
MPI for Python

Release 3.1.4

Lisandro Dalcin

November 07, 2022

Contents

1	Introduction	3
1.1	What is MPI?	3
1.2	What is Python?	3
1.3	Related Projects	4
2	Overview	5
2.1	Communicating Python Objects and Array Data	5
2.2	Communicators	5
2.3	Point-to-Point Communications	6
2.4	Collective Communications	7
2.5	Support for GPU-aware MPI	7
2.6	Dynamic Process Management	8
2.7	One-Sided Communications	8
2.8	Parallel Input/Output	9
2.9	Environmental Management	9
3	Tutorial	10
3.1	Running Python scripts with MPI	12
3.2	Point-to-Point Communication	12
3.3	Collective Communication	13
3.4	MPI-IO	15
3.5	Dynamic Process Management	16
3.6	CUDA-aware MPI + Python GPU arrays	17
3.7	One-Sided Communications	17
3.8	Wrapping with SWIG	18
3.9	Wrapping with F2Py	19
4	mpi4py	20
4.1	Runtime configuration options	20
4.2	Environment variables	22
4.3	Miscellaneous functions	24
5	mpi4py.MPI	25
5.1	Classes	25
5.2	Functions	26
5.3	Attributes	28

6	mpi4py.futures	33
6.1	concurrent.futures	33
6.2	MPIPoolExecutor	34
6.3	MPICommExecutor	37
6.4	Command line	37
6.5	Examples	38
7	mpi4py.util	40
7.1	mpi4py.util.pkl5	40
7.2	mpi4py.util.dtlb	46
8	mpi4py.run	46
8.1	Interface options	47
9	Reference	47
9.1	mpi4py.MPI	47
10	Citation	184
11	Installation	184
11.1	Requirements	184
11.2	Using pip	185
11.3	Using distutils	185
11.4	Testing	187
12	Appendix	187
12.1	MPI-enabled Python interpreter	187
12.2	Building MPI from sources	188
	References	189
	Python Module Index	191
	Index	192

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 *pickleable* 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, MATLAB¹ 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.
- **Pympar**
 - 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 programming. 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 subclasses 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 reduction 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 explicit 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 `python: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 `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):

```
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:

```
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!):

```
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)

```
# 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:

```
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:

```
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:

```
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)
itemsz = datatype.Get_size()

N = comm.Get_size() + 1
win_size = N * itemsz if rank == 0 else 0
win = MPI.Win.Allocate(
    size=win_size,
    disp_unit=itemsz,
    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

<i>initialize</i>	Automatic MPI initialization at import
<i>threads</i>	Request initialization with thread support
<i>thread_level</i>	Level of thread support to request
<i>finalize</i>	Automatic MPI finalization at exit
<i>fast_reduce</i>	Use tree-based reductions for objects
<i>recv_mprobe</i>	Use matched probes to receive objects
<i>errors</i>	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 programmatically:

```
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

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

Communication

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

One-sided operations

<i>Win</i> ([win])	Window handle
--------------------	---------------

Input/Output

<i>File</i> ([file])	File handle
----------------------	-------------

Error handling

<i>Errhandler</i> ([errhandler])	Error handler
<i>Exception</i> ([ierr])	Exception class

Auxiliary

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

5.2 Functions

Version inquiry

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

Initialization and finalization

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

Memory allocation

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

Address manipulation

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

Timer

<i>Wtick()</i>	Return the resolution of <i>Wtime</i>
<i>Wtime()</i>	Return an elapsed time on the calling processor

Error handling

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

Dynamic process management

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

Miscellanea

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

Utilities

<i>get_vendor</i> ()	Information about the underlying MPI implementation
----------------------	---

5.3 Attributes

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

continues on next page

Table 1 – continued from previous page

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

continues on next page

Table 1 – continued from previous page

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

continues on next page

Table 1 – 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 1 – continued from previous page

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

continues on next page

Table 1 – continued from previous page

<i>MESSAGE_NO_PROC</i>	<i>Message</i> MESSAGE_NO_PROC
<i>OP_NULL</i>	<i>Op</i> OP_NULL
<i>MAX</i>	<i>Op</i> MAX
<i>MIN</i>	<i>Op</i> MIN
<i>SUM</i>	<i>Op</i> SUM
<i>PROD</i>	<i>Op</i> PROD
<i>LAND</i>	<i>Op</i> LAND
<i>BAND</i>	<i>Op</i> BAND
<i>LOR</i>	<i>Op</i> LOR
<i>BOR</i>	<i>Op</i> BOR
<i>LXOR</i>	<i>Op</i> LXOR
<i>BXOR</i>	<i>Op</i> BXOR
<i>MAXLOC</i>	<i>Op</i> MAXLOC
<i>MINLOC</i>	<i>Op</i> MINLOC
<i>REPLACE</i>	<i>Op</i> REPLACE
<i>NO_OP</i>	<i>Op</i> NO_OP
<i>GROUP_NULL</i>	<i>Group</i> GROUP_NULL
<i>GROUP_EMPTY</i>	<i>Group</i> GROUP_EMPTY
<i>INFO_NULL</i>	<i>Info</i> INFO_NULL
<i>INFO_ENV</i>	<i>Info</i> INFO_ENV
<i>ERRHANDLER_NULL</i>	<i>Errhandler</i> ERRHANDLER_NULL
<i>ERRORS_RETURN</i>	<i>Errhandler</i> ERRORS_RETURN
<i>ERRORS_ARE_FATAL</i>	<i>Errhandler</i> ERRORS_ARE_FATAL
<i>COMM_NULL</i>	<i>Comm</i> COMM_NULL
<i>COMM_SELF</i>	<i>Intracomm</i> COMM_SELF
<i>COMM_WORLD</i>	<i>Intracomm</i> COMM_WORLD
<i>WIN_NULL</i>	<i>Win</i> WIN_NULL
<i>FILE_NULL</i>	<i>File</i> FILE_NULL
<i>pickle</i>	<i>Pickle</i> 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 additional, 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 management 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 instance 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 management 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:

`python:using-on-cmdline` 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`

```

1 from mpi4py.futures import MPIPoolExecutor
2
3 x0, x1, w = -2.0, +2.0, 640*2
4 y0, y1, h = -1.5, +1.5, 480*2
5 dx = (x1 - x0) / w
6 dy = (y1 - y0) / h
7
8 c = complex(0, 0.65)
9
10 def julia(x, y):
11     z = complex(x, y)
12     n = 255
13     while abs(z) < 3 and n > 1:
14         z = z**2 + c
15         n -= 1
16     return n
17
```

(continues on next page)

(continued from previous page)

```
18 def julia_line(k):
19     line = bytearray(w)
20     y = y1 - k * dy
21     for j in range(w):
22         x = x0 + j * dx
23         line[j] = julia(x, y)
24     return line
25
26 if __name__ == '__main__':
27
28     with MPIPoolExecutor() as executor:
29         image = executor.map(julia_line, range(h))
30         with open('julia.pgm', 'wb') as f:
31             f.write(b'P5 %d %d %d\n' % (w, h, 255))
32             for line in image:
33                 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 implementations¹. 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
```

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.

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.

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 <code>RuntimeError</code> .

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

```
1 import numpy as np
2 from mpi4py import MPI
3 from mpi4py.util import pkl5
4
5 comm = pkl5.Intracomm(MPI.COMM_WORLD) # comm wrapper
6 size = comm.Get_size()
7 rank = comm.Get_rank()
8 dst = (rank + 1) % size
9 src = (rank - 1) % size
10
11 sobj = np.full(1024**3, rank, dtype='i4') # > 4 GiB
12 sreq = comm.isend(sobj, dst, tag=42)
13 robj = comm.recv (None, src, tag=42)
14 sreq.Free()
15
16 assert np.min(robj) == src
17 assert np.max(robj) == src
```

Listing 3: test-pkl5-2.py

```
1 import numpy as np
2 from mpi4py import MPI
3 from mpi4py.util import pkl5
4
5 comm = pkl5.Intracomm(MPI.COMM_WORLD) # comm wrapper
6 size = comm.Get_size()
7 rank = comm.Get_rank()
8 dst = (rank + 1) % size
9 src = (rank - 1) % size
10
11 sobj = np.full(1024**3, rank, dtype='i4') # > 4 GiB
12 sreq = comm.isend(sobj, dst, tag=42)
13
14 status = MPI.Status()
15 rmsg = comm.mprobe(status=status)
16 assert status.Get_source() == src
17 assert status.Get_tag() == 42
18 rreq = rmsg.irecv()
19 robj = rreq.wait()
20
21 sreq.Free()
22 assert np.max(robj) == src
23 assert np.min(robj) == src
```

7.2 mpi4py.util.dtlb

New in version 3.1.0.

The `mpi4py.util.dtlb` module provides converter routines between NumPy and MPI datatypes.

`mpi4py.util.dtlb.from_numpy_dtype(dtype)`

Convert NumPy datatype to MPI datatype.

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

Return type `Datatype`

`mpi4py.util.dtlb.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:

python:using-on-cmdline Documentation on Python command line interface.

9 Reference

mpi4py.MPI

Message Passing Interface.

9.1 mpi4py.MPI

Message Passing Interface.

Classes

<i>Cartcomm</i> ([comm])	Cartesian topology intracommunicator
<i>Comm</i> ([comm])	Communicator
<i>Datatype</i> ([datatype])	Datatype object
<i>Distgraphcomm</i> ([comm])	Distributed graph topology intracommunicator
<i>Errhandler</i> ([errhandler])	Error handler
<i>File</i> ([file])	File handle
<i>Graphcomm</i> ([comm])	General graph topology intracommunicator
<i>Grequest</i> ([request])	Generalized request handle
<i>Group</i> ([group])	Group of processes
<i>Info</i> ([info])	Info object
<i>Intercomm</i> ([comm])	Intercommunicator
<i>Intracomm</i> ([comm])	Intracommunicator
<i>Message</i> ([message])	Matched message handle
<i>Op</i> ([op])	Operation object
<i>Pickle</i> ([dumps, loads, protocol])	Pickle/unpickle Python objects
<i>Prequest</i> ([request])	Persistent request handle
<i>Request</i> ([request])	Request handle
<i>Status</i> ([status])	Status object
<i>Topocomm</i> ([comm])	Topology intracommunicator
<i>Win</i> ([win])	Window handle
<i>memory</i> (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

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

Attributes Summary

<i>coords</i>	coordinates
<i>dim</i>	number of dimensions
<i>dims</i>	dimensions
<i>ndim</i>	number of dimensions
<i>periods</i>	periodicity
<i>topo</i>	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

<i>Abort</i> ([errorcode])	Terminate MPI execution environment
<i>Allgather</i> (sendbuf, recvbuf)	Gather to All, gather data from all processes and distribute it to all other processes in a group
<i>Allgatherv</i> (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
<i>Allreduce</i> (sendbuf, recvbuf[, op])	Reduce to All
<i>Alltoall</i> (sendbuf, recvbuf)	All to All Scatter/Gather, send data from all to all processes in a group
<i>Alltoallv</i> (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
<i>Alltoallw</i> (sendbuf, recvbuf)	Generalized All-to-All communication allowing different counts, displacements and datatypes for each partner

continues on next page

Table 2 – continued from previous page

<i>Barrier()</i>	Barrier synchronization
<i>Bcast</i> (buf[, root])	Broadcast a message from one process to all other processes in a group
<i>Bsend</i> (buf, dest[, tag])	Blocking send in buffered mode
<i>Bsend_init</i> (buf, dest[, tag])	Persistent request for a send in buffered mode
<i>Call_errhandler</i> (errorcode)	Call the error handler installed on a communicator
<i>Clone()</i>	Clone an existing communicator
<i>Compare</i> (comm1, comm2)	Compare two communicators
<i>Create</i> (group)	Create communicator from group
<i>Create_group</i> (group[, tag])	Create communicator from group
<i>Create_keyval</i> ([copy_fn, delete_fn, nopython])	Create a new attribute key for communicators
<i>Delete_attr</i> (keyval)	Delete attribute value associated with a key
<i>Disconnect()</i>	Disconnect from a communicator
<i>Dup</i> ([info])	Duplicate an existing communicator
<i>Dup_with_info</i> (info)	Duplicate an existing communicator
<i>Free()</i>	Free a communicator
<i>Free_keyval</i> (keyval)	Free an attribute key for communicators
<i>Gather</i> (sendbuf, recvbuf[, root])	Gather together values from a group of processes
<i>Gatherv</i> (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
<i>Get_attr</i> (keyval)	Retrieve attribute value by key
<i>Get_errhandler()</i>	Get the error handler for a communicator
<i>Get_group()</i>	Access the group associated with a communicator
<i>Get_info()</i>	Return the hints for a communicator that are currently in use
<i>Get_name()</i>	Get the print name for this communicator
<i>Get_parent()</i>	Return the parent intercommunicator for this process
<i>Get_rank()</i>	Return the rank of this process in a communicator
<i>Get_size()</i>	Return the number of processes in a communicator
<i>Get_topology()</i>	Determine the type of topology (if any) associated with a communicator
<i>Iallgather</i> (sendbuf, recvbuf)	Nonblocking Gather to All
<i>Iallgatherv</i> (sendbuf, recvbuf)	Nonblocking Gather to All Vector
<i>Iallreduce</i> (sendbuf, recvbuf[, op])	Nonblocking Reduce to All
<i>Ialltoall</i> (sendbuf, recvbuf)	Nonblocking All to All Scatter/Gather
<i>Ialltoallv</i> (sendbuf, recvbuf)	Nonblocking All to All Scatter/Gather Vector
<i>Ialltoallw</i> (sendbuf, recvbuf)	Nonblocking Generalized All-to-All
<i>Ibarrier()</i>	Nonblocking Barrier
<i>Ibcast</i> (buf[, root])	Nonblocking Broadcast
<i>Ibsend</i> (buf, dest[, tag])	Nonblocking send in buffered mode
<i>Idup()</i>	Nonblocking duplicate an existing communicator
<i>Igather</i> (sendbuf, recvbuf[, root])	Nonblocking Gather
<i>Igatherv</i> (sendbuf, recvbuf[, root])	Nonblocking Gather Vector
<i>Improbe</i> ([source, tag, status])	Nonblocking test for a matched message
<i>Iprobe</i> ([source, tag, status])	Nonblocking test for a message
<i>Irecv</i> (buf[, source, tag])	Nonblocking receive
<i>Ireduce</i> (sendbuf, recvbuf[, op, root])	Nonblocking Reduce to Root
<i>Ireduce_scatter</i> (sendbuf, recvbuf[, ...])	Nonblocking Reduce-Scatter (vector version)

