
MPI for Python

Release 4.0.0.dev0

Lisandro Dalcin

Mar 04, 2024

Contents

1	Introduction	3
1.1	What is MPI?	4
1.2	What is Python?	4
1.3	Related Projects	4
2	Overview	5
2.1	Communicating Python Objects and Array Data	5
2.2	Communicators	6
2.3	Point-to-Point Communications	6
2.4	Collective Communications	7
2.5	Support for GPU-aware MPI	8
2.6	Dynamic Process Management	8
2.7	One-Sided Communications	9
2.8	Parallel Input/Output	9
2.9	Environmental Management	10
3	Tutorial	11
3.1	Running Python scripts with MPI	12
3.2	Point-to-Point Communication	12
3.3	Collective Communication	14
3.4	Input/Output (MPI-IO)	16
3.5	Dynamic Process Management	17
3.6	GPU-aware MPI + Python GPU arrays	18
3.7	One-Sided Communication (RMA)	18
3.8	Wrapping with SWIG	19
3.9	Wrapping with F2Py	20
4	mpi4py	20
4.1	Runtime configuration options	21
4.2	Environment variables	23
4.3	Miscellaneous functions	26
5	mpi4py.MPI	26
5.1	Classes	26
5.2	Functions	28
5.3	Attributes	30

6	mpi4py.typing	35
7	mpi4py.futures	38
7.1	MPIPoolExecutor	39
7.2	MPICommExecutor	42
7.3	Command line	43
7.4	Parallel tasks	44
7.5	Utilities	44
7.6	Examples	45
7.7	Citation	48
8	mpi4py.util	48
8.1	mpi4py.util.dtlb	48
8.2	mpi4py.util.pkl5	48
8.3	mpi4py.util.pool	55
9	mpi4py.run	57
9.1	Exceptions and deadlocks	58
9.2	Command line	58
10	mpi4py.bench	59
11	Reference	59
11.1	mpi4py.MPI	59
12	Citation	230
13	Installation	231
13.1	Build backends	231
13.2	Using pip	233
13.3	Using conda	233
13.4	Linux	234
13.5	macOS	235
13.6	Windows	235
14	Development	235
14.1	Prerequisites	235
14.2	Building	236
14.3	Testing	237
15	Appendix	237
15.1	MPI-enabled Python interpreter	237
15.2	Building MPI from sources	238
16	LICENSE	239
17	CHANGES	240
17.1	Release 4.0.0 [2024-XX-XX]	240
17.2	Release 3.1.5 [2023-10-04]	241
17.3	Release 3.1.4 [2022-11-02]	241
17.4	Release 3.1.3 [2021-11-25]	242
17.5	Release 3.1.2 [2021-11-04]	242
17.6	Release 3.1.1 [2021-08-14]	242
17.7	Release 3.1.0 [2021-08-12]	242
17.8	Release 3.0.3 [2019-11-04]	243
17.9	Release 3.0.2 [2019-06-11]	243

17.10Release 3.0.1 [2019-02-15]	243
17.11Release 3.0.0 [2017-11-08]	244
17.12Release 2.0.0 [2015-10-18]	244
17.13Release 1.3.1 [2013-08-07]	245
17.14Release 1.3 [2012-01-20]	246
17.15Release 1.2.2 [2010-09-13]	246
17.16Release 1.2.1 [2010-02-26]	246
17.17Release 1.2 [2009-12-29]	246
17.18Release 1.1.0 [2009-06-06]	247
17.19Release 1.0.0 [2009-03-20]	247
References	248
Python Module Index	249
Index	250

Abstract

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

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

1 Introduction

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

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

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

For implementing general-purpose numerical computations, MATLAB¹ is the dominant interpreted programming language. In the open source side, Octave and Scilab are well known, freely distributed software packages providing

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

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.

1.3 Related Projects

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

- OOMPI
 - It has no relation with Python, but is an excellent object oriented approach to MPI.
 - It is a C++ class library specification layered on top of the C bindings that encapsulates MPI into a functional class hierarchy.
 - It provides a flexible and intuitive interface by adding some abstractions, like *Ports* and *Messages*, which enrich and simplify the syntax.
- Pypar
 - Its interface is rather minimal. There is no support for communicators or process topologies.
 - It does not require the Python interpreter to be modified or recompiled, but does not permit interactive parallel runs.
 - General (*picklable*) Python objects of any type can be communicated. There is good support for numeric arrays, practically full MPI bandwidth can be achieved.
- pyMPI

- It rebuilds the Python interpreter providing a built-in module for message passing. It does permit interactive parallel runs, which are useful for learning and debugging.
 - It provides an interface suitable for basic parallel 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 intracomunicator 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 `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 `Intracomm.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 intracomunicator 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 host and device. On the Python side, support for handling GPU arrays have been implemented in many libraries related GPU computation such as CuPy, Numba, PyTorch, and PyArrow. To maximize interoperability across library boundaries, two kinds of zero-copy data exchange protocols have been defined and agreed upon: `DLPack` and `CUDA Array Interface (CAI)`.

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 to 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. New custom error handlers can be created with `Comm.Create_errhandler`.

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

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

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

3 Tutorial

Warning: Under construction. Contributions very welcome!

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

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

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

- Communication of generic Python objects

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

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

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

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

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

- Communication of buffer-like objects

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

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

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

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

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

3.1 Running Python scripts with MPI

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

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

to run the program with 4 processors.

3.2 Point-to-Point Communication

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

```
from mpi4py import MPI

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

if rank == 0:
    data = {'a': 7, 'b': 3.14}
```

(continues on next page)

```

    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)

# 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:
    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()
```

(continues on next page)

```

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:
    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')

```

(continues on next page)

```

comm.Allgather([x, MPI.DOUBLE],
              [xg, MPI.DOUBLE])
y = numpy.dot(A, xg)
return y

```

3.4 Input/Output (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')
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')
comm.Reduce([PI, MPI.DOUBLE], None,
            op=MPI.SUM, root=0)

comm.Disconnect()
```

3.6 GPU-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)
cp.cuda.get_current_stream().synchronize()
comm.Allreduce(sendbuf, recvbuf)

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

3.7 One-Sided Communication (RMA)

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

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

```

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

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

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

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

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

```

3.8 Wrapping with SWIG

- C source:

