

Use matched probes to receive objects.

Type `bool`

Default `True`

See also:

MPI4PY_RC_RECV_MPROBE

`mpi4py.rc.`**`errors`**

Error handling policy.

Type `str`

Default `"exception"`

Choices `"exception"`, `"default"`, `"fatal"`

See also:

MPI4PY_RC_ERRORS

Example

MPI for Python features automatic initialization and finalization of the MPI execution environment. By using the *mpi4py.rc* object, MPI initialization and finalization can be handled programatically:

```
import mpi4py
mpi4py.rc.initialize = False  # do not initialize MPI automatically
mpi4py.rc.finalize = False    # do not finalize MPI automatically

from mpi4py import MPI # import the 'MPI' module

MPI.Init()      # manual initialization of the MPI environment
...             # your finest code here ...
MPI.Finalize()  # manual finalization of the MPI environment
```

4.2 Environment variables

The following environment variables override the corresponding attributes of the *mpi4py.rc* and *MPI.pickle* objects at import time of the *MPI* module.

Note: For variables of boolean type, accepted values are `0` and `1` (interpreted as `False` and `True`, respectively), and strings specifying a YAML boolean value (case-insensitive).

MPI4PY_RC_INITIALIZE

Type `bool`

Default `True`

Whether to automatically initialize MPI at import time of the *mpi4py.MPI* module.

See also:

mpi4py.rc.initialize

New in version 3.1.0.

MPI4PY_RC_FINALIZE

Type `None` | `bool`

Default `None`

Choices `None`, `True`, `False`

Whether to automatically finalize MPI at exit time of the Python process.

See also:

mpi4py.rc.finalize

New in version 3.1.0.

MPI4PY_RC_THREADS

Type `bool`

Default `True`

Whether to initialize MPI with thread support.

See also:

mpi4py.rc.threads

New in version 3.1.0.

MPI4PY_RC_THREAD_LEVEL

Default "multiple"

Choices "single", "funneled", "serialized", "multiple"

The level of required thread support.

See also:

mpi4py.rc.thread_level

New in version 3.1.0.

MPI4PY_RC_FAST_REDUCE

Type bool

Default True

Whether to use tree-based reductions for objects.

See also:

mpi4py.rc.fast_reduce

New in version 3.1.0.

MPI4PY_RC_RECV_MPROBE

Type bool

Default True

Whether to use matched probes to receive objects.

See also:

mpi4py.rc.recv_mprobe

MPI4PY_RC_ERRORS

Default "exception"

Choices "exception", "default", "fatal"

Controls default MPI error handling policy.

See also:

mpi4py.rc.errors

New in version 3.1.0.

MPI4PY_PICKLE_PROTOCOL

Type int

Default pickle.HIGHEST_PROTOCOL

Controls the default pickle protocol to use when communicating Python objects.

See also:

PROTOCOL attribute of the *MPI.pickle* object within the *MPI* module.

New in version 3.1.0.

MPI4PY_PICKLE_THRESHOLD

Type int

Default 262144

Controls the default buffer size threshold for switching from in-band to out-of-band buffer handling when using pickle protocol version 5 or higher.

See also:

Module *mpi4py.util.pkl5*.

New in version 3.1.2.

4.3 Miscellaneous functions

mpi4py.**profile**(*name*, *, *path=None*, *logfile=None*)

Support for the MPI profiling interface.

Parameters

- **name** (str) – Name of the profiler library to load.
- **path** (sequence of str, *optional*) – Additional paths to search for the profiler.
- **logfile** (str, *optional*) – Filename prefix for dumping profiler output.

Return type None

mpi4py.**get_config**()

Return a dictionary with information about MPI.

Return type Dict[str, str]

mpi4py.**get_include**()

Return the directory in the package that contains header files.

Extension modules that need to compile against mpi4py should use this function to locate the appropriate include directory. Using Python distutils (or perhaps NumPy distutils):

```
import mpi4py
Extension('extension_name', ...
          include_dirs=[..., mpi4py.get_include()])
```

Return type str

5 mpi4py.MPI

5.1 Classes

Ancillary

Datatype([datatype])	Datatype object
Status([status])	Status object
Request([request])	Request handle
Prequest([request])	Persistent request handle
Grequest([request])	Generalized request handle
Op([op])	Operation object
Group([group])	Group of processes
Info([info])	Info object

Communication

Comm([comm])	Communicator
Intracomm([comm])	Intracommunicator
Topocomm([comm])	Topology intracommunicator
Cartcomm([comm])	Cartesian topology intracommunicator
Graphcomm([comm])	General graph topology intracommunicator
Distgraphcomm([comm])	Distributed graph topology intracommunicator
Intercomm([comm])	Intercommunicator
Message([message])	Matched message handle

One-sided operations

Win([win])	Window handle

Input/Output

File([file])	File handle

Error handling

Errhandler([errhandler])	Error handler
Exception([ierr])	Exception class

Auxiliary

Pickle([dumps, loads, protocol])	Pickle/unpickle Python objects
memory(buf)	Memory buffer

5.2 Functions

Version inquiry

Get_version()	Obtain the version number of the MPI standard supported by the implementation as a tuple `(version, subversion)`
Get_library_version()	Obtain the version string of the MPI library

Initialization and finalization

Init()	Initialize the MPI execution environment
Init_thread([required])	Initialize the MPI execution environment
Finalize()	Terminate the MPI execution environment
Is_initialized()	Indicates whether *Init* has been called
Is_finalized()	Indicates whether *Finalize* has completed
Query_thread()	Return the level of thread support provided by the MPI library
Is_thread_main()	Indicate whether this thread called *Init* or *Init_thread*

Memory allocation

Alloc_mem(size[, info])	Allocate memory for message passing and RMA
Free_mem(mem)	Free memory allocated with *Alloc_mem()*

Address manipulation

Get_address(location)	Get the address of a location in memory
Aint_add(base, disp)	Return the sum of base address and displacement
Aint_diff(addr1, addr2)	Return the difference between absolute addresses

Timer

Wtick()	Return the resolution of *Wtime*
Wtime()	Return an elapsed time on the calling processor

Error handling

Get_error_class(errorcode)	Convert an *error code* into an *error class*
Get_error_string(errorcode)	Return the *error string* for a given *error class* or *error code*
Add_error_class()	Add an *error class* to the known error classes
Add_error_code(errorclass)	Add an *error code* to an *error class*
Add_error_string(errorcode, string)	Associate an *error string* with an *error class* or *error-code*

Dynamic process management

Open_port([info])	Return an address that can be used to establish connections between groups of MPI processes
Close_port(port_name)	Close a port
Publish_name(service_name, port_name[, info])	Publish a service name
Unpublish_name(service_name, port_name[, info])	Unpublish a service name
Lookup_name(service_name[, info])	Lookup a port name given a service name

Miscellanea

Attach_buffer(buf)	Attach a user-provided buffer for sending in buffered mode
Detach_buffer()	Remove an existing attached buffer
Compute_dims(nnodes, dims)	Return a balanced distribution of processes per coordinate direction
Get_processor_name()	Obtain the name of the calling processor
Register_datarep(datarep, read_fn, write_fn, ...)	Register user-defined data representations
Pcontrol(level)	Control profiling

Utilities

get_vendor()	Infomation about the underlying MPI implementation

5.3 Attributes

UNDEFINED	int UNDEFINED
ANY_SOURCE	int ANY_SOURCE
ANY_TAG	int ANY_TAG
PROC_NULL	int PROC_NULL
ROOT	int ROOT
BOTTOM	Bottom BOTTOM
IN_PLACE	InPlace IN_PLACE
KEYVAL_INVALID	int KEYVAL_INVALID
TAG_UB	int TAG_UB
HOST	int HOST
IO	int IO
WTIME_IS_GLOBAL	int WTIME_IS_GLOBAL
UNIVERSE_SIZE	int UNIVERSE_SIZE
APPNUM	int APPNUM
LASTUSEDCODE	int LASTUSEDCODE
WIN_BASE	int WIN_BASE
WIN_SIZE	int WIN_SIZE
WIN_DISP_UNIT	int WIN_DISP_UNIT
WIN_CREATE_FLAVOR	int WIN_CREATE_FLAVOR
WIN_FLAVOR	int WIN_FLAVOR
WIN_MODEL	int WIN_MODEL
SUCCESS	int SUCCESS
ERR_LASTCODE	int ERR_LASTCODE
ERR_COMM	int ERR_COMM
ERR_GROUP	int ERR_GROUP
ERR_TYPE	int ERR_TYPE
ERR_REQUEST	int ERR_REQUEST
ERR_OP	int ERR_OP
ERR_BUFFER	int ERR_BUFFER
ERR_COUNT	int ERR_COUNT
ERR_TAG	int ERR_TAG
ERR_RANK	int ERR_RANK
ERR_ROOT	int ERR_ROOT
ERR_TRUNCATE	int ERR_TRUNCATE
ERR_IN_STATUS	int ERR_IN_STATUS
ERR_PENDING	int ERR_PENDING
ERR_TOPOLOGY	int ERR_TOPOLOGY
ERR_DIMS	int ERR_DIMS
ERR_ARG	int ERR_ARG
ERR_OTHER	int ERR_OTHER
ERR_UNKNOWN	int ERR_UNKNOWN
ERR_INTERN	int ERR_INTERN
ERR_INFO	int ERR_INFO
ERR_FILE	int ERR_FILE
ERR_WIN	int ERR_WIN
ERR_KEYVAL	int ERR_KEYVAL
ERR_INFO_KEY	int ERR_INFO_KEY
ERR_INFO_VALUE	int ERR_INFO_VALUE

continues on next page

Table 16 – continued from previous page

ERR_INFO_NOKEY	int ERR_INFO_NOKEY
ERR_ACCESS	int ERR_ACCESS
ERR_AMODE	int ERR_AMODE
ERR_BAD_FILE	int ERR_BAD_FILE
ERR_FILE_EXISTS	int ERR_FILE_EXISTS
ERR_FILE_IN_USE	int ERR_FILE_IN_USE
ERR_NO_SPACE	int ERR_NO_SPACE
ERR_NO_SUCH_FILE	int ERR_NO_SUCH_FILE
ERR_IO	int ERR_IO
ERR_READ_ONLY	int ERR_READ_ONLY
ERR_CONVERSION	int ERR_CONVERSION
ERR_DUP_DATAREP	int ERR_DUP_DATAREP
ERR_UNSUPPORTED_DATAREP	int ERR_UNSUPPORTED_DATAREP
ERR_UNSUPPORTED_OPERATION	int ERR_UNSUPPORTED_OPERATION
ERR_NAME	int ERR_NAME
ERR_NO_MEM	int ERR_NO_MEM
ERR_NOT_SAME	int ERR_NOT_SAME
ERR_PORT	int ERR_PORT
ERR_QUOTA	int ERR_QUOTA
ERR_SERVICE	int ERR_SERVICE
ERR_SPAWN	int ERR_SPAWN
ERR_BASE	int ERR_BASE
ERR_SIZE	int ERR_SIZE
ERR_DISP	int ERR_DISP
ERR_ASSERT	int ERR_ASSERT
ERR_LOCKTYPE	int ERR_LOCKTYPE
ERR_RMA_CONFLICT	int ERR_RMA_CONFLICT
ERR_RMA_SYNC	int ERR_RMA_SYNC
ERR_RMA_RANGE	int ERR_RMA_RANGE
ERR_RMA_ATTACH	int ERR_RMA_ATTACH
ERR_RMA_SHARED	int ERR_RMA_SHARED
ERR_RMA_FLAVOR	int ERR_RMA_FLAVOR
ORDER_C	int ORDER_C
ORDER_F	int ORDER_F
ORDER_FORTRAN	int ORDER_FORTRAN
TYPECLASS_INTEGER	int TYPECLASS_INTEGER
TYPECLASS_REAL	int TYPECLASS_REAL
TYPECLASS_COMPLEX	int TYPECLASS_COMPLEX
DISTRIBUTE_NONE	int DISTRIBUTE_NONE
DISTRIBUTE_BLOCK	int DISTRIBUTE_BLOCK
DISTRIBUTE_CYCLIC	int DISTRIBUTE_CYCLIC
DISTRIBUTE_DFLT_DARG	int DISTRIBUTE_DFLT_DARG
COMBINER_NAMED	int COMBINER_NAMED
COMBINER_DUP	int COMBINER_DUP
COMBINER_CONTIGUOUS	int COMBINER_CONTIGUOUS
COMBINER_VECTOR	int COMBINER_VECTOR
COMBINER_HVECTOR	int COMBINER_HVECTOR
COMBINER_INDEXED	int COMBINER_INDEXED
COMBINER_HINDEXED	int COMBINER_HINDEXED
COMBINER_INDEXED_BLOCK	int COMBINER_INDEXED_BLOCK

continues on next page

Table 16 – continued from previous page

COMBINER_HINDEXED_BLOCK	int COMBINER_HINDEXED_BLOCK
COMBINER_STRUCT	int COMBINER_STRUCT
COMBINER_SUBARRAY	int COMBINER_SUBARRAY
COMBINER_DARRAY	int COMBINER_DARRAY
COMBINER_RESIZED	int COMBINER_RESIZED
COMBINER_F90_REAL	int COMBINER_F90_REAL
COMBINER_F90_COMPLEX	int COMBINER_F90_COMPLEX
COMBINER_F90_INTEGER	int COMBINER_F90_INTEGER
IDENT	int IDENT
CONGRUENT	int CONGRUENT
SIMILAR	int SIMILAR
UNEQUAL	int UNEQUAL
CART	int CART
GRAPH	int GRAPH
DIST_GRAPH	int DIST_GRAPH
UNWEIGHTED	int UNWEIGHTED
WEIGHTS_EMPTY	int WEIGHTS_EMPTY
COMM_TYPE_SHARED	int COMM_TYPE_SHARED
BSEND_OVERHEAD	int BSEND_OVERHEAD
WIN_FLAVOR_CREATE	int WIN_FLAVOR_CREATE
WIN_FLAVOR_ALLOCATE	int WIN_FLAVOR_ALLOCATE
WIN_FLAVOR_DYNAMIC	int WIN_FLAVOR_DYNAMIC
WIN_FLAVOR_SHARED	int WIN_FLAVOR_SHARED
WIN_SEPARATE	int WIN_SEPARATE
WIN_UNIFIED	int WIN_UNIFIED
MODE_NOCHECK	int MODE_NOCHECK
MODE_NOSTORE	int MODE_NOSTORE
MODE_NOPUT	int MODE_NOPUT
MODE_NOPRECEDE	int MODE_NOPRECEDE
MODE_NOSUCCEED	int MODE_NOSUCCEED
LOCK_EXCLUSIVE	int LOCK_EXCLUSIVE
LOCK_SHARED	int LOCK_SHARED
MODE_RDONLY	int MODE_RDONLY
MODE_WRONLY	int MODE_WRONLY
MODE_RDWR	int MODE_RDWR
MODE_CREATE	int MODE_CREATE
MODE_EXCL	int MODE_EXCL
MODE_DELETE_ON_CLOSE	int MODE_DELETE_ON_CLOSE
MODE_UNIQUE_OPEN	int MODE_UNIQUE_OPEN
MODE_SEQUENTIAL	int MODE_SEQUENTIAL
MODE_APPEND	int MODE_APPEND
SEEK_SET	int SEEK_SET
SEEK_CUR	int SEEK_CUR
SEEK_END	int SEEK_END
DISPLACEMENT_CURRENT	int DISPLACEMENT_CURRENT
DISP_CUR	int DISP_CUR
THREAD_SINGLE	int THREAD_SINGLE
THREAD_FUNNELED	int THREAD_FUNNELED
THREAD_SERIALIZED	int THREAD_SERIALIZED
THREAD_MULTIPLE	int THREAD_MULTIPLE

continues on next page

Table 16 – continued from previous page

VERSION	int VERSION
SUBVERSION	int SUBVERSION
MAX_PROCESSOR_NAME	int MAX_PROCESSOR_NAME
MAX_ERROR_STRING	int MAX_ERROR_STRING
MAX_PORT_NAME	int MAX_PORT_NAME
MAX_INFO_KEY	int MAX_INFO_KEY
MAX_INFO_VAL	int MAX_INFO_VAL
MAX_OBJECT_NAME	int MAX_OBJECT_NAME
MAX_DATAREP_STRING	int MAX_DATAREP_STRING
MAX_LIBRARY_VERSION_STRING	int MAX_LIBRARY_VERSION_STRING
DATATYPE_NULL	*Datatype* DATATYPE_NULL
UB	*Datatype* UB
LB	*Datatype* LB
PACKED	*Datatype* PACKED
BYTE	*Datatype* BYTE
AINT	*Datatype* AINT
OFFSET	*Datatype* OFFSET
COUNT	*Datatype* COUNT
CHAR	*Datatype* CHAR
WCHAR	*Datatype* WCHAR
SIGNED_CHAR	*Datatype* SIGNED_CHAR
SHORT	*Datatype* SHORT
INT	*Datatype* INT
LONG	*Datatype* LONG
LONG_LONG	*Datatype* LONG_LONG
UNSIGNED_CHAR	*Datatype* UNSIGNED_CHAR
UNSIGNED_SHORT	*Datatype* UNSIGNED_SHORT
UNSIGNED	*Datatype* UNSIGNED
UNSIGNED_LONG	*Datatype* UNSIGNED_LONG
UNSIGNED_LONG_LONG	*Datatype* UNSIGNED_LONG_LONG
FLOAT	*Datatype* FLOAT
DOUBLE	*Datatype* DOUBLE
LONG_DOUBLE	*Datatype* LONG_DOUBLE
C_BOOL	*Datatype* C_BOOL
INT8_T	*Datatype* INT8_T
INT16_T	*Datatype* INT16_T
INT32_T	*Datatype* INT32_T
INT64_T	*Datatype* INT64_T
UINT8_T	*Datatype* UINT8_T
UINT16_T	*Datatype* UINT16_T
UINT32_T	*Datatype* UINT32_T
UINT64_T	*Datatype* UINT64_T
C_COMPLEX	*Datatype* C_COMPLEX
C_FLOAT_COMPLEX	*Datatype* C_FLOAT_COMPLEX
C_DOUBLE_COMPLEX	*Datatype* C_DOUBLE_COMPLEX
C_LONG_DOUBLE_COMPLEX	*Datatype* C_LONG_DOUBLE_COMPLEX
CXX_BOOL	*Datatype* CXX_BOOL
CXX_FLOAT_COMPLEX	*Datatype* CXX_FLOAT_COMPLEX
CXX_DOUBLE_COMPLEX	*Datatype* CXX_DOUBLE_COMPLEX
CXX_LONG_DOUBLE_COMPLEX	*Datatype* CXX_LONG_DOUBLE_COMPLEX

continues on next page

Table 16 – continued from previous page

SHORT_INT	*Datatype* SHORT_INT
INT_INT	*Datatype* INT_INT
TWOINT	*Datatype* TWOINT
LONG_INT	*Datatype* LONG_INT
FLOAT_INT	*Datatype* FLOAT_INT
DOUBLE_INT	*Datatype* DOUBLE_INT
LONG_DOUBLE_INT	*Datatype* LONG_DOUBLE_INT
CHARACTER	*Datatype* CHARACTER
LOGICAL	*Datatype* LOGICAL
INTEGER	*Datatype* INTEGER
REAL	*Datatype* REAL
DOUBLE_PRECISION	*Datatype* DOUBLE_PRECISION
COMPLEX	*Datatype* COMPLEX
DOUBLE_COMPLEX	*Datatype* DOUBLE_COMPLEX
LOGICAL1	*Datatype* LOGICAL1
LOGICAL2	*Datatype* LOGICAL2
LOGICAL4	*Datatype* LOGICAL4
LOGICAL8	*Datatype* LOGICAL8
INTEGER1	*Datatype* INTEGER1
INTEGER2	*Datatype* INTEGER2
INTEGER4	*Datatype* INTEGER4
INTEGER8	*Datatype* INTEGER8
INTEGER16	*Datatype* INTEGER16
REAL2	*Datatype* REAL2
REAL4	*Datatype* REAL4
REAL8	*Datatype* REAL8
REAL16	*Datatype* REAL16
COMPLEX4	*Datatype* COMPLEX4
COMPLEX8	*Datatype* COMPLEX8
COMPLEX16	*Datatype* COMPLEX16
COMPLEX32	*Datatype* COMPLEX32
UNSIGNED_INT	*Datatype* UNSIGNED_INT
SIGNED_SHORT	*Datatype* SIGNED_SHORT
SIGNED_INT	*Datatype* SIGNED_INT
SIGNED_LONG	*Datatype* SIGNED_LONG
SIGNED_LONG_LONG	*Datatype* SIGNED_LONG_LONG
BOOL	*Datatype* BOOL
SINT8_T	*Datatype* SINT8_T
SINT16_T	*Datatype* SINT16_T
SINT32_T	*Datatype* SINT32_T
SINT64_T	*Datatype* SINT64_T
F_BOOL	*Datatype* F_BOOL
F_INT	*Datatype* F_INT
F_FLOAT	*Datatype* F_FLOAT
F_DOUBLE	*Datatype* F_DOUBLE
F_COMPLEX	*Datatype* F_COMPLEX
F_FLOAT_COMPLEX	*Datatype* F_FLOAT_COMPLEX
F_DOUBLE_COMPLEX	*Datatype* F_DOUBLE_COMPLEX
REQUEST_NULL	*Request* REQUEST_NULL
MESSAGE_NULL	*Message* MESSAGE_NULL

continues on next page

Table 16 – continued from previous page

MESSAGE_NO_PROC	*Message* MESSAGE_NO_PROC
OP_NULL	*Op* OP_NULL
MAX	*Op* MAX
MIN	*Op* MIN
SUM	*Op* SUM
PROD	*Op* PROD
LAND	*Op* LAND
BAND	*Op* BAND
LOR	*Op* LOR
BOR	*Op* BOR
LXOR	*Op* LXOR
BXOR	*Op* BXOR
MAXLOC	*Op* MAXLOC
MINLOC	*Op* MINLOC
REPLACE	*Op* REPLACE
NO_OP	*Op* NO_OP
GROUP_NULL	*Group* GROUP_NULL
GROUP_EMPTY	*Group* GROUP_EMPTY
INFO_NULL	*Info* INFO_NULL
INFO_ENV	*Info* INFO_ENV
ERRHANDLER_NULL	*Errhandler* ERRHANDLER_NULL
ERRORS_RETURN	*Errhandler* ERRORS_RETURN
ERRORS_ARE_FATAL	*Errhandler* ERRORS_ARE_FATAL
COMM_NULL	*Comm* COMM_NULL
COMM_SELF	*Intracomm* COMM_SELF
COMM_WORLD	*Intracomm* COMM_WORLD
WIN_NULL	*Win* WIN_NULL
FILE_NULL	*File* FILE_NULL
pickle	*Pickle* pickle

6 mpi4py.futures

New in version 3.0.0.

This package provides a high-level interface for asynchronously executing callables on a pool of worker processes using MPI for inter-process communication.

6.1 concurrent.futures

The *mpi4py.futures* package is based on `concurrent.futures` from the Python standard library. More precisely, *mpi4py.futures* provides the *MPIPoolExecutor* class as a concrete implementation of the abstract class `Executor`. The `submit()` interface schedules a callable to be executed asynchronously and returns a `Future` object representing the execution of the callable. `Future` instances can be queried for the call result or exception. Sets of `Future` instances can be passed to the `wait()` and `as_completed()` functions.

Note: The `concurrent.futures` package was introduced in Python 3.2. A backport targeting Python 2.7 is available on PyPI. The *mpi4py.futures* package uses `concurrent.futures` if available, either from the Python 3 standard library or the Python 2.7 backport if installed. Otherwise, *mpi4py.futures* uses a bundled copy of core functionality backported from Python 3.5 to work with Python 2.7.

See also:

Module `concurrent.futures` Documentation of the `concurrent.futures` standard module.

6.2 MPIPoolExecutor

The *MPIPoolExecutor* class uses a pool of MPI processes to execute calls asynchronously. By performing computations in separate processes, it allows to side-step the global interpreter lock but also means that only picklable objects can be executed and returned. The `__main__` module must be importable by worker processes, thus *MPIPoolExecutor* instances may not work in the interactive interpreter.

MPIPoolExecutor takes advantage of the dynamic process management features introduced in the MPI-2 standard. In particular, the *MPI.Intracomm.Spawn* method of *MPI.COMM_SELF* is used in the master (or parent) process to spawn new worker (or child) processes running a Python interpreter. The master process uses a separate thread (one for each *MPIPoolExecutor* instance) to communicate back and forth with the workers. The worker processes serve the execution of tasks in the main (and only) thread until they are signaled for completion.

Note: The worker processes must import the main script in order to *unpickle* any callable defined in the `__main__` module and submitted from the master process. Furthermore, the callables may need access to other global variables. At the worker processes, *mpi4py.futures* executes the main script code (using the `runpy` module) under the `__worker__` namespace to define the `__main__` module. The `__main__` and `__worker__` modules are added to `sys.modules` (both at the master and worker processes) to ensure proper *pickling* and *unpickling*.

Warning: During the initial import phase at the workers, the main script cannot create and use new *MPIPoolExecutor* instances. Otherwise, each worker would attempt to spawn a new pool of workers, leading to infinite recursion. *mpi4py.futures* detects such recursive attempts to spawn new workers and aborts the MPI execution environment. As the main script code is run under the `__worker__` namespace, the easiest way to avoid spawn recursion is using the idiom `if __name__ == '__main__': ...` in the main script.

class mpi4py.futures.**MPIPoolExecutor**(*max_workers=None*, *initializer=None*, *initargs=()*, ***kwargs*)

An `Executor` subclass that executes calls asynchronously using a pool of at most *max_workers* processes. If *max_workers* is `None` or not given, its value is determined from the *MPI4PY_FUTURES_MAX_WORKERS* environment variable if set, or the MPI universe size if set, otherwise a single worker process is spawned. If *max_workers* is lower than or equal to `0`, then a `ValueError` will be raised.

initializer is an optional callable that is called at the start of each worker process before executing any tasks; *initargs* is a tuple of arguments passed to the initializer. If *initializer* raises an exception, all pending tasks and any attempt to submit new tasks to the pool will raise a `BrokenExecutor` exception.

Other parameters:

- *python_exe*: Path to the Python interpreter executable used to spawn worker processes, otherwise `sys.executable` is used.
- *python_args*: `list` or iterable with additional command line flags to pass to the Python executable. Command line flags determined from inspection of `sys.flags`, `sys.warnoptions` and `sys._xoptions` in are passed unconditionally.
- *mpi_info*: `dict` or iterable yielding `(key, value)` pairs. These `(key, value)` pairs are passed (through an *MPI.Info* object) to the *MPI.Intracomm.Spawn* call used to spawn worker processes. This mechanism

allows telling the MPI runtime system where and how to start the processes. Check the documentation of the backend MPI implementation about the set of keys it interprets and the corresponding format for values.

- *globals*: `dict` or iterable yielding `(name, value)` pairs to initialize the main module namespace in worker processes.
- *main*: If set to `False`, do not import the `__main__` module in worker processes. Setting *main* to `False` prevents worker processes from accessing definitions in the parent `__main__` namespace.
- *path*: `list` or iterable with paths to append to `sys.path` in worker processes to extend the module search path.
- *wdir*: Path to set the current working directory in worker processes using `os.chdir()`. The initial working directory is set by the MPI implementation. Quality MPI implementations should honor a `wdir` info key passed through *mpi_info*, although such feature is not mandatory.
- *env*: `dict` or iterable yielding `(name, value)` pairs with environment variables to update `os.environ` in worker processes. The initial environment is set by the MPI implementation. MPI implementations may allow setting the initial environment through *mpi_info*, however such feature is not required nor recommended by the MPI standard.

submit(*func*, **args*, ***kwargs*)

Schedule the callable, *func*, to be executed as `func(*args, **kwargs)` and returns a `Future` object representing the execution of the callable.

```
executor = MPIPoolExecutor(max_workers=1)
future = executor.submit(pow, 321, 1234)
print(future.result())
```

map(*func*, **iterables*, *timeout=None*, *chunksize=1*, ***kwargs*)

Equivalent to `map(func, *iterables)` except *func* is executed asynchronously and several calls to *func* may be made concurrently, out-of-order, in separate processes. The returned iterator raises a `TimeoutError` if `__next__()` is called and the result isn't available after *timeout* seconds from the original call to `map()`. *timeout* can be an int or a float. If *timeout* is not specified or `None`, there is no limit to the wait time. If a call raises an exception, then that exception will be raised when its value is retrieved from the iterator. This method chops *iterables* into a number of chunks which it submits to the pool as separate tasks. The (approximate) size of these chunks can be specified by setting *chunksize* to a positive integer. For very long iterables, using a large value for *chunksize* can significantly improve performance compared to the default size of one. By default, the returned iterator yields results in-order, waiting for successive tasks to complete . This behavior can be changed by passing the keyword argument *unordered* as `True`, then the result iterator will yield a result as soon as any of the tasks complete.

```
executor = MPIPoolExecutor(max_workers=3)
for result in executor.map(pow, [2]*32, range(32)):
    print(result)
```

starmap(*func*, *iterable*, *timeout=None*, *chunksize=1*, ***kwargs*)

Equivalent to `itertools.starmap(func, iterable)`. Used instead of `map()` when argument parameters are already grouped in tuples from a single iterable (the data has been "pre-zipped"). `map(func, *iterable)` is equivalent to `starmap(func, zip(*iterable))`.

```
executor = MPIPoolExecutor(max_workers=3)
iterable = ((2, n) for n in range(32))
for result in executor.starmap(pow, iterable):
    print(result)
```

shutdown(*wait=True*, *cancel_futures=False*)

Signal the executor that it should free any resources that it is using when the currently pending futures are done executing. Calls to *submit()* and *map()* made after *shutdown()* will raise RuntimeError.

If *wait* is True then this method will not return until all the pending futures are done executing and the resources associated with the executor have been freed. If *wait* is False then this method will return immediately and the resources associated with the executor will be freed when all pending futures are done executing. Regardless of the value of *wait*, the entire Python program will not exit until all pending futures are done executing.

If *cancel_futures* is True, this method will cancel all pending futures that the executor has not started running. Any futures that are completed or running won't be cancelled, regardless of the value of *cancel_futures*.