continues on next page

Table 2 – continued from previous page

<i>Ireduce_scatter_block</i> (sendbuf, recvbuf[, op])	Nonblocking Reduce-Scatter Block (regular, non-vector version)
<i>Irsend</i> (buf, dest[, tag])	Nonblocking send in ready mode
<i>Is_inter</i> ()	Test to see if a comm is an intercommunicator
<i>Is_intra</i> ()	Test to see if a comm is an intracommunicator
<i>Iscatter</i> (sendbuf, recvbuf[, root])	Nonblocking Scatter
<i>Iscatterv</i> (sendbuf, recvbuf[, root])	Nonblocking Scatter Vector
<i>Isend</i> (buf, dest[, tag])	Nonblocking send
<i>Issend</i> (buf, dest[, tag])	Nonblocking send in synchronous mode
<i>Join</i> (fd)	Create a intercommunicator by joining two processes connected by a socket
<i>Mprobe</i> ([source, tag, status])	Blocking test for a matched message
<i>Probe</i> ([source, tag, status])	Blocking test for a message
<i>Recv</i> (buf[, source, tag, status])	Blocking receive
<i>Recv_init</i> (buf[, source, tag])	Create a persistent request for a receive
<i>Reduce</i> (sendbuf, recvbuf[, op, root])	Reduce to Root
<i>Reduce_scatter</i> (sendbuf, recvbuf[, ...])	Reduce-Scatter (vector version)
<i>Reduce_scatter_block</i> (sendbuf, recvbuf[, op])	Reduce-Scatter Block (regular, non-vector version)
<i>Rsend</i> (buf, dest[, tag])	Blocking send in ready mode
<i>Rsend_init</i> (buf, dest[, tag])	Persistent request for a send in ready mode
<i>Scatter</i> (sendbuf, recvbuf[, root])	Scatter data from one process to all other processes in a group
<i>Scatterv</i> (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
<i>Send</i> (buf, dest[, tag])	Blocking send
<i>Send_init</i> (buf, dest[, tag])	Create a persistent request for a standard send
<i>Sendrecv</i> (sendbuf, dest[, sendtag, recvbuf, ...])	Send and receive a message
<i>Sendrecv_replace</i> (buf, dest[, sendtag, ...])	Send and receive a message
<i>Set_attr</i> (keyval, attrval)	Store attribute value associated with a key
<i>Set_errhandler</i> (errhandler)	Set the error handler for a communicator
<i>Set_info</i> (info)	Set new values for the hints associated with a communicator
<i>Set_name</i> (name)	Set the print name for this communicator
<i>Split</i> ([color, key])	Split communicator by color and key
<i>Split_type</i> (split_type[, key, info])	Split communicator by split type
<i>Ssend</i> (buf, dest[, tag])	Blocking send in synchronous mode
<i>Ssend_init</i> (buf, dest[, tag])	Persistent request for a send in synchronous mode
<i>allgather</i> (sendobj)	Gather to All
<i>allreduce</i> (sendobj[, op])	Reduce to All
<i>alltoall</i> (sendobj)	All to All Scatter/Gather
<i>barrier</i> ()	Barrier
<i>bcast</i> (obj[, root])	Broadcast
<i>bsend</i> (obj, dest[, tag])	Send in buffered mode
<i>f2py</i> (arg)	
<i>gather</i> (sendobj[, root])	Gather
<i>ibsend</i> (obj, dest[, tag])	Nonblocking send in buffered mode
<i>improbe</i> ([source, tag, status])	Nonblocking test for a matched message
<i>iprobe</i> ([source, tag, status])	Nonblocking test for a message

continues on next page

Table 2 – continued from previous page

<i>irecv</i> ([buf, source, tag])	Nonblocking receive
<i>isend</i> (obj, dest[, tag])	Nonblocking send
<i>issend</i> (obj, dest[, tag])	Nonblocking send in synchronous mode
<i>mprobe</i> ([source, tag, status])	Blocking test for a matched message
<i>probe</i> ([source, tag, status])	Blocking test for a message
<i>py2f</i> ()	
<i>recv</i> ([buf, source, tag, status])	Receive
<i>reduce</i> (sendobj[, op, root])	Reduce to Root
<i>scatter</i> (sendobj[, root])	Scatter
<i>send</i> (obj, dest[, tag])	Send
<i>sendrecv</i> (sendobj, dest[, sendtag, recvbuf, ...])	Send and Receive
<i>srend</i> (obj, dest[, tag])	Send in synchronous mode

Attributes Summary

<i>group</i>	communicator group
<i>info</i>	communicator info
<i>is_inter</i>	is intercommunicator
<i>is_intra</i>	is intracommunicator
<i>is_topo</i>	is a topology communicator
<i>name</i>	communicator name
<i>rank</i>	rank of this process in communicator
<i>size</i>	number of processes in communicator
<i>topology</i>	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*, *recvcunts=None*, *op=SUM*)

Reduce-Scatter (vector version)

Parameters

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

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

continues on next page

Table 3 – continued from previous page

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

Attributes Summary

<i>combiner</i>	datatype combiner
<i>contents</i>	datatype contents
<i>envelope</i>	datatype envelope
<i>extent</i>	
<i>is_named</i>	is a named datatype
<i>is_predefined</i>	is a predefined datatype
<i>lb</i>	lower bound
<i>name</i>	datatype name
<i>size</i>	
<i>true_extent</i>	true extent
<i>true_lb</i>	true lower bound
<i>true_ub</i>	true upper bound
<i>ub</i>	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, psize, 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]*) –
- **psize** (*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_indexed(*blocklengths, displacements*)

Create an indexed datatype with displacements in bytes

Parameters

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

Return type *Datatype*

Create_indexed_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

<code>Get_dist_neighbors()</code>	Return adjacency information for a distributed graph topology
<code>Get_dist_neighbors_count()</code>	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

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

continues on next page

Table 4 – continued from previous page

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

Attributes Summary

<i>amode</i>	file access mode
<i>atomicity</i>	
<i>group</i>	file group
<i>info</i>	file info
<i>size</i>	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 explicit offset

Parameters

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

Return type None

Read_at_all_end(*buf, status=None*)

Complete a split collective read using explicit 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 [Datype]*) –
- **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 explicit offset

Parameters

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

Return type None

Write_at_all_end(*buf, status=None*)

Complete a split collective write using explicit 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

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

Attributes Summary

<code>dims</code>	number of nodes and edges
<code>edges</code>	
<code>index</code>	
<code>nedges</code>	number of edges
<code>neighbors</code>	
<code>nneighbors</code>	number of neighbors
<code>nnodes</code>	number of nodes
<code>topo</code>	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

<code>Complete()</code>	Notify that a user-defined request is complete
<code>Start(query_fn, free_fn, cancel_fn[, args, ...])</code>	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

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

Attributes Summary

<i>rank</i>	rank of this process in group
<i>size</i>	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

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

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

info update

Parameters

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

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

Attributes Summary

<code>remote_group</code>	remote group
<code>remote_size</code>	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

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

<i>Iprobe</i> (<i>comm</i> [, <i>source</i> , <i>tag</i> , <i>status</i>])	Nonblocking test for a matched message
<i>Irecv</i> (<i>buf</i>)	Nonblocking receive of matched message
<i>Probe</i> (<i>comm</i> [, <i>source</i> , <i>tag</i> , <i>status</i>])	Blocking test for a matched message
<i>Recv</i> (<i>buf</i> [, <i>status</i>])	Blocking receive of matched message
<i>f2py</i> (<i>arg</i>)	
<i>iprobe</i> (<i>comm</i> [, <i>source</i> , <i>tag</i> , <i>status</i>])	Nonblocking test for a matched message
<i>irecv</i> ()	Nonblocking receive of matched message
<i>probe</i> (<i>comm</i> [, <i>source</i> , <i>tag</i> , <i>status</i>])	Blocking test for a matched message
<i>py2f</i> ()	
<i>recv</i> ([<i>status</i>])	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

<code>Create</code> (function[, commute])	Create a user-defined operation
<code>Free</code> ()	Free the operation
<code>Is_commutative</code> ()	Query reduction operations for their commutativity
<code>Reduce_local</code> (inbuf, inoutbuf)	Apply a reduction operator to local data
<code>f2py</code> (arg)	
<code>py2f</code> ()	

Attributes Summary

<code>is_commutative</code>	is commutative
<code>is_predefined</code>	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

<i>dumps</i> (obj[, buffer_callback])	Serialize object to pickle data stream.
<i>loads</i> (data[, buffers])	Deserialize object from pickle data stream.

Attributes Summary

<i>PROTOCOL</i>	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.Prerequest

class `mpi4py.MPI.Prerequest`(*request=None*)

Bases: `mpi4py.MPI.Request`

Persistent request handle

Parameters **request** (*Optional*[`Prerequest`]) –

Return type `Prerequest`

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

Parameters **request** (*Optional*[`Prerequest`]) –

Return type `Prerequest`

Methods Summary

<code>Start()</code>	Initiate a communication with a persistent request
<code>Startall</code> (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* [[Prerequest](#)]) –

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

<i>Cancel()</i>	Cancel a communication request
<i>Free()</i>	Free a communication request
<i>Get_status</i> ([status])	Non-destructive test for the completion of a request
<i>Test</i> ([status])	Test for the completion of a send or receive
<i>Testall</i> (requests[, statuses])	Test for completion of all previously initiated requests
<i>Testany</i> (requests[, status])	Test for completion of any previously initiated request
<i>Testsome</i> (requests[, statuses])	Test for completion of some previously initiated requests
<i>Wait</i> ([status])	Wait for a send or receive to complete
<i>Waitall</i> (requests[, statuses])	Wait for all previously initiated requests to complete
<i>Waitany</i> (requests[, status])	Wait for any previously initiated request to complete
<i>Waitsome</i> (requests[, statuses])	Wait for some previously initiated requests to complete
<i>cancel()</i>	Cancel a communication request
<i>f2py</i> (arg)	
<i>get_status</i> ([status])	Non-destructive test for the completion of a request
<i>py2f</i> ()	
<i>test</i> ([status])	Test for the completion of a send or receive
<i>testall</i> (requests[, statuses])	Test for completion of all previously initiated requests
<i>testany</i> (requests[, status])	Test for completion of any previously initiated request
<i>testsome</i> (requests[, statuses])	Test for completion of some previously initiated requests
<i>wait</i> ([status])	Wait for a send or receive to complete
<i>waitall</i> (requests[, statuses])	Wait for all previously initiated requests to complete
<i>waitany</i> (requests[, status])	Wait for any previously initiated request to complete
<i>waitsome</i> (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

<code>Get_count</code> ([datatype])	Get the number of <i>top level</i> elements
<code>Get_elements</code> (datatype)	Get the number of basic elements in a datatype
<code>Get_error</code> ()	Get message error
<code>Get_source</code> ()	Get message source
<code>Get_tag</code> ()	Get message tag
<code>Is_cancelled</code> ()	Test to see if a request was cancelled
<code>Set_cancelled</code> (flag)	Set the cancelled state associated with a status
<code>Set_elements</code> (datatype, count)	Set the number of elements in a status
<code>Set_error</code> (error)	Set message error
<code>Set_source</code> (source)	Set message source
<code>Set_tag</code> (tag)	Set message tag
<code>f2py</code> (arg)	
<code>py2f</code> ()	

Attributes Summary

<code>cancelled</code>	cancelled state
<code>count</code>	byte count
<code>error</code>	
<code>source</code>	
<code>tag</code>	

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

<i>Ineighbor_allgather</i> (sendbuf, recvbuf)	Nonblocking Neighbor Gather to All
<i>Ineighbor_allgatherv</i> (sendbuf, recvbuf)	Nonblocking Neighbor Gather to All Vector
<i>Ineighbor_alltoall</i> (sendbuf, recvbuf)	Nonblocking Neighbor All-to-All
<i>Ineighbor_alltoallv</i> (sendbuf, recvbuf)	Nonblocking Neighbor All-to-All Vector
<i>Ineighbor_alltoallw</i> (sendbuf, recvbuf)	Nonblocking Neighbor All-to-All Generalized
<i>Neighbor_allgather</i> (sendbuf, recvbuf)	Neighbor Gather to All
<i>Neighbor_allgatherv</i> (sendbuf, recvbuf)	Neighbor Gather to All Vector
<i>Neighbor_alltoall</i> (sendbuf, recvbuf)	Neighbor All-to-All
<i>Neighbor_alltoallv</i> (sendbuf, recvbuf)	Neighbor All-to-All Vector
<i>Neighbor_alltoallw</i> (sendbuf, recvbuf)	Neighbor All-to-All Generalized
<i>neighbor_allgather</i> (sendobj)	Neighbor Gather to All
<i>neighbor_alltoall</i> (sendobj)	Neighbor All to All Scatter/Gather

Attributes Summary

<i>degrees</i>	number of incoming and outgoing neighbors
<i>indegree</i>	number of incoming neighbors
<i>inedges</i>	incoming neighbors
<i>inoutedges</i>	incoming and outgoing neighbors
<i>outdegree</i>	number of outgoing neighbors
<i>outedges</i>	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

<i>Accumulate</i> (origin, target_rank[, target, op])	Accumulate data into the target process
<i>Allocate</i> (size[, disp_unit, info, comm])	Create an window object for one-sided communication
<i>Allocate_shared</i> (size[, disp_unit, info, comm])	Create an window object for one-sided communication
<i>Attach</i> (memory)	Attach a local memory region
<i>Call_errhandler</i> (errorcode)	Call the error handler installed on a window
<i>Compare_and_swap</i> (origin, compare, result, ...)	Perform one-sided atomic compare-and-swap
<i>Complete</i> ()	Completes an RMA operations begun after an <i>Win.Start()</i>
<i>Create</i> (memory[, disp_unit, info, comm])	Create an window object for one-sided communication
<i>Create_dynamic</i> ([info, comm])	Create an window object for one-sided communication
<i>Create_keyval</i> ([copy_fn, delete_fn, nopython])	Create a new attribute key for windows
<i>Delete_attr</i> (keyval)	Delete attribute value associated with a key
<i>Detach</i> (memory)	Detach a local memory region
<i>Fence</i> ([assertion])	Perform an MPI fence synchronization on a window
<i>Fetch_and_op</i> (origin, result, target_rank[, ...])	Perform one-sided read-modify-write
<i>Flush</i> (rank)	Complete all outstanding RMA operations at the given target
<i>Flush_all</i> ()	Complete all outstanding RMA operations at all targets
<i>Flush_local</i> (rank)	Complete locally all outstanding RMA operations at the given target
<i>Flush_local_all</i> ()	Complete locally all outstanding RMA operations at all targets
<i>Free</i> ()	Free a window
<i>Free_keyval</i> (keyval)	Free an attribute key for windows
<i>Get</i> (origin, target_rank[, target])	Get data from a memory window on a remote process.
<i>Get_accumulate</i> (origin, result, target_rank)	Fetch-and-accumulate data into the target process
<i>Get_attr</i> (keyval)	Retrieve attribute value by key
<i>Get_errhandler</i> ()	Get the error handler for a window
<i>Get_group</i> ()	Return a duplicate of the group of the communicator used to create the window
<i>Get_info</i> ()	Return the hints for a windows that are currently in use
<i>Get_name</i> ()	Get the print name associated with the window
<i>Lock</i> (rank[, lock_type, assertion])	Begin an RMA access epoch at the target process
<i>Lock_all</i> ([assertion])	Begin an RMA access epoch at all processes
<i>Post</i> (group[, assertion])	Start an RMA exposure epoch
<i>Put</i> (origin, target_rank[, target])	Put data into a memory window on a remote process.
<i>Raccumulate</i> (origin, target_rank[, target, op])	Fetch-and-accumulate data into the target process
<i>Rget</i> (origin, target_rank[, target])	Get data from a memory window on a remote process.
<i>Rget_accumulate</i> (origin, result, target_rank)	Accumulate data into the target process using remote memory access.
<i>Rput</i> (origin, target_rank[, target])	Put data into a memory window on a remote process.