```

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

(continues on next page)

```
}%
```

```
%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

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 builds 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

<code>initialize</code>	Automatic MPI initialization at import
<code>threads</code>	Request initialization with thread support
<code>thread_level</code>	Level of thread support to request
<code>finalize</code>	Automatic MPI finalization at exit
<code>fast_reduce</code>	Use tree-based reductions for objects
<code>recv_mprobe</code>	Use matched probes to receive objects
<code>irecv_bufsz</code>	Default buffer size in bytes for <code>irecv()</code>
<code>errors</code>	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.irecv_bufsz`

Default buffer size in bytes for `irecv()`.

Type

`int`

Default

`32768`

See also:

[*MPI4PY_RC_IRecv_Bufsz*](#)

New in version 4.0.0.

`mpi4py.rc.errors`

Error handling policy.

Type

`str`

Default
"exception"
Choices
"exception", "default", "abort", "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 4.0.0.

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 4.0.0.

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_IRecv_BUFSZ

Type

`bool`

Default

`True`

Default buffer size in bytes for `irecv()`.

See also:

`mpi4py.rc.irecv_bufsz`

New in version 4.0.0.

MPI4PY_RC_ERRORS

Default

`"exception"`

Choices

`"exception", "default", "abort", "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:

`THRESHOLD` attribute of the `MPI.pickle` object within the `MPI` module.

New in version 3.1.2.

4.3 Miscellaneous functions

`mpi4py.profile(name, *, path=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.

Return type

`None`

`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`

`mpi4py.get_config()`

Return a dictionary with information about MPI.

Changed in version 4.0.0: By default, this function returns an empty dictionary. However, downstream packagers and distributors may alter such behavior. To that end, MPI information must be provided under an `mpi` section within a UTF-8 encoded INI-style configuration file `mpi.cfg` located at the top-level package directory. The configuration file is read and parsed using the `configparser` module.

Return type

`dict[str, str]`

5 mpi4py.MPI

5.1 Classes

Ancillary

<i>Datatype</i>	Datatype object.
<i>Status</i>	Status object.
<i>Request</i>	Request handler.
<i>Prequest</i>	Persistent request handler.
<i>Grequest</i>	Generalized request handler.
<i>Op</i>	Reduction operation.
<i>Group</i>	Group of processes.
<i>Info</i>	Info object.

Communication

<i>Comm</i>	Communication context.
<i>Intracomm</i>	Intracomunicator.
<i>Topocomm</i>	Topology intracomunicator.
<i>Cartcomm</i>	Cartesian topology intracomunicator.
<i>Graphcomm</i>	General graph topology intracomunicator.
<i>Distgraphcomm</i>	Distributed graph topology intracomunicator.
<i>Intercomm</i>	Intercommunicator.
<i>Message</i>	Matched message.

One-sided operations

<i>Win</i>	Remote memory access context.
------------	-------------------------------

Input/Output

<i>File</i>	File I/O context.
-------------	-------------------

Error handling

<i>Errhandler</i>	Error handler.
<i>Exception</i>	Exception class.

Auxiliary

<code>Pickle memory</code>	Pickle/unpickle Python objects. alias of <code>buffer</code>
--------------------------------	---

5.2 Functions

Version inquiry

<code>Get_version()</code>	Obtain the version number of the MPI standard.
<code>Get_library_version()</code>	Obtain the version string of the MPI library.

Initialization and finalization

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

Memory allocation

<code>Alloc_mem(size[, info])</code>	Allocate memory for message passing and remote memory access.
<code>Free_mem(mem)</code>	Free memory allocated with <code>Alloc_mem</code> .

Address manipulation

<code>Get_address(location)</code>	Get the address of a location in memory.
<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.

Timer

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

Error handling

<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>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> .

Dynamic process management

<code>Open_port([info])</code>	Return an address used to connect group of processes.
<code>Close_port(port_name)</code>	Close a port.
<code>Publish_name(service_name, port_name[, info])</code>	Publish a service name.
<code>Unpublish_name(service_name, port_name[, info])</code>	Unpublish a service name.
<code>Lookup_name(service_name[, info])</code>	Lookup a port name given a service name.

Miscellanea

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

Utilities

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

5.3 Attributes

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

continues on next page

Table 1 – continued from previous page

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

continues on next page

Table 1 – continued from previous page

<code>COMBINER_RESIZED</code>	Constant COMBINER_RESIZED of type <code>int</code>
<code>COMBINER_F90_REAL</code>	Constant COMBINER_F90_REAL of type <code>int</code>
<code>COMBINER_F90_COMPLEX</code>	Constant COMBINER_F90_COMPLEX of type <code>int</code>
<code>COMBINER_F90_INTEGER</code>	Constant COMBINER_F90_INTEGER of type <code>int</code>
<code>IDENT</code>	Constant IDENT of type <code>int</code>
<code>CONGRUENT</code>	Constant CONGRUENT of type <code>int</code>
<code>SIMILAR</code>	Constant SIMILAR of type <code>int</code>
<code>UNEQUAL</code>	Constant UNEQUAL of type <code>int</code>
<code>CART</code>	Constant CART of type <code>int</code>
<code>GRAPH</code>	Constant GRAPH of type <code>int</code>
<code>DIST_GRAPH</code>	Constant DIST_GRAPH of type <code>int</code>
<code>UNWEIGHTED</code>	Constant UNWEIGHTED of type <code>int</code>
<code>WEIGHTS_EMPTY</code>	Constant WEIGHTS_EMPTY of type <code>int</code>
<code>COMM_TYPE_SHARED</code>	Constant COMM_TYPE_SHARED of type <code>int</code>
<code>BSEND_OVERHEAD</code>	Constant BSEND_OVERHEAD of type <code>int</code>
<code>WIN_FLAVOR_CREATE</code>	Constant WIN_FLAVOR_CREATE of type <code>int</code>
<code>WIN_FLAVOR_ALLOCATE</code>	Constant WIN_FLAVOR_ALLOCATE of type <code>int</code>
<code>WIN_FLAVOR_DYNAMIC</code>	Constant WIN_FLAVOR_DYNAMIC of type <code>int</code>
<code>WIN_FLAVOR_SHARED</code>	Constant WIN_FLAVOR_SHARED of type <code>int</code>
<code>WIN_SEPARATE</code>	Constant WIN_SEPARATE of type <code>int</code>
<code>WIN_UNIFIED</code>	Constant WIN_UNIFIED of type <code>int</code>
<code>MODE_NOCHECK</code>	Constant MODE_NOCHECK of type <code>int</code>
<code>MODE_NOSTORE</code>	Constant MODE_NOSTORE of type <code>int</code>
<code>MODE_NOPUT</code>	Constant MODE_NOPUT of type <code>int</code>
<code>MODE_NOPRECEDE</code>	Constant MODE_NOPRECEDE of type <code>int</code>
<code>MODE_NOSUCCEED</code>	Constant MODE_NOSUCCEED of type <code>int</code>
<code>LOCK_EXCLUSIVE</code>	Constant LOCK_EXCLUSIVE of type <code>int</code>
<code>LOCK_SHARED</code>	Constant LOCK_SHARED of type <code>int</code>
<code>MODE_RDONLY</code>	Constant MODE_RDONLY of type <code>int</code>
<code>MODE_WRONLY</code>	Constant MODE_WRONLY of type <code>int</code>
<code>MODE_RDWR</code>	Constant MODE_RDWR of type <code>int</code>
<code>MODE_CREATE</code>	Constant MODE_CREATE of type <code>int</code>
<code>MODE_EXCL</code>	Constant MODE_EXCL of type <code>int</code>
<code>MODE_DELETE_ON_CLOSE</code>	Constant MODE_DELETE_ON_CLOSE of type <code>int</code>
<code>MODE_UNIQUE_OPEN</code>	Constant MODE_UNIQUE_OPEN of type <code>int</code>
<code>MODE_SEQUENTIAL</code>	Constant MODE_SEQUENTIAL of type <code>int</code>
<code>MODE_APPEND</code>	Constant MODE_APPEND of type <code>int</code>
<code>SEEK_SET</code>	Constant SEEK_SET of type <code>int</code>
<code>SEEK_CUR</code>	Constant SEEK_CUR of type <code>int</code>
<code>SEEK_END</code>	Constant SEEK_END of type <code>int</code>
<code>DISPLACEMENT_CURRENT</code>	Constant DISPLACEMENT_CURRENT of type <code>int</code>
<code>DISP_CUR</code>	Constant DISP_CUR of type <code>int</code>
<code>THREAD_SINGLE</code>	Constant THREAD_SINGLE of type <code>int</code>
<code>THREAD_FUNNELED</code>	Constant THREAD_FUNNELED of type <code>int</code>
<code>THREAD_SERIALIZED</code>	Constant THREAD_SERIALIZED of type <code>int</code>
<code>THREAD_MULTIPLE</code>	Constant THREAD_MULTIPLE of type <code>int</code>
<code>VERSION</code>	Constant VERSION of type <code>int</code>
<code>SUBVERSION</code>	Constant SUBVERSION of type <code>int</code>
<code>MAX_PROCESSOR_NAME</code>	Constant MAX_PROCESSOR_NAME of type <code>int</code>
<code>MAX_ERROR_STRING</code>	Constant MAX_ERROR_STRING of type <code>int</code>
<code>MAX_PORT_NAME</code>	Constant MAX_PORT_NAME of type <code>int</code>
<code>MAX_INFO_KEY</code>	Constant MAX_INFO_KEY of type <code>int</code>

continues on next page

Table 1 – continued from previous page

<i>MAX_INFO_VAL</i>	Constant <i>MAX_INFO_VAL</i> of type <i>int</i>
<i>MAX_OBJECT_NAME</i>	Constant <i>MAX_OBJECT_NAME</i> of type <i>int</i>
<i>MAX_DATAREP_STRING</i>	Constant <i>MAX_DATAREP_STRING</i> of type <i>int</i>
<i>MAX_LIBRARY_VERSION_STRING</i>	Constant <i>MAX_LIBRARY_VERSION_STRING</i> of type <i>int</i>
<i>DATATYPE_NULL</i>	Object <i>DATATYPE_NULL</i> of type <i>Datatype</i>
<i>PACKED</i>	Object <i>PACKED</i> of type <i>Datatype</i>
<i>BYTE</i>	Object <i>BYTE</i> of type <i>Datatype</i>
<i>AINT</i>	Object <i>AINT</i> of type <i>Datatype</i>
<i>OFFSET</i>	Object <i>OFFSET</i> of type <i>Datatype</i>
<i>COUNT</i>	Object <i>COUNT</i> of type <i>Datatype</i>
<i>CHAR</i>	Object <i>CHAR</i> of type <i>Datatype</i>
<i>WCHAR</i>	Object <i>WCHAR</i> of type <i>Datatype</i>
<i>SIGNED_CHAR</i>	Object <i>SIGNED_CHAR</i> of type <i>Datatype</i>
<i>SHORT</i>	Object <i>SHORT</i> of type <i>Datatype</i>
<i>INT</i>	Object <i>INT</i> of type <i>Datatype</i>
<i>LONG</i>	Object <i>LONG</i> of type <i>Datatype</i>
<i>LONG_LONG</i>	Object <i>LONG_LONG</i> of type <i>Datatype</i>
<i>UNSIGNED_CHAR</i>	Object <i>UNSIGNED_CHAR</i> of type <i>Datatype</i>
<i>UNSIGNED_SHORT</i>	Object <i>UNSIGNED_SHORT</i> of type <i>Datatype</i>
<i>UNSIGNED</i>	Object <i>UNSIGNED</i> of type <i>Datatype</i>
<i>UNSIGNED_LONG</i>	Object <i>UNSIGNED_LONG</i> of type <i>Datatype</i>
<i>UNSIGNED_LONG_LONG</i>	Object <i>UNSIGNED_LONG_LONG</i> of type <i>Datatype</i>
<i>FLOAT</i>	Object <i>FLOAT</i> of type <i>Datatype</i>
<i>DOUBLE</i>	Object <i>DOUBLE</i> of type <i>Datatype</i>
<i>LONG_DOUBLE</i>	Object <i>LONG_DOUBLE</i> of type <i>Datatype</i>
<i>C_BOOL</i>	Object <i>C_BOOL</i> of type <i>Datatype</i>
<i>INT8_T</i>	Object <i>INT8_T</i> of type <i>Datatype</i>
<i>INT16_T</i>	Object <i>INT16_T</i> of type <i>Datatype</i>
<i>INT32_T</i>	Object <i>INT32_T</i> of type <i>Datatype</i>
<i>INT64_T</i>	Object <i>INT64_T</i> of type <i>Datatype</i>
<i>UINT8_T</i>	Object <i>UINT8_T</i> of type <i>Datatype</i>
<i>UINT16_T</i>	Object <i>UINT16_T</i> of type <i>Datatype</i>
<i>UINT32_T</i>	Object <i>UINT32_T</i> of type <i>Datatype</i>
<i>UINT64_T</i>	Object <i>UINT64_T</i> of type <i>Datatype</i>
<i>C_COMPLEX</i>	Object <i>C_COMPLEX</i> of type <i>Datatype</i>
<i>C_FLOAT_COMPLEX</i>	Object <i>C_FLOAT_COMPLEX</i> of type <i>Datatype</i>
<i>C_DOUBLE_COMPLEX</i>	Object <i>C_DOUBLE_COMPLEX</i> of type <i>Datatype</i>
<i>C_LONG_DOUBLE_COMPLEX</i>	Object <i>C_LONG_DOUBLE_COMPLEX</i> of type <i>Datatype</i>
<i>CXX_BOOL</i>	Object <i>CXX_BOOL</i> of type <i>Datatype</i>
<i>CXX_FLOAT_COMPLEX</i>	Object <i>CXX_FLOAT_COMPLEX</i> of type <i>Datatype</i>
<i>CXX_DOUBLE_COMPLEX</i>	Object <i>CXX_DOUBLE_COMPLEX</i> of type <i>Datatype</i>
<i>CXX_LONG_DOUBLE_COMPLEX</i>	Object <i>CXX_LONG_DOUBLE_COMPLEX</i> of type <i>Datatype</i>
<i>SHORT_INT</i>	Object <i>SHORT_INT</i> of type <i>Datatype</i>
<i>INT_INT</i>	Object <i>INT_INT</i> of type <i>Datatype</i>
<i>TWOINT</i>	Object <i>TWOINT</i> of type <i>Datatype</i>
<i>LONG_INT</i>	Object <i>LONG_INT</i> of type <i>Datatype</i>
<i>FLOAT_INT</i>	Object <i>FLOAT_INT</i> of type <i>Datatype</i>
<i>DOUBLE_INT</i>	Object <i>DOUBLE_INT</i> of type <i>Datatype</i>
<i>LONG_DOUBLE_INT</i>	Object <i>LONG_DOUBLE_INT</i> of type <i>Datatype</i>
<i>CHARACTER</i>	Object <i>CHARACTER</i> of type <i>Datatype</i>
<i>LOGICAL</i>	Object <i>LOGICAL</i> of type <i>Datatype</i>
<i>INTEGER</i>	Object <i>INTEGER</i> of type <i>Datatype</i>

continues on next page

Table 1 – continued from previous page

<i>REAL</i>	Object REAL of type <i>Datatype</i>
<i>DOUBLE_PRECISION</i>	Object DOUBLE_PRECISION of type <i>Datatype</i>
<i>COMPLEX</i>	Object COMPLEX of type <i>Datatype</i>
<i>DOUBLE_COMPLEX</i>	Object DOUBLE_COMPLEX of type <i>Datatype</i>
<i>LOGICAL1</i>	Object LOGICAL1 of type <i>Datatype</i>
<i>LOGICAL2</i>	Object LOGICAL2 of type <i>Datatype</i>
<i>LOGICAL4</i>	Object LOGICAL4 of type <i>Datatype</i>
<i>LOGICAL8</i>	Object LOGICAL8 of type <i>Datatype</i>
<i>INTEGER1</i>	Object INTEGER1 of type <i>Datatype</i>
<i>INTEGER2</i>	Object INTEGER2 of type <i>Datatype</i>
<i>INTEGER4</i>	Object INTEGER4 of type <i>Datatype</i>
<i>INTEGER8</i>	Object INTEGER8 of type <i>Datatype</i>
<i>INTEGER16</i>	Object INTEGER16 of type <i>Datatype</i>
<i>REAL2</i>	Object REAL2 of type <i>Datatype</i>
<i>REAL4</i>	Object REAL4 of type <i>Datatype</i>
<i>REAL8</i>	Object REAL8 of type <i>Datatype</i>
<i>REAL16</i>	Object REAL16 of type <i>Datatype</i>
<i>COMPLEX4</i>	Object COMPLEX4 of type <i>Datatype</i>
<i>COMPLEX8</i>	Object COMPLEX8 of type <i>Datatype</i>
<i>COMPLEX16</i>	Object COMPLEX16 of type <i>Datatype</i>
<i>COMPLEX32</i>	Object COMPLEX32 of type <i>Datatype</i>
<i>UNSIGNED_INT</i>	Object UNSIGNED_INT of type <i>Datatype</i>
<i>SIGNED_SHORT</i>	Object SIGNED_SHORT of type <i>Datatype</i>
<i>SIGNED_INT</i>	Object SIGNED_INT of type <i>Datatype</i>
<i>SIGNED_LONG</i>	Object SIGNED_LONG of type <i>Datatype</i>
<i>SIGNED_LONG_LONG</i>	Object SIGNED_LONG_LONG of type <i>Datatype</i>
<i>BOOL</i>	Object BOOL of type <i>Datatype</i>
<i>SINT8_T</i>	Object SINT8_T of type <i>Datatype</i>
<i>SINT16_T</i>	Object SINT16_T of type <i>Datatype</i>
<i>SINT32_T</i>	Object SINT32_T of type <i>Datatype</i>
<i>SINT64_T</i>	Object SINT64_T of type <i>Datatype</i>
<i>F_BOOL</i>	Object F_BOOL of type <i>Datatype</i>
<i>F_INT</i>	Object F_INT of type <i>Datatype</i>
<i>F_FLOAT</i>	Object F_FLOAT of type <i>Datatype</i>
<i>F_DOUBLE</i>	Object F_DOUBLE of type <i>Datatype</i>
<i>F_COMPLEX</i>	Object F_COMPLEX of type <i>Datatype</i>
<i>F_FLOAT_COMPLEX</i>	Object F_FLOAT_COMPLEX of type <i>Datatype</i>
<i>F_DOUBLE_COMPLEX</i>	Object F_DOUBLE_COMPLEX of type <i>Datatype</i>
<i>REQUEST_NULL</i>	Object REQUEST_NULL of type <i>Request</i>
<i>MESSAGE_NULL</i>	Object MESSAGE_NULL of type <i>Message</i>
<i>MESSAGE_NO_PROC</i>	Object MESSAGE_NO_PROC of type <i>Message</i>
<i>OP_NULL</i>	Object OP_NULL of type <i>Op</i>
<i>MAX</i>	Object MAX of type <i>Op</i>
<i>MIN</i>	Object MIN of type <i>Op</i>
<i>SUM</i>	Object SUM of type <i>Op</i>
<i>PROD</i>	Object PROD of type <i>Op</i>
<i>LAND</i>	Object LAND of type <i>Op</i>
<i>BAND</i>	Object BAND of type <i>Op</i>
<i>LOR</i>	Object LOR of type <i>Op</i>
<i>BOR</i>	Object BOR of type <i>Op</i>
<i>LXOR</i>	Object LXOR of type <i>Op</i>
<i>BXOR</i>	Object BXOR of type <i>Op</i>

continues on next page

Table 1 – continued from previous page

<code>MAXLOC</code>	Object <code>MAXLOC</code> of type <code>Op</code>
<code>MINLOC</code>	Object <code>MINLOC</code> of type <code>Op</code>
<code>REPLACE</code>	Object <code>REPLACE</code> of type <code>Op</code>
<code>NO_OP</code>	Object <code>NO_OP</code> of type <code>Op</code>
<code>GROUP_NULL</code>	Object <code>GROUP_NULL</code> of type <code>Group</code>
<code>GROUP_EMPTY</code>	Object <code>GROUP_EMPTY</code> of type <code>Group</code>
<code>INFO_NULL</code>	Object <code>INFO_NULL</code> of type <code>Info</code>
<code>INFO_ENV</code>	Object <code>INFO_ENV</code> of type <code>Info</code>
<code>ERRHANDLER_NULL</code>	Object <code>ERRHANDLER_NULL</code> of type <code>Errhandler</code>
<code>ERRORS_RETURN</code>	Object <code>ERRORS_RETURN</code> of type <code>Errhandler</code>
<code>ERRORS_ARE_FATAL</code>	Object <code>ERRORS_ARE_FATAL</code> of type <code>Errhandler</code>
<code>COMM_NULL</code>	Object <code>COMM_NULL</code> of type <code>Comm</code>
<code>COMM_SELF</code>	Object <code>COMM_SELF</code> of type <code>Intracomm</code>
<code>COMM_WORLD</code>	Object <code>COMM_WORLD</code> of type <code>Intracomm</code>
<code>WIN_NULL</code>	Object <code>WIN_NULL</code> of type <code>Win</code>
<code>FILE_NULL</code>	Object <code>FILE_NULL</code> of type <code>File</code>
<code>pickle</code>	Object <code>pickle</code> of type <code>Pickle</code>

6 mpi4py.typing

New in version 4.0.0.

This module provides `type` aliases used to add type hints to the various functions and methods within the `MPI` module.

See also:

Module `typing`

Documentation of the `typing` standard module.

Types Summary

<code>SupportsBuffer</code>	Python buffer protocol.
<code>SupportsDLPack</code>	DLPack data interchange protocol.
<code>SupportsCAI</code>	CUDA Array Interface (CAI) protocol.
<code>Buffer</code>	Buffer-like object.
<code>Bottom</code>	Start of the address range.
<code>InPlace</code>	In-place buffer argument.
<code>Aint</code>	Address-sized integral type.
<code>Count</code>	Integral type for counts.
<code>Displ</code>	Integral type for displacements.
<code>Offset</code>	Integral type for offsets.
<code>TypeSpec</code>	Datatype specification.
<code>BufSpec</code>	Buffer specification.
<code>BufSpecB</code>	Buffer specification (block).
<code>BufSpecV</code>	Buffer specification (vector).
<code>BufSpecW</code>	Buffer specification (generalized).
<code>TargetSpec</code>	Target specification.

Types Documentation

`mpi4py.typing.SupportsBuffer = <class 'mpi4py.typing.SupportsBuffer'>`

Python buffer protocol.

See also:

[Buffer Protocol](#)

alias of `mpi4py.typing.SupportsBuffer`

`mpi4py.typing.SupportsDLPack = <class 'mpi4py.typing.SupportsDLPack'>`

DLPack data interchange protocol.

See also:

[Python Specification for DLPack](#)

alias of `mpi4py.typing.SupportsDLPack`

`mpi4py.typing.SupportsCAI = <class 'mpi4py.typing.SupportsCAI'>`

CUDA Array Interface (CAI) protocol.

See also:

[CUDA Array Interface \(Version 3\)](#)

alias of `mpi4py.typing.SupportsCAI`

`mpi4py.typing.Buffer`

Buffer-like object.

alias of `Union[SupportsBuffer, SupportsDLPack, SupportsCAI]`

`mpi4py.typing.Bottom`

Start of the address range.

alias of `Optional[BottomType]`

`mpi4py.typing.InPlace`

In-place buffer argument.

alias of `Optional[InPlaceType]`

`mpi4py.typing.Aint = <class 'numbers.Integral'>`

Address-sized integral type.

alias of `numbers.Integral`

`mpi4py.typing.Count = <class 'numbers.Integral'>`

Integral type for counts.

alias of `numbers.Integral`

`mpi4py.typing.Displ = <class 'numbers.Integral'>`

Integral type for displacements.

alias of `numbers.Integral`

`mpi4py.typing.Offset = <class 'numbers.Integral'>`

Integral type for offsets.

alias of `numbers.Integral`

mpi4py.typing.TypeSpec

Datatype specification.

alias of `Union[Datatype, str]`

mpi4py.typing.BufSpec

Buffer specification.

- `Buffer`
- `Tuple[Buffer, Count]`
- `Tuple[Buffer, TypeSpec]`
- `Tuple[Buffer, Count, TypeSpec]`
- `Tuple[Bottom, Count, Datatype]`

alias of `Union[SupportsBuffer, SupportsDLPack, SupportsCAI, Tuple[Union[SupportsBuffer, SupportsDLPack, SupportsCAI], Integral], Tuple[Union[SupportsBuffer, SupportsDLPack, SupportsCAI], Union[Datatype, str]], Tuple[Union[SupportsBuffer, SupportsDLPack, SupportsCAI], Integral, Union[Datatype, str]], Tuple[Optional[BottomType], Integral, Datatype], List[Any]]`

mpi4py.typing.BufSpecB

Buffer specification (block).

- `Buffer`
- `Tuple[Buffer, Count]`
- `Tuple[Buffer, TypeSpec]`
- `Tuple[Buffer, Count, TypeSpec]`

alias of `Union[SupportsBuffer, SupportsDLPack, SupportsCAI, Tuple[Union[SupportsBuffer, SupportsDLPack, SupportsCAI], Integral], Tuple[Union[SupportsBuffer, SupportsDLPack, SupportsCAI], Union[Datatype, str]], Tuple[Union[SupportsBuffer, SupportsDLPack, SupportsCAI], Integral, Union[Datatype, str]], List[Any]]`

mpi4py.typing.BufSpecV

Buffer specification (vector).

- `Buffer`
- `Tuple[Buffer, Sequence[Count]]`
- `Tuple[Buffer, Tuple[Sequence[Count], Sequence[Displ]]]`
- `Tuple[Buffer, TypeSpec]`
- `Tuple[Buffer, Sequence[Count], TypeSpec]`
- `Tuple[Buffer, Tuple[Sequence[Count], Sequence[Displ]], TypeSpec]`
- `Tuple[Buffer, Sequence[Count], Sequence[Displ], TypeSpec]`
- `Tuple[Bottom, Tuple[Sequence[Count], Sequence[Displ]], Datatype]`
- `Tuple[Bottom, Sequence[Count], Sequence[Displ], Datatype]`

alias of `Union[SupportsBuffer, SupportsDLPack, SupportsCAI, Tuple[Union[SupportsBuffer, SupportsDLPack, SupportsCAI], Sequence[Integral]], Tuple[Union[SupportsBuffer, SupportsDLPack, SupportsCAI], Tuple[Sequence[Integral], Sequence[Integral]]], Tuple[Union[SupportsBuffer, SupportsDLPack, SupportsCAI], Union[Datatype, str]]]`