You can avoid having to call this method explicitly if you use the with statement, which will shutdown the executor instance (waiting as if *shutdown()* were called with *wait* set to True).

```
import time
with MPIPoolExecutor(max_workers=1) as executor:
    future = executor.submit(time.sleep, 2)
assert future.done()
```

bootup(*wait=True*)
Signal the executor that it should allocate eagerly any required resources (in particular, MPI worker processes). If *wait* is True, then *bootup()* will not return until the executor resources are ready to process submissions. Resources are automatically allocated in the first call to *submit()*, thus calling *bootup()* explicitly is seldom needed.

MPI4PY_FUTURES_MAX_WORKERS
If the *max_workers* parameter to *MPIPoolExecutor* is None or not given, the *MPI4PY_FUTURES_MAX_WORKERS* environment variable provides fallback value for the maximum number of MPI worker processes to spawn.

Note: As the master process uses a separate thread to perform MPI communication with the workers, the backend MPI implementation should provide support for *MPI.THREAD_MULTIPLE*. However, some popular MPI implementations do not support yet concurrent MPI calls from multiple threads. Additionally, users may decide to initialize MPI with a lower level of thread support. If the level of thread support in the backend MPI is less than *MPI.THREAD_MULTIPLE*, *mpi4py.futures* will use a global lock to serialize MPI calls. If the level of thread support is less than *MPI.THREAD_SERIALIZED*, *mpi4py.futures* will emit a RuntimeWarning.

Warning: If the level of thread support in the backend MPI is less than *MPI.THREAD_SERIALIZED* (i.e, it is either *MPI.THREAD_SINGLE* or *MPI.THREAD_FUNNELED*), in theory *mpi4py.futures* cannot be used. Rather than raising an exception, *mpi4py.futures* emits a warning and takes a "cross-fingers" attitude to continue execution in the hope that serializing MPI calls with a global lock will actually work.

6.3 MPICommExecutor

Legacy MPI-1 implementations (as well as some vendor MPI-2 implementations) do not support the dynamic process management features introduced in the MPI-2 standard. Additionally, job schedulers and batch systems in supercomputing facilities may pose additional complications to applications using the MPI_Comm_spawn() routine.

With these issues in mind, *mpi4py.futures* supports an additonal, more traditional, SPMD-like usage pattern requiring MPI-1 calls only. Python applications are started the usual way, e.g., using the **mpiexec** command. Python code should make a collective call to the *MPICommExecutor* context manager to partition the set of MPI processes within a MPI communicator in one master processes and many workers processes. The master process gets access to an *MPIPoolExecutor* instance to submit tasks. Meanwhile, the worker process follow a different execution path and team-up to execute the tasks submitted from the master.

Besides alleviating the lack of dynamic process managment features in legacy MPI-1 or partial MPI-2 implementations, the *MPICommExecutor* context manager may be useful in classic MPI-based Python applications willing to take advantage of the simple, task-based, master/worker approach available in the *mpi4py.futures* package.

class mpi4py.futures.**MPICommExecutor**(*comm=None*, *root=0*)

Context manager for *MPIPoolExecutor*. This context manager splits a MPI (intra)communicator *comm* (defaults to *MPI.COMM_WORLD* if not provided or `None`) in two disjoint sets: a single master process (with rank *root* in *comm*) and the remaining worker processes. These sets are then connected through an intercommunicator. The target of the `with` statement is assigned either an *MPIPoolExecutor* instance (at the master) or `None` (at the workers).

```
from mpi4py import MPI
from mpi4py.futures import MPICommExecutor

with MPICommExecutor(MPI.COMM_WORLD, root=0) as executor:
    if executor is not None:
       future = executor.submit(abs, -42)
       assert future.result() == 42
       answer = set(executor.map(abs, [-42, 42]))
       assert answer == {42}
```

Warning: If *MPICommExecutor* is passed a communicator of size one (e.g., *MPI.COMM_SELF*), then the executor instace assigned to the target of the `with` statement will execute all submitted tasks in a single worker thread, thus ensuring that task execution still progress asynchronously. However, the *GIL* will prevent the main and worker threads from running concurrently in multicore processors. Moreover, the thread context switching may harm noticeably the performance of CPU-bound tasks. In case of I/O-bound tasks, the *GIL* is not usually an issue, however, as a single worker thread is used, it progress one task at a time. We advice against using *MPICommExecutor* with communicators of size one and suggest refactoring your code to use instead a `ThreadPoolExecutor`.

6.4 Command line

Recalling the issues related to the lack of support for dynamic process managment features in MPI implementations, *mpi4py.futures* supports an alternative usage pattern where Python code (either from scripts, modules, or zip files) is run under command line control of the *mpi4py.futures* package by passing `-m mpi4py.futures` to the **python** executable. The `mpi4py.futures` invocation should be passed a *pyfile* path to a script (or a zipfile/directory containing a `__main__.py` file). Additionally, `mpi4py.futures` accepts `-m mod` to execute a module named *mod*, `-c cmd` to execute a command string *cmd*, or even `-` to read commands from standard input (`sys.stdin`). Summarizing, `mpi4py.futures` can be invoked in the following ways:

- `$ mpiexec -n numprocs python -m mpi4py.futures pyfile [arg] ...`
- `$ mpiexec -n numprocs python -m mpi4py.futures -m mod [arg] ...`
- `$ mpiexec -n numprocs python -m mpi4py.futures -c cmd [arg] ...`
- `$ mpiexec -n numprocs python -m mpi4py.futures - [arg] ...`

Before starting the main script execution, *mpi4py.futures* splits *MPI.COMM_WORLD* in one master (the process with rank 0 in *MPI.COMM_WORLD*) and *numprocs - 1* workers and connects them through an MPI intercommunicator. Afterwards, the master process proceeds with the execution of the user script code, which eventually creates *MPIPoolExecutor* instances to submit tasks. Meanwhile, the worker processes follow a different execution path to serve the master. Upon successful termination of the main script at the master, the entire MPI execution environment exists gracefully. In case of any unhandled exception in the main script, the master process calls `MPI.COMM_WORLD.Abort(1)` to prevent deadlocks and force termination of entire MPI execution environment.

> **Warning:** Running scripts under command line control of *mpi4py.futures* is quite similar to executing a single-process application that spawn additional workers as required. However, there is a very important difference users should be aware of. All *MPIPoolExecutor* instances created at the master will share the pool of workers. Tasks submitted at the master from many different executors will be scheduled for execution in random order as soon as a worker is idle. Any executor can easily starve all the workers (e.g., by calling *MPIPoolExecutor.map()* with long iterables). If that ever happens, submissions from other executors will not be serviced until free workers are available.

See also:

Command line Documentation on Python command line interface.

6.5 Examples

The following `julia.py` script computes the Julia set and dumps an image to disk in binary PGM format. The code starts by importing *MPIPoolExecutor* from the *mpi4py.futures* package. Next, some global constants and functions implement the computation of the Julia set. The computations are protected with the standard `if __name__ == '__main__': ...` idiom. The image is computed by whole scanlines submitting all these tasks at once using the *map* method. The result iterator yields scanlines in-order as the tasks complete. Finally, each scanline is dumped to disk.

Listing 1: `julia.py`

```
from mpi4py.futures import MPIPoolExecutor

x0, x1, w = -2.0, +2.0, 640*2
y0, y1, h = -1.5, +1.5, 480*2
dx = (x1 - x0) / w
dy = (y1 - y0) / h

c = complex(0, 0.65)

def julia(x, y):
    z = complex(x, y)
    n = 255
    while abs(z) < 3 and n > 1:
        z = z**2 + c
        n -= 1
    return n

def julia_line(k):
    line = bytearray(w)
    y = y1 - k * dy
    for j in range(w):
        x = x0 + j * dx
        line[j] = julia(x, y)
    return line

if __name__ == '__main__':
```

(continues on next page)

(continued from previous page)

```python

    with MPIPoolExecutor() as executor:
        image = executor.map(julia_line, range(h))
        with open('julia.pgm', 'wb') as f:
            f.write(b'P5 %d %d %d\n' % (w, h, 255))
            for line in image:
                f.write(line)
```

The recommended way to execute the script is by using the **mpiexec** command specifying one MPI process (master) and (optional but recommended) the desired MPI universe size, which determines the number of additional dynamically spawned processes (workers). The MPI universe size is provided either by a batch system or set by the user via command-line arguments to **mpiexec** or environment variables. Below we provide examples for MPICH and Open MPI implementations[1]. In all of these examples, the **mpiexec** command launches a single master process running the Python interpreter and executing the main script. When required, *mpi4py.futures* spawns the pool of 16 worker processes. The master submits tasks to the workers and waits for the results. The workers receive incoming tasks, execute them, and send back the results to the master.

When using MPICH implementation or its derivatives based on the Hydra process manager, users can set the MPI universe size via the `-usize` argument to **mpiexec**:

```
$ mpiexec -n 1 -usize 17 python julia.py
```

or, alternatively, by setting the `MPIEXEC_UNIVERSE_SIZE` environment variable:

```
$ MPIEXEC_UNIVERSE_SIZE=17 mpiexec -n 1 python julia.py
```

In the Open MPI implementation, the MPI universe size can be set via the `-host` argument to **mpiexec**:

```
$ mpiexec -n 1 -host <hostname>:17 python julia.py
```

Another way to specify the number of workers is to use the *mpi4py.futures*-specific environment variable *MPI4PY_FUTURES_MAX_WORKERS*:

```
$ MPI4PY_FUTURES_MAX_WORKERS=16 mpiexec -n 1 python julia.py
```

Note that in this case, the MPI universe size is ignored.

Alternatively, users may decide to execute the script in a more traditional way, that is, all the MPI processes are started at once. The user script is run under command-line control of *mpi4py.futures* passing the -m flag to the **python** executable:

```
$ mpiexec -n 17 python -m mpi4py.futures julia.py
```

As explained previously, the 17 processes are partitioned in one master and 16 workers. The master process executes the main script while the workers execute the tasks submitted by the master.

GIL See global interpreter lock.

[1] When using an MPI implementation other than MPICH or Open MPI, please check the documentation of the implementation and/or batch system for the ways to specify the desired MPI universe size.

7 mpi4py.util

New in version 3.1.0.

The *mpi4py.util* package collects miscellaneous utilities within the intersection of Python and MPI.

7.1 mpi4py.util.pkl5

New in version 3.1.0.

pickle protocol 5 (see **PEP 574**) introduced support for out-of-band buffers, allowing for more efficient handling of certain object types with large memory footprints.

MPI for Python uses the traditional in-band handling of buffers. This approach is appropriate for communicating non-buffer Python objects, or buffer-like objects with small memory footprints. For point-to-point communication, in-band buffer handling allows for the communication of a pickled stream with a single MPI message, at the expense of additional CPU and memory overhead in the pickling and unpickling steps.

The *mpi4py.util.pkl5* module provides communicator wrapper classes reimplementing pickle-based point-to-point communication methods using pickle protocol 5. Handling out-of-band buffers necessarily involve multiple MPI messages, thus increasing latency and hurting performance in case of small size data. However, in case of large size data, the zero-copy savings of out-of-band buffer handling more than offset the extra latency costs. Additionally, these wrapper methods overcome the infamous 2 GiB message count limit (MPI-1 to MPI-3).

Note: Support for pickle protocol 5 is available in the pickle module within the Python standard library since Python 3.8. Previous Python 3 releases can use the pickle5 backport, which is available on PyPI and can be installed with:

```
python -m pip install pickle5
```

class mpi4py.util.pkl5.**Request**(*request=None*)

Request.

Custom request class for nonblocking communications.

Note: *Request* is not a subclass of *mpi4py.MPI.Request*

Parameters **request** (*Iterable[MPI.Request]*) –

Return type *Request*

Free()

Free a communication request.

Return type None

cancel()

Cancel a communication request.

Return type None

get_status(*status=None*)

Non-destructive test for the completion of a request.

Parameters **status** (*Optional[Status]*) –

Return type bool

test(*status=None*)
Test for the completion of a request.

Parameters **status** (*Optional[Status]*) –

Return type Tuple[bool, Optional[Any]]

wait(*status=None*)
Wait for a request to complete.

Parameters **status** (*Optional[Status]*) –

Return type Any

classmethod testall(*requests*, *statuses=None*)
Test for the completion of all requests.

Classmethod

classmethod waitall(*requests*, *statuses=None*)
Wait for all requests to complete.

Classmethod

class mpi4py.util.pkl5.Message(*message=None*)
Message.

Custom message class for matching probes.

Note: *Message* is not a subclass of *mpi4py.MPI.Message*

Parameters **message** (*Iterable[MPI.Message]*) –

Return type *Message*

recv(*status=None*)
Blocking receive of matched message.

Parameters **status** (*Optional[Status]*) –

Return type Any

irecv()
Nonblocking receive of matched message.

Return type *Request*

classmethod probe(*comm*, *source=ANY_SOURCE*, *tag=ANY_TAG*, *status=None*)
Blocking test for a matched message.

Classmethod

classmethod iprobe(*comm*, *source=ANY_SOURCE*, *tag=ANY_TAG*, *status=None*)
Nonblocking test for a matched message.

Classmethod

class mpi4py.util.pkl5.Comm
Communicator.

Base communicator wrapper class.

send(*obj*, *dest*, *tag=0*)
Blocking send in standard mode.

Parameters

- **obj** (*Any*) –
- **dest** (*int*) –
- **tag** (*int*) –

Return type None

`bsend`(*obj*, *dest*, *tag=0*)
Blocking send in buffered mode.

Parameters

- **obj** (*Any*) –
- **dest** (*int*) –
- **tag** (*int*) –

Return type None

`ssend`(*obj*, *dest*, *tag=0*)
Blocking send in synchronous mode.

Parameters

- **obj** (*Any*) –
- **dest** (*int*) –
- **tag** (*int*) –

Return type None

`isend`(*obj*, *dest*, *tag=0*)
Nonblocking send in standard mode.

Parameters

- **obj** (*Any*) –
- **dest** (*int*) –
- **tag** (*int*) –

Return type *Request*

`ibsend`(*obj*, *dest*, *tag=0*)
Nonblocking send in buffered mode.

Parameters

- **obj** (*Any*) –
- **dest** (*int*) –
- **tag** (*int*) –

Return type *Request*

`issend`(*obj*, *dest*, *tag=0*)
Nonblocking send in synchronous mode.

Parameters

- **obj** (*Any*) –
- **dest** (*int*) –

- **tag** (*int*) –

Return type *Request*

recv(*buf=None*, *source=ANY_SOURCE*, *tag=ANY_TAG*, *status=None*)

Blocking receive.

Parameters

- **buf** (*Optional[Buffer]*) –
- **source** (*int*) –
- **tag** (*int*) –
- **status** (*Optional[Status]*) –

Return type Any

irecv(*buf=None*, *source=ANY_SOURCE*, *tag=ANY_TAG*)

Nonblocking receive.

> **Warning:** This method cannot be supported reliably and raises RuntimeError.

Parameters

- **buf** (*Optional[Buffer]*) –
- **source** (*int*) –
- **tag** (*int*) –

Return type *Request*

sendrecv(*sendobj*, *dest*, *sendtag=0*, *recvbuf=None*, *source=ANY_SOURCE*, *recvtag=ANY_TAG*, *status=None*)

Send and receive.

Parameters

- **sendobj** (*Any*) –
- **dest** (*int*) –
- **sendtag** (*int*) –
- **recvbuf** (*Optional[Buffer]*) –
- **source** (*int*) –
- **recvtag** (*int*) –
- **status** (*Optional[Status]*) –

Return type Any

mprobe(*source=ANY_SOURCE*, *tag=ANY_TAG*, *status=None*)

Blocking test for a matched message.

Parameters

- **source** (*int*) –
- **tag** (*int*) –
- **status** (*Optional[Status]*) –

Return type *Message*

improbe(*source=ANY_SOURCE*, *tag=ANY_TAG*, *status=None*)
Nonblocking test for a matched message.

Parameters

- **source** (*int*) –
- **tag** (*int*) –
- **status** (*Optional[Status]*) –

Return type Optional[*Message*]

bcast(*obj*, *root=0*)
Broadcast.

Parameters

- **obj** (*Any*) –
- **root** (*int*) –

Return type Any

class mpi4py.util.pkl5.**Intracomm**
Intracommunicator.

Intracommunicator wrapper class.

class mpi4py.util.pkl5.**Intercomm**
Intercommunicator.

Intercommunicator wrapper class.

Examples

Listing 2: test-pkl5-1.py

```python
import numpy as np
from mpi4py import MPI
from mpi4py.util import pkl5

comm = pkl5.Intracomm(MPI.COMM_WORLD)  # comm wrapper
size = comm.Get_size()
rank = comm.Get_rank()
dst = (rank + 1) % size
src = (rank - 1) % size

sobj = np.full(1024**3, rank, dtype='i4')  # > 4 GiB
sreq = comm.isend(sobj, dst, tag=42)
robj = comm.recv (None, src, tag=42)
sreq.Free()

assert np.min(robj) == src
assert np.max(robj) == src
```

Listing 3: `test-pkl5-2.py`

```
import numpy as np
from mpi4py import MPI
from mpi4py.util import pkl5

comm = pkl5.Intracomm(MPI.COMM_WORLD)  # comm wrapper
size = comm.Get_size()
rank = comm.Get_rank()
dst = (rank + 1) % size
src = (rank - 1) % size

sobj = np.full(1024**3, rank, dtype='i4')  # > 4 GiB
sreq = comm.isend(sobj, dst, tag=42)

status = MPI.Status()
rmsg = comm.mprobe(status=status)
assert status.Get_source() == src
assert status.Get_tag() == 42
rreq = rmsg.irecv()
robj = rreq.wait()

sreq.Free()
assert np.max(robj) == src
assert np.min(robj) == src
```

7.2 mpi4py.util.dtlib

New in version 3.1.0.

The *mpi4py.util.dtlib* module provides converter routines between NumPy and MPI datatypes.

mpi4py.util.dtlib.**from_numpy_dtype**(*dtype*)

Convert NumPy datatype to MPI datatype.

Parameters **dtype** (*numpy.typing.DTypeLike*) – NumPy dtype-like object.

Return type *Datatype*

mpi4py.util.dtlib.**to_numpy_dtype**(*datatype*)

Convert MPI datatype to NumPy datatype.

Parameters **datatype** (Datatype) – MPI datatype.

Return type numpy.dtype

8 mpi4py.run

New in version 3.0.0.

At import time, *mpi4py* initializes the MPI execution environment calling `MPI_Init_thread()` and installs an exit hook to automatically call `MPI_Finalize()` just before the Python process terminates. Additionally, *mpi4py* overrides the default *ERRORS_ARE_FATAL* error handler in favor of *ERRORS_RETURN*, which allows translating MPI errors in Python exceptions. These departures from standard MPI behavior may be controversial, but are quite convenient within the highly dynamic Python programming environment. Third-party code using *mpi4py* can just `from mpi4py import MPI` and perform MPI calls without the tedious initialization/finalization handling. MPI errors, once translated automatically to Python exceptions, can be dealt with the common `try`…`except`…`finally` clauses; unhandled MPI exceptions will print a traceback which helps in locating problems in source code.

Unfortunately, the interplay of automatic MPI finalization and unhandled exceptions may lead to deadlocks. In unattended runs, these deadlocks will drain the battery of your laptop, or burn precious allocation hours in your supercomputing facility.

Consider the following snippet of Python code. Assume this code is stored in a standard Python script file and run with **`mpiexec`** in two or more processes.

```
from mpi4py import MPI
assert MPI.COMM_WORLD.Get_size() > 1
rank = MPI.COMM_WORLD.Get_rank()
if rank == 0:
    1/0
    MPI.COMM_WORLD.send(None, dest=1, tag=42)
elif rank == 1:
    MPI.COMM_WORLD.recv(source=0, tag=42)
```

Process 0 raises `ZeroDivisionError` exception before performing a send call to process 1. As the exception is not handled, the Python interpreter running in process 0 will proceed to exit with non-zero status. However, as *mpi4py* installed a finalizer hook to call `MPI_Finalize()` before exit, process 0 will block waiting for other processes to also enter the `MPI_Finalize()` call. Meanwhile, process 1 will block waiting for a message to arrive from process 0, thus never reaching to `MPI_Finalize()`. The whole MPI execution environment is irremediably in a deadlock state.

To alleviate this issue, *mpi4py* offers a simple, alternative command line execution mechanism based on using the -m flag and implemented with the `runpy` module. To use this features, Python code should be run passing `-m mpi4py` in the command line invoking the Python interpreter. In case of unhandled exceptions, the finalizer hook will call `MPI_Abort()` on the `MPI_COMM_WORLD` communicator, thus effectively aborting the MPI execution environment.

> **Warning:** When a process is forced to abort, resources (e.g. open files) are not cleaned-up and any registered finalizers (either with the `atexit` module, the Python C/API function `Py_AtExit()`, or even the C standard library function `atexit()`) will not be executed. Thus, aborting execution is an extremely impolite way of ensuring process termination. However, MPI provides no other mechanism to recover from a deadlock state.

8.1 Interface options

The use of `-m mpi4py` to execute Python code on the command line resembles that of the Python interpreter.

- `mpiexec -n` *`numprocs`* `python -m mpi4py` *`pyfile`* `[arg] ...`
- `mpiexec -n` *`numprocs`* `python -m mpi4py -m` *`mod`* `[arg] ...`
- `mpiexec -n` *`numprocs`* `python -m mpi4py -c` *`cmd`* `[arg] ...`
- `mpiexec -n` *`numprocs`* `python -m mpi4py - [arg] ...`

`<pyfile>`

Execute the Python code contained in *pyfile*, which must be a filesystem path referring to either a Python file, a directory containing a `__main__.py` file, or a zipfile containing a `__main__.py` file.

`-m` `<mod>`

Search `sys.path` for the named module *mod* and execute its contents.

`-c` `<cmd>`

Execute the Python code in the *cmd* string command.

`-`

Read commands from standard input (`sys.stdin`).

See also:

Command line Documentation on Python command line interface.

9 Reference

`mpi4py.MPI`	Message Passing Interface.

9.1 mpi4py.MPI

Message Passing Interface.

Classes

`Cartcomm`([comm])	Cartesian topology intracommunicator
`Comm`([comm])	Communicator
`Datatype`([datatype])	Datatype object
`Distgraphcomm`([comm])	Distributed graph topology intracommunicator
`Errhandler`([errhandler])	Error handler
`File`([file])	File handle
`Graphcomm`([comm])	General graph topology intracommunicator
`Grequest`([request])	Generalized request handle
`Group`([group])	Group of processes
`Info`([info])	Info object
`Intercomm`([comm])	Intercommunicator
`Intracomm`([comm])	Intracommunicator
`Message`([message])	Matched message handle

continues on next page

Table 18 – continued from previous page

Op([op])	Operation object
Pickle([dumps, loads, protocol])	Pickle/unpickle Python objects
Prequest([request])	Persistent request handle
Request([request])	Request handle
Status([status])	Status object
Topocomm([comm])	Topology intracommunicator
Win([win])	Window handle
memory(buf)	Memory buffer

mpi4py.MPI.Cartcomm

class mpi4py.MPI.**Cartcomm**(*comm=None*)

Bases: *mpi4py.MPI.Topocomm*

Cartesian topology intracommunicator

Parameters **comm** (*Optional*[Cartcomm]) –

Return type *Cartcomm*

static **__new__**(*cls*, *comm=None*)

Parameters **comm** (*Optional*[Cartcomm]) –

Return type *Cartcomm*

Methods Summary

Get_cart_rank(coords)	Translate logical coordinates to ranks
Get_coords(rank)	Translate ranks to logical coordinates
Get_dim()	Return number of dimensions
Get_topo()	Return information on the cartesian topology
Shift(direction, disp)	Return a tuple (source, dest) of process ranks for data shifting with Comm.Sendrecv()
Sub(remain_dims)	Return cartesian communicators that form lower-dimensional subgrids

Attributes Summary

coords	coordinates
dim	number of dimensions
dims	dimensions
ndim	number of dimensions
periods	periodicity
topo	topology information

Methods Documentation

`Get_cart_rank`(*coords*)

Translate logical coordinates to ranks

Parameters **coords** (*Sequence[int]*) –

Return type int

`Get_coords`(*rank*)

Translate ranks to logical coordinates

Parameters **rank** (*int*) –

Return type List[int]

`Get_dim`()

Return number of dimensions

Return type int

`Get_topo`()

Return information on the cartesian topology

Return type Tuple[List[int], List[int], List[int]]

`Shift`(*direction*, *disp*)

Return a tuple (source, dest) of process ranks for data shifting with Comm.Sendrecv()

Parameters

- **direction** (*int*) –
- **disp** (*int*) –

Return type Tuple[int, int]

`Sub`(*remain_dims*)

Return cartesian communicators that form lower-dimensional subgrids

Parameters **remain_dims** (*Sequence[bool]*) –

Return type *Cartcomm*

Attributes Documentation

`coords`

coordinates

`dim`

number of dimensions

`dims`

dimensions

`ndim`

number of dimensions

`periods`

periodicity

`topo`

topology information

mpi4py.MPI.Comm

class mpi4py.MPI.**Comm**(*comm=None*)

Bases: `object`

Communicator

Parameters **comm** (`Optional[Comm]`) –

Return type *Comm*

static **__new__**(*cls*, *comm=None*)

Parameters **comm** (`Optional[Comm]`) –

Return type *Comm*

Methods Summary

`Abort`([errorcode])	Terminate MPI execution environment
`Allgather`(sendbuf, recvbuf)	Gather to All, gather data from all processes and distribute it to all other processes in a group
`Allgatherv`(sendbuf, recvbuf)	Gather to All Vector, gather data from all processes and distribute it to all other processes in a group providing different amount of data and displacements
`Allreduce`(sendbuf, recvbuf[, op])	Reduce to All
`Alltoall`(sendbuf, recvbuf)	All to All Scatter/Gather, send data from all to all processes in a group
`Alltoallv`(sendbuf, recvbuf)	All to All Scatter/Gather Vector, send data from all to all processes in a group providing different amount of data and displacements
`Alltoallw`(sendbuf, recvbuf)	Generalized All-to-All communication allowing different counts, displacements and datatypes for each partner
`Barrier`()	Barrier synchronization
`Bcast`(buf[, root])	Broadcast a message from one process to all other processes in a group
`Bsend`(buf, dest[, tag])	Blocking send in buffered mode
`Bsend_init`(buf, dest[, tag])	Persistent request for a send in buffered mode
`Call_errhandler`(errorcode)	Call the error handler installed on a communicator
`Clone`()	Clone an existing communicator
`Compare`(comm1, comm2)	Compare two communicators
`Create`(group)	Create communicator from group
`Create_group`(group[, tag])	Create communicator from group
`Create_keyval`([copy_fn, delete_fn, nopython])	Create a new attribute key for communicators
`Delete_attr`(keyval)	Delete attribute value associated with a key
`Disconnect`()	Disconnect from a communicator
`Dup`([info])	Duplicate an existing communicator
`Dup_with_info`(info)	Duplicate an existing communicator
`Free`()	Free a communicator
`Free_keyval`(keyval)	Free an attribute key for communicators
`Gather`(sendbuf, recvbuf[, root])	Gather together values from a group of processes

continues on next page

Table 21 – continued from previous page

Gatherv(sendbuf, recvbuf[, root])	Gather Vector, gather data to one process from all other processes in a group providing different amount of data and displacements at the receiving sides
Get_attr(keyval)	Retrieve attribute value by key
Get_errhandler()	Get the error handler for a communicator
Get_group()	Access the group associated with a communicator
Get_info()	Return the hints for a communicator that are currently in use
Get_name()	Get the print name for this communicator
Get_parent()	Return the parent intercommunicator for this process
Get_rank()	Return the rank of this process in a communicator
Get_size()	Return the number of processes in a communicator
Get_topology()	Determine the type of topology (if any) associated with a communicator
Iallgather(sendbuf, recvbuf)	Nonblocking Gather to All
Iallgatherv(sendbuf, recvbuf)	Nonblocking Gather to All Vector
Iallreduce(sendbuf, recvbuf[, op])	Nonblocking Reduce to All
Ialltoall(sendbuf, recvbuf)	Nonblocking All to All Scatter/Gather
Ialltoallv(sendbuf, recvbuf)	Nonblocking All to All Scatter/Gather Vector
Ialltoallw(sendbuf, recvbuf)	Nonblocking Generalized All-to-All
Ibarrier()	Nonblocking Barrier
Ibcast(buf[, root])	Nonblocking Broadcast
Ibsend(buf, dest[, tag])	Nonblocking send in buffered mode
Idup()	Nonblocking duplicate an existing communicator
Igather(sendbuf, recvbuf[, root])	Nonblocking Gather
Igatherv(sendbuf, recvbuf[, root])	Nonblocking Gather Vector
Improbe([source, tag, status])	Nonblocking test for a matched message
Iprobe([source, tag, status])	Nonblocking test for a message
Irecv(buf[, source, tag])	Nonblocking receive
Ireduce(sendbuf, recvbuf[, op, root])	Nonblocking Reduce to Root
Ireduce_scatter(sendbuf, recvbuf[, ...])	Nonblocking Reduce-Scatter (vector version)
Ireduce_scatter_block(sendbuf, recvbuf[, op])	Nonblocking Reduce-Scatter Block (regular, non-vector version)
Irsend(buf, dest[, tag])	Nonblocking send in ready mode
Is_inter()	Test to see if a comm is an intercommunicator
Is_intra()	Test to see if a comm is an intracommunicator
Iscatter(sendbuf, recvbuf[, root])	Nonblocking Scatter
Iscatterv(sendbuf, recvbuf[, root])	Nonblocking Scatter Vector
Isend(buf, dest[, tag])	Nonblocking send
Issend(buf, dest[, tag])	Nonblocking send in synchronous mode
Join(fd)	Create a intercommunicator by joining two processes connected by a socket
Mprobe([source, tag, status])	Blocking test for a matched message
Probe([source, tag, status])	Blocking test for a message
Recv(buf[, source, tag, status])	Blocking receive
Recv_init(buf[, source, tag])	Create a persistent request for a receive
Reduce(sendbuf, recvbuf[, op, root])	Reduce to Root
Reduce_scatter(sendbuf, recvbuf[, ...])	Reduce-Scatter (vector version)
Reduce_scatter_block(sendbuf, recvbuf[, op])	Reduce-Scatter Block (regular, non-vector version)
Rsend(buf, dest[, tag])	Blocking send in ready mode

continues on next page

Table 21 – continued from previous page

Rsend_init(buf, dest[, tag])	Persistent request for a send in ready mode
Scatter(sendbuf, recvbuf[, root])	Scatter data from one process to all other processes in a group
Scatterv(sendbuf, recvbuf[, root])	Scatter Vector, scatter data from one process to all other processes in a group providing different amount of data and displacements at the sending side
Send(buf, dest[, tag])	Blocking send
Send_init(buf, dest[, tag])	Create a persistent request for a standard send
Sendrecv(sendbuf, dest[, sendtag, recvbuf, ...])	Send and receive a message
Sendrecv_replace(buf, dest[, sendtag, ...])	Send and receive a message
Set_attr(keyval, attrval)	Store attribute value associated with a key
Set_errhandler(errhandler)	Set the error handler for a communicator
Set_info(info)	Set new values for the hints associated with a communicator
Set_name(name)	Set the print name for this communicator
Split([color, key])	Split communicator by color and key
Split_type(split_type[, key, info])	Split communicator by split type
Ssend(buf, dest[, tag])	Blocking send in synchronous mode
Ssend_init(buf, dest[, tag])	Persistent request for a send in synchronous mode
allgather(sendobj)	Gather to All
allreduce(sendobj[, op])	Reduce to All
alltoall(sendobj)	All to All Scatter/Gather
barrier()	Barrier
bcast(obj[, root])	Broadcast
bsend(obj, dest[, tag])	Send in buffered mode
f2py(arg)	
gather(sendobj[, root])	Gather
ibsend(obj, dest[, tag])	Nonblocking send in buffered mode
improbe([source, tag, status])	Nonblocking test for a matched message
iprobe([source, tag, status])	Nonblocking test for a message
irecv([buf, source, tag])	Nonblocking receive
isend(obj, dest[, tag])	Nonblocking send
issend(obj, dest[, tag])	Nonblocking send in synchronous mode
mprobe([source, tag, status])	Blocking test for a matched message
probe([source, tag, status])	Blocking test for a message
py2f()	
recv([buf, source, tag, status])	Receive
reduce(sendobj[, op, root])	Reduce to Root
scatter(sendobj[, root])	Scatter
send(obj, dest[, tag])	Send
sendrecv(sendobj, dest[, sendtag, recvbuf, ...])	Send and Receive
ssend(obj, dest[, tag])	Send in synchronous mode

Attributes Summary

group	communicator group
info	communicator info
is_inter	is intercommunicator
is_intra	is intracommunicator
is_topo	is a topology communicator
name	communicator name
rank	rank of this process in communicator
size	number of processes in communicator
topology	communicator topology type

Methods Documentation

Abort(*errorcode=0*)

Terminate MPI execution environment

> **Warning:** This is a direct call, use it with care!!!.