continues on next page

Table 5 – continued from previous page

<i>Set_attr</i> (keyval, attrval)	Store attribute value associated with a key
<i>Set_errhandler</i> (errhandler)	Set the error handler for a window
<i>Set_info</i> (info)	Set new values for the hints associated with a window
<i>Set_name</i> (name)	Set the print name associated with the window
<i>Shared_query</i> (rank)	Query the process-local address for remote memory segments created with <i>Win.Allocate_shared()</i>
<i>Start</i> (group[, assertion])	Start an RMA access epoch for MPI
<i>Sync</i> ()	Synchronize public and private copies of the given window
<i>Test</i> ()	Test whether an RMA exposure epoch has completed
<i>Unlock</i> (rank)	Complete an RMA access epoch at the target process
<i>Unlock_all</i> ()	Complete an RMA access epoch at all processes
<i>Wait</i> ()	Complete an RMA exposure epoch begun with <i>Win.Post()</i>
<i>f2py</i> (arg)	
<i>py2f</i> ()	
<i>tomemory</i> ()	Return window memory buffer

Attributes Summary

<i>attrs</i>	window attributes
<i>flavor</i>	window create flavor
<i>group</i>	window group
<i>info</i>	window info
<i>model</i>	window memory model
<i>name</i>	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 operations 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

<code>allocate(nbytes[, clear])</code>	Memory allocation
<code>fromaddress(address, nbytes[, readonly])</code>	Memory from address and size in bytes
<code>frombuffer(obj[, readonly])</code>	Memory from buffer-like object
<code>release()</code>	Release the underlying buffer exposed by the memory object
<code>tobytes([order])</code>	Return the data in the buffer as a byte string
<code>toreadonly()</code>	Return a readonly version of the memory object

Attributes Summary

<code>address</code>	Memory address
<code>format</code>	A string with the format of each element
<code>itemsize</code>	The size in bytes of each element
<code>nbytes</code>	Memory size (in bytes)
<code>obj</code>	The underlying object of the memory
<code>readonly</code>	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

<i>Exception</i> (<i>ierr</i>)	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

<code>Get_error_class()</code>	Error class
<code>Get_error_code()</code>	Error code
<code>Get_error_string()</code>	Error string

Attributes Summary

<code>error_class</code>	error class
<code>error_code</code>	error code
<code>error_string</code>	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

<code>Add_error_class()</code>	Add an <i>error class</i> to the known error classes
<code>Add_error_code(errorclass)</code>	Add an <i>error code</i> to an <i>error class</i>
<code>Add_error_string(errorcode, string)</code>	Associate an <i>error string</i> with an <i>error class</i> or <i>error-code</i>
<code>Aint_add(base, disp)</code>	Return the sum of base address and displacement
<code>Aint_diff(addr1, addr2)</code>	Return the difference between absolute addresses
<code>Alloc_mem(size[, info])</code>	Allocate memory for message passing and RMA

continues on next page

Table 6 – continued from previous page

<code>Attach_buffer(buf)</code>	Attach a user-provided buffer for sending in buffered mode
<code>Close_port(port_name)</code>	Close a port
<code>Compute_dims(nnodes, dims)</code>	Return a balanced distribution of processes per coordinate direction
<code>Detach_buffer()</code>	Remove an existing attached buffer
<code>Finalize()</code>	Terminate the MPI execution environment
<code>Free_mem(mem)</code>	Free memory allocated with <code>Alloc_mem()</code>
<code>Get_address(location)</code>	Get the address of a location in memory
<code>Get_error_class(errorcode)</code>	Convert an <i>error code</i> into an <i>error class</i>
<code>Get_error_string(errorcode)</code>	Return the <i>error string</i> for a given <i>error class</i> or <i>error code</i>
<code>Get_library_version()</code>	Obtain the version string of the MPI library
<code>Get_processor_name()</code>	Obtain the name of the calling processor
<code>Get_version()</code>	Obtain the version number of the MPI standard supported by the implementation as a tuple (<i>version</i> , <i>subversion</i>)
<code>Init()</code>	Initialize the MPI execution environment
<code>Init_thread([required])</code>	Initialize the MPI execution environment
<code>Is_finalized()</code>	Indicates whether <code>Finalize</code> has completed
<code>Is_initialized()</code>	Indicates whether <code>Init</code> has been called
<code>Is_thread_main()</code>	Indicate whether this thread called <code>Init</code> or <code>Init_thread</code>
<code>Lookup_name(service_name[, info])</code>	Lookup a port name given a service name
<code>Open_port([info])</code>	Return an address that can be used to establish connections between groups of MPI processes
<code>Pcontrol(level)</code>	Control profiling
<code>Publish_name(service_name, port_name[, info])</code>	Publish a service name
<code>Query_thread()</code>	Return the level of thread support provided by the MPI library
<code>Register_datarep(datarep, read_fn, write_fn, ...)</code>	Register user-defined data representations
<code>Unpublish_name(service_name, port_name[, info])</code>	Unpublish a service name
<code>Wtick()</code>	Return the resolution of <code>Wtime</code>
<code>Wtime()</code>	Return an elapsed time on the calling processor
<code>get_vendor()</code>	Information 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()

Information 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

<i>UNDEFINED</i>	int UNDEFINED
<i>ANY_SOURCE</i>	int ANY_SOURCE
<i>ANY_TAG</i>	int ANY_TAG
<i>PROC_NULL</i>	int PROC_NULL
<i>ROOT</i>	int ROOT
<i>BOTTOM</i>	Bottom BOTTOM
<i>IN_PLACE</i>	InPlace IN_PLACE
<i>KEYVAL_INVALID</i>	int KEYVAL_INVALID
<i>TAG_UB</i>	int TAG_UB
<i>HOST</i>	int HOST
<i>IO</i>	int IO
<i>WTIME_IS_GLOBAL</i>	int WTIME_IS_GLOBAL
<i>UNIVERSE_SIZE</i>	int UNIVERSE_SIZE
<i>APPNUM</i>	int APPNUM
<i>LASTUSED</i>	int LASTUSED
<i>WIN_BASE</i>	int WIN_BASE
<i>WIN_SIZE</i>	int WIN_SIZE
<i>WIN_DISP_UNIT</i>	int WIN_DISP_UNIT
<i>WIN_CREATE_FLAVOR</i>	int WIN_CREATE_FLAVOR
<i>WIN_FLAVOR</i>	int WIN_FLAVOR
<i>WIN_MODEL</i>	int WIN_MODEL
<i>SUCCESS</i>	int SUCCESS
<i>ERR_LASTCODE</i>	int ERR_LASTCODE
<i>ERR_COMM</i>	int ERR_COMM
<i>ERR_GROUP</i>	int ERR_GROUP
<i>ERR_TYPE</i>	int ERR_TYPE
<i>ERR_REQUEST</i>	int ERR_REQUEST
<i>ERR_OP</i>	int ERR_OP
<i>ERR_BUFFER</i>	int ERR_BUFFER
<i>ERR_COUNT</i>	int ERR_COUNT
<i>ERR_TAG</i>	int ERR_TAG
<i>ERR_RANK</i>	int ERR_RANK
<i>ERR_ROOT</i>	int ERR_ROOT
<i>ERR_TRUNCATE</i>	int ERR_TRUNCATE
<i>ERR_IN_STATUS</i>	int ERR_IN_STATUS
<i>ERR_PENDING</i>	int ERR_PENDING
<i>ERR_TOPOLOGY</i>	int ERR_TOPOLOGY

continues on next page

Table 7 – continued from previous page

<i>ERR_DIMS</i>	int ERR_DIMS
<i>ERR_ARG</i>	int ERR_ARG
<i>ERR_OTHER</i>	int ERR_OTHER
<i>ERR_UNKNOWN</i>	int ERR_UNKNOWN
<i>ERR_INTERN</i>	int ERR_INTERN
<i>ERR_INFO</i>	int ERR_INFO
<i>ERR_FILE</i>	int ERR_FILE
<i>ERR_WIN</i>	int ERR_WIN
<i>ERR_KEYVAL</i>	int ERR_KEYVAL
<i>ERR_INFO_KEY</i>	int ERR_INFO_KEY
<i>ERR_INFO_VALUE</i>	int ERR_INFO_VALUE
<i>ERR_INFO_NOKEY</i>	int ERR_INFO_NOKEY
<i>ERR_ACCESS</i>	int ERR_ACCESS
<i>ERR_AMODE</i>	int ERR_AMODE
<i>ERR_BAD_FILE</i>	int ERR_BAD_FILE
<i>ERR_FILE_EXISTS</i>	int ERR_FILE_EXISTS
<i>ERR_FILE_IN_USE</i>	int ERR_FILE_IN_USE
<i>ERR_NO_SPACE</i>	int ERR_NO_SPACE
<i>ERR_NO_SUCH_FILE</i>	int ERR_NO_SUCH_FILE
<i>ERR_IO</i>	int ERR_IO
<i>ERR_READ_ONLY</i>	int ERR_READ_ONLY
<i>ERR_CONVERSION</i>	int ERR_CONVERSION
<i>ERR_DUP_DATAREP</i>	int ERR_DUP_DATAREP
<i>ERR_UNSUPPORTED_DATAREP</i>	int ERR_UNSUPPORTED_DATAREP
<i>ERR_UNSUPPORTED_OPERATION</i>	int ERR_UNSUPPORTED_OPERATION
<i>ERR_NAME</i>	int ERR_NAME
<i>ERR_NO_MEM</i>	int ERR_NO_MEM
<i>ERR_NOT_SAME</i>	int ERR_NOT_SAME
<i>ERR_PORT</i>	int ERR_PORT
<i>ERR_QUOTA</i>	int ERR_QUOTA
<i>ERR_SERVICE</i>	int ERR_SERVICE
<i>ERR_SPAWN</i>	int ERR_SPAWN
<i>ERR_BASE</i>	int ERR_BASE
<i>ERR_SIZE</i>	int ERR_SIZE
<i>ERR_DISP</i>	int ERR_DISP
<i>ERR_ASSERT</i>	int ERR_ASSERT
<i>ERR_LOCKTYPE</i>	int ERR_LOCKTYPE
<i>ERR_RMA_CONFLICT</i>	int ERR_RMA_CONFLICT
<i>ERR_RMA_SYNC</i>	int ERR_RMA_SYNC
<i>ERR_RMA_RANGE</i>	int ERR_RMA_RANGE
<i>ERR_RMA_ATTACH</i>	int ERR_RMA_ATTACH
<i>ERR_RMA_SHARED</i>	int ERR_RMA_SHARED
<i>ERR_RMA_FLAVOR</i>	int ERR_RMA_FLAVOR
<i>ORDER_C</i>	int ORDER_C
<i>ORDER_FORTRAN</i>	int ORDER_FORTRAN
<i>ORDER_F</i>	int ORDER_F
<i>TYPECLASS_INTEGER</i>	int TYPECLASS_INTEGER
<i>TYPECLASS_REAL</i>	int TYPECLASS_REAL
<i>TYPECLASS_COMPLEX</i>	int TYPECLASS_COMPLEX
<i>DISTRIBUTE_NONE</i>	int DISTRIBUTE_NONE

continues on next page

Table 7 – continued from previous page

<i>DISTRIBUTE_BLOCK</i>	int DISTRIBUTE_BLOCK
<i>DISTRIBUTE_CYCLIC</i>	int DISTRIBUTE_CYCLIC
<i>DISTRIBUTE_DFLT_DARG</i>	int DISTRIBUTE_DFLT_DARG
<i>COMBINER_NAMED</i>	int COMBINER_NAMED
<i>COMBINER_DUP</i>	int COMBINER_DUP
<i>COMBINER_CONTIGUOUS</i>	int COMBINER_CONTIGUOUS
<i>COMBINER_VECTOR</i>	int COMBINER_VECTOR
<i>COMBINER_HVECTOR</i>	int COMBINER_HVECTOR
<i>COMBINER_INDEXED</i>	int COMBINER_INDEXED
<i>COMBINER_HINDEXED</i>	int COMBINER_HINDEXED
<i>COMBINER_INDEXED_BLOCK</i>	int COMBINER_INDEXED_BLOCK
<i>COMBINER_HINDEXED_BLOCK</i>	int COMBINER_HINDEXED_BLOCK
<i>COMBINER_STRUCT</i>	int COMBINER_STRUCT
<i>COMBINER_SUBARRAY</i>	int COMBINER_SUBARRAY
<i>COMBINER_DARRAY</i>	int COMBINER_DARRAY
<i>COMBINER_RESIZED</i>	int COMBINER_RESIZED
<i>COMBINER_F90_REAL</i>	int COMBINER_F90_REAL
<i>COMBINER_F90_COMPLEX</i>	int COMBINER_F90_COMPLEX
<i>COMBINER_F90_INTEGER</i>	int COMBINER_F90_INTEGER
<i>IDENT</i>	int IDENT
<i>CONGRUENT</i>	int CONGRUENT
<i>SIMILAR</i>	int SIMILAR
<i>UNEQUAL</i>	int UNEQUAL
<i>CART</i>	int CART
<i>GRAPH</i>	int GRAPH
<i>DIST_GRAPH</i>	int DIST_GRAPH
<i>UNWEIGHTED</i>	int UNWEIGHTED
<i>WEIGHTS_EMPTY</i>	int WEIGHTS_EMPTY
<i>COMM_TYPE_SHARED</i>	int COMM_TYPE_SHARED
<i>BSEND_OVERHEAD</i>	int BSEND_OVERHEAD
<i>WIN_FLAVOR_CREATE</i>	int WIN_FLAVOR_CREATE
<i>WIN_FLAVOR_ALLOCATE</i>	int WIN_FLAVOR_ALLOCATE
<i>WIN_FLAVOR_DYNAMIC</i>	int WIN_FLAVOR_DYNAMIC
<i>WIN_FLAVOR_SHARED</i>	int WIN_FLAVOR_SHARED
<i>WIN_SEPARATE</i>	int WIN_SEPARATE
<i>WIN_UNIFIED</i>	int WIN_UNIFIED
<i>MODE_NOCHECK</i>	int MODE_NOCHECK
<i>MODE_NOSTORE</i>	int MODE_NOSTORE
<i>MODE_NOPUT</i>	int MODE_NOPUT
<i>MODE_NOPRECEDE</i>	int MODE_NOPRECEDE
<i>MODE_NOSUCCEED</i>	int MODE_NOSUCCEED
<i>LOCK_EXCLUSIVE</i>	int LOCK_EXCLUSIVE
<i>LOCK_SHARED</i>	int LOCK_SHARED
<i>MODE_RDONLY</i>	int MODE_RDONLY
<i>MODE_WRONLY</i>	int MODE_WRONLY
<i>MODE_RDWR</i>	int MODE_RDWR
<i>MODE_CREATE</i>	int MODE_CREATE
<i>MODE_EXCL</i>	int MODE_EXCL
<i>MODE_DELETE_ON_CLOSE</i>	int MODE_DELETE_ON_CLOSE
<i>MODE_UNIQUE_OPEN</i>	int MODE_UNIQUE_OPEN