```
Tuple[Union[SupportsBuffer, SupportsDLPack, SupportsCAI], Sequence[Integral], Union[Datatype, str]], Tuple[Union[SupportsBuffer, SupportsDLPack, SupportsCAI], Tuple[Sequence[Integral], Sequence[Integral]], Union[Datatype, str]], Tuple[Union[SupportsBuffer, SupportsDLPack, SupportsCAI], Sequence[Integral], Sequence[Integral], Union[Datatype, str]], Tuple[Optional[BottomType], Tuple[Sequence[Integral], Sequence[Integral], Datatype], Tuple[Optional[BottomType], Sequence[Integral], Sequence[Integral], Datatype], List[Any]]]
```

mpi4py.typing.BufSpecW

Buffer specification (generalized).

- Tuple[*Buffer*, Sequence[*Datatype*]]
- Tuple[*Buffer*, Tuple[Sequence[*Count*], Sequence[*Disp1*]], Sequence[*Datatype*]]
- Tuple[*Buffer*, Sequence[*Count*], Sequence[*Disp1*], Sequence[*Datatype*]]
- Tuple[*Bottom*, Tuple[Sequence[*Count*], Sequence[*Disp1*]], Sequence[*Datatype*]]
- Tuple[*Bottom*, Sequence[*Count*], Sequence[*Disp1*], Sequence[*Datatype*]]