Parameters **errorcode** (*int*) –

Return type NoReturn

Allgather(*sendbuf*, *recvbuf*)

Gather to All, gather data from all processes and distribute it to all other processes in a group

Parameters

- **sendbuf** (*Union[BufSpec, InPlace]*) –
- **recvbuf** (*BufSpecB*) –

Return type None

Allgatherv(*sendbuf*, *recvbuf*)

Gather to All Vector, gather data from all processes and distribute it to all other processes in a group providing different amount of data and displacements

Parameters

- **sendbuf** (*Union[BufSpec, InPlace]*) –
- **recvbuf** (*BufSpecV*) –

Return type None

Allreduce(*sendbuf*, *recvbuf*, *op=SUM*)

Reduce to All

Parameters

- **sendbuf** (*Union[BufSpec, InPlace]*) –
- **recvbuf** (*BufSpec*) –
- **op** (Op) –

Return type None

Alltoall(*sendbuf*, *recvbuf*)
All to All Scatter/Gather, send data from all to all processes in a group

Parameters

- **sendbuf** (*Union[BufSpecB, InPlace]*) –
- **recvbuf** (*BufSpecB*) –

Return type None

Alltoallv(*sendbuf*, *recvbuf*)
All to All Scatter/Gather Vector, send data from all to all processes in a group providing different amount of data and displacements

Parameters

- **sendbuf** (*Union[BufSpecV, InPlace]*) –
- **recvbuf** (*BufSpecV*) –

Return type None

Alltoallw(*sendbuf*, *recvbuf*)
Generalized All-to-All communication allowing different counts, displacements and datatypes for each partner

Parameters

- **sendbuf** (*Union[BufSpecW, InPlace]*) –
- **recvbuf** (*BufSpecW*) –

Return type None

Barrier()
Barrier synchronization

Return type None

Bcast(*buf*, *root=0*)
Broadcast a message from one process to all other processes in a group

Parameters

- **buf** (*BufSpec*) –
- **root** (*int*) –

Return type None

Bsend(*buf*, *dest*, *tag=0*)
Blocking send in buffered mode

Parameters

- **buf** (*BufSpec*) –
- **dest** (*int*) –
- **tag** (*int*) –

Return type None

Bsend_init(*buf*, *dest*, *tag=0*)
Persistent request for a send in buffered mode

Parameters

- **buf** (*BufSpec*) –
- **dest** (*int*) –
- **tag** (*int*) –

Return type *Request*

Call_errhandler(*errorcode*)
Call the error handler installed on a communicator

Parameters **errorcode** (*int*) –

Return type None

Clone()
Clone an existing communicator

Return type *Comm*

classmethod **Compare**(*comm1*, *comm2*)
Compare two communicators

Parameters

- **comm1** (Comm) –
- **comm2** (Comm) –

Return type int

Create(*group*)
Create communicator from group

Parameters **group** (Group) –

Return type *Comm*

Create_group(*group*, *tag=0*)
Create communicator from group

Parameters

- **group** (Group) –
- **tag** (*int*) –

Return type *Comm*

classmethod **Create_keyval**(*copy_fn=None*, *delete_fn=None*, *nopython=False*)
Create a new attribute key for communicators

Parameters

- **copy_fn** (*Optional[Callable[[*Comm*, int, Any], Any]]*) –
- **delete_fn** (*Optional[Callable[[*Comm*, int, Any], None]]*) –
- **nopython** (*bool*) –

Return type int

Delete_attr(*keyval*)
Delete attribute value associated with a key

Parameters **keyval** (*int*) –

Return type None

Disconnect()

Disconnect from a communicator

Return type None

Dup(*info=None*)

Duplicate an existing communicator

Parameters **info** (*Optional[Info]*) –

Return type *Comm*

Dup_with_info(*info*)

Duplicate an existing communicator

Parameters **info** (Info) –

Return type *Comm*

Free()

Free a communicator

Return type None

classmethod Free_keyval(*keyval*)

Free an attribute key for communicators

Parameters **keyval** (*int*) –

Return type int

Gather(*sendbuf*, *recvbuf*, *root=0*)

Gather together values from a group of processes

Parameters

- **sendbuf** (*Union[BufSpec, InPlace]*) –
- **recvbuf** (*Optional[BufSpecB]*) –
- **root** (*int*) –

Return type None

Gatherv(*sendbuf*, *recvbuf*, *root=0*)

Gather Vector, gather data to one process from all other processes in a group providing different amount of data and displacements at the receiving sides

Parameters

- **sendbuf** (*Union[BufSpec, InPlace]*) –
- **recvbuf** (*Optional[BufSpecV]*) –
- **root** (*int*) –

Return type None

Get_attr(*keyval*)

Retrieve attribute value by key

Parameters **keyval** (*int*) –

Return type Optional[Union[int, Any]]

Get_errhandler()

Get the error handler for a communicator

Return type *Errhandler*

`Get_group()`
Access the group associated with a communicator

Return type *Group*

`Get_info()`
Return the hints for a communicator that are currently in use

Return type *Info*

`Get_name()`
Get the print name for this communicator

Return type str

`classmethod Get_parent()`
Return the parent intercommunicator for this process

Return type *Intercomm*

`Get_rank()`
Return the rank of this process in a communicator

Return type int

`Get_size()`
Return the number of processes in a communicator

Return type int

`Get_topology()`
Determine the type of topology (if any) associated with a communicator

Return type int

`Iallgather`(*sendbuf*, *recvbuf*)
Nonblocking Gather to All

Parameters

- **sendbuf** (*Union[BufSpec, InPlace]*) –
- **recvbuf** (*BufSpecB*) –

Return type *Request*

`Iallgatherv`(*sendbuf*, *recvbuf*)
Nonblocking Gather to All Vector

Parameters

- **sendbuf** (*Union[BufSpec, InPlace]*) –
- **recvbuf** (*BufSpecV*) –

Return type *Request*

`Iallreduce`(*sendbuf*, *recvbuf*, *op=SUM*)
Nonblocking Reduce to All

Parameters

- **sendbuf** (*Union[BufSpec, InPlace]*) –
- **recvbuf** (*BufSpec*) –

- **op** (Op) –

Return type *Request*

Ialltoall(*sendbuf*, *recvbuf*)

Nonblocking All to All Scatter/Gather

Parameters

- **sendbuf** (*Union[BufSpecB, InPlace]*) –
- **recvbuf** (*BufSpecB*) –

Return type *Request*

Ialltoallv(*sendbuf*, *recvbuf*)

Nonblocking All to All Scatter/Gather Vector

Parameters

- **sendbuf** (*Union[BufSpecV, InPlace]*) –
- **recvbuf** (*BufSpecV*) –

Return type *Request*

Ialltoallw(*sendbuf*, *recvbuf*)

Nonblocking Generalized All-to-All

Parameters

- **sendbuf** (*Union[BufSpecW, InPlace]*) –
- **recvbuf** (*BufSpecW*) –

Return type *Request*

Ibarrier()

Nonblocking Barrier

Return type *Request*

Ibcast(*buf*, *root=0*)

Nonblocking Broadcast

Parameters

- **buf** (*BufSpec*) –
- **root** (*int*) –

Return type *Request*

Ibsend(*buf*, *dest*, *tag=0*)

Nonblocking send in buffered mode

Parameters

- **buf** (*BufSpec*) –
- **dest** (*int*) –
- **tag** (*int*) –

Return type *Request*

Idup()

Nonblocking duplicate an existing communicator

Return type Tuple[*Comm*, *Request*]

`Igather`(*sendbuf*, *recvbuf*, *root=0*)
Nonblocking Gather

Parameters

- **sendbuf** (`Union[BufSpec, InPlace]`) –
- **recvbuf** (`Optional[BufSpecB]`) –
- **root** (`int`) –

Return type *Request*

`Igatherv`(*sendbuf*, *recvbuf*, *root=0*)
Nonblocking Gather Vector

Parameters

- **sendbuf** (`Union[BufSpec, InPlace]`) –
- **recvbuf** (`Optional[BufSpecV]`) –
- **root** (`int`) –

Return type *Request*

`Improbe`(*source=ANY_SOURCE*, *tag=ANY_TAG*, *status=None*)
Nonblocking test for a matched message

Parameters

- **source** (`int`) –
- **tag** (`int`) –
- **status** (`Optional[Status]`) –

Return type Optional[*Message*]

`Iprobe`(*source=ANY_SOURCE*, *tag=ANY_TAG*, *status=None*)
Nonblocking test for a message

Parameters

- **source** (`int`) –
- **tag** (`int`) –
- **status** (`Optional[Status]`) –

Return type bool

`Irecv`(*buf*, *source=ANY_SOURCE*, *tag=ANY_TAG*)
Nonblocking receive

Parameters

- **buf** (`BufSpec`) –
- **source** (`int`) –
- **tag** (`int`) –

Return type *Request*

`Ireduce`(*sendbuf*, *recvbuf*, *op=SUM*, *root=0*)
Nonblocking Reduce to Root

Parameters

- **sendbuf** (*Union[BufSpec, InPlace]*) –
- **recvbuf** (*Optional[BufSpec]*) –
- **op** (Op) –
- **root** (*int*) –

Return type *Request*

Ireduce_scatter(*sendbuf*, *recvbuf*, *recvcounts=None*, *op=SUM*)

Nonblocking Reduce-Scatter (vector version)

Parameters

- **sendbuf** (*Union[BufSpec, InPlace]*) –
- **recvbuf** (*BufSpec*) –
- **recvcounts** (*Optional[Sequence[int]]*) –
- **op** (Op) –

Return type *Request*

Ireduce_scatter_block(*sendbuf*, *recvbuf*, *op=SUM*)

Nonblocking Reduce-Scatter Block (regular, non-vector version)

Parameters

- **sendbuf** (*Union[BufSpecB, InPlace]*) –
- **recvbuf** (*Union[BufSpec, BufSpecB]*) –
- **op** (Op) –

Return type *Request*

Irsend(*buf*, *dest*, *tag=0*)

Nonblocking send in ready mode

Parameters

- **buf** (*BufSpec*) –
- **dest** (*int*) –
- **tag** (*int*) –

Return type *Request*

Is_inter()

Test to see if a comm is an intercommunicator

Return type bool

Is_intra()

Test to see if a comm is an intracommunicator

Return type bool

Iscatter(*sendbuf*, *recvbuf*, *root=0*)

Nonblocking Scatter

Parameters

- **sendbuf** (*Optional[BufSpecB]*) –

- **recvbuf** (*Union[BufSpec, InPlace]*) –
- **root** (*int*) –

Return type *Request*

Iscatterv(*sendbuf*, *recvbuf*, *root=0*)
Nonblocking Scatter Vector

Parameters

- **sendbuf** (*Optional[BufSpecV]*) –
- **recvbuf** (*Union[BufSpec, InPlace]*) –
- **root** (*int*) –

Return type *Request*

Isend(*buf*, *dest*, *tag=0*)
Nonblocking send

Parameters

- **buf** (*BufSpec*) –
- **dest** (*int*) –
- **tag** (*int*) –

Return type *Request*

Issend(*buf*, *dest*, *tag=0*)
Nonblocking send in synchronous mode

Parameters

- **buf** (*BufSpec*) –
- **dest** (*int*) –
- **tag** (*int*) –

Return type *Request*

classmethod Join(*fd*)
Create a intercommunicator by joining two processes connected by a socket

Parameters **fd** (*int*) –

Return type *Intercomm*

Mprobe(*source=ANY_SOURCE*, *tag=ANY_TAG*, *status=None*)
Blocking test for a matched message

Parameters

- **source** (*int*) –
- **tag** (*int*) –
- **status** (*Optional[Status]*) –

Return type *Message*

Probe(*source=ANY_SOURCE*, *tag=ANY_TAG*, *status=None*)
Blocking test for a message

Note: This function blocks until the message arrives.

Parameters

- **source** (*int*) –
- **tag** (*int*) –
- **status** (*Optional[Status]*) –

Return type Literal[True]

Recv(*buf*, *source=ANY_SOURCE*, *tag=ANY_TAG*, *status=None*)
Blocking receive

Note: This function blocks until the message is received

Parameters

- **buf** (*BufSpec*) –
- **source** (*int*) –
- **tag** (*int*) –
- **status** (*Optional[Status]*) –

Return type None

Recv_init(*buf*, *source=ANY_SOURCE*, *tag=ANY_TAG*)
Create a persistent request for a receive

Parameters

- **buf** (*BufSpec*) –
- **source** (*int*) –
- **tag** (*int*) –

Return type *Prequest*

Reduce(*sendbuf*, *recvbuf*, *op=SUM*, *root=0*)
Reduce to Root

Parameters

- **sendbuf** (*Union[BufSpec, InPlace]*) –
- **recvbuf** (*Optional[BufSpec]*) –
- **op** (Op) –
- **root** (*int*) –

Return type None

Reduce_scatter(*sendbuf*, *recvbuf*, *recvcounts=None*, *op=SUM*)
Reduce-Scatter (vector version)

Parameters

- **sendbuf** (*Union[BufSpec, InPlace]*) –
- **recvbuf** (*BufSpec*) –
- **recvcounts** (*Optional[Sequence[int]]*) –
- **op** (Op) –

Return type None

Reduce_scatter_block(*sendbuf*, *recvbuf*, *op=SUM*)
Reduce-Scatter Block (regular, non-vector version)

Parameters

- **sendbuf** (*Union[BufSpecB, InPlace]*) –
- **recvbuf** (*Union[BufSpec, BufSpecB]*) –
- **op** (Op) –

Return type None

Rsend(*buf*, *dest*, *tag=0*)
Blocking send in ready mode

Parameters

- **buf** (*BufSpec*) –
- **dest** (*int*) –
- **tag** (*int*) –

Return type None

Rsend_init(*buf*, *dest*, *tag=0*)
Persistent request for a send in ready mode

Parameters

- **buf** (*BufSpec*) –
- **dest** (*int*) –
- **tag** (*int*) –

Return type *Request*

Scatter(*sendbuf*, *recvbuf*, *root=0*)
Scatter data from one process to all other processes in a group

Parameters

- **sendbuf** (*Optional[BufSpecB]*) –
- **recvbuf** (*Union[BufSpec, InPlace]*) –
- **root** (*int*) –

Return type None

Scatterv(*sendbuf*, *recvbuf*, *root=0*)
Scatter Vector, scatter data from one process to all other processes in a group providing different amount of data and displacements at the sending side

Parameters

- **sendbuf** (*Optional[BufSpecV]*) –

- **recvbuf** (*Union[BufSpec, InPlace]*) –
- **root** (*int*) –

Return type None

Send(*buf*, *dest*, *tag=0*)

Blocking send

Note: This function may block until the message is received. Whether or not *Send* blocks depends on several factors and is implementation dependent

---

Parameters

- **buf** (*BufSpec*) –
- **dest** (*int*) –
- **tag** (*int*) –

Return type None

Send_init(*buf*, *dest*, *tag=0*)

Create a persistent request for a standard send

Parameters

- **buf** (*BufSpec*) –
- **dest** (*int*) –
- **tag** (*int*) –

Return type *Prequest*

Sendrecv(*sendbuf*, *dest*, *sendtag=0*, *recvbuf=None*, *source=ANY_SOURCE*, *recvtag=ANY_TAG*, *status=None*)

Send and receive a message

Note: This function is guaranteed not to deadlock in situations where pairs of blocking sends and receives may deadlock.

Caution: A common mistake when using this function is to mismatch the tags with the source and destination ranks, which can result in deadlock.

Parameters

- **sendbuf** (*BufSpec*) –
- **dest** (*int*) –
- **sendtag** (*int*) –
- **recvbuf** (*BufSpec*) –
- **source** (*int*) –
- **recvtag** (*int*) –

- **status** (*Optional[Status]*) –

Return type None

Sendrecv_replace(*buf*, *dest*, *sendtag=0*, *source=ANY_SOURCE*, *recvtag=ANY_TAG*, *status=None*)
Send and receive a message

Note: This function is guaranteed not to deadlock in situations where pairs of blocking sends and receives may deadlock.

Caution: A common mistake when using this function is to mismatch the tags with the source and destination ranks, which can result in deadlock.

Parameters

- **buf** (*BufSpec*) –
- **dest** (*int*) –
- **sendtag** (*int*) –
- **source** (*int*) –
- **recvtag** (*int*) –
- **status** (*Optional[Status]*) –

Return type None

Set_attr(*keyval*, *attrval*)
Store attribute value associated with a key

Parameters

- **keyval** (*int*) –
- **attrval** (*Any*) –

Return type None

Set_errhandler(*errhandler*)
Set the error handler for a communicator

Parameters **errhandler** (Errhandler) –

Return type None

Set_info(*info*)
Set new values for the hints associated with a communicator

Parameters **info** (Info) –

Return type None

Set_name(*name*)
Set the print name for this communicator

Parameters **name** (*str*) –

Return type None

Split(*color=0*, *key=0*)

Split communicator by color and key

Parameters

- **color** (*int*) –
- **key** (*int*) –

Return type *Comm*

Split_type(*split_type*, *key=0*, *info=INFO_NULL*)

Split communicator by split type

Parameters

- **split_type** (*int*) –
- **key** (*int*) –
- **info** (Info) –

Return type *Comm*

Ssend(*buf*, *dest*, *tag=0*)

Blocking send in synchronous mode

Parameters

- **buf** (*BufSpec*) –
- **dest** (*int*) –
- **tag** (*int*) –

Return type None

Ssend_init(*buf*, *dest*, *tag=0*)

Persistent request for a send in synchronous mode

Parameters

- **buf** (*BufSpec*) –
- **dest** (*int*) –
- **tag** (*int*) –

Return type *Request*

allgather(*sendobj*)

Gather to All

Parameters **sendobj** (*Any*) –

Return type List[Any]

allreduce(*sendobj*, *op=SUM*)

Reduce to All

Parameters

- **sendobj** (*Any*) –
- **op** (*Union[Op, Callable[[Any, Any], Any]]*) –

Return type Any

`alltoall`(*sendobj*)
: All to All Scatter/Gather

  Parameters **sendobj** (*Sequence[Any]*) –

  Return type List[Any]

`barrier`()
: Barrier

  Return type None

`bcast`(*obj*, *root=0*)
: Broadcast

  Parameters
  - **obj** (*Any*) –
  - **root** (*int*) –

  Return type Any

`bsend`(*obj*, *dest*, *tag=0*)
: Send in buffered mode

  Parameters
  - **obj** (*Any*) –
  - **dest** (*int*) –
  - **tag** (*int*) –

  Return type None

`classmethod f2py`(*arg*)

  Parameters **arg** (*int*) –

  Return type *Comm*

`gather`(*sendobj*, *root=0*)
: Gather

  Parameters
  - **sendobj** (*Any*) –
  - **root** (*int*) –

  Return type Optional[List[Any]]

`ibsend`(*obj*, *dest*, *tag=0*)
: Nonblocking send in buffered mode

  Parameters
  - **obj** (*Any*) –
  - **dest** (*int*) –
  - **tag** (*int*) –

  Return type *Request*

`improbe`(*source=ANY_SOURCE*, *tag=ANY_TAG*, *status=None*)
: Nonblocking test for a matched message

Parameters

- **source** (*int*) –
- **tag** (*int*) –
- **status** (*Optional[Status]*) –

Return type Optional[*Message*]

iprobe(*source=ANY_SOURCE*, *tag=ANY_TAG*, *status=None*)
Nonblocking test for a message

Parameters

- **source** (*int*) –
- **tag** (*int*) –
- **status** (*Optional[Status]*) –

Return type bool

irecv(*buf=None*, *source=ANY_SOURCE*, *tag=ANY_TAG*)
Nonblocking receive

Parameters

- **buf** (*Optional[Buffer]*) –
- **source** (*int*) –
- **tag** (*int*) –

Return type *Request*

isend(*obj*, *dest*, *tag=0*)
Nonblocking send

Parameters

- **obj** (*Any*) –
- **dest** (*int*) –
- **tag** (*int*) –

Return type *Request*

issend(*obj*, *dest*, *tag=0*)
Nonblocking send in synchronous mode

Parameters

- **obj** (*Any*) –
- **dest** (*int*) –
- **tag** (*int*) –

Return type *Request*

mprobe(*source=ANY_SOURCE*, *tag=ANY_TAG*, *status=None*)
Blocking test for a matched message

Parameters

- **source** (*int*) –
- **tag** (*int*) –

- **status** (*Optional[Status]*) –

Return type *Message*

probe(*source=ANY_SOURCE*, *tag=ANY_TAG*, *status=None*)
Blocking test for a message

Parameters

- **source** (*int*) –
- **tag** (*int*) –
- **status** (*Optional[Status]*) –

Return type Literal[True]

py2f()

Return type int

recv(*buf=None*, *source=ANY_SOURCE*, *tag=ANY_TAG*, *status=None*)
Receive

Parameters

- **buf** (*Optional[Buffer]*) –
- **source** (*int*) –
- **tag** (*int*) –
- **status** (*Optional[Status]*) –

Return type Any

reduce(*sendobj*, *op=SUM*, *root=0*)
Reduce to Root

Parameters

- **sendobj** (*Any*) –
- **op** (*Union[Op, Callable[[Any, Any], Any]]*) –
- **root** (*int*) –

Return type Optional[Any]

scatter(*sendobj*, *root=0*)
Scatter

Parameters

- **sendobj** (*Sequence[Any]*) –
- **root** (*int*) –

Return type Any

send(*obj*, *dest*, *tag=0*)
Send

Parameters

- **obj** (*Any*) –
- **dest** (*int*) –

- **tag** (*int*) –

Return type None

sendrecv(*sendobj*, *dest*, *sendtag=0*, *recvbuf=None*, *source=ANY_SOURCE*, *recvtag=ANY_TAG*, *status=None*)

Send and Receive

Parameters

- **sendobj** (*Any*) –
- **dest** (*int*) –
- **sendtag** (*int*) –
- **recvbuf** (*Optional[Buffer]*) –
- **source** (*int*) –
- **recvtag** (*int*) –
- **status** (*Optional[Status]*) –

Return type Any

ssend(*obj*, *dest*, *tag=0*)

Send in synchronous mode

Parameters

- **obj** (*Any*) –
- **dest** (*int*) –
- **tag** (*int*) –

Return type None

Attributes Documentation

group

communicator group

info

communicator info

is_inter

is intercommunicator

is_intra

is intracommunicator

is_topo

is a topology communicator

name

communicator name

rank

rank of this process in communicator

size

number of processes in communicator

topology
communicator topology type

mpi4py.MPI.Datatype

class mpi4py.MPI.**Datatype**(*datatype=None*)

Bases: `object`

Datatype object

Parameters **datatype** (`Optional[Datatype]`) –

Return type *Datatype*

static **__new__**(*cls*, *datatype=None*)

Parameters **datatype** (`Optional[Datatype]`) –

Return type *Datatype*

Methods Summary

`Commit`()	Commit the datatype
`Create_contiguous`(count)	Create a contiguous datatype
`Create_darray`(size, rank, gsizes, distribs, ...)	Create a datatype representing an HPF-like distributed array on Cartesian process grids
`Create_f90_complex`(p, r)	Return a bounded complex datatype
`Create_f90_integer`(r)	Return a bounded integer datatype
`Create_f90_real`(p, r)	Return a bounded real datatype
`Create_hindexed`(blocklengths, displacements)	Create an indexed datatype with displacements in bytes
`Create_hindexed_block`(blocklength, displacements)	Create an indexed datatype with constant-sized blocks and displacements in bytes
`Create_hvector`(count, blocklength, stride)	Create a vector (strided) datatype
`Create_indexed`(blocklengths, displacements)	Create an indexed datatype
`Create_indexed_block`(blocklength, displacements)	Create an indexed datatype with constant-sized blocks
`Create_keyval`([copy_fn, delete_fn, nopython])	Create a new attribute key for datatypes
`Create_resized`(lb, extent)	Create a datatype with a new lower bound and extent
`Create_struct`(blocklengths, displacements, ...)	Create an datatype from a general set of block sizes, displacements and datatypes
`Create_subarray`(sizes, subsizes, starts[, order])	Create a datatype for a subarray of a regular, multidimensional array
`Create_vector`(count, blocklength, stride)	Create a vector (strided) datatype
`Delete_attr`(keyval)	Delete attribute value associated with a key
`Dup`()	Duplicate a datatype
`Free`()	Free the datatype
`Free_keyval`(keyval)	Free an attribute key for datatypes
`Get_attr`(keyval)	Retrieve attribute value by key
`Get_contents`()	Retrieve the actual arguments used in the call that created a datatype

continues on next page

Table 23 – continued from previous page

Get_envelope()	Return information on the number and type of input arguments used in the call that created a datatype
Get_extent()	Return lower bound and extent of datatype
Get_name()	Get the print name for this datatype
Get_size()	Return the number of bytes occupied by entries in the datatype
Get_true_extent()	Return the true lower bound and extent of a datatype
Match_size(typeclass, size)	Find a datatype matching a specified size in bytes
Pack(inbuf, outbuf, position, comm)	Pack into contiguous memory according to datatype.
Pack_external(datarep, inbuf, outbuf, position)	Pack into contiguous memory according to datatype, using a portable data representation (**external32**).
Pack_external_size(datarep, count)	Return the upper bound on the amount of space (in bytes) needed to pack a message according to datatype, using a portable data representation (**external32**).
Pack_size(count, comm)	Return the upper bound on the amount of space (in bytes) needed to pack a message according to datatype.
Set_attr(keyval, attrval)	Store attribute value associated with a key
Set_name(name)	Set the print name for this datatype
Unpack(inbuf, position, outbuf, comm)	Unpack from contiguous memory according to datatype.
Unpack_external(datarep, inbuf, position, outbuf)	Unpack from contiguous memory according to datatype, using a portable data representation (**external32**).
decode()	Convenience method for decoding a datatype
f2py(arg)	
py2f()	

Attributes Summary

combiner	datatype combiner
contents	datatype contents
envelope	datatype envelope
extent	
is_named	is a named datatype
is_predefined	is a predefined datatype
lb	lower bound
name	datatype name
size	
true_extent	true extent
true_lb	true lower bound
true_ub	true upper bound
ub	upper bound

Methods Documentation

Commit()

Commit the datatype

Return type *Datatype*

Create_contiguous(*count*)

Create a contiguous datatype

Parameters **count** (*int*) –

Return type *Datatype*

Create_darray(*size*, *rank*, *gsizes*, *distribs*, *dargs*, *psizes*, *order=ORDER_C*)

Create a datatype representing an HPF-like distributed array on Cartesian process grids

Parameters

- **size** (*int*) –
- **rank** (*int*) –
- **gsizes** (*Sequence[int]*) –
- **distribs** (*Sequence[int]*) –
- **dargs** (*Sequence[int]*) –
- **psizes** (*Sequence[int]*) –
- **order** (*int*) –

Return type *Datatype*

classmethod **Create_f90_complex**(*p*, *r*)

Return a bounded complex datatype

Parameters

- **p** (*int*) –
- **r** (*int*) –

Return type *Datatype*

classmethod **Create_f90_integer**(*r*)

Return a bounded integer datatype

Parameters **r** (*int*) –

Return type *Datatype*

classmethod **Create_f90_real**(*p*, *r*)

Return a bounded real datatype

Parameters

- **p** (*int*) –
- **r** (*int*) –

Return type *Datatype*

Create_hindexed(*blocklengths*, *displacements*)

Create an indexed datatype with displacements in bytes

Parameters

- **blocklengths** (*Sequence[int]*) –
- **displacements** (*Sequence[int]*) –

Return type *Datatype*

`Create_hindexed_block`(*blocklength*, *displacements*)
Create an indexed datatype with constant-sized blocks and displacements in bytes

Parameters

- **blocklength** (*int*) –
- **displacements** (*Sequence[int]*) –

Return type *Datatype*

`Create_hvector`(*count*, *blocklength*, *stride*)
Create a vector (strided) datatype

Parameters

- **count** (*int*) –
- **blocklength** (*int*) –
- **stride** (*int*) –

Return type *Datatype*

`Create_indexed`(*blocklengths*, *displacements*)
Create an indexed datatype

Parameters

- **blocklengths** (*Sequence[int]*) –
- **displacements** (*Sequence[int]*) –

Return type *Datatype*

`Create_indexed_block`(*blocklength*, *displacements*)
Create an indexed datatype with constant-sized blocks

Parameters

- **blocklength** (*int*) –
- **displacements** (*Sequence[int]*) –

Return type *Datatype*

`classmethod Create_keyval`(*copy_fn=None*, *delete_fn=None*, *nopython=False*)
Create a new attribute key for datatypes

Parameters

- **copy_fn** (*Optional[Callable[[Datatype, int, Any], Any]]*) –
- **delete_fn** (*Optional[Callable[[Datatype, int, Any], None]]*) –
- **nopython** (*bool*) –

Return type int

`Create_resized`(*lb*, *extent*)
Create a datatype with a new lower bound and extent

Parameters

- **lb** (*int*) –
- **extent** (*int*) –

Return type *Datatype*

`classmethod Create_struct`(*blocklengths*, *displacements*, *datatypes*)
Create an datatype from a general set of block sizes, displacements and datatypes

Parameters

- **blocklengths** (*Sequence[int]*) –
- **displacements** (*Sequence[int]*) –
- **datatypes** (*Sequence[Datatype]*) –

Return type *Datatype*

`Create_subarray`(*sizes*, *subsizes*, *starts*, *order=ORDER_C*)
Create a datatype for a subarray of a regular, multidimensional array

Parameters

- **sizes** (*Sequence[int]*) –
- **subsizes** (*Sequence[int]*) –
- **starts** (*Sequence[int]*) –
- **order** (*int*) –

Return type *Datatype*

`Create_vector`(*count*, *blocklength*, *stride*)
Create a vector (strided) datatype

Parameters

- **count** (*int*) –
- **blocklength** (*int*) –
- **stride** (*int*) –

Return type *Datatype*

`Delete_attr`(*keyval*)
Delete attribute value associated with a key

Parameters **keyval** (*int*) –

Return type None

`Dup`()
Duplicate a datatype

Return type *Datatype*

`Free`()
Free the datatype

Return type None

`classmethod Free_keyval`(*keyval*)
Free an attribute key for datatypes

Parameters **keyval** (*int*) –

Return type int

`Get_attr`(*keyval*)
Retrieve attribute value by key

Parameters **keyval** (*int*) –

Return type Optional[Union[int, Any]]

`Get_contents`()
Retrieve the actual arguments used in the call that created a datatype

Return type Tuple[List[int], List[int], List[*Datatype*]]

`Get_envelope`()
Return information on the number and type of input arguments used in the call that created a datatype

Return type Tuple[int, int, int, int]

`Get_extent`()
Return lower bound and extent of datatype

Return type Tuple[int, int]

`Get_name`()
Get the print name for this datatype

Return type str

`Get_size`()
Return the number of bytes occupied by entries in the datatype

Return type int

`Get_true_extent`()
Return the true lower bound and extent of a datatype

Return type Tuple[int, int]

`classmethod Match_size`(*typeclass*, *size*)
Find a datatype matching a specified size in bytes

Parameters

- **typeclass** (*int*) –
- **size** (*int*) –

Return type *Datatype*

`Pack`(*inbuf*, *outbuf*, *position*, *comm*)
Pack into contiguous memory according to datatype.