continues on next page

Table 7 – continued from previous page

<i>MODE_SEQUENTIAL</i>	<i>int</i> MODE_SEQUENTIAL
<i>MODE_APPEND</i>	<i>int</i> MODE_APPEND
<i>SEEK_SET</i>	<i>int</i> SEEK_SET
<i>SEEK_CUR</i>	<i>int</i> SEEK_CUR
<i>SEEK_END</i>	<i>int</i> SEEK_END
<i>DISPLACEMENT_CURRENT</i>	<i>int</i> DISPLACEMENT_CURRENT
<i>DISP_CUR</i>	<i>int</i> DISP_CUR
<i>THREAD_SINGLE</i>	<i>int</i> THREAD_SINGLE
<i>THREAD_FUNNELED</i>	<i>int</i> THREAD_FUNNELED
<i>THREAD_SERIALIZED</i>	<i>int</i> THREAD_SERIALIZED
<i>THREAD_MULTIPLE</i>	<i>int</i> THREAD_MULTIPLE
<i>VERSION</i>	<i>int</i> VERSION
<i>SUBVERSION</i>	<i>int</i> SUBVERSION
<i>MAX_PROCESSOR_NAME</i>	<i>int</i> MAX_PROCESSOR_NAME
<i>MAX_ERROR_STRING</i>	<i>int</i> MAX_ERROR_STRING
<i>MAX_PORT_NAME</i>	<i>int</i> MAX_PORT_NAME
<i>MAX_INFO_KEY</i>	<i>int</i> MAX_INFO_KEY
<i>MAX_INFO_VAL</i>	<i>int</i> MAX_INFO_VAL
<i>MAX_OBJECT_NAME</i>	<i>int</i> MAX_OBJECT_NAME
<i>MAX_DATAREP_STRING</i>	<i>int</i> MAX_DATAREP_STRING
<i>MAX_LIBRARY_VERSION_STRING</i>	<i>int</i> MAX_LIBRARY_VERSION_STRING
<i>DATATYPE_NULL</i>	<i>Datatype</i> DATATYPE_NULL
<i>UB</i>	<i>Datatype</i> UB
<i>LB</i>	<i>Datatype</i> LB
<i>PACKED</i>	<i>Datatype</i> PACKED
<i>BYTE</i>	<i>Datatype</i> BYTE
<i>AINT</i>	<i>Datatype</i> AINT
<i>OFFSET</i>	<i>Datatype</i> OFFSET
<i>COUNT</i>	<i>Datatype</i> COUNT
<i>CHAR</i>	<i>Datatype</i> CHAR
<i>WCHAR</i>	<i>Datatype</i> WCHAR
<i>SIGNED_CHAR</i>	<i>Datatype</i> SIGNED_CHAR
<i>SHORT</i>	<i>Datatype</i> SHORT
<i>INT</i>	<i>Datatype</i> INT
<i>LONG</i>	<i>Datatype</i> LONG
<i>LONG_LONG</i>	<i>Datatype</i> LONG_LONG
<i>UNSIGNED_CHAR</i>	<i>Datatype</i> UNSIGNED_CHAR
<i>UNSIGNED_SHORT</i>	<i>Datatype</i> UNSIGNED_SHORT
<i>UNSIGNED</i>	<i>Datatype</i> UNSIGNED
<i>UNSIGNED_LONG</i>	<i>Datatype</i> UNSIGNED_LONG
<i>UNSIGNED_LONG_LONG</i>	<i>Datatype</i> UNSIGNED_LONG_LONG
<i>FLOAT</i>	<i>Datatype</i> FLOAT
<i>DOUBLE</i>	<i>Datatype</i> DOUBLE
<i>LONG_DOUBLE</i>	<i>Datatype</i> LONG_DOUBLE
<i>C_BOOL</i>	<i>Datatype</i> C_BOOL
<i>INT8_T</i>	<i>Datatype</i> INT8_T
<i>INT16_T</i>	<i>Datatype</i> INT16_T
<i>INT32_T</i>	<i>Datatype</i> INT32_T
<i>INT64_T</i>	<i>Datatype</i> INT64_T
<i>UINT8_T</i>	<i>Datatype</i> UINT8_T

continues on next page

Table 7 – continued from previous page

<i>UINT16_T</i>	<i>Datatype</i> <i>UINT16_T</i>
<i>UINT32_T</i>	<i>Datatype</i> <i>UINT32_T</i>
<i>UINT64_T</i>	<i>Datatype</i> <i>UINT64_T</i>
<i>C_COMPLEX</i>	<i>Datatype</i> <i>C_COMPLEX</i>
<i>C_FLOAT_COMPLEX</i>	<i>Datatype</i> <i>C_FLOAT_COMPLEX</i>
<i>C_DOUBLE_COMPLEX</i>	<i>Datatype</i> <i>C_DOUBLE_COMPLEX</i>
<i>C_LONG_DOUBLE_COMPLEX</i>	<i>Datatype</i> <i>C_LONG_DOUBLE_COMPLEX</i>
<i>CXX_BOOL</i>	<i>Datatype</i> <i>CXX_BOOL</i>
<i>CXX_FLOAT_COMPLEX</i>	<i>Datatype</i> <i>CXX_FLOAT_COMPLEX</i>
<i>CXX_DOUBLE_COMPLEX</i>	<i>Datatype</i> <i>CXX_DOUBLE_COMPLEX</i>
<i>CXX_LONG_DOUBLE_COMPLEX</i>	<i>Datatype</i> <i>CXX_LONG_DOUBLE_COMPLEX</i>
<i>SHORT_INT</i>	<i>Datatype</i> <i>SHORT_INT</i>
<i>INT_INT</i>	<i>Datatype</i> <i>INT_INT</i>
<i>TWOINT</i>	<i>Datatype</i> <i>TWOINT</i>
<i>LONG_INT</i>	<i>Datatype</i> <i>LONG_INT</i>
<i>FLOAT_INT</i>	<i>Datatype</i> <i>FLOAT_INT</i>
<i>DOUBLE_INT</i>	<i>Datatype</i> <i>DOUBLE_INT</i>
<i>LONG_DOUBLE_INT</i>	<i>Datatype</i> <i>LONG_DOUBLE_INT</i>
<i>CHARACTER</i>	<i>Datatype</i> <i>CHARACTER</i>
<i>LOGICAL</i>	<i>Datatype</i> <i>LOGICAL</i>
<i>INTEGER</i>	<i>Datatype</i> <i>INTEGER</i>
<i>REAL</i>	<i>Datatype</i> <i>REAL</i>
<i>DOUBLE_PRECISION</i>	<i>Datatype</i> <i>DOUBLE_PRECISION</i>
<i>COMPLEX</i>	<i>Datatype</i> <i>COMPLEX</i>
<i>DOUBLE_COMPLEX</i>	<i>Datatype</i> <i>DOUBLE_COMPLEX</i>
<i>LOGICAL1</i>	<i>Datatype</i> <i>LOGICAL1</i>
<i>LOGICAL2</i>	<i>Datatype</i> <i>LOGICAL2</i>
<i>LOGICAL4</i>	<i>Datatype</i> <i>LOGICAL4</i>
<i>LOGICAL8</i>	<i>Datatype</i> <i>LOGICAL8</i>
<i>INTEGER1</i>	<i>Datatype</i> <i>INTEGER1</i>
<i>INTEGER2</i>	<i>Datatype</i> <i>INTEGER2</i>
<i>INTEGER4</i>	<i>Datatype</i> <i>INTEGER4</i>
<i>INTEGER8</i>	<i>Datatype</i> <i>INTEGER8</i>
<i>INTEGER16</i>	<i>Datatype</i> <i>INTEGER16</i>
<i>REAL2</i>	<i>Datatype</i> <i>REAL2</i>
<i>REAL4</i>	<i>Datatype</i> <i>REAL4</i>
<i>REAL8</i>	<i>Datatype</i> <i>REAL8</i>
<i>REAL16</i>	<i>Datatype</i> <i>REAL16</i>
<i>COMPLEX4</i>	<i>Datatype</i> <i>COMPLEX4</i>
<i>COMPLEX8</i>	<i>Datatype</i> <i>COMPLEX8</i>
<i>COMPLEX16</i>	<i>Datatype</i> <i>COMPLEX16</i>
<i>COMPLEX32</i>	<i>Datatype</i> <i>COMPLEX32</i>
<i>UNSIGNED_INT</i>	<i>Datatype</i> <i>UNSIGNED_INT</i>
<i>SIGNED_SHORT</i>	<i>Datatype</i> <i>SIGNED_SHORT</i>
<i>SIGNED_INT</i>	<i>Datatype</i> <i>SIGNED_INT</i>
<i>SIGNED_LONG</i>	<i>Datatype</i> <i>SIGNED_LONG</i>
<i>SIGNED_LONG_LONG</i>	<i>Datatype</i> <i>SIGNED_LONG_LONG</i>
<i>BOOL</i>	<i>Datatype</i> <i>BOOL</i>
<i>SINT8_T</i>	<i>Datatype</i> <i>SINT8_T</i>
<i>SINT16_T</i>	<i>Datatype</i> <i>SINT16_T</i>

continues on next page

Table 7 – continued from previous page

<i>SINT32_T</i>	<i>Datatype</i> SINT32_T
<i>SINT64_T</i>	<i>Datatype</i> SINT64_T
<i>F_BOOL</i>	<i>Datatype</i> F_BOOL
<i>F_INT</i>	<i>Datatype</i> F_INT
<i>F_FLOAT</i>	<i>Datatype</i> F_FLOAT
<i>F_DOUBLE</i>	<i>Datatype</i> F_DOUBLE
<i>F_COMPLEX</i>	<i>Datatype</i> F_COMPLEX
<i>F_FLOAT_COMPLEX</i>	<i>Datatype</i> F_FLOAT_COMPLEX
<i>F_DOUBLE_COMPLEX</i>	<i>Datatype</i> F_DOUBLE_COMPLEX
<i>REQUEST_NULL</i>	<i>Request</i> REQUEST_NULL
<i>MESSAGE_NULL</i>	<i>Message</i> MESSAGE_NULL
<i>MESSAGE_NO_PROC</i>	<i>Message</i> MESSAGE_NO_PROC
<i>OP_NULL</i>	<i>Op</i> OP_NULL
<i>MAX</i>	<i>Op</i> MAX
<i>MIN</i>	<i>Op</i> MIN
<i>SUM</i>	<i>Op</i> SUM
<i>PROD</i>	<i>Op</i> PROD
<i>LAND</i>	<i>Op</i> LAND
<i>BAND</i>	<i>Op</i> BAND
<i>LOR</i>	<i>Op</i> LOR
<i>BOR</i>	<i>Op</i> BOR
<i>LXOR</i>	<i>Op</i> LXOR
<i>BXOR</i>	<i>Op</i> BXOR
<i>MAXLOC</i>	<i>Op</i> MAXLOC
<i>MINLOC</i>	<i>Op</i> MINLOC
<i>REPLACE</i>	<i>Op</i> REPLACE
<i>NO_OP</i>	<i>Op</i> NO_OP
<i>GROUP_NULL</i>	<i>Group</i> GROUP_NULL
<i>GROUP_EMPTY</i>	<i>Group</i> GROUP_EMPTY
<i>INFO_NULL</i>	<i>Info</i> INFO_NULL
<i>INFO_ENV</i>	<i>Info</i> INFO_ENV
<i>ERRHANDLER_NULL</i>	<i>Errhandler</i> ERRHANDLER_NULL
<i>ERRORS_RETURN</i>	<i>Errhandler</i> ERRORS_RETURN
<i>ERRORS_ARE_FATAL</i>	<i>Errhandler</i> ERRORS_ARE_FATAL
<i>COMM_NULL</i>	<i>Comm</i> COMM_NULL
<i>COMM_SELF</i>	<i>Intracomm</i> COMM_SELF
<i>COMM_WORLD</i>	<i>Intracomm</i> COMM_WORLD
<i>WIN_NULL</i>	<i>Win</i> WIN_NULL
<i>FILE_NULL</i>	<i>File</i> FILE_NULL
<i>pickle</i>	<i>Pickle</i> 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.LASTUSEDPCODE

```
mpi4py.MPI.LASTUSEDPCODE: int = LASTUSEDPCODE  
    int LASTUSEDPCODE
```