```
alias of Union[Tuple[Union[SupportsBuffer, SupportsDLPack, SupportsCAI], Sequence[Datatype]], Tuple[Union[SupportsBuffer, SupportsDLPack, SupportsCAI], Tuple[Sequence[Integral], Sequence[Integral]], Sequence[Datatype]], Tuple[Union[SupportsBuffer, SupportsDLPack, SupportsCAI], Sequence[Integral], Sequence[Integral], Sequence[Datatype]], Tuple[Optional[BottomType], Tuple[Sequence[Integral], Sequence[Integral]], Sequence[Datatype]], Tuple[Optional[BottomType], Sequence[Integral], Sequence[Integral], Sequence[Datatype]], Sequence[Integral], Sequence[Datatype]], List[Any]]]
```

mpi4py.typing.TargetSpec

Target specification.

- *Disp1*
- Tuple[()]
- Tuple[*Disp1*]
- Tuple[*Disp1*, *Count*]
- Tuple[*Disp1*, *Count*, *Datatype*]

```
alias of Union[Integral, Tuple, Tuple[Integral], Tuple[Integral, Integral], Tuple[Integral, Integral, Union[Datatype, str]], List[Any]]
```

7 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.

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.

See also:

Module concurrent.futures

Documentation of the `concurrent.futures` standard module.

7.1 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 pickleable 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 `rmpy` 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` 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.
- *use_pk15*: If set to `True`, use `pickle5` with out-of-band buffers for interprocess communication. If *use_pk15* is set to `None` or not given, its value is determined from the `MPI4PY_FUTURES_USE_PKL5` environment variable. Using `pickle5` with out-of-band buffers may benefit applications dealing with large buffer-like objects like NumPy arrays. See `mpi4py.util.pk15` for additional information.
- *backoff*: `float` value specifying the maximum number of seconds a worker thread or process suspends execution with `time.sleep()` while idle-waiting. If not set, its value is determined from the `MPI4PY_FUTURES_BACKOFF` environment variable if set, otherwise the default value of 0.001 seconds is used. Lower values will reduce latency and increase execution throughput for very short-lived tasks, albeit at the expense of spinning CPU cores and increased energy consumption.

`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 `RuntimError`.

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.

`num_workers`

Number or worker processes in the pool.

`MPI4PY_FUTURES_MAX_WORKERS`

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

New in version 3.1.0.

`MPI4PY_FUTURES_USE_PKL5`

If the `use_pkl5` keyword argument to `MPIPoolExecutor` is `None` or not given, the `MPI4PY_FUTURES_USE_PKL5` environment variable provides a fallback value for whether the executor should use pickle5 with out-of-band buffers for interprocess communication. Accepted values are `0` and `1` (interpreted as `False` and `True`, respectively), and strings specifying a `YAML` boolean value (case-insensitive). Using pickle5 with out-of-band buffers may benefit applications dealing with large buffer-like objects like NumPy arrays. See `mpi4py.util.pkl5` for additional information.

New in version 4.0.0.

`MPI4PY_FUTURES_BACKOFF`

If the `backoff` keyword argument to `MPIPoolExecutor` is not given, the `MPI4PY_FUTURES_BACKOFF` environment variable can be set to a `float` value specifying the maximum number of seconds a worker thread or process

suspends execution with `time.sleep()` while idle-waiting. If not set, the default backoff value is 0.001 seconds. Lower values will reduce latency and increase execution throughput for very short-lived tasks, albeit at the expense of spinning CPU cores and increased energy consumption.

New in version 4.0.0.

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.

7.2 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 `mpieexec` 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 progresses 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 processes one task at a time. We advise against using `MPICommExecutor` with communicators of size one and suggest refactoring your code to use instead a `ThreadPoolExecutor`.

7.3 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 spawns additional workers as required. However, there is a very important difference users should be aware of. All `MPIPoolExecutor` instances created at the master will share the pool of workers. Tasks submitted at the master from many different executors will be scheduled for execution in random order as soon as a worker is idle. Any executor can easily starve all the workers (e.g., by calling `MPIPoolExecutor.map()` with long iterables). If that ever happens, submissions from other executors will not be serviced until free workers are available.

See also:

Command line

Documentation on Python command line interface.

7.4 Parallel tasks

The `mpi4py.futures` package favors an embarrassingly parallel execution model involving a series of sequential tasks independent of each other and executed asynchronously. Albeit unnatural, `MPIPoolExecutor` can still be used for handling workloads involving parallel tasks, where worker processes communicate and coordinate each other via MPI.

`mpi4py.futures.get_comm_workers()`

Access an intracommunicator grouping MPI worker processes.

Executing parallel tasks with `mpi4py.futures` requires following some rules, cf. highlighted lines in example `cpi.py`:

- Use `MPIPoolExecutor.num_workers` to determine the number of worker processes in the executor and **submit exactly one callable per worker process** using the `MPIPoolExecutor.submit()` method.
- The submitted callable must use `get_comm_workers()` to access an intracommunicator grouping MPI worker processes. Afterwards, it is highly recommended calling the `Barrier()` method on the communicator. The barrier synchronization ensures that every worker process is executing the submitted callable exactly once. Afterwards, the parallel task can safely perform any kind of point-to-point or collective operation using the returned communicator.
- The `Future` instances returned by `MPIPoolExecutor.submit()` should be collected in a sequence. Use `wait()` with the sequence of `Future` instances to ensure logical completion of the parallel task.

7.5 Utilities

The `mpi4py.futures` package provides additional utilities for handling `Future` instances.

`mpi4py.futures.collect(fs)`

Gather a collection of futures in a new future.

Parameters

`fs` – Collection of futures.

Returns

New future producing as result a list with results from `fs`.

`mpi4py.futures.compose(future, resulthook=None, excepthook=None)`

Compose the completion of a future with result and exception handlers.

Parameters

- **future** – Input future instance.
- **resulthook** – Function to be called once the input future completes with success. Once the input future finish running with success, its result value is the input argument for `resulthook`. The result of `resulthook` is set as the result of the output future. If `resulthook` is `None`, the output future is completed directly with the result of the input future.
- **excepthook** – Function to be called once the input future completes with failure. Once the input future finish running with failure, its exception value is the input argument for `excepthook`. If `excepthook` returns an `Exception` instance, it is set as the exception of the output future. Otherwise, the result of `excepthook` is set as the result of the output future. If `excepthook` is `None`, the output future is set as failed with the exception from the input future.

Returns

Output future instance to be completed once the input future is completed and either `resulthook` or `excepthook` finish executing.

7.6 Examples

Computing the Julia set

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
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

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.

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:

```
$ env 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 localhost: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`:

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

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

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

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

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

Computing Pi (parallel task)

The number π can be approximated via numerical integration with the simple midpoint rule, that is:

$$\pi = \int_0^1 \frac{4}{1+x^2} dx \approx \frac{1}{n} \sum_{i=1}^n \frac{4}{1 + \left[\frac{1}{n} \left(i - \frac{1}{2} \right) \right]^2}.$$

The following `cpi.py` script computes such approximations using `mpi4py.futures` with a parallel task involving a collective reduction operation. Highlighted lines correspond to the rules discussed in [Parallel tasks](#).

Listing 2: `cpi.py`

```

1 import math
2 import sys
3 from mpi4py.futures import MPIPoolExecutor, wait
4 from mpi4py.futures import get_comm_workers
5
6
7 def compute_pi(n):
8     # Access intracommunicator and synchronize
9     comm = get_comm_workers()
10    comm.Barrier()
```

(continues on next page)

```

11
12     rank = comm.Get_rank()
13     size = comm.Get_size()
14
15     # Local computation
16     h = 1.0 / n
17     s = 0.0
18     for i in range(rank + 1, n + 1, size):
19         x = h * (i - 0.5)
20         s += 4.0 / (1.0 + x**2)
21     pi_partial = s * h
22
23     # Parallel reduce-to-all
24     pi = comm.allreduce(pi_partial)
25
26     # All workers return the same value
27     return pi
28
29
30 if __name__ == '__main__':
31     n = int(sys.argv[1]) if len(sys.argv) > 1 else 256
32
33     with MPIPoolExecutor() as executor:
34         # Submit exactly one callable per worker
35         P = executor.num_workers
36         fs = [executor.submit(compute_pi, n) for _ in range(P)]
37
38         # Wait for all workers to finish
39         wait(fs)
40
41         # Get result from the first future object.
42         # In this particular example, due to using reduce-to-all,
43         # all the other future objects hold the same result value.
44         pi = fs[0].result()
45         print(
46             f"pi: {pi:.16f}, error: {abs(pi - math.pi):.3e}",
47             f"({n:d} intervals, {P:d} workers)",
48         )

```

To run in modern MPI-2 mode:

```

$ env MPI4PY_FUTURES_MAX_WORKERS=4 mpiexec -n 1 python cpi.py 128
pi: 3.1415977398528137, error: 5.086e-06 (128 intervals, 4 workers)

$ env MPI4PY_FUTURES_MAX_WORKERS=8 mpiexec -n 1 python cpi.py 512
pi: 3.1415929714812316, error: 3.179e-07 (512 intervals, 8 workers)

```

To run in legacy MPI-1 mode:

```

$ mpiexec -n 5 python -m mpi4py.futures cpi.py 128
pi: 3.1415977398528137, error: 5.086e-06 (128 intervals, 4 workers)

```

```
$ mpiexec -n 9 python -m mpi4py.futures cpi.py 512
pi: 3.1415929714812316, error: 3.179e-07 (512 intervals, 8 workers)
```

7.7 Citation

If `mpi4py.futures` been significant to a project that leads to an academic publication, please acknowledge our work by citing the following article [[mpi4py-futures](#)]:

8 mpi4py.util

New in version 3.1.0.

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

8.1 mpi4py.util.dtlib

New in version 3.1.0.

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

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

Convert NumPy datatype to MPI datatype.

Parameters

`dtype (DTypeLike)` – NumPy dtype-like object.

Return type

`Datatype`

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

Convert MPI datatype to NumPy datatype.

Parameters

`datatype (Datatype)` – MPI datatype.

Return type

`dtype[Any]`

8.2 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 and collective communication methods using pickle protocol 5. Handling out-of-band buffers necessarily involves 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.pk15.Request`

Request.

Custom request class for nonblocking communications.

Note: `Request` is not a subclass of `mpi4py.MPI.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 (Status / None) –`

Return type

`bool`

`test(status=None)`

Test for the completion of a request.

Parameters

`status (Status / None) –`

Return type

`tuple[bool, Any | None]`

`wait(status=None)`

Wait for a request to complete.