Parameters

- **inbuf** (*BufSpec*) –
- **outbuf** (*BufSpec*) –
- **position** (*int*) –
- **comm** (Comm) –

Return type int

`Pack_external`(*datarep*, *inbuf*, *outbuf*, *position*)
Pack into contiguous memory according to datatype, using a portable data representation (**external32**).

Parameters

- **datarep** (*str*) –
- **inbuf** (*BufSpec*) –
- **outbuf** (*BufSpec*) –
- **position** (*int*) –

Return type int

`Pack_external_size`(*datarep*, *count*)

Return the upper bound on the amount of space (in bytes) needed to pack a message according to datatype, using a portable data representation (**external32**).

Parameters

- **datarep** (*str*) –
- **count** (*int*) –

Return type int

`Pack_size`(*count*, *comm*)

Return the upper bound on the amount of space (in bytes) needed to pack a message according to datatype.

Parameters

- **count** (*int*) –
- **comm** (Comm) –

Return type int

`Set_attr`(*keyval*, *attrval*)

Store attribute value associated with a key

Parameters

- **keyval** (*int*) –
- **attrval** (*Any*) –

Return type None

`Set_name`(*name*)

Set the print name for this datatype

Parameters **name** (*str*) –

Return type None

`Unpack`(*inbuf*, *position*, *outbuf*, *comm*)

Unpack from contiguous memory according to datatype.

Parameters

- **inbuf** (*BufSpec*) –
- **position** (*int*) –
- **outbuf** (*BufSpec*) –
- **comm** (Comm) –

Return type int

Unpack_external(*datarep*, *inbuf*, *position*, *outbuf*)

Unpack from contiguous memory according to datatype, using a portable data representation (**external32**).

Parameters

- **datarep** (*str*) –
- **inbuf** (*BufSpec*) –
- **position** (*int*) –
- **outbuf** (*BufSpec*) –

Return type int

decode()

Convenience method for decoding a datatype

Return type Tuple[*Datatype*, str, Dict[str, Any]]

classmethod f2py(*arg*)

Parameters **arg** (*int*) –

Return type *Datatype*

py2f()

Return type int

Attributes Documentation

combiner

datatype combiner

contents

datatype contents

envelope

datatype envelope

extent

is_named

is a named datatype

is_predefined

is a predefined datatype

lb

lower bound

name

datatype name

size

true_extent

true extent

true_lb

true lower bound

true_ub
: true upper bound

ub
: upper bound

mpi4py.MPI.Distgraphcomm

class mpi4py.MPI.**Distgraphcomm**(*comm=None*)

Bases: *mpi4py.MPI.Topocomm*

Distributed graph topology intracommunicator

Parameters **comm** (*Optional*[Distgraphcomm]) –

Return type *Distgraphcomm*

static **__new__**(*cls*, *comm=None*)

Parameters **comm** (*Optional*[Distgraphcomm]) –

Return type *Distgraphcomm*

Methods Summary

Get_dist_neighbors()	Return adjacency information for a distributed graph topology
Get_dist_neighbors_count()	Return adjacency information for a distributed graph topology

Methods Documentation

Get_dist_neighbors()

Return adjacency information for a distributed graph topology

Return type Tuple[List[int], List[int], Optional[Tuple[List[int], List[int]]]]

Get_dist_neighbors_count()

Return adjacency information for a distributed graph topology

Return type int

mpi4py.MPI.Errhandler

class mpi4py.MPI.**Errhandler**(*errhandler=None*)

Bases: object

Error handler

Parameters **errhandler** (*Optional*[Errhandler]) –

Return type *Errhandler*

static **__new__**(*cls*, *errhandler=None*)

Parameters **errhandler** (*Optional*[Errhandler]) –

Return type *Errhandler*

Methods Summary

Free()	Free an error handler
f2py(arg)	
py2f()	

Methods Documentation

Free()

Free an error handler

Return type None

classmethod **f2py**(*arg*)

Parameters **arg** (*int*) –

Return type *Errhandler*

py2f()

Return type int

mpi4py.MPI.File

class mpi4py.MPI.**File**(*file=None*)

Bases: object

File handle

Parameters **file** (*Optional*[File]) –

Return type *File*

static **__new__**(*cls*, *file=None*)

Parameters **file** (*Optional*[File]) –

Return type *File*

Methods Summary

Call_errhandler(errorcode)	Call the error handler installed on a file
Close()	Close a file
Delete(filename[, info])	Delete a file
Get_amode()	Return the file access mode
Get_atomicity()	Return the atomicity mode
Get_byte_offset(offset)	Return the absolute byte position in the file corresponding to 'offset' etypes relative to the current view
Get_errhandler()	Get the error handler for a file
Get_group()	Return the group of processes that opened the file
Get_info()	Return the hints for a file that that are currently in use
Get_position()	Return the current position of the individual file pointer in etype units relative to the current view
Get_position_shared()	Return the current position of the shared file pointer in etype units relative to the current view
Get_size()	Return the file size
Get_type_extent(datatype)	Return the extent of datatype in the file
Get_view()	Return the file view
Iread(buf)	Nonblocking read using individual file pointer
Iread_all(buf)	Nonblocking collective read using individual file pointer
Iread_at(offset, buf)	Nonblocking read using explicit offset
Iread_at_all(offset, buf)	Nonblocking collective read using explicit offset
Iread_shared(buf)	Nonblocking read using shared file pointer
Iwrite(buf)	Nonblocking write using individual file pointer
Iwrite_all(buf)	Nonblocking collective write using individual file pointer
Iwrite_at(offset, buf)	Nonblocking write using explicit offset
Iwrite_at_all(offset, buf)	Nonblocking collective write using explicit offset
Iwrite_shared(buf)	Nonblocking write using shared file pointer
Open(comm, filename[, amode, info])	Open a file
Preallocate(size)	Preallocate storage space for a file
Read(buf[, status])	Read using individual file pointer
Read_all(buf[, status])	Collective read using individual file pointer
Read_all_begin(buf)	Start a split collective read using individual file pointer
Read_all_end(buf[, status])	Complete a split collective read using individual file pointer
Read_at(offset, buf[, status])	Read using explicit offset
Read_at_all(offset, buf[, status])	Collective read using explicit offset
Read_at_all_begin(offset, buf)	Start a split collective read using explict offset
Read_at_all_end(buf[, status])	Complete a split collective read using explict offset
Read_ordered(buf[, status])	Collective read using shared file pointer
Read_ordered_begin(buf)	Start a split collective read using shared file pointer
Read_ordered_end(buf[, status])	Complete a split collective read using shared file pointer
Read_shared(buf[, status])	Read using shared file pointer
Seek(offset[, whence])	Update the individual file pointer
Seek_shared(offset[, whence])	Update the shared file pointer
Set_atomicity(flag)	Set the atomicity mode

continues on next page

Table 27 – continued from previous page

Set_errhandler(errhandler)	Set the error handler for a file
Set_info(info)	Set new values for the hints associated with a file
Set_size(size)	Sets the file size
Set_view([disp, etype, filetype, datarep, info])	Set the file view
Sync()	Causes all previous writes to be transferred to the storage device
Write(buf[, status])	Write using individual file pointer
Write_all(buf[, status])	Collective write using individual file pointer
Write_all_begin(buf)	Start a split collective write using individual file pointer
Write_all_end(buf[, status])	Complete a split collective write using individual file pointer
Write_at(offset, buf[, status])	Write using explicit offset
Write_at_all(offset, buf[, status])	Collective write using explicit offset
Write_at_all_begin(offset, buf)	Start a split collective write using explict offset
Write_at_all_end(buf[, status])	Complete a split collective write using explict offset
Write_ordered(buf[, status])	Collective write using shared file pointer
Write_ordered_begin(buf)	Start a split collective write using shared file pointer
Write_ordered_end(buf[, status])	Complete a split collective write using shared file pointer
Write_shared(buf[, status])	Write using shared file pointer
f2py(arg)	
py2f()	

Attributes Summary

amode	file access mode
atomicity	
group	file group
info	file info
size	file size

Methods Documentation

Call_errhandler(*errorcode*)

Call the error handler installed on a file

Parameters **errorcode** (*int*) –

Return type None

Close()

Close a file

Return type None

classmethod Delete(*filename*, *info=INFO_NULL*)

Delete a file

Parameters

- **filename** (*str*) –
- **info** (Info) –

Return type None

Get_amode()

Return the file access mode

Return type int

Get_atomicity()

Return the atomicity mode

Return type bool

Get_byte_offset(*offset*)

Return the absolute byte position in the file corresponding to 'offset' etypes relative to the current view

Parameters **offset** (*int*) –

Return type int

Get_errhandler()

Get the error handler for a file

Return type *Errhandler*

Get_group()

Return the group of processes that opened the file

Return type *Group*

Get_info()

Return the hints for a file that that are currently in use

Return type *Info*

Get_position()

Return the current position of the individual file pointer in etype units relative to the current view

Return type int

Get_position_shared()

Return the current position of the shared file pointer in etype units relative to the current view

Return type int

Get_size()

Return the file size

Return type int

Get_type_extent(*datatype*)

Return the extent of datatype in the file

Parameters **datatype** (Datatype) –

Return type int

Get_view()

Return the file view

Return type Tuple[int, *Datatype*, *Datatype*, str]

Iread(*buf*)
Nonblocking read using individual file pointer

Parameters **buf** (*BufSpec*) –

Return type *Request*

Iread_all(*buf*)
Nonblocking collective read using individual file pointer

Parameters **buf** (*BufSpec*) –

Return type *Request*

Iread_at(*offset*, *buf*)
Nonblocking read using explicit offset

Parameters

- **offset** (*int*) –
- **buf** (*BufSpec*) –

Return type *Request*

Iread_at_all(*offset*, *buf*)
Nonblocking collective read using explicit offset

Parameters

- **offset** (*int*) –
- **buf** (*BufSpec*) –

Return type *Request*

Iread_shared(*buf*)
Nonblocking read using shared file pointer

Parameters **buf** (*BufSpec*) –

Return type *Request*

Iwrite(*buf*)
Nonblocking write using individual file pointer

Parameters **buf** (*BufSpec*) –

Return type *Request*

Iwrite_all(*buf*)
Nonblocking collective write using individual file pointer

Parameters **buf** (*BufSpec*) –

Return type *Request*

Iwrite_at(*offset*, *buf*)
Nonblocking write using explicit offset

Parameters

- **offset** (*int*) –
- **buf** (*BufSpec*) –

Return type *Request*

Iwrite_at_all(*offset*, *buf*)
Nonblocking collective write using explicit offset

Parameters

- **offset** (*int*) –
- **buf** (*BufSpec*) –

Return type *Request*

Iwrite_shared(*buf*)
Nonblocking write using shared file pointer

Parameters **buf** (*BufSpec*) –

Return type *Request*

classmethod Open(*comm*, *filename*, *amode=MODE_RDONLY*, *info=INFO_NULL*)
Open a file

Parameters

- **comm** (Intracomm) –
- **filename** (*str*) –
- **amode** (*int*) –
- **info** (Info) –

Return type *File*

Preallocate(*size*)
Preallocate storage space for a file

Parameters **size** (*int*) –

Return type None

Read(*buf*, *status=None*)
Read using individual file pointer

Parameters

- **buf** (*BufSpec*) –
- **status** (*Optional*[Status]) –

Return type None

Read_all(*buf*, *status=None*)
Collective read using individual file pointer

Parameters

- **buf** (*BufSpec*) –
- **status** (*Optional*[Status]) –

Return type None

Read_all_begin(*buf*)
Start a split collective read using individual file pointer

Parameters **buf** (*BufSpec*) –

Return type None

Read_all_end(*buf*, *status=None*)

Complete a split collective read using individual file pointer

Parameters

- **buf** (*BufSpec*) –
- **status** (*Optional[Status]*) –

Return type None

Read_at(*offset*, *buf*, *status=None*)

Read using explicit offset

Parameters

- **offset** (*int*) –
- **buf** (*BufSpec*) –
- **status** (*Optional[Status]*) –

Return type None

Read_at_all(*offset*, *buf*, *status=None*)

Collective read using explicit offset

Parameters

- **offset** (*int*) –
- **buf** (*BufSpec*) –
- **status** (*Optional[Status]*) –

Return type None

Read_at_all_begin(*offset*, *buf*)

Start a split collective read using explict offset

Parameters

- **offset** (*int*) –
- **buf** (*BufSpec*) –

Return type None

Read_at_all_end(*buf*, *status=None*)

Complete a split collective read using explict offset

Parameters

- **buf** (*BufSpec*) –
- **status** (*Optional[Status]*) –

Return type None

Read_ordered(*buf*, *status=None*)

Collective read using shared file pointer

Parameters

- **buf** (*BufSpec*) –
- **status** (*Optional[Status]*) –

Return type None

`Read_ordered_begin`(*buf*)
Start a split collective read using shared file pointer

Parameters **buf** (*BufSpec*) –

Return type None

`Read_ordered_end`(*buf*, *status=None*)
Complete a split collective read using shared file pointer

Parameters

- **buf** (*BufSpec*) –
- **status** (*Optional[Status]*) –

Return type None

`Read_shared`(*buf*, *status=None*)
Read using shared file pointer

Parameters

- **buf** (*BufSpec*) –
- **status** (*Optional[Status]*) –

Return type None

`Seek`(*offset*, *whence=SEEK_SET*)
Update the individual file pointer

Parameters

- **offset** (*int*) –
- **whence** (*int*) –

Return type None

`Seek_shared`(*offset*, *whence=SEEK_SET*)
Update the shared file pointer

Parameters

- **offset** (*int*) –
- **whence** (*int*) –

Return type None

`Set_atomicity`(*flag*)
Set the atomicity mode

Parameters **flag** (*bool*) –

Return type None

`Set_errhandler`(*errhandler*)
Set the error handler for a file

Parameters **errhandler** (*Errhandler*) –

Return type None

`Set_info`(*info*)
Set new values for the hints associated with a file

Parameters **info** (*Info*) –

Return type None

Set_size(*size*)

Sets the file size

Parameters **size** (*int*) –

Return type None

Set_view(*disp=0*, *etype=BYTE*, *filetype=None*, *datarep='native'*, *info=INFO_NULL*)

Set the file view

Parameters

- **disp** (*int*) –
- **etype** (Datatype) –
- **filetype** (*Optional*[Datatype]) –
- **datarep** (*str*) –
- **info** (Info) –

Return type None

Sync()

Causes all previous writes to be transferred to the storage device

Return type None

Write(*buf*, *status=None*)

Write using individual file pointer

Parameters

- **buf** (*BufSpec*) –
- **status** (*Optional*[Status]) –

Return type None

Write_all(*buf*, *status=None*)

Collective write using individual file pointer

Parameters

- **buf** (*BufSpec*) –
- **status** (*Optional*[Status]) –

Return type None

Write_all_begin(*buf*)

Start a split collective write using individual file pointer

Parameters **buf** (*BufSpec*) –

Return type None

Write_all_end(*buf*, *status=None*)

Complete a split collective write using individual file pointer

Parameters

- **buf** (*BufSpec*) –
- **status** (*Optional*[Status]) –

Return type None

Write_at(*offset*, *buf*, *status=None*)
Write using explicit offset

Parameters

- **offset** (*int*) –
- **buf** (*BufSpec*) –
- **status** (*Optional*[*Status*]) –

Return type None

Write_at_all(*offset*, *buf*, *status=None*)
Collective write using explicit offset

Parameters

- **offset** (*int*) –
- **buf** (*BufSpec*) –
- **status** (*Optional*[*Status*]) –

Return type None

Write_at_all_begin(*offset*, *buf*)
Start a split collective write using explict offset

Parameters

- **offset** (*int*) –
- **buf** (*BufSpec*) –

Return type None

Write_at_all_end(*buf*, *status=None*)
Complete a split collective write using explict offset

Parameters

- **buf** (*BufSpec*) –
- **status** (*Optional*[*Status*]) –

Return type None

Write_ordered(*buf*, *status=None*)
Collective write using shared file pointer

Parameters

- **buf** (*BufSpec*) –
- **status** (*Optional*[*Status*]) –

Return type None

Write_ordered_begin(*buf*)
Start a split collective write using shared file pointer

Parameters **buf** (*BufSpec*) –

Return type None

Write_ordered_end(*buf*, *status=None*)

Complete a split collective write using shared file pointer

Parameters

- **buf** (*BufSpec*) –
- **status** (*Optional[Status]*) –

Return type None

Write_shared(*buf*, *status=None*)

Write using shared file pointer

Parameters

- **buf** (*BufSpec*) –
- **status** (*Optional[Status]*) –

Return type None

classmethod f2py(*arg*)

Parameters **arg** (*int*) –

Return type *File*

py2f()

Return type int

Attributes Documentation

amode

file access mode

atomicity

group

file group

info

file info

size

file size

mpi4py.MPI.Graphcomm

class mpi4py.MPI.**Graphcomm**(*comm=None*)

Bases: *mpi4py.MPI.Topocomm*

General graph topology intracommunicator

Parameters **comm** (*Optional[Graphcomm]*) –

Return type *Graphcomm*

static __new__(*cls*, *comm=None*)

Parameters **comm** (*Optional[Graphcomm]*) –

Return type *Graphcomm*

Methods Summary

Get_dims()	Return the number of nodes and edges
Get_neighbors(rank)	Return list of neighbors of a process
Get_neighbors_count(rank)	Return number of neighbors of a process
Get_topo()	Return index and edges

Attributes Summary

dims	number of nodes and edges
edges	
index	
nedges	number of edges
neighbors	
nneighbors	number of neighbors
nnodes	number of nodes
topo	topology information

Methods Documentation

Get_dims()

Return the number of nodes and edges

Return type Tuple[int, int]

Get_neighbors(*rank*)

Return list of neighbors of a process

Parameters **rank** (*int*) –

Return type List[int]

Get_neighbors_count(*rank*)

Return number of neighbors of a process

Parameters **rank** (*int*) –

Return type int

Get_topo()

Return index and edges

Return type Tuple[List[int], List[int]]

Attributes Documentation

dims
: number of nodes and edges

edges

index

nedges
: number of edges

neighbors

nneighbors
: number of neighbors

nnodes
: number of nodes

topo
: topology information

mpi4py.MPI.Grequest

class mpi4py.MPI.**Grequest**(*request=None*)

Bases: *mpi4py.MPI.Request*

Generalized request handle

Parameters **request** (*Optional[Grequest]*) –

Return type *Grequest*

static **__new__**(*cls*, *request=None*)

Parameters **request** (*Optional[Grequest]*) –

Return type *Grequest*

Methods Summary

Complete()	Notify that a user-defined request is complete
Start(query_fn, free_fn, cancel_fn[, args, ...])	Create and return a user-defined request

Methods Documentation

Complete()

Notify that a user-defined request is complete

Return type None

classmethod **Start**(*query_fn*, *free_fn*, *cancel_fn*, *args=None*, *kargs=None*)

Create and return a user-defined request

Parameters

- **query_fn** (*Callable[..., None]*) –

- **free_fn** (*Callable[..., None]*) –
- **cancel_fn** (*Callable[..., None]*) –
- **args** (*Optional[Tuple[Any]]*) –
- **kargs** (*Optional[Dict[str, Any]]*) –

Return type *Grequest*

mpi4py.MPI.Group

class mpi4py.MPI.**Group**(*group=None*)

Bases: object

Group of processes

Parameters **group** (*Optional[Group]*) –

Return type *Group*

static **__new__**(*cls*, *group=None*)

Parameters **group** (*Optional[Group]*) –

Return type *Group*

Methods Summary

Compare(group1, group2)	Compare two groups
Difference(group1, group2)	Produce a group from the difference of two existing groups
Dup()	Duplicate a group
Excl(ranks)	Produce a group by reordering an existing group and taking only unlisted members
Free()	Free a group
Get_rank()	Return the rank of this process in a group
Get_size()	Return the size of a group
Incl(ranks)	Produce a group by reordering an existing group and taking only listed members
Intersection(group1, group2)	Produce a group as the intersection of two existing groups
Range_excl(ranks)	Create a new group by excluding ranges of processes from an existing group
Range_incl(ranks)	Create a new group from ranges of of ranks in an existing group
Translate_ranks(group1, ranks1[, group2])	Translate the ranks of processes in one group to those in another group
Union(group1, group2)	Produce a group by combining two existing groups
f2py(arg)	
py2f()	

Attributes Summary

rank	rank of this process in group
size	number of processes in group

Methods Documentation

classmethod Compare(*group1*, *group2*)

Compare two groups

Parameters

- **group1** (Group) –
- **group2** (Group) –

Return type int

classmethod Difference(*group1*, *group2*)

Produce a group from the difference of two existing groups

Parameters

- **group1** (Group) –
- **group2** (Group) –

Return type *Group*

Dup()

Duplicate a group

Return type *Group*

Excl(*ranks*)

Produce a group by reordering an existing group and taking only unlisted members

Parameters **ranks** (*Sequence[int]*) –

Return type *Group*

Free()

Free a group

Return type None

Get_rank()

Return the rank of this process in a group

Return type int

Get_size()

Return the size of a group

Return type int

Incl(*ranks*)

Produce a group by reordering an existing group and taking only listed members

Parameters **ranks** (*Sequence[int]*) –

Return type *Group*

`classmethod Intersection`(*group1*, *group2*)

Produce a group as the intersection of two existing groups

Parameters

- **group1** (Group) –
- **group2** (Group) –

Return type *Group*

`Range_excl`(*ranks*)

Create a new group by excluding ranges of processes from an existing group

Parameters **ranks** (*Sequence[Tuple[int, int, int]]*) –

Return type *Group*

`Range_incl`(*ranks*)

Create a new group from ranges of of ranks in an existing group

Parameters **ranks** (*Sequence[Tuple[int, int, int]]*) –

Return type *Group*

`classmethod Translate_ranks`(*group1*, *ranks1*, *group2=None*)

Translate the ranks of processes in one group to those in another group

Parameters

- **ranks1** (*Sequence[int]*) –
- **group1** (Group) –
- **group2** (*Optional[Group]*) –

Return type List[int]

`classmethod Union`(*group1*, *group2*)

Produce a group by combining two existing groups

Parameters

- **group1** (Group) –
- **group2** (Group) –

Return type *Group*

`classmethod f2py`(*arg*)

Parameters **arg** (*int*) –

Return type *Group*

`py2f`()

Return type int

Attributes Documentation

rank
: rank of this process in group

size
: number of processes in group

mpi4py.MPI.Info

class mpi4py.MPI.**Info**(*info=None*)

Bases: object

Info object

Parameters **info** (*Optional[Info]*) –

Return type *Info*

static **__new__**(*cls*, *info=None*)

Parameters **info** (*Optional[Info]*) –

Return type *Info*

Methods Summary

Create()	Create a new, empty info object
Delete(key)	Remove a (key, value) pair from info
Dup()	Duplicate an existing info object, creating a new object, with the same (key, value) pairs and the same ordering of keys
Free()	Free a info object
Get(key[, maxlen])	Retrieve the value associated with a key
Get_nkeys()	Return the number of currently defined keys in info
Get_nthkey(n)	Return the nth defined key in info.
Set(key, value)	Add the (key, value) pair to info, and overrides the value if a value for the same key was previously set
clear()	info clear
copy()	info copy
f2py(arg)	
get(key[, default])	info get
items()	info items
keys()	info keys
pop(key, *default)	info pop
popitem()	info popitem
py2f()	
update([other])	info update
values()	info values

Methods Documentation

classmethod Create()

Create a new, empty info object

Return type *Info*

Delete(*key*)

Remove a (key, value) pair from info

Parameters **key** (*str*) –

Return type None

Dup()

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

Return type *Info*

Free()

Free a info object

Return type None

Get(*key*, *maxlen=- 1*)

Retrieve the value associated with a key

Parameters

- **key** (*str*) –
- **maxlen** (*int*) –

Return type Optional[str]

Get_nkeys()

Return the number of currently defined keys in info

Return type int

Get_nthkey(*n*)

Return the nth defined key in info. Keys are numbered in the range [0, N) where N is the value returned by *Info.Get_nkeys()*

Parameters **n** (*int*) –

Return type str

Set(*key*, *value*)

Add the (key, value) pair to info, and overrides the value if a value for the same key was previously set

Parameters

- **key** (*str*) –
- **value** (*str*) –

Return type None

clear()

info clear

Return type None

copy()

info copy

Return type *Info*

classmethod f2py(*arg*)

Parameters **arg** (*int*) –

Return type *Info*

get(*key*, *default=None*)

info get

Parameters

- **key** (*str*) –
- **default** (*Optional[str]*) –

Return type Optional[str]

items()

info items

Return type List[Tuple[str, str]]

keys()

info keys

Return type List[str]

pop(*key*, **default*)

info pop

Parameters

- **key** (*str*) –
- **default** (*str*) –

Return type str

popitem()

info popitem

Return type Tuple[str, str]

py2f()

Return type int

update(*other=()*, ***kwds*)

info update

Parameters

- **other** (*Union[Info, Mapping[str, str], Iterable[Tuple[str, str]]]*) –
- **kwds** (*str*) –

Return type None

values()

info values

Return type List[str]

mpi4py.MPI.Intercomm

class mpi4py.MPI.**Intercomm**(*comm=None*)

Bases: *mpi4py.MPI.Comm*

Intercommunicator

Parameters **comm** (*Optional[Intercomm]*) –

Return type *Intercomm*

static __new__(*cls*, *comm=None*)

Parameters **comm** (*Optional[Intercomm]*) –

Return type *Intercomm*

Methods Summary

Get_remote_group()	Access the remote group associated with the inter-communicator
Get_remote_size()	Intercommunicator remote size
Merge([high])	Merge intercommunicator

Attributes Summary

remote_group	remote group
remote_size	number of remote processes

Methods Documentation

Get_remote_group()

Access the remote group associated with the inter-communicator

Return type *Group*

Get_remote_size()

Intercommunicator remote size

Return type int

Merge(*high=False*)

Merge intercommunicator

Parameters **high** (*bool*) –

Return type *Intracomm*

Attributes Documentation

remote_group
remote group

remote_size
number of remote processes

mpi4py.MPI.Intracomm

class mpi4py.MPI.**Intracomm**(*comm=None*)

Bases: *mpi4py.MPI.Comm*

Intracommunicator

Parameters **comm** (*Optional[Intracomm]*) –

Return type *Intracomm*

static __new__(*cls*, *comm=None*)

Parameters **comm** (*Optional[Intracomm]*) –

Return type *Intracomm*

Methods Summary

Accept(port_name[, info, root])	Accept a request to form a new intercommunicator
Cart_map(dims[, periods])	Return an optimal placement for the calling process on the physical machine
Connect(port_name[, info, root])	Make a request to form a new intercommunicator
Create_cart(dims[, periods, reorder])	Create cartesian communicator
Create_dist_graph(sources, degrees, destinations)	Create distributed graph communicator
Create_dist_graph_adjacent(sources, destinations)	Create distributed graph communicator
Create_graph(index, edges[, reorder])	Create graph communicator
Create_intercomm(local_leader, peer_comm, ...)	Create intercommunicator
Exscan(sendbuf, recvbuf[, op])	Exclusive Scan
Graph_map(index, edges)	Return an optimal placement for the calling process on the physical machine
Iexscan(sendbuf, recvbuf[, op])	Inclusive Scan
Iscan(sendbuf, recvbuf[, op])	Inclusive Scan
Scan(sendbuf, recvbuf[, op])	Inclusive Scan
Spawn(command[, args, maxprocs, info, root, ...])	Spawn instances of a single MPI application
Spawn_multiple(command[, args, maxprocs, ...])	Spawn instances of multiple MPI applications
exscan(sendobj[, op])	Exclusive Scan
scan(sendobj[, op])	Inclusive Scan