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 previously 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 = /opt/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 or 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:

```
#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

- [mpi4py](#), 20
- [mpi4py.futures](#), 33
- [mpi4py.MPI](#), 47
- [mpi4py.run](#), 46
- [mpi4py.util](#), 40
- [mpi4py.util.dtlb](#), 46
- [mpi4py.util.pkl5](#), 40

Index

Symbols

`__init__()` (*mpi4py.MPI.Pickle* method), 110
`__new__()` (*mpi4py.MPI.Cartcomm* static method), 48
`__new__()` (*mpi4py.MPI.Comm* static method), 50
`__new__()` (*mpi4py.MPI.Datatype* static method), 72
`__new__()` (*mpi4py.MPI.Distgraphcomm* static method), 80
`__new__()` (*mpi4py.MPI.Errhandler* static method), 81
`__new__()` (*mpi4py.MPI.Exception* static method), 133
`__new__()` (*mpi4py.MPI.File* static method), 81
`__new__()` (*mpi4py.MPI.Graphcomm* static method), 92
`__new__()` (*mpi4py.MPI.Grequest* static method), 93
`__new__()` (*mpi4py.MPI.Group* static method), 94
`__new__()` (*mpi4py.MPI.Info* static method), 97
`__new__()` (*mpi4py.MPI.Intercomm* static method), 100
`__new__()` (*mpi4py.MPI.Intracomm* static method), 101
`__new__()` (*mpi4py.MPI.Message* static method), 106
`__new__()` (*mpi4py.MPI.Op* static method), 108
`__new__()` (*mpi4py.MPI.Prequest* static method), 111
`__new__()` (*mpi4py.MPI.Request* static method), 112
`__new__()` (*mpi4py.MPI.Status* static method), 116
`__new__()` (*mpi4py.MPI.Topocomm* static method), 119
`__new__()` (*mpi4py.MPI.Win* static method), 122
`__new__()` (*mpi4py.MPI.memory* static method), 131
`--configure`
 command line option, 185
`--mpi`
 command line option, 185
`--mpicc`
 command line option, 185
`-c`
 command line option, 47
`-m`
 command line option, 47

A

`Abort()` (*mpi4py.MPI.Comm* method), 53
`Accept()` (*mpi4py.MPI.Intracomm* method), 102
`Accumulate()` (*mpi4py.MPI.Win* method), 124
`Add_error_class()` (in module *mpi4py.MPI*), 135
`Add_error_code()` (in module *mpi4py.MPI*), 136
`Add_error_string()` (in module *mpi4py.MPI*), 136
`address` (*mpi4py.MPI.memory* attribute), 133
`AINT` (in module *mpi4py.MPI*), 168
`Aint_add()` (in module *mpi4py.MPI*), 136
`Aint_diff()` (in module *mpi4py.MPI*), 136
`Allgather()` (*mpi4py.MPI.Comm* method), 53
`allgather()` (*mpi4py.MPI.Comm* method), 67
`Allgatherv()` (*mpi4py.MPI.Comm* method), 53
`Alloc_mem()` (in module *mpi4py.MPI*), 137

`allocate()` (*mpi4py.MPI.memory* static method), 132
`Allocate()` (*mpi4py.MPI.Win* class method), 124
`Allocate_shared()` (*mpi4py.MPI.Win* class method), 125
`Allreduce()` (*mpi4py.MPI.Comm* method), 54
`allreduce()` (*mpi4py.MPI.Comm* method), 67
`Alltoall()` (*mpi4py.MPI.Comm* method), 54
`alltoall()` (*mpi4py.MPI.Comm* method), 67
`Alltoallv()` (*mpi4py.MPI.Comm* method), 54
`Alltoallw()` (*mpi4py.MPI.Comm* method), 54
`amode` (*mpi4py.MPI.File* attribute), 91
`ANY_SOURCE` (in module *mpi4py.MPI*), 148
`ANY_TAG` (in module *mpi4py.MPI*), 148
`APPNUM` (in module *mpi4py.MPI*), 149
`atomicity` (*mpi4py.MPI.File* attribute), 91
`Attach()` (*mpi4py.MPI.Win* method), 125
`Attach_buffer()` (in module *mpi4py.MPI*), 137
`attrs` (*mpi4py.MPI.Win* attribute), 131

B

`BAND` (in module *mpi4py.MPI*), 180
`Barrier()` (*mpi4py.MPI.Comm* method), 54
`barrier()` (*mpi4py.MPI.Comm* method), 67
`Bcast()` (*mpi4py.MPI.Comm* method), 54
`bcast()` (*mpi4py.MPI.Comm* method), 67
`bcast()` (*mpi4py.util.pkl5.Comm* method), 44
`BOOL` (in module *mpi4py.MPI*), 177
`bootup()` (*mpi4py.futures.MPIPoolExecutor* method), 36
`BOR` (in module *mpi4py.MPI*), 181
`BOTTOM` (in module *mpi4py.MPI*), 148
`Bsend()` (*mpi4py.MPI.Comm* method), 55
`bsend()` (*mpi4py.MPI.Comm* method), 67
`bsend()` (*mpi4py.util.pkl5.Comm* method), 42
`Bsend_init()` (*mpi4py.MPI.Comm* method), 55
`BSEND_OVERHEAD` (in module *mpi4py.MPI*), 162
`BXOR` (in module *mpi4py.MPI*), 181
`BYTE` (in module *mpi4py.MPI*), 168

C

`C_BOOL` (in module *mpi4py.MPI*), 170
`C_COMPLEX` (in module *mpi4py.MPI*), 171
`C_DOUBLE_COMPLEX` (in module *mpi4py.MPI*), 172
`C_FLOAT_COMPLEX` (in module *mpi4py.MPI*), 171
`C_LONG_DOUBLE_COMPLEX` (in module *mpi4py.MPI*), 172
`Call_errhandler()` (*mpi4py.MPI.Comm* method), 55
`Call_errhandler()` (*mpi4py.MPI.File* method), 83
`Call_errhandler()` (*mpi4py.MPI.Win* method), 125
`Cancel()` (*mpi4py.MPI.Request* method), 113
`cancel()` (*mpi4py.MPI.Request* method), 115
`cancel()` (*mpi4py.util.pkl5.Request* method), 40

cancelled (*mpi4py.MPI.Status* attribute), 119
 CART (in module *mpi4py.MPI*), 161
 Cart_map() (*mpi4py.MPI.Intracomm* method), 102
 Cartcomm (class in *mpi4py.MPI*), 48
 CHAR (in module *mpi4py.MPI*), 168
 CHARACTER (in module *mpi4py.MPI*), 173
 clear() (*mpi4py.MPI.Info* method), 99
 Clone() (*mpi4py.MPI.Comm* method), 55
 Close() (*mpi4py.MPI.File* method), 83
 Close_port() (in module *mpi4py.MPI*), 137
 combiner (*mpi4py.MPI.Datatype* attribute), 79
 COMBINER_CONTIGUOUS (in module *mpi4py.MPI*), 159
 COMBINER_DARRAY (in module *mpi4py.MPI*), 160
 COMBINER_DUP (in module *mpi4py.MPI*), 159
 COMBINER_F90_COMPLEX (in module *mpi4py.MPI*), 161
 COMBINER_F90_INTEGER (in module *mpi4py.MPI*), 161
 COMBINER_F90_REAL (in module *mpi4py.MPI*), 160
 COMBINER_HINDEXED (in module *mpi4py.MPI*), 160
 COMBINER_HINDEXED_BLOCK (in module *mpi4py.MPI*), 160
 COMBINER_HVECTOR (in module *mpi4py.MPI*), 159
 COMBINER_INDEXED (in module *mpi4py.MPI*), 159
 COMBINER_INDEXED_BLOCK (in module *mpi4py.MPI*), 160
 COMBINER_NAMED (in module *mpi4py.MPI*), 159
 COMBINER_RESIZED (in module *mpi4py.MPI*), 160
 COMBINER_STRUCT (in module *mpi4py.MPI*), 160
 COMBINER_SUBARRAY (in module *mpi4py.MPI*), 160
 COMBINER_VECTOR (in module *mpi4py.MPI*), 159
 Comm (class in *mpi4py.MPI*), 50
 Comm (class in *mpi4py.util.pkl5*), 42
 COMM_NULL (in module *mpi4py.MPI*), 183
 COMM_SELF (in module *mpi4py.MPI*), 183
 COMM_TYPE_SHARED (in module *mpi4py.MPI*), 162
 COMM_WORLD (in module *mpi4py.MPI*), 183
 command line option
 --configure, 185
 --mpi, 185
 --mpicc, 185
 -c, 47
 -m, 47
 Commit() (*mpi4py.MPI.Datatype* method), 74
 Compare() (*mpi4py.MPI.Comm* class method), 55
 Compare() (*mpi4py.MPI.Group* class method), 95
 Compare_and_swap() (*mpi4py.MPI.Win* method), 125
 Complete() (*mpi4py.MPI.Grequest* method), 94
 Complete() (*mpi4py.MPI.Win* method), 125
 COMPLEX (in module *mpi4py.MPI*), 174
 COMPLEX16 (in module *mpi4py.MPI*), 176
 COMPLEX32 (in module *mpi4py.MPI*), 176
 COMPLEX4 (in module *mpi4py.MPI*), 176
 COMPLEX8 (in module *mpi4py.MPI*), 176
 Compute_dims() (in module *mpi4py.MPI*), 137
 CONGRUENT (in module *mpi4py.MPI*), 161
 Connect() (*mpi4py.MPI.Intracomm* method), 102
 contents (*mpi4py.MPI.Datatype* attribute), 79
 coords (*mpi4py.MPI.Cartcomm* attribute), 50
 copy() (*mpi4py.MPI.Info* method), 99
 COUNT (in module *mpi4py.MPI*), 168
 count (*mpi4py.MPI.Status* attribute), 119
 Create() (*mpi4py.MPI.Comm* method), 55
 Create() (*mpi4py.MPI.Info* class method), 98
 Create() (*mpi4py.MPI.Op* class method), 109
 Create() (*mpi4py.MPI.Win* class method), 125
 Create_cart() (*mpi4py.MPI.Intracomm* method), 103
 Create_contiguous() (*mpi4py.MPI.Datatype* method), 74
 Create_darray() (*mpi4py.MPI.Datatype* method), 74
 Create_dist_graph() (*mpi4py.MPI.Intracomm* method), 103
 Create_dist_graph_adjacent() (*mpi4py.MPI.Intracomm* method), 103
 Create_dynamic() (*mpi4py.MPI.Win* class method), 126
 Create_f90_complex() (*mpi4py.MPI.Datatype* class method), 74
 Create_f90_integer() (*mpi4py.MPI.Datatype* class method), 74
 Create_f90_real() (*mpi4py.MPI.Datatype* class method), 74
 Create_graph() (*mpi4py.MPI.Intracomm* method), 103
 Create_group() (*mpi4py.MPI.Comm* method), 55
 Create_hindexed() (*mpi4py.MPI.Datatype* method), 74
 Create_hindexed_block() (*mpi4py.MPI.Datatype* method), 75
 Create_hvector() (*mpi4py.MPI.Datatype* method), 75
 Create_indexed() (*mpi4py.MPI.Datatype* method), 75
 Create_indexed_block() (*mpi4py.MPI.Datatype* method), 75
 Create_intercomm() (*mpi4py.MPI.Intracomm* method), 104
 Create_keyval() (*mpi4py.MPI.Comm* class method), 56
 Create_keyval() (*mpi4py.MPI.Datatype* class method), 75
 Create_keyval() (*mpi4py.MPI.Win* class method), 126
 Create_resized() (*mpi4py.MPI.Datatype* method), 76
 Create_struct() (*mpi4py.MPI.Datatype* class method), 76
 Create_subarray() (*mpi4py.MPI.Datatype* method), 76
 Create_vector() (*mpi4py.MPI.Datatype* method), 76
 CXX_BOOL (in module *mpi4py.MPI*), 172
 CXX_DOUBLE_COMPLEX (in module *mpi4py.MPI*), 172
 CXX_FLOAT_COMPLEX (in module *mpi4py.MPI*), 172
 CXX_LONG_DOUBLE_COMPLEX (in module *mpi4py.MPI*), 172

D

Datatype (class in *mpi4py.MPI*), 72
DATATYPE_NULL (in module *mpi4py.MPI*), 167
decode() (*mpi4py.MPI.Datatype* method), 79
degrees (*mpi4py.MPI.Topocomm* attribute), 122
Delete() (*mpi4py.MPI.File* class method), 83
Delete() (*mpi4py.MPI.Info* method), 98
Delete_attr() (*mpi4py.MPI.Comm* method), 56
Delete_attr() (*mpi4py.MPI.Datatype* method), 76
Delete_attr() (*mpi4py.MPI.Win* method), 126
Detach() (*mpi4py.MPI.Win* method), 126
Detach_buffer() (in module *mpi4py.MPI*), 137
Difference() (*mpi4py.MPI.Group* class method), 95
dim (*mpi4py.MPI.Cartcomm* attribute), 50
dims (*mpi4py.MPI.Cartcomm* attribute), 50
dims (*mpi4py.MPI.Graphcomm* attribute), 93
Disconnect() (*mpi4py.MPI.Comm* method), 56
DISP_CUR (in module *mpi4py.MPI*), 165
DISPLACEMENT_CURRENT (in module *mpi4py.MPI*), 165
DIST_GRAPH (in module *mpi4py.MPI*), 162
Distgraphcomm (class in *mpi4py.MPI*), 80
DISTRIBUTE_BLOCK (in module *mpi4py.MPI*), 158
DISTRIBUTE_CYCLIC (in module *mpi4py.MPI*), 159
DISTRIBUTE_DFLT_DARG (in module *mpi4py.MPI*), 159
DISTRIBUTE_NONE (in module *mpi4py.MPI*), 158
DOUBLE (in module *mpi4py.MPI*), 170
DOUBLE_COMPLEX (in module *mpi4py.MPI*), 174
DOUBLE_INT (in module *mpi4py.MPI*), 173
DOUBLE_PRECISION (in module *mpi4py.MPI*), 174
dumps() (*mpi4py.MPI.Pickle* method), 111
Dup() (*mpi4py.MPI.Comm* method), 56
Dup() (*mpi4py.MPI.Datatype* method), 76
Dup() (*mpi4py.MPI.Group* method), 96
Dup() (*mpi4py.MPI.Info* method), 98
Dup_with_info() (*mpi4py.MPI.Comm* method), 56

E

edges (*mpi4py.MPI.Graphcomm* attribute), 93
envelope (*mpi4py.MPI.Datatype* attribute), 79
environment variable
LD_LIBRARY_PATH, 189
MPI4PY_FUTURES_MAX_WORKERS, 34, 36, 39
MPI4PY_PICKLE_PROTOCOL, 11, 23
MPI4PY_PICKLE_THRESHOLD, 24