Parameters

`status (Status / None) –`

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.pk15.Message
```

Message.

Custom message class for matching probes.

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

```
recv(status=None)
```

Blocking receive of matched message.

Parameters

status (*Status* / *None*) –

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.pk15.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** (`Buffer` / `None`) –
- **source** (`int`) –
- **tag** (`int`) –
- **status** (`Status` / `None`) –

Return type

`Any`

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

Nonblocking receive.

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

Parameters

- **buf** (`Buffer` / `None`) –
- **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** (`Buffer` / `None`) –
- **source** (`int`) –
- **recvtag** (`int`) –
- **status** (`Status` / `None`) –

Return type

`Any`

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

Blocking test for a matched message.

Parameters

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

- **status** (`Status` / `None`) –

Return type

`Message`

improbe(*source*=`ANY_SOURCE`, *tag*=`ANY_TAG`, *status*=`None`)

Nonblocking test for a matched message.

Parameters

- **source** (`int`) –
- **tag** (`int`) –
- **status** (`Status` / `None`) –

Return type

`Message` | `None`

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

Broadcast.

New in version 3.1.0.

Parameters

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

Return type

`Any`

gather(*sendobj*, *root*=0)

Gather.

New in version 4.0.0.

Parameters

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

Return type

`list[Any]` | `None`

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

Scatter.

New in version 4.0.0.

Parameters

- **sendobj** (`Sequence[Any]` / `None`) –
- **root** (`int`) –

Return type

`Any`

allgather(*sendobj*)

Gather to All.

New in version 4.0.0.

Parameters
sendobj (*Any*) –

Return type
list[*Any*]

alltoall(*sendobj*)
All to All Scatter/Gather.
New in version 4.0.0.

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

Return type
list[*Any*]

```
class mpi4py.util.pk15.Intracomm
    Intracommunicator.
    Intracommunicator wrapper class.

class mpi4py.util.pk15.Intercomm
    Intercommunicator.
    Intercommunicator wrapper class.
```

Examples

Listing 3: test-pk15-1.py

```
1 import numpy as np
2 from mpi4py import MPI
3 from mpi4py.util import pk15
4
5 comm = pk15.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 4: test-pk15-2.py

```
1 import numpy as np
2 from mpi4py import MPI
3 from mpi4py.util import pk15
4
5 comm = pk15.Intracomm(MPI.COMM_WORLD) # comm wrapper
```

(continues on next page)

```

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

```

8.3 mpi4py.util.pool

New in version 4.0.0.

See also:

This module intends to be a drop-in replacement for the `multiprocessing.Pool` interface from the Python standard library. The `Pool` class exposed here is implemented as a thin wrapper around `MPIPoolExecutor`.

Note: The `mpi4py.futures` package offers a higher level interface for asynchronously pushing tasks to MPI worker process, allowing for a clear separation between submitting tasks and waiting for the results.

`class mpi4py.util.pool.Pool`
 Pool using MPI processes as workers.

`__init__(processes=None, initializer=None, initargs=(), **kwargs)`

Initialize a new Pool instance.

Parameters

- `processes` – Number of worker processes.
- `initializer` – An callable used to initialize workers processes.
- `initargs` – A tuple of arguments to pass to the initializer.

Note: Additional keyword arguments are passed down to the `MPIPoolExecutor` constructor.

Warning: The `maxtasksperchild` and `context` arguments of `multiprocessing.pool.Pool` are not supported. Specifying `maxtasksperchild` or `context` with a value other than `None` will issue a warning of category `UserWarning`.

apply(*func, args=(), kwds={}*)

Call *func* with arguments *args* and keyword arguments *kwds*.

Equivalent to *func(*args, **kwds)*.

apply_async(*func, args=(), kwds={}, callback=None, error_callback=None*)

Asynchronous version of [apply\(\)](#) returning [ApplyResult](#).

map(*func, iterable, chunkszie=None*)

Apply *func* to each element in *iterable*.

Equivalent to *list(map(func, iterable))*.

Block until all results are ready and return them in a [list](#).

The *iterable* is chopped into a number of chunks which are submitted as separate tasks. The (approximate) size of these chunks can be specified by setting *chunkszie* to a positive integer.

Consider using [imap\(\)](#) or [imap_unordered\(\)](#) with explicit *chunkszie* for better efficiency.

map_async(*func, iterable, chunkszie=None, callback=None, error_callback=None*)

Asynchronous version of [map\(\)](#) returning [MapResult](#).

imap(*func, iterable, chunkszie=1*)

Like [map\(\)](#) but return an [iterator](#).

Equivalent to *map(func, iterable)*.

imap_unordered(*func, iterable, chunkszie=1*)

Like [imap\(\)](#) but ordering of results is arbitrary.

starmap(*func, iterable, chunkszie=None*)

Apply *func* to each argument tuple in *iterable*.

Equivalent to *list(itertools.starmap(func, iterable))*.

Block until all results are ready and return them in a [list](#).

The *iterable* is chopped into a number of chunks which are submitted as separate tasks. The (approximate) size of these chunks can be specified by setting *chunkszie* to a positive integer.

Consider using [istarmap\(\)](#) or [istarmap_unordered\(\)](#) with explicit *chunkszie* for better efficiency.

starmap_async(*func, iterable, chunkszie=None, callback=None, error_callback=None*)

Asynchronous version of [starmap\(\)](#) returning [MapResult](#).

istarmap(*func, iterable, chunkszie=1*)

Like [starmap\(\)](#) but return an [iterator](#).

Equivalent to *itertools.starmap(func, iterable)*.

istarmap_unordered(*func, iterable, chunkszie=1*)

Like [istarmap\(\)](#) but ordering of results is arbitrary.

close()

Prevent any more tasks from being submitted to the pool.

terminate()

Stop the worker processes without completing pending tasks.

join()

Wait for the worker processes to exit.

```

class mpi4py.util.pool.ThreadPool
    Bases: Pool

    Pool using threads as workers.

class mpi4py.util.pool.AsyncResult
    Asynchronous result.

    get(timeout=None)
        Return the result when it arrives.

        If timeout is not None and the result does not arrive within timeout seconds then raise TimeoutError.
        If the remote call raised an exception then that exception will be reraised.

    wait(timeout=None)
        Wait until the result is available or timeout seconds pass.

    ready()
        Return whether the call has completed.

    successful()
        Return whether the call completed without raising an exception.

        If the result is not ready then raise ValueError.

class mpi4py.util.pool.ApplyResult
    Bases: AsyncResult

    Result type of apply_async().

class mpi4py.util.pool.MapResult
    Bases: AsyncResult

    Result type of map_async() and starmap_async().

```

9 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.

9.1 Exceptions and deadlocks

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.

Listing 5: `deadlock.py`

```
1 from mpi4py import MPI
2 assert MPI.COMM_WORLD.Get_size() > 1
3 rank = MPI.COMM_WORLD.Get_rank()
4 if rank == 0:
5     1/0
6     MPI.COMM_WORLD.send(None, dest=1, tag=42)
7 elif rank == 1:
8     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 irredeemably 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.

9.2 Command line

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

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

`<pyfile>`

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

`-m <mod>`

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

`-c <cmd>`

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

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

See also:

Command line

Documentation on Python command line interface.

10 mpi4py.bench

New in version 3.0.0.

11 Reference

<code>mpi4py.MPI</code>	Message Passing Interface.
-------------------------	----------------------------

11.1 mpi4py.MPI

Message Passing Interface.

Classes

<code>BottomType</code>	Type of <code>BOTTOM</code> .
<code>BufferAutomaticType</code>	Type of <code>BUFFER_AUTOMATIC</code> .
<code>Cartcomm</code>	Cartesian topology intracommunicator.
<code>Comm</code>	Communication context.
<code>Datatype</code>	Datatype object.
<code>Distgraphcomm</code>	Distributed graph topology intracommunicator.
<code>Errhandler</code>	Error handler.
<code>File</code>	File I/O context.
<code>Graphcomm</code>	General graph topology intracommunicator.
<code>Grequest</code>	Generalized request handler.
<code>Group</code>	Group of processes.
<code>InPlaceType</code>	Type of <code>IN_PLACE</code> .
<code>Info</code>	Info object.
<code>Intercomm</code>	Intercommunicator.
<code>Intracomm</code>	Intracommunicator.
<code>Message</code>	Matched message.
<code>Op</code>	Reduction operation.
<code>Pickle</code>	Pickle/unpickle Python objects.
<code>Prequest</code>	Persistent request handler.
<code>Request</code>	Request handler.
<code>Session</code>	Session context.
<code>Status</code>	Status object.
<code>Topocomm</code>	Topology intracommunicator.
<code>Win</code>	Remote memory access context.
<code>buffer</code>	Buffer.
<code>memory</code>	alias of <code>buffer</code>

mpi4py.MPI.BottomType

```
class mpi4py.MPI.BottomType
```

Bases: `int`

Type of `BOTTOM`.

```
static __new__(cls)
```

Return type

Self

mpi4py.MPI.BufferAutomaticType

```
class mpi4py.MPI.BufferAutomaticType
```

Bases: `int`

Type of `BUFFER_AUTOMATIC`.