Methods Documentation

Accept(*port_name*, *info=INFO_NULL*, *root=0*)
Accept a request to form a new intercommunicator

Parameters

- **port_name** (*str*) –
- **info** (Info) –
- **root** (*int*) –

Return type *Intercomm*

Cart_map(*dims*, *periods=None*)
Return an optimal placement for the calling process on the physical machine

Parameters

- **dims** (*Sequence[int]*) –
- **periods** (*Optional[Sequence[bool]]*) –

Return type int

Connect(*port_name*, *info=INFO_NULL*, *root=0*)
Make a request to form a new intercommunicator

Parameters

- **port_name** (*str*) –
- **info** (Info) –
- **root** (*int*) –

Return type *Intercomm*

Create_cart(*dims*, *periods=None*, *reorder=False*)
Create cartesian communicator

Parameters

- **dims** (*Sequence[int]*) –
- **periods** (*Optional[Sequence[bool]]*) –
- **reorder** (*bool*) –

Return type *Cartcomm*

Create_dist_graph(*sources*, *degrees*, *destinations*, *weights=None*, *info=INFO_NULL*, *reorder=False*)
Create distributed graph communicator

Parameters

- **sources** (*Sequence[int]*) –
- **degrees** (*Sequence[int]*) –
- **destinations** (*Sequence[int]*) –
- **weights** (*Optional[Sequence[int]]*) –
- **info** (Info) –
- **reorder** (*bool*) –

Return type *Distgraphcomm*

Create_dist_graph_adjacent(*sources*, *destinations*, *sourceweights=None*, *destweights=None*, *info=INFO_NULL*, *reorder=False*)

Create distributed graph communicator

Parameters

- **sources** (*Sequence[int]*) –
- **destinations** (*Sequence[int]*) –
- **sourceweights** (*Optional[Sequence[int]]*) –
- **destweights** (*Optional[Sequence[int]]*) –
- **info** (Info) –
- **reorder** (*bool*) –

Return type *Distgraphcomm*

Create_graph(*index*, *edges*, *reorder=False*)

Create graph communicator

Parameters

- **index** (*Sequence[int]*) –
- **edges** (*Sequence[int]*) –
- **reorder** (*bool*) –

Return type *Graphcomm*

Create_intercomm(*local_leader*, *peer_comm*, *remote_leader*, *tag=0*)

Create intercommunicator

Parameters

- **local_leader** (*int*) –
- **peer_comm** (Intracomm) –
- **remote_leader** (*int*) –
- **tag** (*int*) –

Return type *Intercomm*

Exscan(*sendbuf*, *recvbuf*, *op=SUM*)

Exclusive Scan

Parameters

- **sendbuf** (*Union[BufSpec, InPlace]*) –
- **recvbuf** (*BufSpec*) –
- **op** (Op) –

Return type None

Graph_map(*index*, *edges*)

Return an optimal placement for the calling process on the physical machine

Parameters

- **index** (*Sequence[int]*) –

- **edges** (*Sequence[int]*) –

Return type int

Iexscan(*sendbuf*, *recvbuf*, *op=SUM*)
Inclusive Scan

Parameters

- **sendbuf** (*Union[BufSpec, InPlace]*) –
- **recvbuf** (*BufSpec*) –
- **op** (Op) –

Return type *Request*

Iscan(*sendbuf*, *recvbuf*, *op=SUM*)
Inclusive Scan

Parameters

- **sendbuf** (*Union[BufSpec, InPlace]*) –
- **recvbuf** (*BufSpec*) –
- **op** (Op) –

Return type *Request*

Scan(*sendbuf*, *recvbuf*, *op=SUM*)
Inclusive Scan

Parameters

- **sendbuf** (*Union[BufSpec, InPlace]*) –
- **recvbuf** (*BufSpec*) –
- **op** (Op) –

Return type None

Spawn(*command*, *args=None*, *maxprocs=1*, *info=INFO_NULL*, *root=0*, *errcodes=None*)
Spawn instances of a single MPI application

Parameters

- **command** (*str*) –
- **args** (*Optional[Sequence[str]]*) –
- **maxprocs** (*int*) –
- **info** (Info) –
- **root** (*int*) –
- **errcodes** (*Optional[list]*) –

Return type *Intercomm*

Spawn_multiple(*command*, *args=None*, *maxprocs=None*, *info=INFO_NULL*, *root=0*, *errcodes=None*)
Spawn instances of multiple MPI applications

Parameters

- **command** (*Sequence[str]*) –
- **args** (*Optional[Sequence[Sequence[str]]]*) –

- **maxprocs** (*Optional[Sequence[int]]*) –
- **info** (*Union[Info, Sequence[Info]]*) –
- **root** (*int*) –
- **errcodes** (*Optional[list]*) –

Return type *Intercomm*

exscan(*sendobj*, *op=SUM*)
Exclusive Scan

Parameters

- **sendobj** (*Any*) –
- **op** (*Union[Op, Callable[[Any, Any], Any]]*) –

Return type Any

scan(*sendobj*, *op=SUM*)
Inclusive Scan

Parameters

- **sendobj** (*Any*) –
- **op** (*Union[Op, Callable[[Any, Any], Any]]*) –

Return type Any

mpi4py.MPI.Message

class mpi4py.MPI.**Message**(*message=None*)
Bases: object

Matched message handle

Parameters **message** (*Optional[Message]*) –

Return type *Message*

static **__new__**(*cls*, *message=None*)

Parameters **message** (*Optional[Message]*) –

Return type *Message*

Methods Summary

Iprobe(comm[, source, tag, status])	Nonblocking test for a matched message
Irecv(buf)	Nonblocking receive of matched message
Probe(comm[, source, tag, status])	Blocking test for a matched message
Recv(buf[, status])	Blocking receive of matched message
f2py(arg)	
iprobe(comm[, source, tag, status])	Nonblocking test for a matched message
irecv()	Nonblocking receive of matched message

continues on next page

Table 38 – continued from previous page

probe(comm[, source, tag, status])	Blocking test for a matched message
py2f()	
recv([status])	Blocking receive of matched message

Methods Documentation

classmethod Iprobe(*comm*, *source=ANY_SOURCE*, *tag=ANY_TAG*, *status=None*)

Nonblocking test for a matched message

Parameters

- **comm** (Comm) –
- **source** (*int*) –
- **tag** (*int*) –
- **status** (*Optional[Status]*) –

Return type Optional[*Message*]

Irecv(*buf*)

Nonblocking receive of matched message

Parameters **buf** (*BufSpec*) –

Return type *Request*

classmethod Probe(*comm*, *source=ANY_SOURCE*, *tag=ANY_TAG*, *status=None*)

Blocking test for a matched message

Parameters

- **comm** (Comm) –
- **source** (*int*) –
- **tag** (*int*) –
- **status** (*Optional[Status]*) –

Return type *Message*

Recv(*buf*, *status=None*)

Blocking receive of matched message

Parameters

- **buf** (*BufSpec*) –
- **status** (*Optional[Status]*) –

Return type None

classmethod f2py(*arg*)

Parameters **arg** (*int*) –

Return type *Message*

classmethod iprobe(*comm*, *source=ANY_SOURCE*, *tag=ANY_TAG*, *status=None*)

Nonblocking test for a matched message

Parameters

- **comm** (Comm) –
- **source** (*int*) –
- **tag** (*int*) –
- **status** (*Optional[Status]*) –

Return type Optional[*Message*]

irecv()

Nonblocking receive of matched message

Return type *Request*

classmethod probe(*comm*, *source=ANY_SOURCE*, *tag=ANY_TAG*, *status=None*)

Blocking test for a matched message

Parameters

- **comm** (Comm) –
- **source** (*int*) –
- **tag** (*int*) –
- **status** (*Optional[Status]*) –

Return type *Message*

py2f()

Return type int

recv(*status=None*)

Blocking receive of matched message

Parameters **status** (*Optional[Status]*) –

Return type Any

mpi4py.MPI.Op

class mpi4py.MPI.**Op**(*op=None*)

Bases: object

Operation object

Parameters **op** (*Optional[Op]*) –

Return type *Op*

static __new__(*cls*, *op=None*)

Parameters **op** (*Optional[Op]*) –

Return type *Op*

Methods Summary

Create(function[, commute])	Create a user-defined operation
Free()	Free the operation
Is_commutative()	Query reduction operations for their commutativity
Reduce_local(inbuf, inoutbuf)	Apply a reduction operator to local data
f2py(arg)	
py2f()	

Attributes Summary

is_commutative	is commutative
is_predefined	is a predefined operation

Methods Documentation

classmethod Create(*function*, *commute=False*)

Create a user-defined operation

Parameters

- **function** (*Callable[[Buffer, Buffer, Datatype], None]*) –
- **commute** (*bool*) –

Return type *Op*

Free()

Free the operation

Return type None

Is_commutative()

Query reduction operations for their commutativity

Return type bool

Reduce_local(*inbuf*, *inoutbuf*)

Apply a reduction operator to local data

Parameters

- **inbuf** (*BufSpec*) –
- **inoutbuf** (*BufSpec*) –

Return type None

classmethod f2py(*arg*)

Parameters **arg** (*int*) –

Return type *Op*

py2f()

Return type int

Attributes Documentation

is_commutative
is commutative

is_predefined
is a predefined operation

mpi4py.MPI.Pickle

class mpi4py.MPI.**Pickle**(*dumps=None*, *loads=None*, *protocol=None*)

Bases: `object`

Pickle/unpickle Python objects

Parameters

- **dumps** (*Optional[Callable[[Any, int], bytes]]*) –
- **loads** (*Optional[Callable[[Buffer], Any]]*) –
- **protocol** (*Optional[int]*) –

Return type None

__init__(*dumps=None*, *loads=None*, *protocol=None*)

Parameters

- **dumps** (*Optional[Callable[[Any, int], bytes]]*) –
- **loads** (*Optional[Callable[[Buffer], Any]]*) –
- **protocol** (*Optional[int]*) –

Return type None

Methods Summary

dumps(obj[, buffer_callback])	Serialize object to pickle data stream.
loads(data[, buffers])	Deserialize object from pickle data stream.

Attributes Summary

PROTOCOL	pickle protocol

Methods Documentation

dumps(*obj*, *buffer_callback=None*)

Serialize object to pickle data stream.

Parameters

- **obj** (*Any*) –
- **buffer_callback** (*Optional[Callable[[Buffer], Any]]*) –

Return type bytes

loads(*data*, *buffers=None*)

Deserialize object from pickle data stream.

Parameters

- **data** (*Buffer*) –
- **buffers** (*Optional[Iterable[Buffer]]*) –

Return type Any

Attributes Documentation

PROTOCOL

pickle protocol

mpi4py.MPI.Prequest

class mpi4py.MPI.**Prequest**(*request=None*)

Bases: *mpi4py.MPI.Request*

Persistent request handle

Parameters **request** (*Optional[Prequest]*) –

Return type *Prequest*

static **__new__**(*cls*, *request=None*)

Parameters **request** (*Optional[Prequest]*) –

Return type *Prequest*

Methods Summary

Start()	Initiate a communication with a persistent request
Startall(requests)	Start a collection of persistent requests

Methods Documentation

Start()

Initiate a communication with a persistent request

Return type None

classmethod **Startall**(*requests*)

Start a collection of persistent requests

Parameters **requests** (*List*[Prequest]) –

Return type None

mpi4py.MPI.Request

class mpi4py.MPI.**Request**(*request=None*)

Bases: object

Request handle

Parameters **request** (*Optional*[Request]) –

Return type *Request*

static **__new__**(*cls*, *request=None*)

Parameters **request** (*Optional*[Request]) –

Return type *Request*

Methods Summary

Cancel()	Cancel a communication request
Free()	Free a communication request
Get_status([status])	Non-destructive test for the completion of a request
Test([status])	Test for the completion of a send or receive
Testall(requests[, statuses])	Test for completion of all previously initiated requests
Testany(requests[, status])	Test for completion of any previously initiated request
Testsome(requests[, statuses])	Test for completion of some previously initiated requests
Wait([status])	Wait for a send or receive to complete
Waitall(requests[, statuses])	Wait for all previously initiated requests to complete
Waitany(requests[, status])	Wait for any previously initiated request to complete
Waitsome(requests[, statuses])	Wait for some previously initiated requests to complete
cancel()	Cancel a communication request
f2py(arg)	
get_status([status])	Non-destructive test for the completion of a request
py2f()	
test([status])	Test for the completion of a send or receive
testall(requests[, statuses])	Test for completion of all previously initiated requests

continues on next page

Table 44 – continued from previous page

`testany`(requests[, status])	Test for completion of any previously initiated request
`testsome`(requests[, statuses])	Test for completion of some previously initiated requests
`wait`([status])	Wait for a send or receive to complete
`waitall`(requests[, statuses])	Wait for all previously initiated requests to complete
`waitany`(requests[, status])	Wait for any previously initiated request to complete
`waitsome`(requests[, statuses])	Wait for some previously initiated requests to complete

Methods Documentation

Cancel()

Cancel a communication request

Return type None

Free()

Free a communication request

Return type None

Get_status(*status=None*)

Non-destructive test for the completion of a request

Parameters **status** (*Optional[Status]*) –

Return type bool

Test(*status=None*)

Test for the completion of a send or receive

Parameters **status** (*Optional[Status]*) –

Return type bool

classmethod **Testall**(*requests*, *statuses=None*)

Test for completion of all previously initiated requests

Parameters

- **requests** (*Sequence[Request]*) –
- **statuses** (*Optional[List[Status]]*) –

Return type bool

classmethod **Testany**(*requests*, *status=None*)

Test for completion of any previously initiated request

Parameters

- **requests** (*Sequence[Request]*) –
- **status** (*Optional[Status]*) –

Return type Tuple[int, bool]

classmethod **Testsome**(*requests*, *statuses=None*)

Test for completion of some previously initiated requests

Parameters

- **requests** (*Sequence[Request]*) –

- **statuses** (*Optional[List[Status]]*) –

Return type Optional[List[int]]

Wait(*status=None*)
Wait for a send or receive to complete

Parameters **status** (*Optional[Status]*) –

Return type Literal[True]

classmethod Waitall(*requests, statuses=None*)
Wait for all previously initiated requests to complete

Parameters

- **requests** (*Sequence[Request]*) –
- **statuses** (*Optional[List[Status]]*) –

Return type Literal[True]

classmethod Waitany(*requests, status=None*)
Wait for any previously initiated request to complete

Parameters

- **requests** (*Sequence[Request]*) –
- **status** (*Optional[Status]*) –

Return type int

classmethod Waitsome(*requests, statuses=None*)
Wait for some previously initiated requests to complete

Parameters

- **requests** (*Sequence[Request]*) –
- **statuses** (*Optional[List[Status]]*) –

Return type Optional[List[int]]

cancel()
Cancel a communication request

Return type None

classmethod f2py(*arg*)

Parameters **arg** (*int*) –

Return type *Request*

get_status(*status=None*)
Non-destructive test for the completion of a request

Parameters **status** (*Optional[Status]*) –

Return type bool

py2f()

Return type int

test(*status=None*)
Test for the completion of a send or receive

Parameters **status** (*Optional[Status]*) –

Return type Tuple[bool, Optional[Any]]

classmethod testall(*requests*, *statuses=None*)
Test for completion of all previously initiated requests

Parameters

- **requests** (*Sequence[Request]*) –
- **statuses** (*Optional[List[Status]]*) –

Return type Tuple[bool, Optional[List[Any]]]

classmethod testany(*requests*, *status=None*)
Test for completion of any previously initiated request

Parameters

- **requests** (*Sequence[Request]*) –
- **status** (*Optional[Status]*) –

Return type Tuple[int, bool, Optional[Any]]

classmethod testsome(*requests*, *statuses=None*)
Test for completion of some previously initiated requests

Parameters

- **requests** (*Sequence[Request]*) –
- **statuses** (*Optional[List[Status]]*) –

Return type Tuple[Optional[List[int]], Optional[List[Any]]]

wait(*status=None*)
Wait for a send or receive to complete

Parameters **status** (*Optional[Status]*) –

Return type Any

classmethod waitall(*requests*, *statuses=None*)
Wait for all previously initiated requests to complete

Parameters

- **requests** (*Sequence[Request]*) –
- **statuses** (*Optional[List[Status]]*) –

Return type List[Any]

classmethod waitany(*requests*, *status=None*)
Wait for any previously initiated request to complete

Parameters

- **requests** (*Sequence[Request]*) –
- **status** (*Optional[Status]*) –

Return type Tuple[int, Any]

classmethod waitsome(*requests*, *statuses=None*)

Wait for some previously initiated requests to complete

Parameters

- **requests** (*Sequence[Request]*) –
- **statuses** (*Optional[List[Status]]*) –

Return type Tuple[Optional[List[int]], Optional[List[Any]]]

mpi4py.MPI.Status

class mpi4py.MPI.Status(*status=None*)

Bases: `object`

Status object

Parameters **status** (*Optional[Status]*) –

Return type *Status*

static __new__(*cls*, *status=None*)

Parameters **status** (*Optional[Status]*) –

Return type *Status*

Methods Summary

Get_count([datatype])	Get the number of *top level* elements
Get_elements(datatype)	Get the number of basic elements in a datatype
Get_error()	Get message error
Get_source()	Get message source
Get_tag()	Get message tag
Is_cancelled()	Test to see if a request was cancelled
Set_cancelled(flag)	Set the cancelled state associated with a status
Set_elements(datatype, count)	Set the number of elements in a status
Set_error(error)	Set message error
Set_source(source)	Set message source
Set_tag(tag)	Set message tag
f2py(arg)	
py2f()	

Attributes Summary

cancelled	cancelled state
count	byte count
error	
source	
tag	

Methods Documentation

Get_count(*datatype=BYTE*)
Get the number of *top level* elements

Parameters **datatype** (Datatype) –

Return type int

Get_elements(*datatype*)
Get the number of basic elements in a datatype

Parameters **datatype** (Datatype) –

Return type int

Get_error()
Get message error

Return type int

Get_source()
Get message source

Return type int

Get_tag()
Get message tag

Return type int

Is_cancelled()
Test to see if a request was cancelled

Return type bool

Set_cancelled(*flag*)
Set the cancelled state associated with a status

Note: This should be only used when implementing query callback functions for generalized requests

Parameters **flag** (*bool*) –

Return type None

`Set_elements`(*datatype*, *count*)

Set the number of elements in a status

Note: This should be only used when implementing query callback functions for generalized requests

Parameters

- **datatype** (Datatype) –
- **count** (*int*) –

Return type None

`Set_error`(*error*)

Set message error

Parameters **error** (*int*) –

Return type None

`Set_source`(*source*)

Set message source

Parameters **source** (*int*) –

Return type None

`Set_tag`(*tag*)

Set message tag

Parameters **tag** (*int*) –

Return type None

`classmethod f2py`(*arg*)

Parameters **arg** (*List[int]*) –

Return type *Status*

`py2f`()

Return type List[int]

Attributes Documentation

`cancelled`

cancelled state

`count`

byte count

`error`

`source`

`tag`

mpi4py.MPI.Topocomm

class mpi4py.MPI.**Topocomm**(*comm=None*)

Bases: *mpi4py.MPI.Intracomm*

Topology intracommunicator

Parameters **comm** (*Optional[Topocomm]*) –

Return type *Topocomm*

static **__new__**(*cls*, *comm=None*)

Parameters **comm** (*Optional[Topocomm]*) –

Return type *Topocomm*

Methods Summary

Ineighbor_allgather(sendbuf, recvbuf)	Nonblocking Neighbor Gather to All
Ineighbor_allgatherv(sendbuf, recvbuf)	Nonblocking Neighbor Gather to All Vector
Ineighbor_alltoall(sendbuf, recvbuf)	Nonblocking Neighbor All-to-All
Ineighbor_alltoallv(sendbuf, recvbuf)	Nonblocking Neighbor All-to-All Vector
Ineighbor_alltoallw(sendbuf, recvbuf)	Nonblocking Neighbor All-to-All Generalized
Neighbor_allgather(sendbuf, recvbuf)	Neighbor Gather to All
Neighbor_allgatherv(sendbuf, recvbuf)	Neighbor Gather to All Vector
Neighbor_alltoall(sendbuf, recvbuf)	Neighbor All-to-All
Neighbor_alltoallv(sendbuf, recvbuf)	Neighbor All-to-All Vector
Neighbor_alltoallw(sendbuf, recvbuf)	Neighbor All-to-All Generalized
neighbor_allgather(sendobj)	Neighbor Gather to All
neighbor_alltoall(sendobj)	Neighbor All to All Scatter/Gather

Attributes Summary

degrees	number of incoming and outgoing neighbors
indegree	number of incoming neighbors
inedges	incoming neighbors
inoutedges	incoming and outgoing neighbors
outdegree	number of outgoing neighbors
outedges	outgoing neighbors

Methods Documentation

Ineighbor_allgather(*sendbuf*, *recvbuf*)

Nonblocking Neighbor Gather to All

Parameters

- **sendbuf** (*BufSpec*) –
- **recvbuf** (*BufSpecB*) –

Return type *Request*

Ineighbor_allgatherv(*sendbuf*, *recvbuf*)
Nonblocking Neighbor Gather to All Vector

Parameters

- **sendbuf** (*BufSpec*) –
- **recvbuf** (*BufSpecV*) –

Return type *Request*

Ineighbor_alltoall(*sendbuf*, *recvbuf*)
Nonblocking Neighbor All-to-All

Parameters

- **sendbuf** (*BufSpecB*) –
- **recvbuf** (*BufSpecB*) –

Return type *Request*

Ineighbor_alltoallv(*sendbuf*, *recvbuf*)
Nonblocking Neighbor All-to-All Vector

Parameters

- **sendbuf** (*BufSpecV*) –
- **recvbuf** (*BufSpecV*) –

Return type *Request*

Ineighbor_alltoallw(*sendbuf*, *recvbuf*)
Nonblocking Neighbor All-to-All Generalized

Parameters

- **sendbuf** (*BufSpecW*) –
- **recvbuf** (*BufSpecW*) –

Return type *Request*

Neighbor_allgather(*sendbuf*, *recvbuf*)
Neighbor Gather to All

Parameters

- **sendbuf** (*BufSpec*) –
- **recvbuf** (*BufSpecB*) –

Return type None

Neighbor_allgatherv(*sendbuf*, *recvbuf*)
Neighbor Gather to All Vector

Parameters

- **sendbuf** (*BufSpec*) –
- **recvbuf** (*BufSpecV*) –

Return type None

Neighbor_alltoall(*sendbuf*, *recvbuf*)
Neighbor All-to-All

Parameters

- **sendbuf** (*BufSpecB*) –
- **recvbuf** (*BufSpecB*) –

Return type None

Neighbor_alltoallv(*sendbuf*, *recvbuf*)
Neighbor All-to-All Vector

Parameters

- **sendbuf** (*BufSpecV*) –
- **recvbuf** (*BufSpecV*) –

Return type None

Neighbor_alltoallw(*sendbuf*, *recvbuf*)
Neighbor All-to-All Generalized

Parameters

- **sendbuf** (*BufSpecW*) –
- **recvbuf** (*BufSpecW*) –

Return type None

neighbor_allgather(*sendobj*)
Neighbor Gather to All

Parameters **sendobj** (*Any*) –

Return type List[Any]

neighbor_alltoall(*sendobj*)
Neighbor All to All Scatter/Gather

Parameters **sendobj** (*List[Any]*) –

Return type List[Any]

Attributes Documentation

degrees
number of incoming and outgoing neighbors

indegree
number of incoming neighbors

inedges
incoming neighbors

inoutedges
incoming and outgoing neighbors

outdegree
number of outgoing neighbors

outedges
outgoing neighbors

mpi4py.MPI.Win

class mpi4py.MPI.**Win**(*win=None*)

Bases: `object`

Window handle

Parameters **win** (`Optional[Win]`) –

Return type *Win*

static **__new__**(*cls*, *win=None*)

Parameters **win** (`Optional[Win]`) –

Return type *Win*

Methods Summary

`Accumulate`(origin, target_rank[, target, op])	Accumulate data into the target process
`Allocate`(size[, disp_unit, info, comm])	Create an window object for one-sided communication
`Allocate_shared`(size[, disp_unit, info, comm])	Create an window object for one-sided communication
`Attach`(memory)	Attach a local memory region
`Call_errhandler`(errorcode)	Call the error handler installed on a window
`Compare_and_swap`(origin, compare, result, ...)	Perform one-sided atomic compare-and-swap
`Complete`()	Completes an RMA operations begun after an `Win.Start()`
`Create`(memory[, disp_unit, info, comm])	Create an window object for one-sided communication
`Create_dynamic`([info, comm])	Create an window object for one-sided communication
`Create_keyval`([copy_fn, delete_fn, nopython])	Create a new attribute key for windows
`Delete_attr`(keyval)	Delete attribute value associated with a key
`Detach`(memory)	Detach a local memory region
`Fence`([assertion])	Perform an MPI fence synchronization on a window
`Fetch_and_op`(origin, result, target_rank[, ...])	Perform one-sided read-modify-write
`Flush`(rank)	Complete all outstanding RMA operations at the given target
`Flush_all`()	Complete all outstanding RMA operations at all targets
`Flush_local`(rank)	Complete locally all outstanding RMA operations at the given target
`Flush_local_all`()	Complete locally all outstanding RMA opera- tions at all targets
`Free`()	Free a window
`Free_keyval`(keyval)	Free an attribute key for windows
`Get`(origin, target_rank[, target])	Get data from a memory window on a remote process.
`Get_accumulate`(origin, result, target_rank)	Fetch-and-accumulate data into the target process
`Get_attr`(keyval)	Retrieve attribute value by key
`Get_errhandler`()	Get the error handler for a window

continues on next page

Table 49 – continued from previous page

Get_group()	Return a duplicate of the group of the communicator used to create the window
Get_info()	Return the hints for a windows that are currently in use
Get_name()	Get the print name associated with the window
Lock(rank[, lock_type, assertion])	Begin an RMA access epoch at the target process
Lock_all([assertion])	Begin an RMA access epoch at all processes
Post(group[, assertion])	Start an RMA exposure epoch
Put(origin, target_rank[, target])	Put data into a memory window on a remote process.
Raccumulate(origin, target_rank[, target, op])	Fetch-and-accumulate data into the target process
Rget(origin, target_rank[, target])	Get data from a memory window on a remote process.
Rget_accumulate(origin, result, target_rank)	Accumulate data into the target process using remote memory access.
Rput(origin, target_rank[, target])	Put data into a memory window on a remote process.
Set_attr(keyval, attrval)	Store attribute value associated with a key
Set_errhandler(errhandler)	Set the error handler for a window
Set_info(info)	Set new values for the hints associated with a window
Set_name(name)	Set the print name associated with the window
Shared_query(rank)	Query the process-local address for remote memory segments created with Win.Allocate_shared()
Start(group[, assertion])	Start an RMA access epoch for MPI
Sync()	Synchronize public and private copies of the given window
Test()	Test whether an RMA exposure epoch has completed
Unlock(rank)	Complete an RMA access epoch at the target process
Unlock_all()	Complete an RMA access epoch at all processes
Wait()	Complete an RMA exposure epoch begun with Win.Post()
f2py(arg)	
py2f()	
tomemory()	Return window memory buffer

Attributes Summary

attrs	window attributes
flavor	window create flavor
group	window group
info	window info
model	window memory model
name	window name

Methods Documentation

Accumulate(*origin*, *target_rank*, *target=None*, *op=SUM*)
Accumulate data into the target process

Parameters

- **origin** (*BufSpec*) –
- **target_rank** (*int*) –
- **target** (*Optional[TargetSpec]*) –
- **op** (Op) –

Return type None

classmethod Allocate(*size*, *disp_unit=1*, *info=INFO_NULL*, *comm=COMM_SELF*)
Create an window object for one-sided communication

Parameters

- **size** (*int*) –
- **disp_unit** (*int*) –
- **info** (Info) –
- **comm** (Intracomm) –

Return type *Win*

classmethod Allocate_shared(*size*, *disp_unit=1*, *info=INFO_NULL*, *comm=COMM_SELF*)
Create an window object for one-sided communication

Parameters

- **size** (*int*) –
- **disp_unit** (*int*) –
- **info** (Info) –
- **comm** (Intracomm) –

Return type *Win*

Attach(*memory*)
Attach a local memory region

Parameters **memory** (*Buffer*) –

Return type None

Call_errhandler(*errorcode*)
Call the error handler installed on a window

Parameters **errorcode** (*int*) –

Return type None

Compare_and_swap(*origin*, *compare*, *result*, *target_rank*, *target_disp=0*)
Perform one-sided atomic compare-and-swap

Parameters

- **origin** (*BufSpec*) –
- **compare** (*BufSpec*) –

- **result** (*BufSpec*) –
- **target_rank** (*int*) –
- **target_disp** (*int*) –

Return type None

Complete()

Completes an RMA operations begun after an *Win.Start()*

Return type None

classmethod Create(*memory*, *disp_unit=1*, *info=INFO_NULL*, *comm=COMM_SELF*)

Create an window object for one-sided communication

Parameters

- **memory** (*Union[Buffer, Bottom, None]*) –
- **disp_unit** (*int*) –
- **info** (Info) –
- **comm** (Intracomm) –

Return type *Win*

classmethod Create_dynamic(*info=INFO_NULL*, *comm=COMM_SELF*)

Create an window object for one-sided communication

Parameters

- **info** (Info) –
- **comm** (Intracomm) –

Return type *Win*

classmethod Create_keyval(*copy_fn=None*, *delete_fn=None*, *nopython=False*)

Create a new attribute key for windows

Parameters

- **copy_fn** (*Optional[Callable[[Win, int, Any], Any]]*) –
- **delete_fn** (*Optional[Callable[[Win, int, Any], None]]*) –
- **nopython** (*bool*) –

Return type int

Delete_attr(*keyval*)

Delete attribute value associated with a key

Parameters **keyval** (*int*) –

Return type None

Detach(*memory*)

Detach a local memory region

Parameters **memory** (*Buffer*) –

Return type None

Fence(*assertion=0*)

Perform an MPI fence synchronization on a window

Parameters **assertion** (*int*) –

Return type None

Fetch_and_op(*origin*, *result*, *target_rank*, *target_disp=0*, *op=SUM*)

Perform one-sided read-modify-write

Parameters

- **origin** (*BufSpec*) –
- **result** (*BufSpec*) –
- **target_rank** (*int*) –
- **target_disp** (*int*) –
- **op** (Op) –

Return type None

Flush(*rank*)

Complete all outstanding RMA operations at the given target

Parameters **rank** (*int*) –

Return type None

Flush_all()

Complete all outstanding RMA operations at all targets

Return type None

Flush_local(*rank*)

Complete locally all outstanding RMA operations at the given target

Parameters **rank** (*int*) –

Return type None

Flush_local_all()

Complete locally all outstanding RMA opera- tions at all targets

Return type None

Free()

Free a window

Return type None

classmethod **Free_keyval**(*keyval*)

Free an attribute key for windows

Parameters **keyval** (*int*) –

Return type int

Get(*origin*, *target_rank*, *target=None*)

Get data from a memory window on a remote process.