MPI4PY_RC_ERRORS, 21, 23
MPI4PY_RC_FAST_REDUCE, 21, 23
MPI4PY_RC_FINALIZE, 21, 22
MPI4PY_RC_INITIALIZE, 20, 22
MPI4PY_RC_RECV_MPROBE, 21, 23
MPI4PY_RC_THREAD_LEVEL, 21, 23
MPI4PY_RC_THREADS, 20, 22
MPICC, 185
MPICH_USE_SHLIB, 189

MPIEXEC_UNIVERSE_SIZE, 39

PATH, 188

ERR_ACCESS (in module *mpi4py.MPI*), 154
ERR_AMODE (in module *mpi4py.MPI*), 154
ERR_ARG (in module *mpi4py.MPI*), 152
ERR_ASSERT (in module *mpi4py.MPI*), 157
ERR_BAD_FILE (in module *mpi4py.MPI*), 154
ERR_BASE (in module *mpi4py.MPI*), 156
ERR_BUFFER (in module *mpi4py.MPI*), 151
ERR_COMM (in module *mpi4py.MPI*), 150
ERR_CONVERSION (in module *mpi4py.MPI*), 155
ERR_COUNT (in module *mpi4py.MPI*), 151
ERR_DIMS (in module *mpi4py.MPI*), 152
ERR_DISP (in module *mpi4py.MPI*), 156
ERR_DUP_DATAREP (in module *mpi4py.MPI*), 155
ERR_FILE (in module *mpi4py.MPI*), 153
ERR_FILE_EXISTS (in module *mpi4py.MPI*), 154
ERR_FILE_IN_USE (in module *mpi4py.MPI*), 154
ERR_GROUP (in module *mpi4py.MPI*), 151
ERR_IN_STATUS (in module *mpi4py.MPI*), 152
ERR_INFO (in module *mpi4py.MPI*), 153
ERR_INFO_KEY (in module *mpi4py.MPI*), 153
ERR_INFO_NOKEY (in module *mpi4py.MPI*), 154
ERR_INFO_VALUE (in module *mpi4py.MPI*), 153
ERR_INTERN (in module *mpi4py.MPI*), 153
ERR_IO (in module *mpi4py.MPI*), 155
ERR_KEYVAL (in module *mpi4py.MPI*), 153
ERR_LASTCODE (in module *mpi4py.MPI*), 150
ERR_LOCKTYPE (in module *mpi4py.MPI*), 157
ERR_NAME (in module *mpi4py.MPI*), 155
ERR_NO_MEM (in module *mpi4py.MPI*), 155
ERR_NO_SPACE (in module *mpi4py.MPI*), 154
ERR_NO_SUCH_FILE (in module *mpi4py.MPI*), 154
ERR_NOT_SAME (in module *mpi4py.MPI*), 156
ERR_OP (in module *mpi4py.MPI*), 151
ERR_OTHER (in module *mpi4py.MPI*), 152
ERR_PENDING (in module *mpi4py.MPI*), 152
ERR_PORT (in module *mpi4py.MPI*), 156
ERR_QUOTA (in module *mpi4py.MPI*), 156
ERR_RANK (in module *mpi4py.MPI*), 151
ERR_READ_ONLY (in module *mpi4py.MPI*), 155
ERR_REQUEST (in module *mpi4py.MPI*), 151
ERR_RMA_ATTACH (in module *mpi4py.MPI*), 157
ERR_RMA_CONFLICT (in module *mpi4py.MPI*), 157
ERR_RMA_FLAVOR (in module *mpi4py.MPI*), 157
ERR_RMA_RANGE (in module *mpi4py.MPI*), 157
ERR_RMA_SHARED (in module *mpi4py.MPI*), 157
ERR_RMA_SYNC (in module *mpi4py.MPI*), 157
ERR_ROOT (in module *mpi4py.MPI*), 152
ERR_SERVICE (in module *mpi4py.MPI*), 156
ERR_SIZE (in module *mpi4py.MPI*), 156
ERR_SPAWN (in module *mpi4py.MPI*), 156
ERR_TAG (in module *mpi4py.MPI*), 151
ERR_TOPOLOGY (in module *mpi4py.MPI*), 152

ERR_TRUNCATE (in module *mpi4py.MPI*), 152
 ERR_TYPE (in module *mpi4py.MPI*), 151
 ERR_UNKNOWN (in module *mpi4py.MPI*), 153
 ERR_UNSUPPORTED_DATAREP (in module *mpi4py.MPI*), 155
 ERR_UNSUPPORTED_OPERATION (in module *mpi4py.MPI*), 155
 ERR_WIN (in module *mpi4py.MPI*), 153
 Errhandler (class in *mpi4py.MPI*), 81
 ERRHANDLER_NULL (in module *mpi4py.MPI*), 183
 error (*mpi4py.MPI.Status* attribute), 119
 error_class (*mpi4py.MPI.Exception* attribute), 134
 error_code (*mpi4py.MPI.Exception* attribute), 134
 error_string (*mpi4py.MPI.Exception* attribute), 134
 errors (*mpi4py.mpi4py.rc* attribute), 21
 ERRORS_ARE_FATAL (in module *mpi4py.MPI*), 183
 ERRORS_RETURN (in module *mpi4py.MPI*), 183
 Exception, 133
 Excl() (*mpi4py.MPI.Group* method), 96
 Exscan() (*mpi4py.MPI.Intracomm* method), 104
 exscan() (*mpi4py.MPI.Intracomm* method), 105
 extent (*mpi4py.MPI.Datatype* attribute), 79

F

f2py() (*mpi4py.MPI.Comm* class method), 68
 f2py() (*mpi4py.MPI.Datatype* class method), 79
 f2py() (*mpi4py.MPI.Errhandler* class method), 81
 f2py() (*mpi4py.MPI.File* class method), 91
 f2py() (*mpi4py.MPI.Group* class method), 97
 f2py() (*mpi4py.MPI.Info* class method), 99
 f2py() (*mpi4py.MPI.Message* class method), 107
 f2py() (*mpi4py.MPI.Op* class method), 109
 f2py() (*mpi4py.MPI.Request* class method), 115
 f2py() (*mpi4py.MPI.Status* class method), 119
 f2py() (*mpi4py.MPI.Win* class method), 131
 F_BOOL (in module *mpi4py.MPI*), 177
 F_COMPLEX (in module *mpi4py.MPI*), 178
 F_DOUBLE (in module *mpi4py.MPI*), 178
 F_DOUBLE_COMPLEX (in module *mpi4py.MPI*), 178
 F_FLOAT (in module *mpi4py.MPI*), 178
 F_FLOAT_COMPLEX (in module *mpi4py.MPI*), 178
 F_INT (in module *mpi4py.MPI*), 178
 fast_reduce (*mpi4py.mpi4py.rc* attribute), 21
 Fence() (*mpi4py.MPI.Win* method), 126
 Fetch_and_op() (*mpi4py.MPI.Win* method), 126
 File (class in *mpi4py.MPI*), 81
 FILE_NULL (in module *mpi4py.MPI*), 184
 finalize (*mpi4py.mpi4py.rc* attribute), 21
 Finalize() (in module *mpi4py.MPI*), 138
 flavor (*mpi4py.MPI.Win* attribute), 131
 FLOAT (in module *mpi4py.MPI*), 170
 FLOAT_INT (in module *mpi4py.MPI*), 173
 Flush() (*mpi4py.MPI.Win* method), 126
 Flush_all() (*mpi4py.MPI.Win* method), 127

Flush_local() (*mpi4py.MPI.Win* method), 127
 Flush_local_all() (*mpi4py.MPI.Win* method), 127
 format (*mpi4py.MPI.memory* attribute), 133
 Free() (*mpi4py.MPI.Comm* method), 56
 Free() (*mpi4py.MPI.Datatype* method), 76
 Free() (*mpi4py.MPI.Errhandler* method), 81
 Free() (*mpi4py.MPI.Group* method), 96
 Free() (*mpi4py.MPI.Info* method), 98
 Free() (*mpi4py.MPI.Op* method), 109
 Free() (*mpi4py.MPI.Request* method), 113
 Free() (*mpi4py.MPI.Win* method), 127
 Free() (*mpi4py.util.pkl5.Request* method), 40
 Free_keyval() (*mpi4py.MPI.Comm* class method), 56
 Free_keyval() (*mpi4py.MPI.Datatype* class method), 77
 Free_keyval() (*mpi4py.MPI.Win* class method), 127
 Free_mem() (in module *mpi4py.MPI*), 138
 from_numpy_dtype() (in module *mpi4py.util.dtlb*), 46
 fromaddress() (*mpi4py.MPI.memory* static method), 132
 frombuffer() (*mpi4py.MPI.memory* static method), 132

G

Gather() (*mpi4py.MPI.Comm* method), 56
 gather() (*mpi4py.MPI.Comm* method), 68
 Gatherv() (*mpi4py.MPI.Comm* method), 57
 Get() (*mpi4py.MPI.Info* method), 98
 get() (*mpi4py.MPI.Info* method), 99
 Get() (*mpi4py.MPI.Win* method), 127
 Get_accumulate() (*mpi4py.MPI.Win* method), 127
 Get_address() (in module *mpi4py.MPI*), 138
 Get_amode() (*mpi4py.MPI.File* method), 84
 Get_atomicity() (*mpi4py.MPI.File* method), 84
 Get_attr() (*mpi4py.MPI.Comm* method), 57
 Get_attr() (*mpi4py.MPI.Datatype* method), 77
 Get_attr() (*mpi4py.MPI.Win* method), 127
 Get_byte_offset() (*mpi4py.MPI.File* method), 84
 Get_cart_rank() (*mpi4py.MPI.Cartcomm* method), 49
 get_config() (in module *mpi4py*), 24
 Get_contents() (*mpi4py.MPI.Datatype* method), 77
 Get_coords() (*mpi4py.MPI.Cartcomm* method), 49
 Get_count() (*mpi4py.MPI.Status* method), 117
 Get_dim() (*mpi4py.MPI.Cartcomm* method), 49
 Get_dims() (*mpi4py.MPI.Graphcomm* method), 92
 Get_dist_neighbors() (*mpi4py.MPI.Distgraphcomm* method), 80
 Get_dist_neighbors_count() (*mpi4py.MPI.Distgraphcomm* method), 80
 Get_elements() (*mpi4py.MPI.Status* method), 117
 Get_envelope() (*mpi4py.MPI.Datatype* method), 77
 Get_errhandler() (*mpi4py.MPI.Comm* method), 57
 Get_errhandler() (*mpi4py.MPI.File* method), 84
 Get_errhandler() (*mpi4py.MPI.Win* method), 128
 Get_error() (*mpi4py.MPI.Status* method), 117

Get_error_class() (in module *mpi4py.MPI*), 138
 Get_error_class() (*mpi4py.MPI.Exception* method), 134
 Get_error_code() (*mpi4py.MPI.Exception* method), 134
 Get_error_string() (in module *mpi4py.MPI*), 138
 Get_error_string() (*mpi4py.MPI.Exception* method), 134
 Get_extent() (*mpi4py.MPI.Datatype* method), 77
 Get_group() (*mpi4py.MPI.Comm* method), 57
 Get_group() (*mpi4py.MPI.File* method), 84
 Get_group() (*mpi4py.MPI.Win* method), 128
 get_include() (in module *mpi4py*), 24
 Get_info() (*mpi4py.MPI.Comm* method), 57
 Get_info() (*mpi4py.MPI.File* method), 84
 Get_info() (*mpi4py.MPI.Win* method), 128
 Get_library_version() (in module *mpi4py.MPI*), 138
 Get_name() (*mpi4py.MPI.Comm* method), 57
 Get_name() (*mpi4py.MPI.Datatype* method), 77
 Get_name() (*mpi4py.MPI.Win* method), 128
 Get_neighbors() (*mpi4py.MPI.Graphcomm* method), 92
 Get_neighbors_count() (*mpi4py.MPI.Graphcomm* method), 93
 Get_nkeys() (*mpi4py.MPI.Info* method), 99
 Get_nthkey() (*mpi4py.MPI.Info* method), 99
 Get_parent() (*mpi4py.MPI.Comm* class method), 57
 Get_position() (*mpi4py.MPI.File* method), 84
 Get_position_shared() (*mpi4py.MPI.File* method), 84
 Get_processor_name() (in module *mpi4py.MPI*), 139
 Get_rank() (*mpi4py.MPI.Comm* method), 57
 Get_rank() (*mpi4py.MPI.Group* method), 96
 Get_remote_group() (*mpi4py.MPI.Intercomm* method), 101
 Get_remote_size() (*mpi4py.MPI.Intercomm* method), 101
 Get_size() (*mpi4py.MPI.Comm* method), 57
 Get_size() (*mpi4py.MPI.Datatype* method), 77
 Get_size() (*mpi4py.MPI.File* method), 84
 Get_size() (*mpi4py.MPI.Group* method), 96
 Get_source() (*mpi4py.MPI.Status* method), 118
 Get_status() (*mpi4py.MPI.Request* method), 113
 get_status() (*mpi4py.MPI.Request* method), 115
 get_status() (*mpi4py.util.pkl5.Request* method), 41
 Get_tag() (*mpi4py.MPI.Status* method), 118
 Get_topo() (*mpi4py.MPI.Cartcomm* method), 49
 Get_topo() (*mpi4py.MPI.Graphcomm* method), 93
 Get_topology() (*mpi4py.MPI.Comm* method), 57
 Get_true_extent() (*mpi4py.MPI.Datatype* method), 77
 Get_type_extent() (*mpi4py.MPI.File* method), 84
 get_vendor() (in module *mpi4py.MPI*), 142
 Get_version() (in module *mpi4py.MPI*), 139
 Get_view() (*mpi4py.MPI.File* method), 84
 GIL, 40
 GRAPH (in module *mpi4py.MPI*), 161
 Graph_map() (*mpi4py.MPI.Intracomm* method), 104
 Graphcomm (class in *mpi4py.MPI*), 92
 Grequest (class in *mpi4py.MPI*), 93
 Group (class in *mpi4py.MPI*), 94
 group (*mpi4py.MPI.Comm* attribute), 71
 group (*mpi4py.MPI.File* attribute), 91
 group (*mpi4py.MPI.Win* attribute), 131
 GROUP_EMPTY (in module *mpi4py.MPI*), 182
 GROUP_NULL (in module *mpi4py.MPI*), 182

H

 HOST (in module *mpi4py.MPI*), 149

I

 Iallgather() (*mpi4py.MPI.Comm* method), 57
 Iallgatherv() (*mpi4py.MPI.Comm* method), 58
 Iallreduce() (*mpi4py.MPI.Comm* method), 58
 Ialltoall() (*mpi4py.MPI.Comm* method), 58
 Ialltoallv() (*mpi4py.MPI.Comm* method), 58
 Ialltoallw() (*mpi4py.MPI.Comm* method), 58
 Ibarrier() (*mpi4py.MPI.Comm* method), 58
 Ibcast() (*mpi4py.MPI.Comm* method), 59
 Ibsend() (*mpi4py.MPI.Comm* method), 59
 ibsend() (*mpi4py.MPI.Comm* method), 68
 ibsend() (*mpi4py.util.pkl5.Comm* method), 42
 IDENT (in module *mpi4py.MPI*), 161
 Idup() (*mpi4py.MPI.Comm* method), 59
 Iexscan() (*mpi4py.MPI.Intracomm* method), 104
 Igather() (*mpi4py.MPI.Comm* method), 59
 Igatherv() (*mpi4py.MPI.Comm* method), 59
 Improbe() (*mpi4py.MPI.Comm* method), 59
 improbe() (*mpi4py.MPI.Comm* method), 68