```
static __new__(cls)
```

Return type

Self

mpi4py.MPI.Cartcomm

```
class mpi4py.MPI.Cartcomm
```

Bases: `Topocomm`

Cartesian topology intracommunicator.

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

Parameters

`comm` (`Cartcomm` / `None`) –

Return type

Self

Methods Summary

<code>Get_cart_rank(coords)</code>	Translate logical coordinates to ranks.
<code>Get_coords(rank)</code>	Translate ranks to logical coordinates.
<code>Get_dim()</code>	Return number of dimensions.
<code>Get_topo()</code>	Return information on the cartesian topology.
<code>Shift(direction, disp)</code>	Return a process ranks for data shifting with <code>Sendrecv</code> .
<code>Sub(remain_dims)</code>	Return a lower-dimensional Cartesian topology.

Attributes Summary

<code>coords</code>	Coordinates.
<code>dim</code>	Number of dimensions.
<code>dims</code>	Dimensions.
<code>ndim</code>	Number of dimensions.
<code>periods</code>	Periodicity.
<code>topo</code>	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 process ranks for data shifting with `Sendrecv`.

Parameters

- `direction` (`int`) –
- `disp` (`int`) –

Return type

`tuple[int, int]`

`Sub(remain_dims)`

Return a lower-dimensional Cartesian topology.

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

Bases: `object`

Communication context.

static __new__(cls, comm=None)

Parameters

`comm` (`Comm` / `None`) –

Return type

`Self`

Methods Summary

<code>Abort([errorcode])</code>	Terminate the MPI execution environment.
<code>Ack_failed([num_to_ack])</code>	Acknowledge failures on a communicator.
<code>Agree(flag)</code>	Blocking agreement.
<code>Allgather(sendbuf, recvbuf)</code>	Gather to All.
<code>Allgather_init(sendbuf, recvbuf[, info])</code>	Persistent Gather to All.
<code>Allgatherv(sendbuf, recvbuf)</code>	Gather to All Vector.
<code>Allgatherv_init(sendbuf, recvbuf[, info])</code>	Persistent Gather to All Vector.
<code>Allreduce(sendbuf, recvbuf[, op])</code>	Reduce to All.
<code>Allreduce_init(sendbuf, recvbuf[, op, info])</code>	Persistent Reduce to All.
<code>Alltoall(sendbuf, recvbuf)</code>	All to All Scatter/Gather.
<code>Alltoall_init(sendbuf, recvbuf[, info])</code>	Persistent All to All Scatter/Gather.
<code>Alltoallv(sendbuf, recvbuf)</code>	All to All Scatter/Gather Vector.

continues on next page

Table 2 – continued from previous page

<code>Alltoallv_init</code> (sendbuf, recvbuf[, info])	Persistent All to All Scatter/Gather Vector.
<code>Alltoallw</code> (sendbuf, recvbuf)	All to All Scatter/Gather General.
<code>Alltoallw_init</code> (sendbuf, recvbuf[, info])	Persistent All to All Scatter/Gather General.
<code>Attach_buffer</code> (buf)	Attach a user-provided buffer for sending in buffered mode.
<code>Barrier()</code>	Barrier synchronization.
<code>Barrier_init</code> ([info])	Persistent Barrier.
<code>Bcast</code> (buf[, root])	Broadcast data from one process to all other processes.
<code>Bcast_init</code> (buf[, root, info])	Persistent Broadcast.
<code>Bsend</code> (buf, dest[, tag])	Blocking send in buffered mode.
<code>Bsend_init</code> (buf, dest[, tag])	Persistent request for a send in buffered mode.
<code>Call_errhandler</code> (errorcode)	Call the error handler installed on a communicator.
<code>Clone()</code>	Clone an existing communicator.
<code>Compare</code> (comm)	Compare two communicators.
<code>Create</code> (group)	Create communicator from group.
<code>Create_errhandler</code> (errhandler_fn)	Create a new error handler for communicators.
<code>Create_keyval</code> ([copy_fn, delete_fn, nopython])	Create a new attribute key for communicators.
<code>Delete_attr</code> (keyval)	Delete attribute value associated with a key.
<code>Detach_buffer</code> ()	Remove an existing attached buffer.
<code>Disconnect()</code>	Disconnect from a communicator.
<code>Dup</code> ([info])	Duplicate a communicator.
<code>Dup_with_info</code> (info)	Duplicate a communicator with hints.
<code>Flush_buffer</code> ()	Block until all buffered messages have been transmitted.
<code>Free()</code>	Free a communicator.
<code>Free_keyval</code> (keyval)	Free an attribute key for communicators.
<code>Gather</code> (sendbuf, recvbuf[, root])	Gather data to one process from all other processes.
<code>Gather_init</code> (sendbuf, recvbuf[, root, info])	Persistent Gather.
<code>Gatherv</code> (sendbuf, recvbuf[, root])	Gather Vector.
<code>Gatherv_init</code> (sendbuf, recvbuf[, root, info])	Persistent Gather Vector.
<code>Get_attr</code> (keyval)	Retrieve attribute value by key.
<code>Get_errhandler</code> ()	Get the error handler for a communicator.
<code>Get_failed</code> ()	Extract the group of failed processes.
<code>Get_group</code> ()	Access the group associated with a communicator.
<code>Get_info</code> ()	Return the current hints for a communicator.
<code>Get_name</code> ()	Get the print name for this communicator.
<code>Get_parent</code> ()	Return the parent intercommunicator for this process.
<code>Get_rank</code> ()	Return the rank of this process in a communicator.
<code>Get_size</code> ()	Return the number of processes in a communicator.
<code>Get_topology</code> ()	Return the type of topology (if any) associated with a communicator.
<code>Iagree</code> (flag)	Nonblocking agreement.
<code>Iallgather</code> (sendbuf, recvbuf)	Nonblocking Gather to All.
<code>Iallgatherv</code> (sendbuf, recvbuf)	Nonblocking Gather to All Vector.
<code>Iallreduce</code> (sendbuf, recvbuf[, op])	Nonblocking Reduce to All.
<code>Ialltoall</code> (sendbuf, recvbuf)	Nonblocking All to All Scatter/Gather.
<code>Ialltoallv</code> (sendbuf, recvbuf)	Nonblocking All to All Scatter/Gather Vector.
<code>Ialltoallw</code> (sendbuf, recvbuf)	Nonblocking All to All Scatter/Gather General.
<code>Ibarrier</code> ()	Nonblocking Barrier.
<code>Ibcast</code> (buf[, root])	Nonblocking Broadcast.
<code>Ibsend</code> (buf, dest[, tag])	Nonblocking send in buffered mode.

continues on next page

Table 2 – continued from previous page

<i>Idup</i> ([info])	Nonblocking duplicate a communicator.
<i>Idup_with_info</i> (info)	Nonblocking duplicate a communicator with hints.
<i>Iflush_buffer</i> ()	Nonblocking flush for buffered messages.
<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).
<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> ()	Return whether the communicator is an intercommunicator.
<i>Is_intra</i> ()	Return whether the communicator is an intracommmunicator.
<i>Is_revoked</i> ()	Indicate whether the communicator has been revoked.
<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>Isendrecv</i> (sendbuf, dest[, sendtag, recvbuf, ...])	Nonblocking send and receive.
<i>Isendrecv_replace</i> (buf, dest[, sendtag, ...])	Send and receive a message.
<i>Ishrink</i> ()	Nonblocking shrink a communicator to remove all failed processes.
<i>Issend</i> (buf, dest[, tag])	Nonblocking send in synchronous mode.
<i>Join</i> (fd)	Interconnect two processes connected by a socket.
<i>Mprobe</i> ([source, tag, status])	Blocking test for a matched message.
<i>Precv_init</i> (buf, partitions[, source, tag, info])	Create request for a partitioned recv operation.
<i>Probe</i> ([source, tag, status])	Blocking test for a message.
<i>Psend_init</i> (buf, partitions, dest[, tag, info])	Create request for a partitioned send operation.
<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_init</i> (sendbuf, recvbuf[, op, root, info])	Persistent 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>Reduce_scatter_block_init</i> (sendbuf, recvbuf)	Persistent Reduce-Scatter Block (regular, non-vector version).
<i>Reduce_scatter_init</i> (sendbuf, recvbuf[, ...])	Persistent Reduce-Scatter (vector version).
<i>Revoke</i> ()	Revoke a communicator.
<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.
<i>Scatter_init</i> (sendbuf, recvbuf[, root, info])	Persistent Scatter.
<i>Scatterv</i> (sendbuf, recvbuf[, root])	Scatter Vector.
<i>Scatterv_init</i> (sendbuf, recvbuf[, root, info])	Persistent Scatter Vector.
<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.

continues on next page

Table 2 – continued from previous page

<code>Set_attr(keyval, attrval)</code>	Store attribute value associated with a key.
<code>Set_errhandler(errhandler)</code>	Set the error handler for a communicator.
<code>Set_info(info)</code>	Set new values for the hints associated with a communicator.
<code>Set_name(name)</code>	Set the print name for this communicator.
<code>Shrink()</code>	Shrink a communicator to remove all failed processes.
<code>Split([color, key])</code>	Split communicator by color and key.
<code>Split_type(split_type[, key, info])</code>	Split communicator by split type.
<code>Ssend(buf, dest[, tag])</code>	Blocking send in synchronous mode.
<code>Ssend_init(buf, dest[, tag])</code>	Persistent request for a send in synchronous mode.
<code>allgather(sendobj)</code>	Gather to All.
<code>allreduce(sendobj[, op])</code>	Reduce to All.
<code>alltoall(sendobj)</code>	All to All Scatter/Gather.
<code>barrier()</code>	Barrier synchronization.
<code>bcast(obj[, root])</code>	Broadcast.
<code>bsend(obj, dest[, tag])</code>	Send in buffered mode.
<code>f2py(arg)</code>	
<code>fromhandle(handle)</code>	Create object from MPI handle.
<code>gather(sendobj[, root])</code>	Gather.
<code>ibsend(obj, dest[, tag])</code>	Nonblocking send in buffered mode.
<code>improbe([source, tag, status])</code>	Nonblocking test for a matched message.
<code>iprobe([source, tag, status])</code>	Nonblocking test for a message.
<code>irecv([buf, source, tag])</code>	Nonblocking receive.
<code>isend(obj, dest[, tag])</code>	Nonblocking send.
<code>issend(obj, dest[, tag])</code>	Nonblocking send in synchronous mode.
<code>mprobe([source, tag, status])</code>	Blocking test for a matched message.
<code>probe([source, tag, status])</code>	Blocking test for a message.
<code>py2f()</code>	
<code>recv([buf, source, tag, status])</code>	Receive.
<code>reduce(sendobj[, op, root])</code>	Reduce to Root.
<code>scatter(sendobj[, root])</code>	Scatter.
<code>send(obj, dest[, tag])</code>	Send in standard mode.
<code>sendrecv(sendobj, dest[, sendtag, recvbuf, ...])</code>	Send and Receive.
<code>ssend(obj, dest[, tag])</code>	Send in synchronous mode.

Attributes Summary

<code>group</code>	Group.
<code>handle</code>	MPI handle.
<code>info</code>	Info hints.
<code>is_inter</code>	Is intercommunicator.
<code>is_intra</code>	Is intracommunicator.
<code>is_topo</code>	Is a topology.
<code>name</code>	Print name.
<code>rank</code>	Rank of this process.
<code>size</code>	Number of processes.
<code>topology</code>	Topology type.

Methods Documentation

Abort(*errorcode*=0)

Terminate the MPI execution environment.

Warning: The invocation of this method prevents the execution of various Python exit and cleanup mechanisms. Use this method as a last resort to prevent parallel deadlocks in case of unrecoverable errors.

Parameters

errorcode (*int*) –

Return type

NoReturn

Ack_failed(*num_to_ack*=None)

Acknowledge failures on a communicator.

Parameters

num_to_ack (*int* / *None*) –

Return type

int

Agree(*flag*)

Blocking agreement.

Parameters

flag (*int*) –

Return type

int

Allgather(*sendbuf*, *recvbuf*)

Gather to All.

Gather data from all processes and broadcast the combined data to all other processes.

Parameters

- **sendbuf** (*BufSpec* / *InPlace*) –
- **recvbuf** (*BufSpecB*) –

Return type

None

Allgather_init(*sendbuf*, *recvbuf*, *info*=INFO_NULL)

Persistent Gather to All.

Parameters

- **sendbuf** (*BufSpec* / *InPlace*) –
- **recvbuf** (*BufSpecB*) –
- **info** (*Info*) –

Return type

Request

Allgatherv(*sendbuf*, *recvbuf*)

Gather to All Vector.

Gather data from all processes and send it to all other processes providing different amounts of data and displacements.

Parameters

- **sendbuf** ([BufSpec](#) / [InPlace](#)) –
- **recvbuf** ([BufSpecV](#)) –

Return type

[None](#)

Allgatherv_init(*sendbuf*, *recvbuf*, *info*=[INFO_NULL](#))

Persistent Gather to All Vector.

Parameters

- **sendbuf** ([BufSpec](#) / [InPlace](#)) –
- **recvbuf** ([BufSpecV](#)) –
- **info** ([Info](#)) –

Return type

[Prequest](#)

Allreduce(*sendbuf*, *recvbuf*, *op*=[SUM](#))

Reduce to All.

Parameters

- **sendbuf** ([BufSpec](#) / [InPlace](#)) –
- **recvbuf** ([BufSpec](#)) –
- **op** ([Op](#)) –

Return type

[None](#)

Allreduce_init(*sendbuf*, *recvbuf*, *op*=[SUM](#), *info*=[INFO_NULL](#))

Persistent Reduce to All.

Parameters

- **sendbuf** ([BufSpec](#) / [InPlace](#)) –
- **recvbuf** ([BufSpec](#)) –
- **op** ([Op](#)) –
- **info** ([Info](#)) –

Return type

[Prequest](#)

Alltoall(*sendbuf*, *recvbuf*)

All to All Scatter/Gather.

Send data to all processes and recv data from all processes.

Parameters

- **sendbuf** ([BufSpecB](#) / [InPlace](#)) –

- **recvbuf** (`BufSpecB`) –

Return type

`None`

Alltoall_init(*sendbuf*, *recvbuf*, *info*=`INFO_NULL`)

Persistent All to All Scatter/Gather.

Parameters

- **sendbuf** (`BufSpecB` / `InPlace`) –
- **recvbuf** (`BufSpecB`) –
- **info** (`Info`) –

Return type

`Prequest`

Alltoallv(*sendbuf*, *recvbuf*)

All to All Scatter/Gather Vector.

Send data to all processes and recv data from all processes providing different amounts of data and displacements.

Parameters

- **sendbuf** (`BufSpecV` / `InPlace`) –
- **recvbuf** (`BufSpecV`) –

Return type

`None`

Alltoallv_init(*sendbuf*, *recvbuf*, *info*=`INFO_NULL`)

Persistent All to All Scatter/Gather Vector.

Parameters

- **sendbuf** (`BufSpecV` / `InPlace`) –
- **recvbuf** (`BufSpecV`) –
- **info** (`Info`) –

Return type

`Prequest`

Alltoallw(*sendbuf*, *recvbuf*)

All to All Scatter/Gather General.

Send/recv data to/from all processes allowing the specification of different counts, displacements, and datatypes for each dest/source.

Parameters

- **sendbuf** (`BufSpecW` / `InPlace`) –
- **recvbuf** (`BufSpecW`) –

Return type

`None`

Alltoallw_init(*sendbuf*, *recvbuf*, *info*=*INFO_NULL*)

Persistent All to All Scatter/Gather General.

Parameters

- **sendbuf** (*BufSpecW* / *InPlace*) –
- **recvbuf** (*BufSpecW*) –
- **info** (*Info*) –

Return type

Prequest

Attach_buffer(*buf*)

Attach a user-provided buffer for sending in buffered mode.

Parameters

- **buf** (*Buffer* / *None*) –

Return type

None

Barrier()

Barrier synchronization.

Return type

None