Parameters

- **origin** (*BufSpec*) –
- **target_rank** (*int*) –
- **target** (*Optional[TargetSpec]*) –

Return type None

Get_accumulate(*origin*, *result*, *target_rank*, *target=None*, *op=SUM*)
Fetch-and-accumulate data into the target process

Parameters

- **origin** (*BufSpec*) –
- **result** (*BufSpec*) –
- **target_rank** (*int*) –
- **target** (*Optional[TargetSpec]*) –
- **op** (Op) –

Return type None

Get_attr(*keyval*)
Retrieve attribute value by key

Parameters **keyval** (*int*) –

Return type Optional[Union[int, Any]]

Get_errhandler()
Get the error handler for a window

Return type *Errhandler*

Get_group()
Return a duplicate of the group of the communicator used to create the window

Return type *Group*

Get_info()
Return the hints for a windows that are currently in use

Return type *Info*

Get_name()
Get the print name associated with the window

Return type str

Lock(*rank*, *lock_type=LOCK_EXCLUSIVE*, *assertion=0*)
Begin an RMA access epoch at the target process

Parameters

- **rank** (*int*) –
- **lock_type** (*int*) –
- **assertion** (*int*) –

Return type None

Lock_all(*assertion=0*)
Begin an RMA access epoch at all processes

Parameters **assertion** (*int*) –

Return type None

Post(*group*, *assertion=0*)
Start an RMA exposure epoch

Parameters

- **group** (Group) –
- **assertion** (*int*) –

Return type None

Put(*origin*, *target_rank*, *target=None*)
Put data into a memory window on a remote process.

Parameters

- **origin** (*BufSpec*) –
- **target_rank** (*int*) –
- **target** (*Optional[TargetSpec]*) –

Return type None

Raccumulate(*origin*, *target_rank*, *target=None*, *op=SUM*)
Fetch-and-accumulate data into the target process

Parameters

- **origin** (*BufSpec*) –
- **target_rank** (*int*) –
- **target** (*Optional[TargetSpec]*) –
- **op** (Op) –

Return type *Request*

Rget(*origin*, *target_rank*, *target=None*)
Get data from a memory window on a remote process.

Parameters

- **origin** (*BufSpec*) –
- **target_rank** (*int*) –
- **target** (*Optional[TargetSpec]*) –

Return type *Request*

Rget_accumulate(*origin*, *result*, *target_rank*, *target=None*, *op=SUM*)
Accumulate data into the target process using remote memory access.

Parameters

- **origin** (*BufSpec*) –
- **result** (*BufSpec*) –
- **target_rank** (*int*) –
- **target** (*Optional[TargetSpec]*) –
- **op** (Op) –

Return type *Request*

Rput(*origin*, *target_rank*, *target=None*)
Put data into a memory window on a remote process.

Parameters

- **origin** (*BufSpec*) –

- **target_rank** (*int*) –
- **target** (*Optional[TargetSpec]*) –

Return type *Request*

Set_attr(*keyval*, *attrval*)
Store attribute value associated with a key

Parameters

- **keyval** (*int*) –
- **attrval** (*Any*) –

Return type None

Set_errhandler(*errhandler*)
Set the error handler for a window

Parameters **errhandler** (Errhandler) –

Return type None

Set_info(*info*)
Set new values for the hints associated with a window

Parameters **info** (Info) –

Return type None

Set_name(*name*)
Set the print name associated with the window

Parameters **name** (*str*) –

Return type None

Shared_query(*rank*)
Query the process-local address for remote memory segments created with *Win.Allocate_shared()*

Parameters **rank** (*int*) –

Return type Tuple[*memory*, int]

Start(*group*, *assertion=0*)
Start an RMA access epoch for MPI

Parameters

- **group** (Group) –
- **assertion** (*int*) –

Return type None

Sync()
Synchronize public and private copies of the given window

Return type None

Test()
Test whether an RMA exposure epoch has completed

Return type bool

Unlock(*rank*)
Complete an RMA access epoch at the target process

Parameters **rank** (*int*) –

Return type None

Unlock_all()

Complete an RMA access epoch at all processes

Return type None

Wait()

Complete an RMA exposure epoch begun with *Win.Post()*

Return type Literal[True]

classmethod **f2py**(*arg*)

Parameters **arg** (*int*) –

Return type *Win*

py2f()

Return type int

tomemory()

Return window memory buffer

Return type *memory*

Attributes Documentation

attrs

window attributes

flavor

window create flavor

group

window group

info

window info

model

window memory model

name

window name

mpi4py.MPI.memory

class mpi4py.MPI.**memory**(*buf*)

Bases: object

Memory buffer

Parameters **buf** (*Buffer*) –

Return type *memory*

static `__new__`(*cls*, *buf*)

Parameters **buf** (*Buffer*) –

Return type *memory*

Methods Summary

`allocate`(nbytes[, clear])	Memory allocation
`fromaddress`(address, nbytes[, readonly])	Memory from address and size in bytes
`frombuffer`(obj[, readonly])	Memory from buffer-like object
`release`()	Release the underlying buffer exposed by the memory object
`tobytes`([order])	Return the data in the buffer as a byte string
`toreadonly`()	Return a readonly version of the memory object

Attributes Summary

`address`	Memory address
`format`	A string with the format of each element
`itemsize`	The size in bytes of each element
`nbytes`	Memory size (in bytes)
`obj`	The underlying object of the memory
`readonly`	Boolean indicating whether the memory is read-only

Methods Documentation

static `allocate`(*nbytes*, *clear=False*)

Memory allocation

Parameters

- **nbytes** (*int*) –
- **clear** (*bool*) –

Return type *memory*

static `fromaddress`(*address*, *nbytes*, *readonly=False*)

Memory from address and size in bytes

Parameters

- **address** (*int*) –
- **nbytes** (*int*) –
- **readonly** (*bool*) –

Return type *memory*

static `frombuffer`(*obj*, *readonly=False*)

Memory from buffer-like object

Parameters

- **obj** (*Buffer*) –
- **readonly** (*bool*) –

Return type *memory*

release()

Release the underlying buffer exposed by the memory object

Return type None

tobytes(*order=None*)

Return the data in the buffer as a byte string

Parameters **order** (*Optional[str]*) –

Return type bytes

toreadonly()

Return a readonly version of the memory object

Return type *memory*

Attributes Documentation

address

Memory address

format

A string with the format of each element

itemsize

The size in bytes of each element

nbytes

Memory size (in bytes)

obj

The underlying object of the memory

readonly

Boolean indicating whether the memory is read-only

Exceptions

Exception([ierr])	Exception class

mpi4py.MPI.Exception

exception mpi4py.MPI.**Exception**(*ierr=SUCCESS*)

Bases: RuntimeError

Exception class

Parameters **ierr** (*int*) –

Return type *Exception*

static **__new__**(*cls*, *ierr=SUCCESS*)

Parameters **ierr** (*int*) –

Return type *Exception*

Methods Summary

Get_error_class()	Error class
Get_error_code()	Error code
Get_error_string()	Error string

Attributes Summary

error_class	error class
error_code	error code
error_string	error string

Methods Documentation

Get_error_class()

Error class

Return type int

Get_error_code()

Error code

Return type int

Get_error_string()

Error string

Return type str

Attributes Documentation

error_class

error class

error_code

error code

error_string

error string

Functions

Add_error_class()	Add an *error class* to the known error classes
Add_error_code(errorclass)	Add an *error code* to an *error class*
Add_error_string(errorcode, string)	Associate an *error string* with an *error class* or *error-code*
Aint_add(base, disp)	Return the sum of base address and displacement
Aint_diff(addr1, addr2)	Return the difference between absolute addresses
Alloc_mem(size[, info])	Allocate memory for message passing and RMA
Attach_buffer(buf)	Attach a user-provided buffer for sending in buffered mode
Close_port(port_name)	Close a port
Compute_dims(nnodes, dims)	Return a balanced distribution of processes per coordinate direction
Detach_buffer()	Remove an existing attached buffer
Finalize()	Terminate the MPI execution environment
Free_mem(mem)	Free memory allocated with *Alloc_mem()*
Get_address(location)	Get the address of a location in memory
Get_error_class(errorcode)	Convert an *error code* into an *error class*
Get_error_string(errorcode)	Return the *error string* for a given *error class* or *error code*
Get_library_version()	Obtain the version string of the MPI library
Get_processor_name()	Obtain the name of the calling processor
Get_version()	Obtain the version number of the MPI standard supported by the implementation as a tuple `(version, subversion)`
Init()	Initialize the MPI execution environment
Init_thread([required])	Initialize the MPI execution environment
Is_finalized()	Indicates whether *Finalize* has completed
Is_initialized()	Indicates whether *Init* has been called
Is_thread_main()	Indicate whether this thread called *Init* or *Init_thread*
Lookup_name(service_name[, info])	Lookup a port name given a service name
Open_port([info])	Return an address that can be used to establish connections between groups of MPI processes
Pcontrol(level)	Control profiling
Publish_name(service_name, port_name[, info])	Publish a service name
Query_thread()	Return the level of thread support provided by the MPI library
Register_datarep(datarep, read_fn, write_fn, ...)	Register user-defined data representations
Unpublish_name(service_name, port_name[, info])	Unpublish a service name
Wtick()	Return the resolution of *Wtime*
Wtime()	Return an elapsed time on the calling processor
get_vendor()	Infomation about the underlying MPI implementation

mpi4py.MPI.Add_error_class

`mpi4py.MPI.Add_error_class()`

Add an *error class* to the known error classes

Return type int

mpi4py.MPI.Add_error_code

`mpi4py.MPI.Add_error_code(errorclass)`

Add an *error code* to an *error class*

Parameters **errorclass** (*int*) –

Return type int

mpi4py.MPI.Add_error_string

`mpi4py.MPI.Add_error_string(errorcode, string)`

Associate an *error string* with an *error class* or *errorcode*

Parameters

- **errorcode** (*int*) –
- **string** (*str*) –

Return type None

mpi4py.MPI.Aint_add

`mpi4py.MPI.Aint_add(base, disp)`

Return the sum of base address and displacement

Parameters

- **base** (*int*) –
- **disp** (*int*) –

Return type int

mpi4py.MPI.Aint_diff

`mpi4py.MPI.Aint_diff(addr1, addr2)`

Return the difference between absolute addresses

Parameters

- **addr1** (*int*) –
- **addr2** (*int*) –

Return type int

mpi4py.MPI.Alloc_mem

`mpi4py.MPI.Alloc_mem(size, info=INFO_NULL)`

Allocate memory for message passing and RMA

Parameters

- **size** (*int*) –
- **info** (Info) –

Return type *memory*

mpi4py.MPI.Attach_buffer

`mpi4py.MPI.Attach_buffer(buf)`

Attach a user-provided buffer for sending in buffered mode

Parameters **buf** (*Buffer*) –

Return type None

mpi4py.MPI.Close_port

`mpi4py.MPI.Close_port(port_name)`

Close a port

Parameters **port_name** (*str*) –

Return type None

mpi4py.MPI.Compute_dims

`mpi4py.MPI.Compute_dims(nnodes, dims)`

Return a balanced distribution of processes per coordinate direction

Parameters

- **nnodes** (*int*) –
- **dims** (*Union[int, Sequence[int]]*) –

Return type List[int]

mpi4py.MPI.Detach_buffer

`mpi4py.MPI.Detach_buffer()`

Remove an existing attached buffer

Return type Buffer

mpi4py.MPI.Finalize

mpi4py.MPI.**Finalize**()

Terminate the MPI execution environment

Return type None

mpi4py.MPI.Free_mem

mpi4py.MPI.**Free_mem**(*mem*)

Free memory allocated with *Alloc_mem()*

Parameters **mem** (memory) –

Return type None

mpi4py.MPI.Get_address

mpi4py.MPI.**Get_address**(*location*)

Get the address of a location in memory

Parameters **location** (*Union[Buffer, Bottom]*) –

Return type int

mpi4py.MPI.Get_error_class

mpi4py.MPI.**Get_error_class**(*errorcode*)

Convert an *error code* into an *error class*

Parameters **errorcode** (*int*) –

Return type int

mpi4py.MPI.Get_error_string

mpi4py.MPI.**Get_error_string**(*errorcode*)

Return the *error string* for a given *error class* or *error code*

Parameters **errorcode** (*int*) –

Return type str

mpi4py.MPI.Get_library_version

mpi4py.MPI.**Get_library_version**()

Obtain the version string of the MPI library

Return type str

mpi4py.MPI.Get_processor_name

mpi4py.MPI.**Get_processor_name**()

Obtain the name of the calling processor

Return type str

mpi4py.MPI.Get_version

mpi4py.MPI.**Get_version**()

Obtain the version number of the MPI standard supported by the implementation as a tuple `(version, subversion)`

Return type Tuple[int, int]

mpi4py.MPI.Init

mpi4py.MPI.**Init**()

Initialize the MPI execution environment

Return type None

mpi4py.MPI.Init_thread

mpi4py.MPI.**Init_thread**(*required=THREAD_MULTIPLE*)

Initialize the MPI execution environment

Parameters **required** (*int*) –

Return type int

mpi4py.MPI.Is_finalized

mpi4py.MPI.**Is_finalized**()

Indicates whether *Finalize* has completed

Return type bool

mpi4py.MPI.Is_initialized

mpi4py.MPI.**Is_initialized**()

Indicates whether *Init* has been called

Return type bool

mpi4py.MPI.Is_thread_main

mpi4py.MPI.**Is_thread_main**()

Indicate whether this thread called *Init* or *Init_thread*

Return type bool

mpi4py.MPI.Lookup_name

mpi4py.MPI.**Lookup_name**(*service_name*, *info=INFO_NULL*)

Lookup a port name given a service name

Parameters

- **service_name** (*str*) –
- **info** (Info) –

Return type str

mpi4py.MPI.Open_port

mpi4py.MPI.**Open_port**(*info=INFO_NULL*)

Return an address that can be used to establish connections between groups of MPI processes

Parameters **info** (Info) –

Return type str

mpi4py.MPI.Pcontrol

mpi4py.MPI.**Pcontrol**(*level*)

Control profiling

Parameters **level** (*int*) –

Return type None

mpi4py.MPI.Publish_name

mpi4py.MPI.**Publish_name**(*service_name*, *port_name*, *info=INFO_NULL*)

Publish a service name

Parameters

- **service_name** (*str*) –
- **port_name** (*str*) –
- **info** (Info) –

Return type None

mpi4py.MPI.Query_thread

`mpi4py.MPI.Query_thread()`

Return the level of thread support provided by the MPI library

Return type int

mpi4py.MPI.Register_datarep

`mpi4py.MPI.Register_datarep(datarep, read_fn, write_fn, extent_fn)`

Register user-defined data representations

Parameters

- **datarep** (*str*) –
- **read_fn** (*Callable[[Buffer, Datatype, int, Buffer, int], None]*) –
- **write_fn** (*Callable[[Buffer, Datatype, int, Buffer, int], None]*) –
- **extent_fn** (*Callable[[Datatype], int]*) –

Return type None

mpi4py.MPI.Unpublish_name

`mpi4py.MPI.Unpublish_name(service_name, port_name, info=INFO_NULL)`

Unpublish a service name

Parameters

- **service_name** (*str*) –
- **port_name** (*str*) –
- **info** (Info) –

Return type None

mpi4py.MPI.Wtick

`mpi4py.MPI.Wtick()`

Return the resolution of *Wtime*

Return type float

mpi4py.MPI.Wtime

`mpi4py.MPI.Wtime()`

Return an elapsed time on the calling processor

Return type float

mpi4py.MPI.get_vendor

`mpi4py.MPI.get_vendor()`

Infomation about the underlying MPI implementation

Returns

- a string with the name of the MPI implementation
- an integer 3-tuple version `(major, minor, micro)`

Return type Tuple[str, Tuple[int, int, int]]

Attributes

UNDEFINED	`int` `UNDEFINED`
ANY_SOURCE	`int` `ANY_SOURCE`
ANY_TAG	`int` `ANY_TAG`
PROC_NULL	`int` `PROC_NULL`
ROOT	`int` `ROOT`
BOTTOM	`Bottom` `BOTTOM`
IN_PLACE	`InPlace` `IN_PLACE`
KEYVAL_INVALID	`int` `KEYVAL_INVALID`
TAG_UB	`int` `TAG_UB`
HOST	`int` `HOST`
IO	`int` `IO`
WTIME_IS_GLOBAL	`int` `WTIME_IS_GLOBAL`
UNIVERSE_SIZE	`int` `UNIVERSE_SIZE`
APPNUM	`int` `APPNUM`
LASTUSEDCODE	`int` `LASTUSEDCODE`
WIN_BASE	`int` `WIN_BASE`
WIN_SIZE	`int` `WIN_SIZE`
WIN_DISP_UNIT	`int` `WIN_DISP_UNIT`
WIN_CREATE_FLAVOR	`int` `WIN_CREATE_FLAVOR`
WIN_FLAVOR	`int` `WIN_FLAVOR`
WIN_MODEL	`int` `WIN_MODEL`
SUCCESS	`int` `SUCCESS`
ERR_LASTCODE	`int` `ERR_LASTCODE`
ERR_COMM	`int` `ERR_COMM`
ERR_GROUP	`int` `ERR_GROUP`
ERR_TYPE	`int` `ERR_TYPE`
ERR_REQUEST	`int` `ERR_REQUEST`
ERR_OP	`int` `ERR_OP`
ERR_BUFFER	`int` `ERR_BUFFER`
ERR_COUNT	`int` `ERR_COUNT`
ERR_TAG	`int` `ERR_TAG`
ERR_RANK	`int` `ERR_RANK`
ERR_ROOT	`int` `ERR_ROOT`
ERR_TRUNCATE	`int` `ERR_TRUNCATE`
ERR_IN_STATUS	`int` `ERR_IN_STATUS`
ERR_PENDING	`int` `ERR_PENDING`
ERR_TOPOLOGY	`int` `ERR_TOPOLOGY`

continues on next page

Table 57 – continued from previous page

ERR_DIMS	int ERR_DIMS
ERR_ARG	int ERR_ARG
ERR_OTHER	int ERR_OTHER
ERR_UNKNOWN	int ERR_UNKNOWN
ERR_INTERN	int ERR_INTERN
ERR_INFO	int ERR_INFO
ERR_FILE	int ERR_FILE
ERR_WIN	int ERR_WIN
ERR_KEYVAL	int ERR_KEYVAL
ERR_INFO_KEY	int ERR_INFO_KEY
ERR_INFO_VALUE	int ERR_INFO_VALUE
ERR_INFO_NOKEY	int ERR_INFO_NOKEY
ERR_ACCESS	int ERR_ACCESS
ERR_AMODE	int ERR_AMODE
ERR_BAD_FILE	int ERR_BAD_FILE
ERR_FILE_EXISTS	int ERR_FILE_EXISTS
ERR_FILE_IN_USE	int ERR_FILE_IN_USE
ERR_NO_SPACE	int ERR_NO_SPACE
ERR_NO_SUCH_FILE	int ERR_NO_SUCH_FILE
ERR_IO	int ERR_IO
ERR_READ_ONLY	int ERR_READ_ONLY
ERR_CONVERSION	int ERR_CONVERSION
ERR_DUP_DATAREP	int ERR_DUP_DATAREP
ERR_UNSUPPORTED_DATAREP	int ERR_UNSUPPORTED_DATAREP
ERR_UNSUPPORTED_OPERATION	int ERR_UNSUPPORTED_OPERATION
ERR_NAME	int ERR_NAME
ERR_NO_MEM	int ERR_NO_MEM
ERR_NOT_SAME	int ERR_NOT_SAME
ERR_PORT	int ERR_PORT
ERR_QUOTA	int ERR_QUOTA
ERR_SERVICE	int ERR_SERVICE
ERR_SPAWN	int ERR_SPAWN
ERR_BASE	int ERR_BASE
ERR_SIZE	int ERR_SIZE
ERR_DISP	int ERR_DISP
ERR_ASSERT	int ERR_ASSERT
ERR_LOCKTYPE	int ERR_LOCKTYPE
ERR_RMA_CONFLICT	int ERR_RMA_CONFLICT
ERR_RMA_SYNC	int ERR_RMA_SYNC
ERR_RMA_RANGE	int ERR_RMA_RANGE
ERR_RMA_ATTACH	int ERR_RMA_ATTACH
ERR_RMA_SHARED	int ERR_RMA_SHARED
ERR_RMA_FLAVOR	int ERR_RMA_FLAVOR
ORDER_C	int ORDER_C
ORDER_FORTRAN	int ORDER_FORTRAN
ORDER_F	int ORDER_F
TYPECLASS_INTEGER	int TYPECLASS_INTEGER
TYPECLASS_REAL	int TYPECLASS_REAL
TYPECLASS_COMPLEX	int TYPECLASS_COMPLEX
DISTRIBUTE_NONE	int DISTRIBUTE_NONE

continues on next page

Table 57 – continued from previous page

DISTRIBUTE_BLOCK	int DISTRIBUTE_BLOCK
DISTRIBUTE_CYCLIC	int DISTRIBUTE_CYCLIC
DISTRIBUTE_DFLT_DARG	int DISTRIBUTE_DFLT_DARG
COMBINER_NAMED	int COMBINER_NAMED
COMBINER_DUP	int COMBINER_DUP
COMBINER_CONTIGUOUS	int COMBINER_CONTIGUOUS
COMBINER_VECTOR	int COMBINER_VECTOR
COMBINER_HVECTOR	int COMBINER_HVECTOR
COMBINER_INDEXED	int COMBINER_INDEXED
COMBINER_HINDEXED	int COMBINER_HINDEXED
COMBINER_INDEXED_BLOCK	int COMBINER_INDEXED_BLOCK
COMBINER_HINDEXED_BLOCK	int COMBINER_HINDEXED_BLOCK
COMBINER_STRUCT	int COMBINER_STRUCT
COMBINER_SUBARRAY	int COMBINER_SUBARRAY
COMBINER_DARRAY	int COMBINER_DARRAY
COMBINER_RESIZED	int COMBINER_RESIZED
COMBINER_F90_REAL	int COMBINER_F90_REAL
COMBINER_F90_COMPLEX	int COMBINER_F90_COMPLEX
COMBINER_F90_INTEGER	int COMBINER_F90_INTEGER
IDENT	int IDENT
CONGRUENT	int CONGRUENT
SIMILAR	int SIMILAR
UNEQUAL	int UNEQUAL
CART	int CART
GRAPH	int GRAPH
DIST_GRAPH	int DIST_GRAPH
UNWEIGHTED	int UNWEIGHTED
WEIGHTS_EMPTY	int WEIGHTS_EMPTY
COMM_TYPE_SHARED	int COMM_TYPE_SHARED
BSEND_OVERHEAD	int BSEND_OVERHEAD
WIN_FLAVOR_CREATE	int WIN_FLAVOR_CREATE
WIN_FLAVOR_ALLOCATE	int WIN_FLAVOR_ALLOCATE
WIN_FLAVOR_DYNAMIC	int WIN_FLAVOR_DYNAMIC
WIN_FLAVOR_SHARED	int WIN_FLAVOR_SHARED
WIN_SEPARATE	int WIN_SEPARATE
WIN_UNIFIED	int WIN_UNIFIED
MODE_NOCHECK	int MODE_NOCHECK
MODE_NOSTORE	int MODE_NOSTORE
MODE_NOPUT	int MODE_NOPUT
MODE_NOPRECEDE	int MODE_NOPRECEDE
MODE_NOSUCCEED	int MODE_NOSUCCEED
LOCK_EXCLUSIVE	int LOCK_EXCLUSIVE
LOCK_SHARED	int LOCK_SHARED
MODE_RDONLY	int MODE_RDONLY
MODE_WRONLY	int MODE_WRONLY
MODE_RDWR	int MODE_RDWR
MODE_CREATE	int MODE_CREATE
MODE_EXCL	int MODE_EXCL
MODE_DELETE_ON_CLOSE	int MODE_DELETE_ON_CLOSE
MODE_UNIQUE_OPEN	int MODE_UNIQUE_OPEN

continues on next page

Table 57 – continued from previous page

MODE_SEQUENTIAL	int MODE_SEQUENTIAL
MODE_APPEND	int MODE_APPEND
SEEK_SET	int SEEK_SET
SEEK_CUR	int SEEK_CUR
SEEK_END	int SEEK_END
DISPLACEMENT_CURRENT	int DISPLACEMENT_CURRENT
DISP_CUR	int DISP_CUR
THREAD_SINGLE	int THREAD_SINGLE
THREAD_FUNNELED	int THREAD_FUNNELED
THREAD_SERIALIZED	int THREAD_SERIALIZED
THREAD_MULTIPLE	int THREAD_MULTIPLE
VERSION	int VERSION
SUBVERSION	int SUBVERSION
MAX_PROCESSOR_NAME	int MAX_PROCESSOR_NAME
MAX_ERROR_STRING	int MAX_ERROR_STRING
MAX_PORT_NAME	int MAX_PORT_NAME
MAX_INFO_KEY	int MAX_INFO_KEY
MAX_INFO_VAL	int MAX_INFO_VAL
MAX_OBJECT_NAME	int MAX_OBJECT_NAME
MAX_DATAREP_STRING	int MAX_DATAREP_STRING
MAX_LIBRARY_VERSION_STRING	int MAX_LIBRARY_VERSION_STRING
DATATYPE_NULL	*Datatype* DATATYPE_NULL
UB	*Datatype* UB
LB	*Datatype* LB
PACKED	*Datatype* PACKED
BYTE	*Datatype* BYTE
AINT	*Datatype* AINT
OFFSET	*Datatype* OFFSET
COUNT	*Datatype* COUNT
CHAR	*Datatype* CHAR
WCHAR	*Datatype* WCHAR
SIGNED_CHAR	*Datatype* SIGNED_CHAR
SHORT	*Datatype* SHORT
INT	*Datatype* INT
LONG	*Datatype* LONG
LONG_LONG	*Datatype* LONG_LONG
UNSIGNED_CHAR	*Datatype* UNSIGNED_CHAR
UNSIGNED_SHORT	*Datatype* UNSIGNED_SHORT
UNSIGNED	*Datatype* UNSIGNED
UNSIGNED_LONG	*Datatype* UNSIGNED_LONG
UNSIGNED_LONG_LONG	*Datatype* UNSIGNED_LONG_LONG
FLOAT	*Datatype* FLOAT
DOUBLE	*Datatype* DOUBLE
LONG_DOUBLE	*Datatype* LONG_DOUBLE
C_BOOL	*Datatype* C_BOOL
INT8_T	*Datatype* INT8_T
INT16_T	*Datatype* INT16_T
INT32_T	*Datatype* INT32_T
INT64_T	*Datatype* INT64_T
UINT8_T	*Datatype* UINT8_T

continues on next page

Table 57 – continued from previous page

UINT16_T	*Datatype* UINT16_T
UINT32_T	*Datatype* UINT32_T
UINT64_T	*Datatype* UINT64_T
C_COMPLEX	*Datatype* C_COMPLEX
C_FLOAT_COMPLEX	*Datatype* C_FLOAT_COMPLEX
C_DOUBLE_COMPLEX	*Datatype* C_DOUBLE_COMPLEX
C_LONG_DOUBLE_COMPLEX	*Datatype* C_LONG_DOUBLE_COMPLEX
CXX_BOOL	*Datatype* CXX_BOOL
CXX_FLOAT_COMPLEX	*Datatype* CXX_FLOAT_COMPLEX
CXX_DOUBLE_COMPLEX	*Datatype* CXX_DOUBLE_COMPLEX
CXX_LONG_DOUBLE_COMPLEX	*Datatype* CXX_LONG_DOUBLE_COMPLEX
SHORT_INT	*Datatype* SHORT_INT
INT_INT	*Datatype* INT_INT
TWOINT	*Datatype* TWOINT
LONG_INT	*Datatype* LONG_INT
FLOAT_INT	*Datatype* FLOAT_INT
DOUBLE_INT	*Datatype* DOUBLE_INT
LONG_DOUBLE_INT	*Datatype* LONG_DOUBLE_INT
CHARACTER	*Datatype* CHARACTER
LOGICAL	*Datatype* LOGICAL
INTEGER	*Datatype* INTEGER
REAL	*Datatype* REAL
DOUBLE_PRECISION	*Datatype* DOUBLE_PRECISION
COMPLEX	*Datatype* COMPLEX
DOUBLE_COMPLEX	*Datatype* DOUBLE_COMPLEX
LOGICAL1	*Datatype* LOGICAL1
LOGICAL2	*Datatype* LOGICAL2
LOGICAL4	*Datatype* LOGICAL4
LOGICAL8	*Datatype* LOGICAL8
INTEGER1	*Datatype* INTEGER1
INTEGER2	*Datatype* INTEGER2
INTEGER4	*Datatype* INTEGER4
INTEGER8	*Datatype* INTEGER8
INTEGER16	*Datatype* INTEGER16
REAL2	*Datatype* REAL2
REAL4	*Datatype* REAL4
REAL8	*Datatype* REAL8
REAL16	*Datatype* REAL16
COMPLEX4	*Datatype* COMPLEX4
COMPLEX8	*Datatype* COMPLEX8
COMPLEX16	*Datatype* COMPLEX16
COMPLEX32	*Datatype* COMPLEX32
UNSIGNED_INT	*Datatype* UNSIGNED_INT
SIGNED_SHORT	*Datatype* SIGNED_SHORT
SIGNED_INT	*Datatype* SIGNED_INT
SIGNED_LONG	*Datatype* SIGNED_LONG
SIGNED_LONG_LONG	*Datatype* SIGNED_LONG_LONG
BOOL	*Datatype* BOOL
SINT8_T	*Datatype* SINT8_T
SINT16_T	*Datatype* SINT16_T