 improbe() (*mpi4py.util.pkl5.Comm* method), 44
 IN_PLACE (in module *mpi4py.MPI*), 148
 Incl() (*mpi4py.MPI.Group* method), 96
 indegree (*mpi4py.MPI.Topocomm* attribute), 122
 index (*mpi4py.MPI.Graphcomm* attribute), 93
 inedges (*mpi4py.MPI.Topocomm* attribute), 122
 Ineighbor_allgather() (*mpi4py.MPI.Topocomm* method), 120
 Ineighbor_allgatherv() (*mpi4py.MPI.Topocomm* method), 120
 Ineighbor_alltoall() (*mpi4py.MPI.Topocomm* method), 120
 Ineighbor_alltoallv() (*mpi4py.MPI.Topocomm* method), 121
 Ineighbor_alltoallw() (*mpi4py.MPI.Topocomm* method), 121
 Info (class in *mpi4py.MPI*), 97
 info (*mpi4py.MPI.Comm* attribute), 71
 info (*mpi4py.MPI.File* attribute), 91

info (*mpi4py.MPI.Win attribute*), 131
 INFO_ENV (*in module mpi4py.MPI*), 183
 INFO_NULL (*in module mpi4py.MPI*), 183
 Init() (*in module mpi4py.MPI*), 139
 Init_thread() (*in module mpi4py.MPI*), 139
 initialize (*mpi4py.mpi4py.rc attribute*), 20
 inoutedges (*mpi4py.MPI.Topocomm attribute*), 122
 INT (*in module mpi4py.MPI*), 169
 INT16_T (*in module mpi4py.MPI*), 170
 INT32_T (*in module mpi4py.MPI*), 171
 INT64_T (*in module mpi4py.MPI*), 171
 INT8_T (*in module mpi4py.MPI*), 170
 INT_INT (*in module mpi4py.MPI*), 172
 INTEGER (*in module mpi4py.MPI*), 173
 INTEGER1 (*in module mpi4py.MPI*), 175
 INTEGER16 (*in module mpi4py.MPI*), 175
 INTEGER2 (*in module mpi4py.MPI*), 175
 INTEGER4 (*in module mpi4py.MPI*), 175
 INTEGER8 (*in module mpi4py.MPI*), 175
 Intercomm (*class in mpi4py.MPI*), 100
 Intercomm (*class in mpi4py.util.pkl5*), 44
 Intersection() (*mpi4py.MPI.Group class method*), 96
 Intracomm (*class in mpi4py.MPI*), 101
 Intracomm (*class in mpi4py.util.pkl5*), 44
 IO (*in module mpi4py.MPI*), 149
 Iprobe() (*mpi4py.MPI.Comm method*), 60
 iprobe() (*mpi4py.MPI.Comm method*), 68
 Iprobe() (*mpi4py.MPI.Message class method*), 107
 iprobe() (*mpi4py.MPI.Message class method*), 107
 iprobe() (*mpi4py.util.pkl5.Message class method*), 42
 Iread() (*mpi4py.MPI.File method*), 85
 Iread_all() (*mpi4py.MPI.File method*), 85
 Iread_at() (*mpi4py.MPI.File method*), 85
 Iread_at_all() (*mpi4py.MPI.File method*), 85
 Iread_shared() (*mpi4py.MPI.File method*), 85
 Irecv() (*mpi4py.MPI.Comm method*), 60
 irecv() (*mpi4py.MPI.Comm method*), 68
 Irecv() (*mpi4py.MPI.Message method*), 107
 irecv() (*mpi4py.MPI.Message method*), 108
 irecv() (*mpi4py.util.pkl5.Comm method*), 43
 irecv() (*mpi4py.util.pkl5.Message method*), 41
 Ireduce() (*mpi4py.MPI.Comm method*), 60
 Ireduce_scatter() (*mpi4py.MPI.Comm method*), 60
 Ireduce_scatter_block() (*mpi4py.MPI.Comm method*), 60
 Irsend() (*mpi4py.MPI.Comm method*), 61
 Is_cancelled() (*mpi4py.MPI.Status method*), 118
 is_commutative (*mpi4py.MPI.Op attribute*), 110
 Is_commutative() (*mpi4py.MPI.Op method*), 109
 Is_finalized() (*in module mpi4py.MPI*), 139
 Is_initialized() (*in module mpi4py.MPI*), 139
 is_inter (*mpi4py.MPI.Comm attribute*), 71
 Is_inter() (*mpi4py.MPI.Comm method*), 61
 is_intra (*mpi4py.MPI.Comm attribute*), 71
 Is_intra() (*mpi4py.MPI.Comm method*), 61
 is_named (*mpi4py.MPI.Datatype attribute*), 79
 is_predefined (*mpi4py.MPI.Datatype attribute*), 79
 is_predefined (*mpi4py.MPI.Op attribute*), 110
 Is_thread_main() (*in module mpi4py.MPI*), 140
 is_topo (*mpi4py.MPI.Comm attribute*), 71
 Iscan() (*mpi4py.MPI.Intracomm method*), 104
 Iscatter() (*mpi4py.MPI.Comm method*), 61
 Iscatterv() (*mpi4py.MPI.Comm method*), 61
 Isend() (*mpi4py.MPI.Comm method*), 61
 isend() (*mpi4py.MPI.Comm method*), 69
 isend() (*mpi4py.util.pkl5.Comm method*), 42
 Issend() (*mpi4py.MPI.Comm method*), 61
 issend() (*mpi4py.MPI.Comm method*), 69
 issend() (*mpi4py.util.pkl5.Comm method*), 43
 items() (*mpi4py.MPI.Info method*), 99
 itemsize (*mpi4py.MPI.memory attribute*), 133
 Iwrite() (*mpi4py.MPI.File method*), 85
 Iwrite_all() (*mpi4py.MPI.File method*), 85
 Iwrite_at() (*mpi4py.MPI.File method*), 85
 Iwrite_at_all() (*mpi4py.MPI.File method*), 86
 Iwrite_shared() (*mpi4py.MPI.File method*), 86

J

Join() (*mpi4py.MPI.Comm class method*), 62

K

keys() (*mpi4py.MPI.Info method*), 99
 KEYVAL_INVALID (*in module mpi4py.MPI*), 148

L

LAND (*in module mpi4py.MPI*), 180
 LASTUSEDPCODE (*in module mpi4py.MPI*), 149
 LB (*in module mpi4py.MPI*), 168
 lb (*mpi4py.MPI.Datatype attribute*), 79
 LD_LIBRARY_PATH, 189
 loads() (*mpi4py.MPI.Pickle method*), 111
 Lock() (*mpi4py.MPI.Win method*), 128
 Lock_all() (*mpi4py.MPI.Win method*), 128
 LOCK_EXCLUSIVE (*in module mpi4py.MPI*), 164
 LOCK_SHARED (*in module mpi4py.MPI*), 164
 LOGICAL (*in module mpi4py.MPI*), 173
 LOGICAL1 (*in module mpi4py.MPI*), 174
 LOGICAL2 (*in module mpi4py.MPI*), 174
 LOGICAL4 (*in module mpi4py.MPI*), 174
 LOGICAL8 (*in module mpi4py.MPI*), 174
 LONG (*in module mpi4py.MPI*), 169
 LONG_DOUBLE (*in module mpi4py.MPI*), 170
 LONG_DOUBLE_INT (*in module mpi4py.MPI*), 173
 LONG_INT (*in module mpi4py.MPI*), 173
 LONG_LONG (*in module mpi4py.MPI*), 169
 Lookup_name() (*in module mpi4py.MPI*), 140
 LOR (*in module mpi4py.MPI*), 180

LXOR (in module *mpi4py.MPI*), 181

M

map() (*mpi4py.futures.MPIPoolExecutor* method), 35
Match_size() (*mpi4py.MPI.Datatype* class method), 77
MAX (in module *mpi4py.MPI*), 179
MAX_DATAREP_STRING (in module *mpi4py.MPI*), 167
MAX_ERROR_STRING (in module *mpi4py.MPI*), 166
MAX_INFO_KEY (in module *mpi4py.MPI*), 167
MAX_INFO_VAL (in module *mpi4py.MPI*), 167
MAX_LIBRARY_VERSION_STRING (in module *mpi4py.MPI*), 167
MAX_OBJECT_NAME (in module *mpi4py.MPI*), 167
MAX_PORT_NAME (in module *mpi4py.MPI*), 167
MAX_PROCESSOR_NAME (in module *mpi4py.MPI*), 166
MAXLOC (in module *mpi4py.MPI*), 181
memory (class in *mpi4py.MPI*), 131
Merge() (*mpi4py.MPI.Intercomm* method), 101
Message (class in *mpi4py.MPI*), 106
Message (class in *mpi4py.util.pkl5*), 41
MESSAGE_NO_PROC (in module *mpi4py.MPI*), 179
MESSAGE_NULL (in module *mpi4py.MPI*), 178
MIN (in module *mpi4py.MPI*), 179
MINLOC (in module *mpi4py.MPI*), 182
MODE_APPEND (in module *mpi4py.MPI*), 165
MODE_CREATE (in module *mpi4py.MPI*), 164
MODE_DELETE_ON_CLOSE (in module *mpi4py.MPI*), 164
MODE_EXCL (in module *mpi4py.MPI*), 164
MODE_NOCHECK (in module *mpi4py.MPI*), 163
MODE_NOPRECEDE (in module *mpi4py.MPI*), 163
MODE_NOPUT (in module *mpi4py.MPI*), 163
MODE_NOSTORE (in module *mpi4py.MPI*), 163
MODE_NOSUCCEED (in module *mpi4py.MPI*), 163
MODE_RDONLY (in module *mpi4py.MPI*), 164
MODE_RDWR (in module *mpi4py.MPI*), 164
MODE_SEQUENTIAL (in module *mpi4py.MPI*), 165
MODE_UNIQUE_OPEN (in module *mpi4py.MPI*), 165
MODE_WRONLY (in module *mpi4py.MPI*), 164
model (*mpi4py.MPI.Win* attribute), 131
module
 mpi4py, 20
 mpi4py.futures, 33
 mpi4py.MPI, 47
 mpi4py.run, 46
 mpi4py.util, 40
 mpi4py.util.dtlb, 46
 mpi4py.util.pkl5, 40
mpi4py
 module, 20
mpi4py.futures
 module, 33
mpi4py.MPI
 module, 47
mpi4py.rc (in module *mpi4py*), 20

mpi4py.run
 module, 46
mpi4py.util
 module, 40
mpi4py.util.dtlb
 module, 46
mpi4py.util.pkl5
 module, 40
MPI4PY_FUTURES_MAX_WORKERS, 34, 36, 39
MPI4PY_PICKLE_PROTOCOL, 11
MPI4PY_RC_ERRORS, 21
MPI4PY_RC_FAST_REDUCE, 21
MPI4PY_RC_FINALIZE, 21
MPI4PY_RC_INITIALIZE, 20
MPI4PY_RC_RECV_MPROBE, 21
MPI4PY_RC_THREAD_LEVEL, 21
MPI4PY_RC_THREADS, 20
MPICC, 185
MPICH_USE_SHLIB, 189
MPICommExecutor (class in *mpi4py.futures*), 37
MPIEXEC_UNIVERSE_SIZE, 39
MPIPoolExecutor (class in *mpi4py.futures*), 34
Mprobe() (*mpi4py.MPI.Comm* method), 62
mprobe() (*mpi4py.MPI.Comm* method), 69
mprobe() (*mpi4py.util.pkl5.Comm* method), 44

N

name (*mpi4py.MPI.Comm* attribute), 71
name (*mpi4py.MPI.Datatype* attribute), 80
name (*mpi4py.MPI.Win* attribute), 131
nbytes (*mpi4py.MPI.memory* attribute), 133
ndim (*mpi4py.MPI.Cartcomm* attribute), 50
nedges (*mpi4py.MPI.Graphcomm* attribute), 93
Neighbor_allgather() (*mpi4py.MPI.Topocomm* method), 121
neighbor_allgather() (*mpi4py.MPI.Topocomm* method), 122
Neighbor_allgatherv() (*mpi4py.MPI.Topocomm* method), 121
Neighbor_alltoall() (*mpi4py.MPI.Topocomm* method), 121
neighbor_alltoall() (*mpi4py.MPI.Topocomm* method), 122
Neighbor_alltoallv() (*mpi4py.MPI.Topocomm* method), 121
Neighbor_alltoallw() (*mpi4py.MPI.Topocomm* method), 121
neighbors (*mpi4py.MPI.Graphcomm* attribute), 93
nneighbors (*mpi4py.MPI.Graphcomm* attribute), 93
nnodes (*mpi4py.MPI.Graphcomm* attribute), 93
NO_OP (in module *mpi4py.MPI*), 182

O

obj (*mpi4py.MPI.memory* attribute), 133

OFFSET (in module *mpi4py.MPI*), 168
 Op (class in *mpi4py.MPI*), 108
 OP_NULL (in module *mpi4py.MPI*), 179
 Open() (*mpi4py.MPI.File* class method), 86
 Open_port() (in module *mpi4py.MPI*), 140
 ORDER_C (in module *mpi4py.MPI*), 158
 ORDER_F (in module *mpi4py.MPI*), 158
 ORDER_FORTRAN (in module *mpi4py.MPI*), 158
 outdegree (*mpi4py.MPI.Topocomm* attribute), 122
 outedges (*mpi4py.MPI.Topocomm* attribute), 122

P

Pack() (*mpi4py.MPI.Datatype* method), 77
 Pack_external() (*mpi4py.MPI.Datatype* method), 78
 Pack_external_size() (*mpi4py.MPI.Datatype* method), 78
 Pack_size() (*mpi4py.MPI.Datatype* method), 78
 PACKED (in module *mpi4py.MPI*), 168
 PATH, 188
 Pcontrol() (in module *mpi4py.MPI*), 140
 periods (*mpi4py.MPI.Cartcomm* attribute), 50
 Pickle (class in *mpi4py.MPI*), 110
 pickle (in module *mpi4py.MPI*), 184
 pop() (*mpi4py.MPI.Info* method), 100
 popitem() (*mpi4py.MPI.Info* method), 100
 Post() (*mpi4py.MPI.Win* method), 128
 Preallocate() (*mpi4py.MPI.File* method), 86
 Prequest (class in *mpi4py.MPI*), 111
 Probe() (*mpi4py.MPI.Comm* method), 62
 probe() (*mpi4py.MPI.Comm* method), 69
 Probe() (*mpi4py.MPI.Message* class method), 107
 probe() (*mpi4py.MPI.Message* class method), 108
 probe() (*mpi4py.util.pkl5.Message* class method), 41
 PROC_NULL (in module *mpi4py.MPI*), 148
 PROD (in module *mpi4py.MPI*), 180
 profile() (in module *mpi4py*), 24
 PROTOCOL (*mpi4py.MPI.Pickle* attribute), 111
 Publish_name() (in module *mpi4py.MPI*), 140
 Put() (*mpi4py.MPI.Win* method), 128
 py2f() (*mpi4py.MPI.Comm* method), 69
 py2f() (*mpi4py.MPI.Datatype* method), 79
 py2f() (*mpi4py.MPI.Errhandler* method), 81
 py2f() (*mpi4py.MPI.File* method), 91
 py2f() (*mpi4py.MPI.Group* method), 97
 py2f() (*mpi4py.MPI.Info* method), 100
 py2f() (*mpi4py.MPI.Message* method), 108
 py2f() (*mpi4py.MPI.Op* method), 109
 py2f() (*mpi4py.MPI.Request* method), 115
 py2f() (*mpi4py.MPI.Status* method), 119
 py2f() (*mpi4py.MPI.Win* method), 131
 Python Enhancement Proposals
 PEP 574, 40

Q

Query_thread() (in module *mpi4py.MPI*), 141