Barrier_init(*info*=*INFO_NULL*)

Persistent Barrier.

Parameters

- **info** (*Info*) –

Return type

Prequest

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

Broadcast data from one process to all other processes.

Parameters

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

Return type

None

Bcast_init(*buf*, *root*=0, *info*=*INFO_NULL*)

Persistent Broadcast.

Parameters

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

Return type

Prequest

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

Self

Compare(*comm*)

Compare two communicators.

Parameters

comm ([Comm](#)) –

Return type

[int](#)

Create(*group*)

Create communicator from group.

Parameters

group ([Group](#)) –

Return type

[Comm](#)

classmethod Create_errhandler(*errhandler_fn*)

Create a new error handler for communicators.

Parameters

errhandler_fn (*Callable*[*Comm*, *int*], *None*) –

Return type

Errhandler

classmethod Create_keyval(*copy_fn=None*, *delete_fn=None*, *nopython=False*)

Create a new attribute key for communicators.

Parameters

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

Return type

int

Delete_attr(*keyval*)

Delete attribute value associated with a key.

Parameters

keyval (*int*) –

Return type

None

Detach_buffer()

Remove an existing attached buffer.

Return type

Buffer | *None*

Disconnect()

Disconnect from a communicator.

Return type

None

Dup(*info=None*)

Duplicate a communicator.

Parameters

info (*Info* / *None*) –

Return type

Self

Dup_with_info(*info*)

Duplicate a communicator with hints.

Parameters

info (*Info*) –

Return type

Self

Flush_buffer()

Block until all buffered messages have been transmitted.

Return type

None

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 data to one process from all other processes.

Parameters

- **sendbuf** (*BufSpec* / *InPlace*) –
- **recvbuf** (*BufSpecB* / *None*) –
- **root** (*int*) –

Return type

None

Gather_init(sendbuf, recvbuf, root=0, info=INFO_NULL)

Persistent Gather.

Parameters

- **sendbuf** (*BufSpec* / *InPlace*) –
- **recvbuf** (*BufSpecB* / *None*) –
- **root** (*int*) –
- **info** (*Info*) –

Return type

Prequest

Gatherv(sendbuf, recvbuf, root=0)

Gather Vector.

Gather data to one process from all other processes providing different amounts of data and displacements.

Parameters

- **sendbuf** (*BufSpec* / *InPlace*) –
- **recvbuf** (*BufSpecV* / *None*) –
- **root** (*int*) –

Return type

None

Gatherv_init(*sendbuf*, *recvbuf*, *root*=0, *info*=INFO_NULL)

Persistent Gather Vector.

Parameters

- **sendbuf** ([BufSpec](#) / [InPlace](#)) –
- **recvbuf** ([BufSpecV](#) / [None](#)) –
- **root** ([int](#)) –
- **info** ([Info](#)) –

Return type

[Request](#)

Get_attr(*keyval*)

Retrieve attribute value by key.

Parameters

- keyval** ([int](#)) –

Return type

[int](#) | [Any](#) | [None](#)

Get_errhandler()

Get the error handler for a communicator.

Return type

[Errhandler](#)

Get_failed()

Extract the group of failed processes.

Return type

[Group](#)

Get_group()

Access the group associated with a communicator.

Return type

[Group](#)

Get_info()

Return the current hints for a communicator.

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()

Return the type of topology (if any) associated with a communicator.

Return type

int

Iagree(*flag*)

Nonblocking agreement.

Parameters

flag ([Buffer](#)) –

Return type

Request

Iallgather(*sendbuf*, *recvbuf*)

Nonblocking Gather to All.

Parameters

- **sendbuf** ([BufSpec](#) / [InPlace](#)) –
- **recvbuf** ([BufSpecB](#)) –

Return type

Request

Iallgatherv(*sendbuf*, *recvbuf*)

Nonblocking Gather to All Vector.

Parameters

- **sendbuf** ([BufSpec](#) / [InPlace](#)) –
- **recvbuf** ([BufSpecV](#)) –

Return type

Request

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

Nonblocking Reduce to All.

Parameters

- **sendbuf** ([BufSpec](#) / [InPlace](#)) –
- **recvbuf** ([BufSpec](#)) –
- **op** ([Op](#)) –

Return type

Request

Ialltoall(*sendbuf*, *recvbuf*)

Nonblocking All to All Scatter/Gather.

Parameters

- **sendbuf** ([BufSpecB](#) / [InPlace](#)) –
- **recvbuf** ([BufSpecB](#)) –

Return type

Request

Ialltoallv(*sendbuf*, *recvbuf*)

Nonblocking All to All Scatter/Gather Vector.

Parameters

- **sendbuf** ([BufSpecV](#) / [InPlace](#)) –
- **recvbuf** ([BufSpecV](#)) –

Return type

Request

Ialltoallw(*sendbuf*, *recvbuf*)

Nonblocking All to All Scatter/Gather General.

Parameters

- **sendbuf** ([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(*info=None*)

Nonblocking duplicate a communicator.

Parameters

info (*Info* / *None*) –

Return type

tuple[Self, Request]

Idup_with_info(*info*)

Nonblocking duplicate a communicator with hints.

Parameters

info (*Info*) –

Return type

tuple[Self, Request]

Iflush_buffer()

Nonblocking flush for buffered messages.

Return type

Request

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

Nonblocking Gather.

Parameters

- **sendbuf** (*BufSpec* / *InPlace*) –
- **recvbuf** (*BufSpecB* / *None*) –
- **root** (*int*) –

Return type

Request

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

Nonblocking Gather Vector.

Parameters

- **sendbuf** (*BufSpec* / *InPlace*) –
- **recvbuf** (*BufSpecV* / *None*) –
- **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** (*Status* / *None*) –

Return type

Message | None

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

Nonblocking test for a message.

Parameters

- **source** (*int*) –
- **tag** (*int*) –
- **status** (*Status* / *None*) –

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** (*BufSpec* / *InPlace*) –
- **recvbuf** (*BufSpec* / *None*) –
- **op** (*Op*) –
- **root** (*int*) –

Return type

Request

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

Nonblocking Reduce-Scatter (vector version).

Parameters

- **sendbuf** (*BufSpec* / *InPlace*) –
- **recvbuf** (*BufSpec*) –
- **recvcounts** (*Sequence*[*int*] / *None*) –
- **op** (*Op*) –

Return type

Request

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

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

Parameters

- **sendbuf** (*BufSpecB* / *InPlace*) –
- **recvbuf** (*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()

Return whether the communicator is an intercommunicator.

Return type

bool

Is_intra()

Return whether the communicator is an intracommunicator.

Return type

bool

Is_revoked()

Indicate whether the communicator has been revoked.

Return type

bool

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

Nonblocking Scatter.

Parameters

- **sendbuf** (*BufSpecB* / *None*) –
- **recvbuf** (*BufSpec* / *InPlace*) –
- **root** (*int*) –

Return type

Request

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

Nonblocking Scatter Vector.

Parameters

- **sendbuf** (`BufSpecV / None`) –
- **recvbuf** (`BufSpec / InPlace`) –
- **root** (`int`) –

Return type

Request

I`send`(`buf, dest, tag=0`)

Nonblocking send.

Parameters

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

Return type

Request

I`sendrecv`(`sendbuf, dest, sendtag=0, recvbuf=None, source=ANY_SOURCE, recvtag=ANY_TAG`)

Nonblocking send and receive.

Parameters

- **sendbuf** (`BufSpec`) –
- **dest** (`int`) –
- **sendtag** (`int`) –
- **recvbuf** (`BufSpec / None`) –
- **source** (`int`) –
- **recvtag** (`int`) –

Return type

Request

I`sendrecv_replace`(`buf, dest, sendtag=0, source=ANY_SOURCE, recvtag=ANY_TAG`)

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*) –

Return type
Request

Ishrink()

Nonblocking shrink a communicator to remove all failed processes.

Return type
tuple[Comm, Request]

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

Nonblocking send in synchronous mode.

Parameters

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

Return type
Request

classmethod Join(*fd*)

Interconnect 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** (*Status* / *None*) –

Return type
Message

Precv_init(*buf*, *partitions*, *source*=ANY_SOURCE, *tag*=ANY_TAG, *info*=INFO_NULL)

Create request for a partitioned recv operation.

Parameters

- **buf** (*BufSpec*) –
- **partitions** (*int*) –
- **source** (*int*) –
- **tag** (*int*) –
- **info** (*Info*) –

Return type
Prequest

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** (*Status* / *None*) –

Return type

Literal[True]

Psend_init(*buf*, *partitions*, *dest*, *tag*=0, *info*=INFO_NULL)

Create request for a partitioned send operation.

Parameters

- **buf** (*BufSpec*) –
- **partitions** (*int*) –
- **dest** (*int*) –
- **tag** (*int*) –
- **info** (*Info*) –

Return type

Prequest

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** (*Status* / *None*) –

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** (*BufSpec* / *InPlace*) –
- **recvbuf** (*BufSpec* / *None*) –
- **op** (*Op*) –
- **root** (*int*) –

Return type

None

Reduce_init(*sendbuf*, *recvbuf*, *op*=*SUM*, *root*=0, *info*=*INFO_NULL*)

Persistent Reduce to Root.

Parameters

- **sendbuf** (*BufSpec* / *InPlace*) –
- **recvbuf** (*BufSpec* / *None*) –
- **op** (*Op*) –
- **root** (*int*) –
- **info** (*Info*) –

Return type

Prequest

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

Reduce-Scatter (vector version).

Parameters

- **sendbuf** (*BufSpec* / *InPlace*) –
- **recvbuf** (*BufSpec*) –
- **recvcounts** (*Sequence*[*int*] / *None*) –
- **op** (*Op*) –

Return type

None

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

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

Parameters

- **sendbuf** (*BufSpecB* / *InPlace*) –
- **recvbuf** (*BufSpec* / *BufSpecB*) –
- **op** (*Op*) –

Return type

None

Reduce_scatter_block_init(*sendbuf*, *recvbuf*, *op*=SUM, *info*=INFO_NULL)

Persistent Reduce-Scatter Block (regular, non-vector version).

Parameters

- **sendbuf** (`BufSpecB / InPlace`) –
- **recvbuf** (`BufSpec / BufSpecB`) –
- **op** (`Op`) –
- **info** (`Info`) –

Return type

`Request`

Reduce_scatter_init(*sendbuf*, *recvbuf*, *recvcounts*=None, *op*=SUM, *info*=INFO_NULL)

Persistent Reduce-Scatter (vector version).

Parameters

- **sendbuf** (`BufSpec / InPlace`) –
- **recvbuf** (`BufSpec`) –
- **recvcounts** (`Sequence[int] / None`) –
- **op** (`Op`) –
- **info** (`Info`) –

Return type

`Request`

Revoke()

Revoke a communicator.

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.

Parameters

- **sendbuf** (`BufSpecB` / `None`) –
- **recvbuf** (`BufSpec` / `InPlace`) –
- **root** (`int`) –

Return type

`None`

Scatter_init(*sendbuf*, *recvbuf*, *root*=0, *info*=`INFO_NULL`)

Persistent Scatter.

Parameters

- **sendbuf** (`BufSpecB` / `None`) –
- **recvbuf** (`BufSpec` / `InPlace`) –
- **root** (`int`) –
- **info** (`Info`) –

Return type

`Request`

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

Scatter Vector.

Scatter data from one process to all other processes providing different amounts of data and displacements.

Parameters

- **sendbuf** (`BufSpecV` / `None`) –
- **recvbuf** (`BufSpec` / `InPlace`) –
- **root** (`int`) –

Return type

`None`

Scatterv_init(*sendbuf*, *recvbuf*, *root*=0, *info*=`INFO_NULL`)

Persistent Scatter Vector.

Parameters

- **sendbuf** (`BufSpecV` / `None`) –
- **recvbuf** (`BufSpec` / `InPlace`) –
- **root** (`int`) –
- **info** (`Info`) –

Return type

`Request`

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

Blocking send.

Note: This function may block until the message is received. Whether *Send* blocks or not 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](#) / [None](#)) –
- **source** ([int](#)) –
- **recvtag** ([int](#)) –
- **status** ([Status](#) / [None](#)) –

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** ([Status](#) / [None](#)) –

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

Shrink()

Shrink a communicator to remove all failed processes.

Return type

Comm

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** (*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 synchronization.

Note: This method is equivalent to *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

classmethod fromhandle(handle)

Create object from MPI handle.

Parameters

- **handle** (*int*) –

Return type

Comm

gather(sendobj, root=0)

Gather.

Parameters

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

Return type

list[*Any*] | None

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** (*Status* | None) –

Return type

Message | None

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

Nonblocking test for a message.

Parameters

- **source** (*int*) –
- **tag** (*int*) –
- **status** (*Status* / *None*) –

Return type

bool

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

Nonblocking receive.

Parameters

- **buf** (*Buffer* / *None*) –
- **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** (*Status* / *None*) –

Return type

Message

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

Blocking test for a message.

Parameters

- **source** (*int*) –
- **tag** (*int*) –
- **status** (*Status* / *None*) –

Return type

Literal[True]

py2f()

Return type

int

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

Receive.

Parameters

- **buf** (*Buffer* / *None*) –
- **source** (*int*) –
- **tag** (*int*) –
- **status** (*Status* / *None*) –

Return type

Any

reduce(*sendobj*, *op*=SUM, *root*=0)

Reduce to Root.

Parameters

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

Return type

Any | None

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

Scatter.

Parameters

- **sendobj** (*Sequence*[*Any*] / *None*) –
- **root** (*int*) –

Return type

Any

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

Send in standard mode.

Parameters

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

Return type

None

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

Send and Receive.

Parameters

- **sendobj** (*Any*) –
- **dest** (*int*) –
- **sendtag** (*int*) –
- **recvbuf** (*Buffer* / None) –
- **source** (*int*) –
- **recvtag** (*int*) –
- **status** (*Status* / None) –

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

Group.

handle

MPI handle.

info

Info hints.

is_inter
Is intercommunicator.

is_intra
Is intracomunicator.

is_topo
Is a topology.

name
Print name.

rank
Rank of this process.

size
Number of processes.

topology
Topology type.

mpi4py.MPI.Datatype