continues on next page

Table 57 – continued from previous page

SINT32_T	*Datatype* SINT32_T
SINT64_T	*Datatype* SINT64_T
F_BOOL	*Datatype* F_BOOL
F_INT	*Datatype* F_INT
F_FLOAT	*Datatype* F_FLOAT
F_DOUBLE	*Datatype* F_DOUBLE
F_COMPLEX	*Datatype* F_COMPLEX
F_FLOAT_COMPLEX	*Datatype* F_FLOAT_COMPLEX
F_DOUBLE_COMPLEX	*Datatype* F_DOUBLE_COMPLEX
REQUEST_NULL	*Request* REQUEST_NULL
MESSAGE_NULL	*Message* MESSAGE_NULL
MESSAGE_NO_PROC	*Message* MESSAGE_NO_PROC
OP_NULL	*Op* OP_NULL
MAX	*Op* MAX
MIN	*Op* MIN
SUM	*Op* SUM
PROD	*Op* PROD
LAND	*Op* LAND
BAND	*Op* BAND
LOR	*Op* LOR
BOR	*Op* BOR
LXOR	*Op* LXOR
BXOR	*Op* BXOR
MAXLOC	*Op* MAXLOC
MINLOC	*Op* MINLOC
REPLACE	*Op* REPLACE
NO_OP	*Op* NO_OP
GROUP_NULL	*Group* GROUP_NULL
GROUP_EMPTY	*Group* GROUP_EMPTY
INFO_NULL	*Info* INFO_NULL
INFO_ENV	*Info* INFO_ENV
ERRHANDLER_NULL	*Errhandler* ERRHANDLER_NULL
ERRORS_RETURN	*Errhandler* ERRORS_RETURN
ERRORS_ARE_FATAL	*Errhandler* ERRORS_ARE_FATAL
COMM_NULL	*Comm* COMM_NULL
COMM_SELF	*Intracomm* COMM_SELF
COMM_WORLD	*Intracomm* COMM_WORLD
WIN_NULL	*Win* WIN_NULL
FILE_NULL	*File* FILE_NULL
pickle	*Pickle* pickle

mpi4py.MPI.UNDEFINED

`mpi4py.MPI.UNDEFINED: int = UNDEFINED`

int UNDEFINED

mpi4py.MPI.ANY_SOURCE

`mpi4py.MPI.ANY_SOURCE: int = ANY_SOURCE`

int ANY_SOURCE

mpi4py.MPI.ANY_TAG

`mpi4py.MPI.ANY_TAG: int = ANY_TAG`

int ANY_TAG

mpi4py.MPI.PROC_NULL

`mpi4py.MPI.PROC_NULL: int = PROC_NULL`

int PROC_NULL

mpi4py.MPI.ROOT

`mpi4py.MPI.ROOT: int = ROOT`

int ROOT

mpi4py.MPI.BOTTOM

`mpi4py.MPI.BOTTOM: Bottom = BOTTOM`

Bottom BOTTOM

mpi4py.MPI.IN_PLACE

`mpi4py.MPI.IN_PLACE: InPlace = IN_PLACE`

InPlace IN_PLACE

mpi4py.MPI.KEYVAL_INVALID

`mpi4py.MPI.KEYVAL_INVALID: int = KEYVAL_INVALID`

int KEYVAL_INVALID

mpi4py.MPI.TAG_UB

mpi4py.MPI.**TAG_UB**: int = **TAG_UB**
    int TAG_UB

mpi4py.MPI.HOST

mpi4py.MPI.**HOST**: int = **HOST**
    int HOST

mpi4py.MPI.IO

mpi4py.MPI.**IO**: int = **IO**
    int IO

mpi4py.MPI.WTIME_IS_GLOBAL

mpi4py.MPI.**WTIME_IS_GLOBAL**: int = **WTIME_IS_GLOBAL**
    int WTIME_IS_GLOBAL

mpi4py.MPI.UNIVERSE_SIZE

mpi4py.MPI.**UNIVERSE_SIZE**: int = **UNIVERSE_SIZE**
    int UNIVERSE_SIZE

mpi4py.MPI.APPNUM

mpi4py.MPI.**APPNUM**: int = **APPNUM**
    int APPNUM

mpi4py.MPI.LASTUSEDCODE

mpi4py.MPI.**LASTUSEDCODE**: int = **LASTUSEDCODE**
    int LASTUSEDCODE

mpi4py.MPI.WIN_BASE

mpi4py.MPI.**WIN_BASE**: int = **WIN_BASE**
    int WIN_BASE

mpi4py.MPI.WIN_SIZE

mpi4py.MPI.**WIN_SIZE: int = WIN_SIZE**

int WIN_SIZE

mpi4py.MPI.WIN_DISP_UNIT

mpi4py.MPI.**WIN_DISP_UNIT: int = WIN_DISP_UNIT**

int WIN_DISP_UNIT

mpi4py.MPI.WIN_CREATE_FLAVOR

mpi4py.MPI.**WIN_CREATE_FLAVOR: int = WIN_CREATE_FLAVOR**

int WIN_CREATE_FLAVOR

mpi4py.MPI.WIN_FLAVOR

mpi4py.MPI.**WIN_FLAVOR: int = WIN_FLAVOR**

int WIN_FLAVOR

mpi4py.MPI.WIN_MODEL

mpi4py.MPI.**WIN_MODEL: int = WIN_MODEL**

int WIN_MODEL

mpi4py.MPI.SUCCESS

mpi4py.MPI.**SUCCESS: int = SUCCESS**

int SUCCESS

mpi4py.MPI.ERR_LASTCODE

mpi4py.MPI.**ERR_LASTCODE: int = ERR_LASTCODE**

int ERR_LASTCODE

mpi4py.MPI.ERR_COMM

mpi4py.MPI.**ERR_COMM: int = ERR_COMM**

int ERR_COMM

mpi4py.MPI.ERR_GROUP

mpi4py.MPI.**ERR_GROUP**: int = ERR_GROUP

int ERR_GROUP

mpi4py.MPI.ERR_TYPE

mpi4py.MPI.**ERR_TYPE**: int = ERR_TYPE

int ERR_TYPE

mpi4py.MPI.ERR_REQUEST

mpi4py.MPI.**ERR_REQUEST**: int = ERR_REQUEST

int ERR_REQUEST

mpi4py.MPI.ERR_OP

mpi4py.MPI.**ERR_OP**: int = ERR_OP

int ERR_OP

mpi4py.MPI.ERR_BUFFER

mpi4py.MPI.**ERR_BUFFER**: int = ERR_BUFFER

int ERR_BUFFER

mpi4py.MPI.ERR_COUNT

mpi4py.MPI.**ERR_COUNT**: int = ERR_COUNT

int ERR_COUNT

mpi4py.MPI.ERR_TAG

mpi4py.MPI.**ERR_TAG**: int = ERR_TAG

int ERR_TAG

mpi4py.MPI.ERR_RANK

mpi4py.MPI.**ERR_RANK**: int = ERR_RANK

int ERR_RANK

mpi4py.MPI.ERR_ROOT

mpi4py.MPI.**ERR_ROOT: int = ERR_ROOT**
    int ERR_ROOT

mpi4py.MPI.ERR_TRUNCATE

mpi4py.MPI.**ERR_TRUNCATE: int = ERR_TRUNCATE**
    int ERR_TRUNCATE

mpi4py.MPI.ERR_IN_STATUS

mpi4py.MPI.**ERR_IN_STATUS: int = ERR_IN_STATUS**
    int ERR_IN_STATUS

mpi4py.MPI.ERR_PENDING

mpi4py.MPI.**ERR_PENDING: int = ERR_PENDING**
    int ERR_PENDING

mpi4py.MPI.ERR_TOPOLOGY

mpi4py.MPI.**ERR_TOPOLOGY: int = ERR_TOPOLOGY**
    int ERR_TOPOLOGY

mpi4py.MPI.ERR_DIMS

mpi4py.MPI.**ERR_DIMS: int = ERR_DIMS**
    int ERR_DIMS

mpi4py.MPI.ERR_ARG

mpi4py.MPI.**ERR_ARG: int = ERR_ARG**
    int ERR_ARG

mpi4py.MPI.ERR_OTHER

mpi4py.MPI.**ERR_OTHER: int = ERR_OTHER**
    int ERR_OTHER

mpi4py.MPI.ERR_UNKNOWN

mpi4py.MPI.**ERR_UNKNOWN: int = ERR_UNKNOWN**

int ERR_UNKNOWN

mpi4py.MPI.ERR_INTERN

mpi4py.MPI.**ERR_INTERN: int = ERR_INTERN**

int ERR_INTERN

mpi4py.MPI.ERR_INFO

mpi4py.MPI.**ERR_INFO: int = ERR_INFO**

int ERR_INFO

mpi4py.MPI.ERR_FILE

mpi4py.MPI.**ERR_FILE: int = ERR_FILE**

int ERR_FILE

mpi4py.MPI.ERR_WIN

mpi4py.MPI.**ERR_WIN: int = ERR_WIN**

int ERR_WIN

mpi4py.MPI.ERR_KEYVAL

mpi4py.MPI.**ERR_KEYVAL: int = ERR_KEYVAL**

int ERR_KEYVAL

mpi4py.MPI.ERR_INFO_KEY

mpi4py.MPI.**ERR_INFO_KEY: int = ERR_INFO_KEY**

int ERR_INFO_KEY

mpi4py.MPI.ERR_INFO_VALUE

mpi4py.MPI.**ERR_INFO_VALUE: int = ERR_INFO_VALUE**

int ERR_INFO_VALUE

mpi4py.MPI.ERR_INFO_NOKEY

mpi4py.MPI.**ERR_INFO_NOKEY**: int = **ERR_INFO_NOKEY**

int ERR_INFO_NOKEY

mpi4py.MPI.ERR_ACCESS

mpi4py.MPI.**ERR_ACCESS**: int = **ERR_ACCESS**

int ERR_ACCESS

mpi4py.MPI.ERR_AMODE

mpi4py.MPI.**ERR_AMODE**: int = **ERR_AMODE**

int ERR_AMODE

mpi4py.MPI.ERR_BAD_FILE

mpi4py.MPI.**ERR_BAD_FILE**: int = **ERR_BAD_FILE**

int ERR_BAD_FILE

mpi4py.MPI.ERR_FILE_EXISTS

mpi4py.MPI.**ERR_FILE_EXISTS**: int = **ERR_FILE_EXISTS**

int ERR_FILE_EXISTS

mpi4py.MPI.ERR_FILE_IN_USE

mpi4py.MPI.**ERR_FILE_IN_USE**: int = **ERR_FILE_IN_USE**

int ERR_FILE_IN_USE

mpi4py.MPI.ERR_NO_SPACE

mpi4py.MPI.**ERR_NO_SPACE**: int = **ERR_NO_SPACE**

int ERR_NO_SPACE

mpi4py.MPI.ERR_NO_SUCH_FILE

mpi4py.MPI.**ERR_NO_SUCH_FILE**: int = **ERR_NO_SUCH_FILE**

int ERR_NO_SUCH_FILE

mpi4py.MPI.ERR_IO

mpi4py.MPI.**ERR_IO**: int = **ERR_IO**
    int ERR_IO

mpi4py.MPI.ERR_READ_ONLY

mpi4py.MPI.**ERR_READ_ONLY**: int = **ERR_READ_ONLY**
    int ERR_READ_ONLY

mpi4py.MPI.ERR_CONVERSION

mpi4py.MPI.**ERR_CONVERSION**: int = **ERR_CONVERSION**
    int ERR_CONVERSION

mpi4py.MPI.ERR_DUP_DATAREP

mpi4py.MPI.**ERR_DUP_DATAREP**: int = **ERR_DUP_DATAREP**
    int ERR_DUP_DATAREP

mpi4py.MPI.ERR_UNSUPPORTED_DATAREP

mpi4py.MPI.**ERR_UNSUPPORTED_DATAREP**: int = **ERR_UNSUPPORTED_DATAREP**
    int ERR_UNSUPPORTED_DATAREP

mpi4py.MPI.ERR_UNSUPPORTED_OPERATION

mpi4py.MPI.**ERR_UNSUPPORTED_OPERATION**: int = **ERR_UNSUPPORTED_OPERATION**
    int ERR_UNSUPPORTED_OPERATION

mpi4py.MPI.ERR_NAME

mpi4py.MPI.**ERR_NAME**: int = **ERR_NAME**
    int ERR_NAME

mpi4py.MPI.ERR_NO_MEM

mpi4py.MPI.**ERR_NO_MEM**: int = **ERR_NO_MEM**
    int ERR_NO_MEM

mpi4py.MPI.ERR_NOT_SAME

mpi4py.MPI.**ERR_NOT_SAME: int = ERR_NOT_SAME**

int ERR_NOT_SAME

mpi4py.MPI.ERR_PORT

mpi4py.MPI.**ERR_PORT: int = ERR_PORT**

int ERR_PORT

mpi4py.MPI.ERR_QUOTA

mpi4py.MPI.**ERR_QUOTA: int = ERR_QUOTA**

int ERR_QUOTA

mpi4py.MPI.ERR_SERVICE

mpi4py.MPI.**ERR_SERVICE: int = ERR_SERVICE**

int ERR_SERVICE

mpi4py.MPI.ERR_SPAWN

mpi4py.MPI.**ERR_SPAWN: int = ERR_SPAWN**

int ERR_SPAWN

mpi4py.MPI.ERR_BASE

mpi4py.MPI.**ERR_BASE: int = ERR_BASE**

int ERR_BASE

mpi4py.MPI.ERR_SIZE

mpi4py.MPI.**ERR_SIZE: int = ERR_SIZE**

int ERR_SIZE

mpi4py.MPI.ERR_DISP

mpi4py.MPI.**ERR_DISP: int = ERR_DISP**

int ERR_DISP

mpi4py.MPI.ERR_ASSERT

mpi4py.MPI.**ERR_ASSERT**: int = ERR_ASSERT
    int ERR_ASSERT

mpi4py.MPI.ERR_LOCKTYPE

mpi4py.MPI.**ERR_LOCKTYPE**: int = ERR_LOCKTYPE
    int ERR_LOCKTYPE

mpi4py.MPI.ERR_RMA_CONFLICT

mpi4py.MPI.**ERR_RMA_CONFLICT**: int = ERR_RMA_CONFLICT
    int ERR_RMA_CONFLICT

mpi4py.MPI.ERR_RMA_SYNC

mpi4py.MPI.**ERR_RMA_SYNC**: int = ERR_RMA_SYNC
    int ERR_RMA_SYNC

mpi4py.MPI.ERR_RMA_RANGE

mpi4py.MPI.**ERR_RMA_RANGE**: int = ERR_RMA_RANGE
    int ERR_RMA_RANGE

mpi4py.MPI.ERR_RMA_ATTACH

mpi4py.MPI.**ERR_RMA_ATTACH**: int = ERR_RMA_ATTACH
    int ERR_RMA_ATTACH

mpi4py.MPI.ERR_RMA_SHARED

mpi4py.MPI.**ERR_RMA_SHARED**: int = ERR_RMA_SHARED
    int ERR_RMA_SHARED

mpi4py.MPI.ERR_RMA_FLAVOR

mpi4py.MPI.**ERR_RMA_FLAVOR**: int = ERR_RMA_FLAVOR
    int ERR_RMA_FLAVOR

mpi4py.MPI.ORDER_C

mpi4py.MPI.**ORDER_C**: int = ORDER_C
    int ORDER_C

mpi4py.MPI.ORDER_FORTRAN

mpi4py.MPI.**ORDER_FORTRAN**: int = ORDER_FORTRAN
    int ORDER_FORTRAN

mpi4py.MPI.ORDER_F

mpi4py.MPI.**ORDER_F**: int = ORDER_F
    int ORDER_F

mpi4py.MPI.TYPECLASS_INTEGER

mpi4py.MPI.**TYPECLASS_INTEGER**: int = TYPECLASS_INTEGER
    int TYPECLASS_INTEGER

mpi4py.MPI.TYPECLASS_REAL

mpi4py.MPI.**TYPECLASS_REAL**: int = TYPECLASS_REAL
    int TYPECLASS_REAL

mpi4py.MPI.TYPECLASS_COMPLEX

mpi4py.MPI.**TYPECLASS_COMPLEX**: int = TYPECLASS_COMPLEX
    int TYPECLASS_COMPLEX

mpi4py.MPI.DISTRIBUTE_NONE

mpi4py.MPI.**DISTRIBUTE_NONE**: int = DISTRIBUTE_NONE
    int DISTRIBUTE_NONE

mpi4py.MPI.DISTRIBUTE_BLOCK

mpi4py.MPI.**DISTRIBUTE_BLOCK**: int = DISTRIBUTE_BLOCK
    int DISTRIBUTE_BLOCK

mpi4py.MPI.DISTRIBUTE_CYCLIC

mpi4py.MPI.**DISTRIBUTE_CYCLIC: int = DISTRIBUTE_CYCLIC**

int DISTRIBUTE_CYCLIC

mpi4py.MPI.DISTRIBUTE_DFLT_DARG

mpi4py.MPI.**DISTRIBUTE_DFLT_DARG: int = DISTRIBUTE_DFLT_DARG**

int DISTRIBUTE_DFLT_DARG

mpi4py.MPI.COMBINER_NAMED

mpi4py.MPI.**COMBINER_NAMED: int = COMBINER_NAMED**

int COMBINER_NAMED

mpi4py.MPI.COMBINER_DUP

mpi4py.MPI.**COMBINER_DUP: int = COMBINER_DUP**

int COMBINER_DUP

mpi4py.MPI.COMBINER_CONTIGUOUS

mpi4py.MPI.**COMBINER_CONTIGUOUS: int = COMBINER_CONTIGUOUS**

int COMBINER_CONTIGUOUS

mpi4py.MPI.COMBINER_VECTOR

mpi4py.MPI.**COMBINER_VECTOR: int = COMBINER_VECTOR**

int COMBINER_VECTOR

mpi4py.MPI.COMBINER_HVECTOR

mpi4py.MPI.**COMBINER_HVECTOR: int = COMBINER_HVECTOR**

int COMBINER_HVECTOR

mpi4py.MPI.COMBINER_INDEXED

mpi4py.MPI.**COMBINER_INDEXED: int = COMBINER_INDEXED**

int COMBINER_INDEXED

mpi4py.MPI.COMBINER_HINDEXED

mpi4py.MPI.**COMBINER_HINDEXED: int = COMBINER_HINDEXED**
int COMBINER_HINDEXED

mpi4py.MPI.COMBINER_INDEXED_BLOCK

mpi4py.MPI.**COMBINER_INDEXED_BLOCK: int = COMBINER_INDEXED_BLOCK**
int COMBINER_INDEXED_BLOCK

mpi4py.MPI.COMBINER_HINDEXED_BLOCK

mpi4py.MPI.**COMBINER_HINDEXED_BLOCK: int = COMBINER_HINDEXED_BLOCK**
int COMBINER_HINDEXED_BLOCK

mpi4py.MPI.COMBINER_STRUCT

mpi4py.MPI.**COMBINER_STRUCT: int = COMBINER_STRUCT**
int COMBINER_STRUCT

mpi4py.MPI.COMBINER_SUBARRAY

mpi4py.MPI.**COMBINER_SUBARRAY: int = COMBINER_SUBARRAY**
int COMBINER_SUBARRAY

mpi4py.MPI.COMBINER_DARRAY

mpi4py.MPI.**COMBINER_DARRAY: int = COMBINER_DARRAY**
int COMBINER_DARRAY

mpi4py.MPI.COMBINER_RESIZED

mpi4py.MPI.**COMBINER_RESIZED: int = COMBINER_RESIZED**
int COMBINER_RESIZED

mpi4py.MPI.COMBINER_F90_REAL

mpi4py.MPI.**COMBINER_F90_REAL: int = COMBINER_F90_REAL**
int COMBINER_F90_REAL

mpi4py.MPI.COMBINER_F90_COMPLEX

mpi4py.MPI.**COMBINER_F90_COMPLEX: int = COMBINER_F90_COMPLEX**
    int COMBINER_F90_COMPLEX

mpi4py.MPI.COMBINER_F90_INTEGER

mpi4py.MPI.**COMBINER_F90_INTEGER: int = COMBINER_F90_INTEGER**
    int COMBINER_F90_INTEGER

mpi4py.MPI.IDENT

mpi4py.MPI.**IDENT: int = IDENT**
    int IDENT

mpi4py.MPI.CONGRUENT

mpi4py.MPI.**CONGRUENT: int = CONGRUENT**
    int CONGRUENT

mpi4py.MPI.SIMILAR

mpi4py.MPI.**SIMILAR: int = SIMILAR**
    int SIMILAR

mpi4py.MPI.UNEQUAL

mpi4py.MPI.**UNEQUAL: int = UNEQUAL**
    int UNEQUAL

mpi4py.MPI.CART

mpi4py.MPI.**CART: int = CART**
    int CART

mpi4py.MPI.GRAPH

mpi4py.MPI.**GRAPH: int = GRAPH**
    int GRAPH

mpi4py.MPI.DIST_GRAPH

mpi4py.MPI.**DIST_GRAPH**: int = **DIST_GRAPH**
    int DIST_GRAPH

mpi4py.MPI.UNWEIGHTED

mpi4py.MPI.**UNWEIGHTED**: int = **UNWEIGHTED**
    int UNWEIGHTED

mpi4py.MPI.WEIGHTS_EMPTY

mpi4py.MPI.**WEIGHTS_EMPTY**: int = **WEIGHTS_EMPTY**
    int WEIGHTS_EMPTY

mpi4py.MPI.COMM_TYPE_SHARED

mpi4py.MPI.**COMM_TYPE_SHARED**: int = **COMM_TYPE_SHARED**
    int COMM_TYPE_SHARED

mpi4py.MPI.BSEND_OVERHEAD

mpi4py.MPI.**BSEND_OVERHEAD**: int = **BSEND_OVERHEAD**
    int BSEND_OVERHEAD

mpi4py.MPI.WIN_FLAVOR_CREATE

mpi4py.MPI.**WIN_FLAVOR_CREATE**: int = **WIN_FLAVOR_CREATE**
    int WIN_FLAVOR_CREATE

mpi4py.MPI.WIN_FLAVOR_ALLOCATE

mpi4py.MPI.**WIN_FLAVOR_ALLOCATE**: int = **WIN_FLAVOR_ALLOCATE**
    int WIN_FLAVOR_ALLOCATE

mpi4py.MPI.WIN_FLAVOR_DYNAMIC

mpi4py.MPI.**WIN_FLAVOR_DYNAMIC**: int = **WIN_FLAVOR_DYNAMIC**
    int WIN_FLAVOR_DYNAMIC

mpi4py.MPI.WIN_FLAVOR_SHARED

mpi4py.MPI.**WIN_FLAVOR_SHARED: int = WIN_FLAVOR_SHARED**

int WIN_FLAVOR_SHARED

mpi4py.MPI.WIN_SEPARATE

mpi4py.MPI.**WIN_SEPARATE: int = WIN_SEPARATE**

int WIN_SEPARATE

mpi4py.MPI.WIN_UNIFIED

mpi4py.MPI.**WIN_UNIFIED: int = WIN_UNIFIED**

int WIN_UNIFIED

mpi4py.MPI.MODE_NOCHECK

mpi4py.MPI.**MODE_NOCHECK: int = MODE_NOCHECK**

int MODE_NOCHECK

mpi4py.MPI.MODE_NOSTORE

mpi4py.MPI.**MODE_NOSTORE: int = MODE_NOSTORE**

int MODE_NOSTORE

mpi4py.MPI.MODE_NOPUT

mpi4py.MPI.**MODE_NOPUT: int = MODE_NOPUT**

int MODE_NOPUT

mpi4py.MPI.MODE_NOPRECEDE

mpi4py.MPI.**MODE_NOPRECEDE: int = MODE_NOPRECEDE**

int MODE_NOPRECEDE

mpi4py.MPI.MODE_NOSUCCEED

mpi4py.MPI.**MODE_NOSUCCEED: int = MODE_NOSUCCEED**

int MODE_NOSUCCEED

mpi4py.MPI.LOCK_EXCLUSIVE

mpi4py.MPI.**LOCK_EXCLUSIVE: int = LOCK_EXCLUSIVE**
    int LOCK_EXCLUSIVE

mpi4py.MPI.LOCK_SHARED

mpi4py.MPI.**LOCK_SHARED: int = LOCK_SHARED**
    int LOCK_SHARED

mpi4py.MPI.MODE_RDONLY

mpi4py.MPI.**MODE_RDONLY: int = MODE_RDONLY**
    int MODE_RDONLY

mpi4py.MPI.MODE_WRONLY

mpi4py.MPI.**MODE_WRONLY: int = MODE_WRONLY**
    int MODE_WRONLY

mpi4py.MPI.MODE_RDWR

mpi4py.MPI.**MODE_RDWR: int = MODE_RDWR**
    int MODE_RDWR

mpi4py.MPI.MODE_CREATE

mpi4py.MPI.**MODE_CREATE: int = MODE_CREATE**
    int MODE_CREATE

mpi4py.MPI.MODE_EXCL

mpi4py.MPI.**MODE_EXCL: int = MODE_EXCL**
    int MODE_EXCL

mpi4py.MPI.MODE_DELETE_ON_CLOSE

mpi4py.MPI.**MODE_DELETE_ON_CLOSE: int = MODE_DELETE_ON_CLOSE**
    int MODE_DELETE_ON_CLOSE

mpi4py.MPI.MODE_UNIQUE_OPEN

mpi4py.MPI.**MODE_UNIQUE_OPEN: int = MODE_UNIQUE_OPEN**

int MODE_UNIQUE_OPEN

mpi4py.MPI.MODE_SEQUENTIAL

mpi4py.MPI.**MODE_SEQUENTIAL: int = MODE_SEQUENTIAL**

int MODE_SEQUENTIAL

mpi4py.MPI.MODE_APPEND

mpi4py.MPI.**MODE_APPEND: int = MODE_APPEND**

int MODE_APPEND

mpi4py.MPI.SEEK_SET

mpi4py.MPI.**SEEK_SET: int = SEEK_SET**

int SEEK_SET

mpi4py.MPI.SEEK_CUR

mpi4py.MPI.**SEEK_CUR: int = SEEK_CUR**

int SEEK_CUR

mpi4py.MPI.SEEK_END

mpi4py.MPI.**SEEK_END: int = SEEK_END**

int SEEK_END

mpi4py.MPI.DISPLACEMENT_CURRENT

mpi4py.MPI.**DISPLACEMENT_CURRENT: int = DISPLACEMENT_CURRENT**

int DISPLACEMENT_CURRENT

mpi4py.MPI.DISP_CUR

mpi4py.MPI.**DISP_CUR: int = DISP_CUR**

int DISP_CUR

mpi4py.MPI.THREAD_SINGLE

mpi4py.MPI.**THREAD_SINGLE**: int = **THREAD_SINGLE**
    int THREAD_SINGLE

mpi4py.MPI.THREAD_FUNNELED

mpi4py.MPI.**THREAD_FUNNELED**: int = **THREAD_FUNNELED**
    int THREAD_FUNNELED

mpi4py.MPI.THREAD_SERIALIZED

mpi4py.MPI.**THREAD_SERIALIZED**: int = **THREAD_SERIALIZED**
    int THREAD_SERIALIZED

mpi4py.MPI.THREAD_MULTIPLE

mpi4py.MPI.**THREAD_MULTIPLE**: int = **THREAD_MULTIPLE**
    int THREAD_MULTIPLE

mpi4py.MPI.VERSION

mpi4py.MPI.**VERSION**: int = **VERSION**
    int VERSION

mpi4py.MPI.SUBVERSION

mpi4py.MPI.**SUBVERSION**: int = **SUBVERSION**
    int SUBVERSION

mpi4py.MPI.MAX_PROCESSOR_NAME

mpi4py.MPI.**MAX_PROCESSOR_NAME**: int = **MAX_PROCESSOR_NAME**
    int MAX_PROCESSOR_NAME

mpi4py.MPI.MAX_ERROR_STRING

mpi4py.MPI.**MAX_ERROR_STRING**: int = **MAX_ERROR_STRING**
    int MAX_ERROR_STRING

mpi4py.MPI.MAX_PORT_NAME

mpi4py.MPI.**MAX_PORT_NAME: int = MAX_PORT_NAME**
int MAX_PORT_NAME

mpi4py.MPI.MAX_INFO_KEY

mpi4py.MPI.**MAX_INFO_KEY: int = MAX_INFO_KEY**
int MAX_INFO_KEY

mpi4py.MPI.MAX_INFO_VAL

mpi4py.MPI.**MAX_INFO_VAL: int = MAX_INFO_VAL**
int MAX_INFO_VAL

mpi4py.MPI.MAX_OBJECT_NAME

mpi4py.MPI.**MAX_OBJECT_NAME: int = MAX_OBJECT_NAME**
int MAX_OBJECT_NAME

mpi4py.MPI.MAX_DATAREP_STRING

mpi4py.MPI.**MAX_DATAREP_STRING: int = MAX_DATAREP_STRING**
int MAX_DATAREP_STRING

mpi4py.MPI.MAX_LIBRARY_VERSION_STRING

mpi4py.MPI.**MAX_LIBRARY_VERSION_STRING: int = MAX_LIBRARY_VERSION_STRING**
int MAX_LIBRARY_VERSION_STRING

mpi4py.MPI.DATATYPE_NULL

mpi4py.MPI.**DATATYPE_NULL:** ***Datatype*** **= DATATYPE_NULL**
Datatype DATATYPE_NULL