R

Raccumulate() (*mpi4py.MPI.Win* method), 129
 Range_excl() (*mpi4py.MPI.Group* method), 96
 Range_incl() (*mpi4py.MPI.Group* method), 96
 rank (*mpi4py.MPI.Comm* attribute), 71
 rank (*mpi4py.MPI.Group* attribute), 97
 Read() (*mpi4py.MPI.File* method), 86
 Read_all() (*mpi4py.MPI.File* method), 86
 Read_all_begin() (*mpi4py.MPI.File* method), 87
 Read_all_end() (*mpi4py.MPI.File* method), 87
 Read_at() (*mpi4py.MPI.File* method), 87
 Read_at_all() (*mpi4py.MPI.File* method), 87
 Read_at_all_begin() (*mpi4py.MPI.File* method), 87
 Read_at_all_end() (*mpi4py.MPI.File* method), 87
 Read_ordered() (*mpi4py.MPI.File* method), 88
 Read_ordered_begin() (*mpi4py.MPI.File* method), 88
 Read_ordered_end() (*mpi4py.MPI.File* method), 88
 Read_shared() (*mpi4py.MPI.File* method), 88
 readonly (*mpi4py.MPI.memory* attribute), 133
 REAL (in module *mpi4py.MPI*), 174
 REAL16 (in module *mpi4py.MPI*), 176
 REAL2 (in module *mpi4py.MPI*), 175
 REAL4 (in module *mpi4py.MPI*), 175
 REAL8 (in module *mpi4py.MPI*), 175
 Recv() (*mpi4py.MPI.Comm* method), 62
 recv() (*mpi4py.MPI.Comm* method), 70
 Recv() (*mpi4py.MPI.Message* method), 107
 recv() (*mpi4py.MPI.Message* method), 108
 recv() (*mpi4py.util.pkl5.Comm* method), 43
 recv() (*mpi4py.util.pkl5.Message* method), 41
 Recv_init() (*mpi4py.MPI.Comm* method), 63
 recv_mprobe (*mpi4py.mpi4py.rc* attribute), 21
 Reduce() (*mpi4py.MPI.Comm* method), 63
 reduce() (*mpi4py.MPI.Comm* method), 70
 Reduce_local() (*mpi4py.MPI.Op* method), 109
 Reduce_scatter() (*mpi4py.MPI.Comm* method), 63
 Reduce_scatter_block() (*mpi4py.MPI.Comm* method), 63
 Register_datarep() (in module *mpi4py.MPI*), 141
 release() (*mpi4py.MPI.memory* method), 132
 remote_group (*mpi4py.MPI.Intercomm* attribute), 101
 remote_size (*mpi4py.MPI.Intercomm* attribute), 101
 REPLACE (in module *mpi4py.MPI*), 182
 Request (class in *mpi4py.MPI*), 112
 Request (class in *mpi4py.util.pkl5*), 40
 REQUEST_NULL (in module *mpi4py.MPI*), 178
 Rget() (*mpi4py.MPI.Win* method), 129
 Rget_accumulate() (*mpi4py.MPI.Win* method), 129
 ROOT (in module *mpi4py.MPI*), 148
 Rput() (*mpi4py.MPI.Win* method), 129

Rsend() (*mpi4py.MPI.Comm* method), 63
Rsend_init() (*mpi4py.MPI.Comm* method), 64

S

Scan() (*mpi4py.MPI.Intracomm* method), 105
scan() (*mpi4py.MPI.Intracomm* method), 106
Scatter() (*mpi4py.MPI.Comm* method), 64
scatter() (*mpi4py.MPI.Comm* method), 70
Scatterv() (*mpi4py.MPI.Comm* method), 64
Seek() (*mpi4py.MPI.File* method), 88
SEEK_CUR (in module *mpi4py.MPI*), 165
SEEK_END (in module *mpi4py.MPI*), 165
SEEK_SET (in module *mpi4py.MPI*), 165
Seek_shared() (*mpi4py.MPI.File* method), 88
Send() (*mpi4py.MPI.Comm* method), 64
send() (*mpi4py.MPI.Comm* method), 70
send() (*mpi4py.util.pkl5.Comm* method), 42
Send_init() (*mpi4py.MPI.Comm* method), 64
Sendrecv() (*mpi4py.MPI.Comm* method), 65
sendrecv() (*mpi4py.MPI.Comm* method), 70
sendrecv() (*mpi4py.util.pkl5.Comm* method), 43
Sendrecv_replace() (*mpi4py.MPI.Comm* method), 65
Set() (*mpi4py.MPI.Info* method), 99
Set_atomicsity() (*mpi4py.MPI.File* method), 88
Set_attr() (*mpi4py.MPI.Comm* method), 66
Set_attr() (*mpi4py.MPI.Datatype* method), 78
Set_attr() (*mpi4py.MPI.Win* method), 129
Set_cancelled() (*mpi4py.MPI.Status* method), 118
Set_elements() (*mpi4py.MPI.Status* method), 118
Set_errhandler() (*mpi4py.MPI.Comm* method), 66
Set_errhandler() (*mpi4py.MPI.File* method), 89
Set_errhandler() (*mpi4py.MPI.Win* method), 130
Set_error() (*mpi4py.MPI.Status* method), 118
Set_info() (*mpi4py.MPI.Comm* method), 66
Set_info() (*mpi4py.MPI.File* method), 89
Set_info() (*mpi4py.MPI.Win* method), 130
Set_name() (*mpi4py.MPI.Comm* method), 66
Set_name() (*mpi4py.MPI.Datatype* method), 78
Set_name() (*mpi4py.MPI.Win* method), 130
Set_size() (*mpi4py.MPI.File* method), 89
Set_source() (*mpi4py.MPI.Status* method), 118
Set_tag() (*mpi4py.MPI.Status* method), 118
Set_view() (*mpi4py.MPI.File* method), 89
Shared_query() (*mpi4py.MPI.Win* method), 130
Shift() (*mpi4py.MPI.Cartcomm* method), 49
SHORT (in module *mpi4py.MPI*), 169
SHORT_INT (in module *mpi4py.MPI*), 172
shutdown() (*mpi4py.futures.MPIPoolExecutor* method), 35
SIGNED_CHAR (in module *mpi4py.MPI*), 169
SIGNED_INT (in module *mpi4py.MPI*), 176
SIGNED_LONG (in module *mpi4py.MPI*), 177
SIGNED_LONG_LONG (in module *mpi4py.MPI*), 177
SIGNED_SHORT (in module *mpi4py.MPI*), 176

SIMILAR (in module *mpi4py.MPI*), 161
SINT16_T (in module *mpi4py.MPI*), 177
SINT32_T (in module *mpi4py.MPI*), 177
SINT64_T (in module *mpi4py.MPI*), 177
SINT8_T (in module *mpi4py.MPI*), 177
size (*mpi4py.MPI.Comm* attribute), 71
size (*mpi4py.MPI.Datatype* attribute), 80
size (*mpi4py.MPI.File* attribute), 91
size (*mpi4py.MPI.Group* attribute), 97
source (*mpi4py.MPI.Status* attribute), 119
Spawn() (*mpi4py.MPI.Intracomm* method), 105
Spawn_multiple() (*mpi4py.MPI.Intracomm* method), 105
Split() (*mpi4py.MPI.Comm* method), 66
Split_type() (*mpi4py.MPI.Comm* method), 66
Ssend() (*mpi4py.MPI.Comm* method), 66
ssend() (*mpi4py.MPI.Comm* method), 71
ssend() (*mpi4py.util.pkl5.Comm* method), 42
Ssend_init() (*mpi4py.MPI.Comm* method), 67
starmap() (*mpi4py.futures.MPIPoolExecutor* method), 35
Start() (*mpi4py.MPI.Grequest* class method), 94
Start() (*mpi4py.MPI.Prerequest* method), 112
Start() (*mpi4py.MPI.Win* method), 130
Startall() (*mpi4py.MPI.Prerequest* class method), 112
Status (class in *mpi4py.MPI*), 116
Sub() (*mpi4py.MPI.Cartcomm* method), 49
submit() (*mpi4py.futures.MPIPoolExecutor* method), 35
SUBVERSION (in module *mpi4py.MPI*), 166
SUCCESS (in module *mpi4py.MPI*), 150
SUM (in module *mpi4py.MPI*), 179
Sync() (*mpi4py.MPI.File* method), 89
Sync() (*mpi4py.MPI.Win* method), 130

T

tag (*mpi4py.MPI.Status* attribute), 119
TAG_UB (in module *mpi4py.MPI*), 149
Test() (*mpi4py.MPI.Request* method), 113
test() (*mpi4py.MPI.Request* method), 115
Test() (*mpi4py.MPI.Win* method), 130
test() (*mpi4py.util.pkl5.Request* method), 41
Testall() (*mpi4py.MPI.Request* class method), 114
testall() (*mpi4py.MPI.Request* class method), 115
testall() (*mpi4py.util.pkl5.Request* class method), 41
Testany() (*mpi4py.MPI.Request* class method), 114
testany() (*mpi4py.MPI.Request* class method), 115
Testsome() (*mpi4py.MPI.Request* class method), 114
testsome() (*mpi4py.MPI.Request* class method), 115
THREAD_FUNNELED (in module *mpi4py.MPI*), 166
thread_level (*mpi4py.mpi4py.rc* attribute), 20
THREAD_MULTIPLE (in module *mpi4py.MPI*), 166
THREAD_SERIALIZED (in module *mpi4py.MPI*), 166
THREAD_SINGLE (in module *mpi4py.MPI*), 166
threads (*mpi4py.mpi4py.rc* attribute), 20

to_numpy_dtype() (in module *mpi4py.util.dtlb*), 46
 tobytes() (*mpi4py.MPI.memory* method), 133
 tomemory() (*mpi4py.MPI.Win* method), 131
 topo (*mpi4py.MPI.Cartcomm* attribute), 50
 topo (*mpi4py.MPI.Graphcomm* attribute), 93
 Topocomm (class in *mpi4py.MPI*), 119
 topology (*mpi4py.MPI.Comm* attribute), 71
 toreadonly() (*mpi4py.MPI.memory* method), 133
 Translate_ranks() (*mpi4py.MPI.Group* class method), 96
 true_extent (*mpi4py.MPI.Datatype* attribute), 80
 true_lb (*mpi4py.MPI.Datatype* attribute), 80
 true_ub (*mpi4py.MPI.Datatype* attribute), 80
 TWOINT (in module *mpi4py.MPI*), 173
 TYPECLASS_COMPLEX (in module *mpi4py.MPI*), 158
 TYPECLASS_INTEGER (in module *mpi4py.MPI*), 158
 TYPECLASS_REAL (in module *mpi4py.MPI*), 158

U

UB (in module *mpi4py.MPI*), 167
 ub (*mpi4py.MPI.Datatype* attribute), 80
 UINT16_T (in module *mpi4py.MPI*), 171
 UINT32_T (in module *mpi4py.MPI*), 171
 UINT64_T (in module *mpi4py.MPI*), 171
 UINT8_T (in module *mpi4py.MPI*), 171
 UNDEFINED (in module *mpi4py.MPI*), 148
 UNEQUAL (in module *mpi4py.MPI*), 161
 Union() (*mpi4py.MPI.Group* class method), 97
 UNIVERSE_SIZE (in module *mpi4py.MPI*), 149
 Unlock() (*mpi4py.MPI.Win* method), 130
 Unlock_all() (*mpi4py.MPI.Win* method), 130
 Unpack() (*mpi4py.MPI.Datatype* method), 78
 Unpack_external() (*mpi4py.MPI.Datatype* method), 79
 Unpublish_name() (in module *mpi4py.MPI*), 141
 UNSIGNED (in module *mpi4py.MPI*), 169
 UNSIGNED_CHAR (in module *mpi4py.MPI*), 169
 UNSIGNED_INT (in module *mpi4py.MPI*), 176
 UNSIGNED_LONG (in module *mpi4py.MPI*), 170
 UNSIGNED_LONG_LONG (in module *mpi4py.MPI*), 170
 UNSIGNED_SHORT (in module *mpi4py.MPI*), 169
 UNWEIGHTED (in module *mpi4py.MPI*), 162
 update() (*mpi4py.MPI.Info* method), 100

V

values() (*mpi4py.MPI.Info* method), 100
 VERSION (in module *mpi4py.MPI*), 166

W

Wait() (*mpi4py.MPI.Request* method), 114
 wait() (*mpi4py.MPI.Request* method), 116
 Wait() (*mpi4py.MPI.Win* method), 130
 wait() (*mpi4py.util.pkl5.Request* method), 41

Waitall() (*mpi4py.MPI.Request* class method), 114
 waitall() (*mpi4py.MPI.Request* class method), 116
 waitall() (*mpi4py.util.pkl5.Request* class method), 41
 Waitany() (*mpi4py.MPI.Request* class method), 114
 waitany() (*mpi4py.MPI.Request* class method), 116
 Waitsome() (*mpi4py.MPI.Request* class method), 114
 waitsome() (*mpi4py.MPI.Request* class method), 116
 WCHAR (in module *mpi4py.MPI*), 168
 WEIGHTS_EMPTY (in module *mpi4py.MPI*), 162
 Win (class in *mpi4py.MPI*), 122
 WIN_BASE (in module *mpi4py.MPI*), 149
 WIN_CREATE_FLAVOR (in module *mpi4py.MPI*), 150
 WIN_DISP_UNIT (in module *mpi4py.MPI*), 150
 WIN_FLAVOR (in module *mpi4py.MPI*), 150
 WIN_FLAVOR_ALLOCATE (in module *mpi4py.MPI*), 162
 WIN_FLAVOR_CREATE (in module *mpi4py.MPI*), 162
 WIN_FLAVOR_DYNAMIC (in module *mpi4py.MPI*), 162
 WIN_FLAVOR_SHARED (in module *mpi4py.MPI*), 163
 WIN_MODEL (in module *mpi4py.MPI*), 150
 WIN_NULL (in module *mpi4py.MPI*), 184
 WIN_SEPARATE (in module *mpi4py.MPI*), 163
 WIN_SIZE (in module *mpi4py.MPI*), 150
 WIN_UNIFIED (in module *mpi4py.MPI*), 163
 Write() (*mpi4py.MPI.File* method), 89
 Write_all() (*mpi4py.MPI.File* method), 89
 Write_all_begin() (*mpi4py.MPI.File* method), 90
 Write_all_end() (*mpi4py.MPI.File* method), 90
 Write_at() (*mpi4py.MPI.File* method), 90
 Write_at_all() (*mpi4py.MPI.File* method), 90
 Write_at_all_begin() (*mpi4py.MPI.File* method), 90
 Write_at_all_end() (*mpi4py.MPI.File* method), 90
 Write_ordered() (*mpi4py.MPI.File* method), 91
 Write_ordered_begin() (*mpi4py.MPI.File* method), 91
 Write_ordered_end() (*mpi4py.MPI.File* method), 91
 Write_shared() (*mpi4py.MPI.File* method), 91
 Wtick() (in module *mpi4py.MPI*), 141
 Wtime() (in module *mpi4py.MPI*), 141
 WTIME_IS_GLOBAL (in module *mpi4py.MPI*), 149