mpi4py.MPI.UB

mpi4py.MPI.**UB:** ***Datatype*** **= UB**
Datatype UB

mpi4py.MPI.LB

mpi4py.MPI.**LB**: *Datatype* = **LB**
Datatype LB

mpi4py.MPI.PACKED

mpi4py.MPI.**PACKED**: *Datatype* = **PACKED**
Datatype PACKED

mpi4py.MPI.BYTE

mpi4py.MPI.**BYTE**: *Datatype* = **BYTE**
Datatype BYTE

mpi4py.MPI.AINT

mpi4py.MPI.**AINT**: *Datatype* = **AINT**
Datatype AINT

mpi4py.MPI.OFFSET

mpi4py.MPI.**OFFSET**: *Datatype* = **OFFSET**
Datatype OFFSET

mpi4py.MPI.COUNT

mpi4py.MPI.**COUNT**: *Datatype* = **COUNT**
Datatype COUNT

mpi4py.MPI.CHAR

mpi4py.MPI.**CHAR**: *Datatype* = **CHAR**
Datatype CHAR

mpi4py.MPI.WCHAR

mpi4py.MPI.**WCHAR**: *Datatype* = **WCHAR**
Datatype WCHAR

mpi4py.MPI.SIGNED_CHAR

mpi4py.MPI.**SIGNED_CHAR**: *Datatype* = **SIGNED_CHAR**
    Datatype SIGNED_CHAR

mpi4py.MPI.SHORT

mpi4py.MPI.**SHORT**: *Datatype* = **SHORT**
    Datatype SHORT

mpi4py.MPI.INT

mpi4py.MPI.**INT**: *Datatype* = **INT**
    Datatype INT

mpi4py.MPI.LONG

mpi4py.MPI.**LONG**: *Datatype* = **LONG**
    Datatype LONG

mpi4py.MPI.LONG_LONG

mpi4py.MPI.**LONG_LONG**: *Datatype* = **LONG_LONG**
    Datatype LONG_LONG

mpi4py.MPI.UNSIGNED_CHAR

mpi4py.MPI.**UNSIGNED_CHAR**: *Datatype* = **UNSIGNED_CHAR**
    Datatype UNSIGNED_CHAR

mpi4py.MPI.UNSIGNED_SHORT

mpi4py.MPI.**UNSIGNED_SHORT**: *Datatype* = **UNSIGNED_SHORT**
    Datatype UNSIGNED_SHORT

mpi4py.MPI.UNSIGNED

mpi4py.MPI.**UNSIGNED**: *Datatype* = **UNSIGNED**
    Datatype UNSIGNED

mpi4py.MPI.UNSIGNED_LONG

mpi4py.MPI.**UNSIGNED_LONG**: *Datatype* = **UNSIGNED_LONG**
    Datatype UNSIGNED_LONG

mpi4py.MPI.UNSIGNED_LONG_LONG

mpi4py.MPI.**UNSIGNED_LONG_LONG**: *Datatype* = **UNSIGNED_LONG_LONG**
    Datatype UNSIGNED_LONG_LONG

mpi4py.MPI.FLOAT

mpi4py.MPI.**FLOAT**: *Datatype* = **FLOAT**
    Datatype FLOAT

mpi4py.MPI.DOUBLE

mpi4py.MPI.**DOUBLE**: *Datatype* = **DOUBLE**
    Datatype DOUBLE

mpi4py.MPI.LONG_DOUBLE

mpi4py.MPI.**LONG_DOUBLE**: *Datatype* = **LONG_DOUBLE**
    Datatype LONG_DOUBLE

mpi4py.MPI.C_BOOL

mpi4py.MPI.**C_BOOL**: *Datatype* = **C_BOOL**
    Datatype C_BOOL

mpi4py.MPI.INT8_T

mpi4py.MPI.**INT8_T**: *Datatype* = **INT8_T**
    Datatype INT8_T

mpi4py.MPI.INT16_T

mpi4py.MPI.**INT16_T**: *Datatype* = **INT16_T**
    Datatype INT16_T

mpi4py.MPI.INT32_T

mpi4py.MPI.**INT32_T**: *Datatype* = **INT32_T**

Datatype INT32_T

mpi4py.MPI.INT64_T

mpi4py.MPI.**INT64_T**: *Datatype* = **INT64_T**

Datatype INT64_T

mpi4py.MPI.UINT8_T

mpi4py.MPI.**UINT8_T**: *Datatype* = **UINT8_T**

Datatype UINT8_T

mpi4py.MPI.UINT16_T

mpi4py.MPI.**UINT16_T**: *Datatype* = **UINT16_T**

Datatype UINT16_T

mpi4py.MPI.UINT32_T

mpi4py.MPI.**UINT32_T**: *Datatype* = **UINT32_T**

Datatype UINT32_T

mpi4py.MPI.UINT64_T

mpi4py.MPI.**UINT64_T**: *Datatype* = **UINT64_T**

Datatype UINT64_T

mpi4py.MPI.C_COMPLEX

mpi4py.MPI.**C_COMPLEX**: *Datatype* = **C_COMPLEX**

Datatype C_COMPLEX

mpi4py.MPI.C_FLOAT_COMPLEX

mpi4py.MPI.**C_FLOAT_COMPLEX**: *Datatype* = **C_FLOAT_COMPLEX**

Datatype C_FLOAT_COMPLEX

mpi4py.MPI.C_DOUBLE_COMPLEX

mpi4py.MPI.**C_DOUBLE_COMPLEX**: *Datatype* = **C_DOUBLE_COMPLEX**
    Datatype C_DOUBLE_COMPLEX

mpi4py.MPI.C_LONG_DOUBLE_COMPLEX

mpi4py.MPI.**C_LONG_DOUBLE_COMPLEX**: *Datatype* = **C_LONG_DOUBLE_COMPLEX**
    Datatype C_LONG_DOUBLE_COMPLEX

mpi4py.MPI.CXX_BOOL

mpi4py.MPI.**CXX_BOOL**: *Datatype* = **CXX_BOOL**
    Datatype CXX_BOOL

mpi4py.MPI.CXX_FLOAT_COMPLEX

mpi4py.MPI.**CXX_FLOAT_COMPLEX**: *Datatype* = **CXX_FLOAT_COMPLEX**
    Datatype CXX_FLOAT_COMPLEX

mpi4py.MPI.CXX_DOUBLE_COMPLEX

mpi4py.MPI.**CXX_DOUBLE_COMPLEX**: *Datatype* = **CXX_DOUBLE_COMPLEX**
    Datatype CXX_DOUBLE_COMPLEX

mpi4py.MPI.CXX_LONG_DOUBLE_COMPLEX

mpi4py.MPI.**CXX_LONG_DOUBLE_COMPLEX**: *Datatype* = **CXX_LONG_DOUBLE_COMPLEX**
    Datatype CXX_LONG_DOUBLE_COMPLEX

mpi4py.MPI.SHORT_INT

mpi4py.MPI.**SHORT_INT**: *Datatype* = **SHORT_INT**
    Datatype SHORT_INT

mpi4py.MPI.INT_INT

mpi4py.MPI.**INT_INT**: *Datatype* = **INT_INT**
    Datatype INT_INT

mpi4py.MPI.TWOINT

`mpi4py.MPI.TWOINT: Datatype = TWOINT`

Datatype TWOINT

mpi4py.MPI.LONG_INT

`mpi4py.MPI.LONG_INT: Datatype = LONG_INT`

Datatype LONG_INT

mpi4py.MPI.FLOAT_INT

`mpi4py.MPI.FLOAT_INT: Datatype = FLOAT_INT`

Datatype FLOAT_INT

mpi4py.MPI.DOUBLE_INT

`mpi4py.MPI.DOUBLE_INT: Datatype = DOUBLE_INT`

Datatype DOUBLE_INT

mpi4py.MPI.LONG_DOUBLE_INT

`mpi4py.MPI.LONG_DOUBLE_INT: Datatype = LONG_DOUBLE_INT`

Datatype LONG_DOUBLE_INT

mpi4py.MPI.CHARACTER

`mpi4py.MPI.CHARACTER: Datatype = CHARACTER`

Datatype CHARACTER

mpi4py.MPI.LOGICAL

`mpi4py.MPI.LOGICAL: Datatype = LOGICAL`

Datatype LOGICAL

mpi4py.MPI.INTEGER

`mpi4py.MPI.INTEGER: Datatype = INTEGER`

Datatype INTEGER

mpi4py.MPI.REAL

mpi4py.MPI.**REAL**: *Datatype* = REAL
    Datatype REAL

mpi4py.MPI.DOUBLE_PRECISION

mpi4py.MPI.**DOUBLE_PRECISION**: *Datatype* = DOUBLE_PRECISION
    Datatype DOUBLE_PRECISION

mpi4py.MPI.COMPLEX

mpi4py.MPI.**COMPLEX**: *Datatype* = COMPLEX
    Datatype COMPLEX

mpi4py.MPI.DOUBLE_COMPLEX

mpi4py.MPI.**DOUBLE_COMPLEX**: *Datatype* = DOUBLE_COMPLEX
    Datatype DOUBLE_COMPLEX

mpi4py.MPI.LOGICAL1

mpi4py.MPI.**LOGICAL1**: *Datatype* = LOGICAL1
    Datatype LOGICAL1

mpi4py.MPI.LOGICAL2

mpi4py.MPI.**LOGICAL2**: *Datatype* = LOGICAL2
    Datatype LOGICAL2

mpi4py.MPI.LOGICAL4

mpi4py.MPI.**LOGICAL4**: *Datatype* = LOGICAL4
    Datatype LOGICAL4

mpi4py.MPI.LOGICAL8

mpi4py.MPI.**LOGICAL8**: *Datatype* = LOGICAL8
    Datatype LOGICAL8

mpi4py.MPI.INTEGER1

mpi4py.MPI.**INTEGER1:** *Datatype* **= INTEGER1**

Datatype INTEGER1

mpi4py.MPI.INTEGER2

mpi4py.MPI.**INTEGER2:** *Datatype* **= INTEGER2**

Datatype INTEGER2

mpi4py.MPI.INTEGER4

mpi4py.MPI.**INTEGER4:** *Datatype* **= INTEGER4**

Datatype INTEGER4

mpi4py.MPI.INTEGER8

mpi4py.MPI.**INTEGER8:** *Datatype* **= INTEGER8**

Datatype INTEGER8

mpi4py.MPI.INTEGER16

mpi4py.MPI.**INTEGER16:** *Datatype* **= INTEGER16**

Datatype INTEGER16

mpi4py.MPI.REAL2

mpi4py.MPI.**REAL2:** *Datatype* **= REAL2**

Datatype REAL2

mpi4py.MPI.REAL4

mpi4py.MPI.**REAL4:** *Datatype* **= REAL4**

Datatype REAL4

mpi4py.MPI.REAL8

mpi4py.MPI.**REAL8:** *Datatype* **= REAL8**

Datatype REAL8

mpi4py.MPI.REAL16

mpi4py.MPI.**REAL16**: *Datatype* = **REAL16**

Datatype REAL16

mpi4py.MPI.COMPLEX4

mpi4py.MPI.**COMPLEX4**: *Datatype* = **COMPLEX4**

Datatype COMPLEX4

mpi4py.MPI.COMPLEX8

mpi4py.MPI.**COMPLEX8**: *Datatype* = **COMPLEX8**

Datatype COMPLEX8

mpi4py.MPI.COMPLEX16

mpi4py.MPI.**COMPLEX16**: *Datatype* = **COMPLEX16**

Datatype COMPLEX16

mpi4py.MPI.COMPLEX32

mpi4py.MPI.**COMPLEX32**: *Datatype* = **COMPLEX32**

Datatype COMPLEX32

mpi4py.MPI.UNSIGNED_INT

mpi4py.MPI.**UNSIGNED_INT**: *Datatype* = **UNSIGNED_INT**

Datatype UNSIGNED_INT

mpi4py.MPI.SIGNED_SHORT

mpi4py.MPI.**SIGNED_SHORT**: *Datatype* = **SIGNED_SHORT**

Datatype SIGNED_SHORT

mpi4py.MPI.SIGNED_INT

mpi4py.MPI.**SIGNED_INT**: *Datatype* = **SIGNED_INT**

Datatype SIGNED_INT

mpi4py.MPI.SIGNED_LONG

mpi4py.MPI.**SIGNED_LONG**: *Datatype* = SIGNED_LONG
    Datatype SIGNED_LONG

mpi4py.MPI.SIGNED_LONG_LONG

mpi4py.MPI.**SIGNED_LONG_LONG**: *Datatype* = SIGNED_LONG_LONG
    Datatype SIGNED_LONG_LONG

mpi4py.MPI.BOOL

mpi4py.MPI.**BOOL**: *Datatype* = BOOL
    Datatype BOOL

mpi4py.MPI.SINT8_T

mpi4py.MPI.**SINT8_T**: *Datatype* = SINT8_T
    Datatype SINT8_T

mpi4py.MPI.SINT16_T

mpi4py.MPI.**SINT16_T**: *Datatype* = SINT16_T
    Datatype SINT16_T

mpi4py.MPI.SINT32_T

mpi4py.MPI.**SINT32_T**: *Datatype* = SINT32_T
    Datatype SINT32_T

mpi4py.MPI.SINT64_T

mpi4py.MPI.**SINT64_T**: *Datatype* = SINT64_T
    Datatype SINT64_T

mpi4py.MPI.F_BOOL

mpi4py.MPI.**F_BOOL**: *Datatype* = F_BOOL
    Datatype F_BOOL

mpi4py.MPI.F_INT

mpi4py.MPI.**F_INT**: *Datatype* = **F_INT**
    Datatype F_INT

mpi4py.MPI.F_FLOAT

mpi4py.MPI.**F_FLOAT**: *Datatype* = **F_FLOAT**
    Datatype F_FLOAT

mpi4py.MPI.F_DOUBLE

mpi4py.MPI.**F_DOUBLE**: *Datatype* = **F_DOUBLE**
    Datatype F_DOUBLE

mpi4py.MPI.F_COMPLEX

mpi4py.MPI.**F_COMPLEX**: *Datatype* = **F_COMPLEX**
    Datatype F_COMPLEX

mpi4py.MPI.F_FLOAT_COMPLEX

mpi4py.MPI.**F_FLOAT_COMPLEX**: *Datatype* = **F_FLOAT_COMPLEX**
    Datatype F_FLOAT_COMPLEX

mpi4py.MPI.F_DOUBLE_COMPLEX

mpi4py.MPI.**F_DOUBLE_COMPLEX**: *Datatype* = **F_DOUBLE_COMPLEX**
    Datatype F_DOUBLE_COMPLEX

mpi4py.MPI.REQUEST_NULL

mpi4py.MPI.**REQUEST_NULL**: *Request* = **REQUEST_NULL**
    Request REQUEST_NULL

mpi4py.MPI.MESSAGE_NULL

mpi4py.MPI.**MESSAGE_NULL**: *Message* = **MESSAGE_NULL**
    Message MESSAGE_NULL

mpi4py.MPI.MESSAGE_NO_PROC

mpi4py.MPI.**MESSAGE_NO_PROC**: *Message* = **MESSAGE_NO_PROC**
    Message MESSAGE_NO_PROC

mpi4py.MPI.OP_NULL

mpi4py.MPI.**OP_NULL**: *Op* = **OP_NULL**
    Op OP_NULL

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type Any

mpi4py.MPI.MAX

mpi4py.MPI.**MAX**: *Op* = **MAX**
    Op MAX

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type Any

mpi4py.MPI.MIN

mpi4py.MPI.**MIN**: *Op* = **MIN**
    Op MIN

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type Any

mpi4py.MPI.SUM

mpi4py.MPI.**SUM**: *Op* = **SUM**
    Op SUM

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type Any

mpi4py.MPI.PROD

`mpi4py.MPI.PROD: Op = PROD`

Op PROD

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type Any

mpi4py.MPI.LAND

`mpi4py.MPI.LAND: Op = LAND`

Op LAND

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type Any

mpi4py.MPI.BAND

`mpi4py.MPI.BAND: Op = BAND`

Op BAND

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type Any

mpi4py.MPI.LOR

`mpi4py.MPI.LOR: Op = LOR`

Op LOR

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type Any

mpi4py.MPI.BOR

`mpi4py.MPI.BOR: Op = BOR`

Op BOR

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type Any

mpi4py.MPI.LXOR

`mpi4py.MPI.LXOR: Op = LXOR`

Op LXOR

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type Any

mpi4py.MPI.BXOR

`mpi4py.MPI.BXOR: Op = BXOR`

Op BXOR

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type Any

mpi4py.MPI.MAXLOC

`mpi4py.MPI.MAXLOC: Op = MAXLOC`

Op MAXLOC

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type Any

mpi4py.MPI.MINLOC

mpi4py.MPI.**MINLOC**: *Op* = **MINLOC**

Op MINLOC

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type Any

mpi4py.MPI.REPLACE

mpi4py.MPI.**REPLACE**: *Op* = **REPLACE**

Op REPLACE

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type Any

mpi4py.MPI.NO_OP

mpi4py.MPI.**NO_OP**: *Op* = **NO_OP**

Op NO_OP

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type Any

mpi4py.MPI.GROUP_NULL

mpi4py.MPI.**GROUP_NULL**: *Group* = **GROUP_NULL**

Group GROUP_NULL

mpi4py.MPI.GROUP_EMPTY

mpi4py.MPI.**GROUP_EMPTY**: *Group* = **GROUP_EMPTY**

Group GROUP_EMPTY

mpi4py.MPI.INFO_NULL

mpi4py.MPI.**INFO_NULL**: *Info* = **INFO_NULL**

Info INFO_NULL

mpi4py.MPI.INFO_ENV

mpi4py.MPI.**INFO_ENV**: *Info* = **INFO_ENV**

Info INFO_ENV

mpi4py.MPI.ERRHANDLER_NULL

mpi4py.MPI.**ERRHANDLER_NULL**: *Errhandler* = **ERRHANDLER_NULL**

Errhandler ERRHANDLER_NULL

mpi4py.MPI.ERRORS_RETURN

mpi4py.MPI.**ERRORS_RETURN**: *Errhandler* = **ERRORS_RETURN**

Errhandler ERRORS_RETURN

mpi4py.MPI.ERRORS_ARE_FATAL

mpi4py.MPI.**ERRORS_ARE_FATAL**: *Errhandler* = **ERRORS_ARE_FATAL**

Errhandler ERRORS_ARE_FATAL

mpi4py.MPI.COMM_NULL

mpi4py.MPI.**COMM_NULL**: *Comm* = **COMM_NULL**

Comm COMM_NULL

mpi4py.MPI.COMM_SELF

mpi4py.MPI.**COMM_SELF**: *Intracomm* = **COMM_SELF**

Intracomm COMM_SELF

mpi4py.MPI.COMM_WORLD

mpi4py.MPI.**COMM_WORLD**: *Intracomm* = **COMM_WORLD**

Intracomm COMM_WORLD

mpi4py.MPI.WIN_NULL

`mpi4py.MPI.WIN_NULL: Win = WIN_NULL`

Win WIN_NULL

mpi4py.MPI.FILE_NULL

`mpi4py.MPI.FILE_NULL: File = FILE_NULL`

File FILE_NULL

mpi4py.MPI.pickle

`mpi4py.MPI.pickle: Pickle = <mpi4py.MPI.Pickle object>`

Pickle pickle

10 Citation

If MPI for Python been significant to a project that leads to an academic publication, please acknowledge that fact by citing the project.

- L. Dalcin and Y.-L. L. Fang, *mpi4py: Status Update After 12 Years of Development*, Computing in Science & Engineering, 23(4):47-54, 2021. https://doi.org/10.1109/MCSE.2021.3083216
- L. Dalcin, P. Kler, R. Paz, and A. Cosimo, *Parallel Distributed Computing using Python*, Advances in Water Resources, 34(9):1124-1139, 2011. https://doi.org/10.1016/j.advwatres.2011.04.013
- L. Dalcin, R. Paz, M. Storti, and J. D'Elia, *MPI for Python: performance improvements and MPI-2 extensions*, Journal of Parallel and Distributed Computing, 68(5):655-662, 2008. https://doi.org/10.1016/j.jpdc.2007.09.005
- L. Dalcin, R. Paz, and M. Storti, *MPI for Python*, Journal of Parallel and Distributed Computing, 65(9):1108-1115, 2005. https://doi.org/10.1016/j.jpdc.2005.03.010

11 Installation

11.1 Requirements

You need to have the following software properly installed in order to build *MPI for Python*:

- A working MPI implementation, preferably supporting MPI-3 and built with shared/dynamic libraries.

  Note: If you want to build some MPI implementation from sources, check the instructions at *Building MPI from sources* in the appendix.

- Python 2.7, 3.5 or above.

  Note: Some MPI-1 implementations **do require** the actual command line arguments to be passed in `MPI_Init()`. In this case, you will need to use a rebuilt, MPI-enabled, Python interpreter executable. *MPI for Python* has some support for alleviating you from this task. Check the instructions at *MPI-enabled Python interpreter* in the appendix.

11.2 Using pip

If you already have a working MPI (either if you installed it from sources or by using a pre-built package from your favourite GNU/Linux distribution) and the **mpicc** compiler wrapper is on your search path, you can use **pip**:

```
$ python -m pip install mpi4py
```

Note: If the **mpicc** compiler wrapper is not on your search path (or if it has a different name) you can use **env** to pass the environment variable `MPICC` providing the full path to the MPI compiler wrapper executable:

```
$ env MPICC=/path/to/mpicc python -m pip install mpi4py
```

Warning: **pip** keeps previouly built wheel files on its cache for future reuse. If you want to reinstall the *mpi4py* package using a different or updated MPI implementation, you have to either first remove the cached wheel file with:

```
$ python -m pip cache remove mpi4py
```

or ask **pip** to disable the cache:

```
$ python -m pip install --no-cache-dir mpi4py
```

11.3 Using distutils

The *MPI for Python* package is available for download at the project website generously hosted by GitHub. You can use **curl** or **wget** to get a release tarball.

- Using **curl**:

```
$ curl -O https://github.com/mpi4py/mpi4py/releases/download/X.Y.Z/mpi4py-X.Y.Z.tar.
↪gz
```

- Using **wget**:

```
$ wget https://github.com/mpi4py/mpi4py/releases/download/X.Y.Z/mpi4py-X.Y.Z.tar.gz
```

After unpacking the release tarball:

```
$ tar -zxf mpi4py-X.Y.Z.tar.gz
$ cd mpi4py-X.Y.Z
```

the package is ready for building.

MPI for Python uses a standard distutils-based build system. However, some distutils commands (like *build*) have additional options:

--mpicc=
Lets you specify a special location or name for the **mpicc** compiler wrapper.

--mpi=
Lets you pass a section with MPI configuration within a special configuration file.

--configure
Runs exhaustive tests for checking about missing MPI types, constants, and functions. This option should be

passed in order to build *MPI for Python* against old MPI-1 or MPI-2 implementations, possibly providing a subset of MPI-3.

If you use a MPI implementation providing a **mpicc** compiler wrapper (e.g., MPICH, Open MPI), it will be used for compilation and linking. This is the preferred and easiest way of building *MPI for Python*.

If **mpicc** is located somewhere in your search path, simply run the *build* command:

```
$ python setup.py build
```

If **mpicc** is not in your search path or the compiler wrapper has a different name, you can run the *build* command specifying its location:

```
$ python setup.py build --mpicc=/where/you/have/mpicc
```

Alternatively, you can provide all the relevant information about your MPI implementation by editing the file called `mpi.cfg`. You can use the default section `[mpi]` or add a new, custom section, for example `[other_mpi]` (see the examples provided in the `mpi.cfg` file as a starting point to write your own section):

```
[mpi]

include_dirs         = /usr/local/mpi/include
libraries            = mpi
library_dirs         = /usr/local/mpi/lib
runtime_library_dirs = /usr/local/mpi/lib

[other_mpi]

include_dirs         = /opt/mpi/include ...
libraries            = mpi ...
library_dirs         = /opt/mpi/lib ...
runtime_library_dirs = /op/mpi/lib ...

...
```

and then run the *build* command, perhaps specifying you custom configuration section:

```
$ python setup.py build --mpi=other_mpi
```

After building, the package is ready for install.

If you have root privileges (either by log-in as the root user of by using **sudo**) and you want to install *MPI for Python* in your system for all users, just do:

```
$ python setup.py install
```

The previous steps will install the `mpi4py` package at standard location `prefix/lib/pythonX.X/site-packages`.

If you do not have root privileges or you want to install *MPI for Python* for your private use, just do:

```
$ python setup.py install --user
```

11.4 Testing

To quickly test the installation:

```
$ mpiexec -n 5 python -m mpi4py.bench helloworld
Hello, World! I am process 0 of 5 on localhost.
Hello, World! I am process 1 of 5 on localhost.
Hello, World! I am process 2 of 5 on localhost.
Hello, World! I am process 3 of 5 on localhost.
Hello, World! I am process 4 of 5 on localhost.
```

If you installed from source, issuing at the command line:

```
$ mpiexec -n 5 python demo/helloworld.py
```

or (in the case of ancient MPI-1 implementations):

```
$ mpirun -np 5 python `pwd`/demo/helloworld.py
```

will launch a five-process run of the Python interpreter and run the test script `demo/helloworld.py` from the source distribution.

You can also run all the *unittest* scripts:

```
$ mpiexec -n 5 python test/runtests.py
```

or, if you have nose unit testing framework installed:

```
$ mpiexec -n 5 nosetests -w test
```

or, if you have py.test unit testing framework installed:

```
$ mpiexec -n 5 py.test test/
```

12 Appendix

12.1 MPI-enabled Python interpreter

> **Warning:** These days it is no longer required to use the MPI-enabled Python interpreter in most cases, and, therefore, it is not built by default anymore because it is too difficult to reliably build a Python interpreter across different distributions. If you know that you still **really** need it, see below on how to use the `build_exe` and `install_exe` commands.

Some MPI-1 implementations (notably, MPICH 1) **do require** the actual command line arguments to be passed at the time `MPI_Init()` is called. In this case, you will need to use a re-built, MPI-enabled, Python interpreter binary executable. A basic implementation (targeting Python 2.X) of what is required is shown below:

```c
#include <Python.h>
#include <mpi.h>

int main(int argc, char *argv[])
```

(continues on next page)

(continued from previous page)

```
{
   int status, flag;
   MPI_Init(&argc, &argv);
   status = Py_Main(argc, argv);
   MPI_Finalized(&flag);
   if (!flag) MPI_Finalize();
   return status;
}
```

The source code above is straightforward; compiling it should also be. However, the linking step is more tricky: special flags have to be passed to the linker depending on your platform. In order to alleviate you for such low-level details, *MPI for Python* provides some pure-distutils based support to build and install an MPI-enabled Python interpreter executable:

```
$ cd mpi4py-X.X.X
$ python setup.py build_exe [--mpi=<name>|--mpicc=/path/to/mpicc]
$ [sudo] python setup.py install_exe [--install-dir=$HOME/bin]
```

After the above steps you should have the MPI-enabled interpreter installed as `prefix/bin/pythonX.X-mpi` (or `$HOME/bin/pythonX.X-mpi`). Assuming that `prefix/bin` (or `$HOME/bin`) is listed on your `PATH`, you should be able to enter your MPI-enabled Python interactively, for example:

```
$ python2.7-mpi
Python 2.7.8 (default, Nov 10 2014, 08:19:18)
[GCC 4.9.2 20141101 (Red Hat 4.9.2-1)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.executable
'/usr/bin/python2.7-mpi'
>>>
```

12.2 Building MPI from sources

In the list below you have some executive instructions for building some of the open-source MPI implementations out there with support for shared/dynamic libraries on POSIX environments.

- *MPICH*

  ```
  $ tar -zxf mpich-X.X.X.tar.gz
  $ cd mpich-X.X.X
  $ ./configure --enable-shared --prefix=/usr/local/mpich
  $ make
  $ make install
  ```

- *Open MPI*

  ```
  $ tar -zxf openmpi-X.X.X tar.gz
  $ cd openmpi-X.X.X
  $ ./configure --prefix=/usr/local/openmpi
  $ make all
  $ make install
  ```

- *MPICH 1*

```
$ tar -zxf mpich-X.X.X.tar.gz
$ cd mpich-X.X.X
$ ./configure --enable-sharedlib --prefix=/usr/local/mpich1
$ make
$ make install
```

Perhaps you will need to set the `LD_LIBRARY_PATH` environment variable (using **`export`**, **`setenv`** or what applies to your system) pointing to the directory containing the MPI libraries . In case of getting runtime linking errors when running MPI programs, the following lines can be added to the user login shell script (`.profile`, `.bashrc`, etc.).

- *MPICH*

```
MPI_DIR=/usr/local/mpich
export LD_LIBRARY_PATH=$MPI_DIR/lib:$LD_LIBRARY_PATH
```

- *Open MPI*

```
MPI_DIR=/usr/local/openmpi
export LD_LIBRARY_PATH=$MPI_DIR/lib:$LD_LIBRARY_PATH
```

- *MPICH 1*

```
MPI_DIR=/usr/local/mpich1
export LD_LIBRARY_PATH=$MPI_DIR/lib/shared:$LD_LIBRARY_PATH:
export MPICH_USE_SHLIB=yes
```

Warning: MPICH 1 support for dynamic libraries is not completely transparent. Users should set the environment variable `MPICH_USE_SHLIB` to `yes` in order to avoid link problems when using the **`mpicc`** compiler wrapper.

References

[mpi-std1] MPI Forum. MPI: A Message Passing Interface Standard. International Journal of Supercomputer Applications, volume 8, number 3-4, pages 159-416, 1994.

[mpi-std2] MPI Forum. MPI: A Message Passing Interface Standard. High Performance Computing Applications, volume 12, number 1-2, pages 1-299, 1998.

[mpi-using] William Gropp, Ewing Lusk, and Anthony Skjellum. Using MPI: portable parallel programming with the message-passing interface. MIT Press, 1994.

[mpi-ref] Mark Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. MPI - The Complete Reference, volume 1, The MPI Core. MIT Press, 2nd. edition, 1998.

[mpi-mpich] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. Parallel Computing, 22(6):789-828, September 1996.

[mpi-openmpi] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, September 2004.

[Hinsen97] Konrad Hinsen. The Molecular Modelling Toolkit: a case study of a large scientific application in Python. In Proceedings of the 6th International Python Conference, pages 29-35, San Jose, Ca., October 1997.

[Beazley97] David M. Beazley and Peter S. Lomdahl. Feeding a large-scale physics application to Python. In Proceedings of the 6th International Python Conference, pages 21-29, San Jose, Ca., October 1997.

Python Module Index

m

Index

Symbols

A

B

C

F

G

H

I

J

K

L

P

Q

R

S

T

U

V

W