```
class mpi4py.MPI.Datatype
    Bases: object
    Datatype object.
    static __new__(cls, datatype=None)

    Parameters
        datatype (Datatype / None) -
    Return type
        Self
```

Methods Summary

<code>Commit()</code>	Commit the datatype.
<code>Create_contiguous(count)</code>	Create a contiguous datatype.
<code>Create_darray(size, rank, gsizes, distribs, ...)</code>	Create a datatype for a distributed array on Cartesian process grids.
<code>Create_f90_complex(p, r)</code>	Return a bounded complex datatype.
<code>Create_f90_integer(r)</code>	Return a bounded integer datatype.
<code>Create_f90_real(p, r)</code>	Return a bounded real datatype.
<code>Create_hindexed(blocklengths, displacements)</code>	Create an indexed datatype.
<code>Create_hindexed_block(blocklength, displacements)</code>	Create an indexed datatype with constant-sized blocks.
<code>Create_hvector(count, blocklength, stride)</code>	Create a vector (strided) datatype with stride in bytes.
<code>Create_indexed(blocklengths, displacements)</code>	Create an indexed datatype.
<code>Create_indexed_block(blocklength, displacements)</code>	Create an indexed datatype with constant-sized blocks.
<code>Create_keyval([copy_fn, delete_fn, nopython])</code>	Create a new attribute key for datatypes.

continues on next page

Table 3 – continued from previous page

<code>Create_resized(lb, extent)</code>	Create a datatype with a new lower bound and extent.
<code>Create_struct(blocklengths, displacements, ...)</code>	Create a general composite (struct) datatype.
<code>Create_subarray(sizes, subsizes, starts[, order])</code>	Create a datatype for a subarray of a multidimensional array.
<code>Create_vector(count, blocklength, stride)</code>	Create a vector (strided) datatype.
<code>Delete_attr(keyval)</code>	Delete attribute value associated with a key.
<code>Dup()</code>	Duplicate a datatype.
<code>Free()</code>	Free the datatype.
<code>Free_keyval(keyval)</code>	Free an attribute key for datatypes.
<code>Get_attr(keyval)</code>	Retrieve attribute value by key.
<code>Get_contents()</code>	Return the input arguments used to create a datatype.
<code>Get_envelope()</code>	Return the number of input arguments used to create a datatype.
<code>Get_extent()</code>	Return lower bound and extent of datatype.
<code>Get_name()</code>	Get the print name for this datatype.
<code>Get_size()</code>	Return the number of bytes occupied by entries in the datatype.
<code>Get_true_extent()</code>	Return the true lower bound and extent of a datatype.
<code>Get_value_index(value, index)</code>	Return a predefined pair datatype.
<code>Match_size(typeclass, size)</code>	Find a datatype matching a specified size in bytes.
<code>Pack(inbuf, outbuf, position, comm)</code>	Pack into contiguous memory according to datatype.
<code>Pack_external(datarep, inbuf, outbuf, position)</code>	Pack into contiguous memory according to datatype.
<code>Pack_external_size(datarep, count)</code>	Determine the amount of space needed to pack a message.
<code>Pack_size(count, comm)</code>	Determine the amount of space needed to pack a message.
<code>Set_attr(keyval, attrval)</code>	Store attribute value associated with a key.
<code>Set_name(name)</code>	Set the print name for this datatype.
<code>Unpack(inbuf, position, outbuf, comm)</code>	Unpack from contiguous memory according to datatype.
<code>Unpack_external(datarep, inbuf, position, outbuf)</code>	Unpack from contiguous memory according to datatype.
<code>decode()</code>	Convenience method for decoding a datatype.
<code>f2py(arg)</code>	
<code>fromcode(code)</code>	Get predefined MPI datatype from character code or type string.
<code>fromhandle(handle)</code>	Create object from MPI handle.
<code>py2f()</code>	
<code>tocode()</code>	Get character code or type string from predefined MPI datatype.

Attributes Summary

<code>combiner</code>	Combiner.
<code>contents</code>	Contents.
<code>envelope</code>	Envelope.
<code>extent</code>	Extent.
<code>handle</code>	MPI handle.
<code>is_named</code>	Is a named datatype.
<code>is_predefined</code>	Is a predefined datatype.
<code>lb</code>	Lower bound.
<code>name</code>	Print name.
<code>size</code>	Size (in bytes).
<code>true_extent</code>	True extent.
<code>true_lb</code>	True lower bound.
<code>true_ub</code>	True upper bound.
<code>typechar</code>	Character code.
<code>typestr</code>	Type string.
<code>ub</code>	Upper bound.

Methods Documentation

`Commit()`

Commit the datatype.

Return type

Self

`Create_contiguous(count)`

Create a contiguous datatype.

Parameters

`count (int)` –

Return type

Self

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

Create a datatype for a distributed array on Cartesian process grids.

Parameters

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

Return type

Self

classmethod `Create_f90_complex(p, r)`

Return a bounded complex datatype.

Parameters

- `p` (`int`) –
- `r` (`int`) –

Return type

Self

classmethod `Create_f90_integer(r)`

Return a bounded integer datatype.

Parameters

`r` (`int`) –

Return type

Self

classmethod `Create_f90_real(p, r)`

Return a bounded real datatype.

Parameters

- `p` (`int`) –
- `r` (`int`) –

Return type

Self

Create_indexed(*blocklengths*, *displacements*)

Create an indexed datatype.

Note: Displacements are measured in bytes.

Parameters

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

Return type

Self

Create_indexed_block(*blocklength*, *displacements*)

Create an indexed datatype with constant-sized blocks.

Note: Displacements are measured in bytes.

Parameters

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

Return type*Self***Create_hvector**(*count*, *blocklength*, *stride*)

Create a vector (strided) datatype with stride in bytes.

Parameters

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

Return type*Self***Create_indexed**(*blocklengths*, *displacements*)

Create an indexed datatype.

Parameters

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

Return type*Self***Create_indexed_block**(*blocklength*, *displacements*)

Create an indexed datatype with constant-sized blocks.

Parameters

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

Return type*Self***classmethod Create_keyval**(*copy_fn=None*, *delete_fn=None*, *nopython=False*)

Create a new attribute key for datatypes.

Parameters

- **copy_fn** (*Callable[[Datatype, int, Any], Any] / None*) –
- **delete_fn** (*Callable[[Datatype, int, Any], None] / 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*Self*

classmethod **Create_struct**(*blocklengths*, *displacements*, *datatypes*)

Create a general composite (struct) datatype.

Note: Displacements are measured in bytes.

Parameters

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

Return type

Self

Create_subarray(*sizes*, *subsizes*, *starts*, *order=ORDER_C*)

Create a datatype for a subarray of a multidimensional array.

Parameters

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

Return type

Self

Create_vector(*count*, *blocklength*, *stride*)

Create a vector (strided) datatype.

Parameters

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

Return type

Self

Delete_attr(*keyval*)

Delete attribute value associated with a key.

Parameters

keyval (*int*) –

Return type

None

Dup()

Duplicate a datatype.

Return type

Self

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

int | *Any* | None

Get_contents()

Return the input arguments used to create a datatype.

Return type

tuple[list[int], list[int], list[int], list[Datatype]]

Get_envelope()

Return the number of input arguments used to create a datatype.

Return type

tuple[int, 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 **Get_value_index**(*value, index*)

Return a predefined pair datatype.

Parameters

- **value** ([Datatype](#)) –
- **index** ([Datatype](#)) –

Return type

Self

classmethod **Match_size**(*typeclass, size*)

Find a datatype matching a specified size in bytes.

Parameters

- **typeclass** ([int](#)) –
- **size** ([int](#)) –

Return type

Self

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.

Uses the portable data representation **external32**.

Parameters

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

Return type

[int](#)

Pack_external_size(*datarep, count*)

Determine the amount of space needed to pack a message.

Uses the portable data representation **external32**.

Note: Returns an upper bound measured in bytes.

Parameters

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

Return type

int

Pack_size(*count, comm*)

Determine the amount of space needed to pack a message.

Note: Returns an upper bound measured in bytes.

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.
Uses the 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

classmethod fromcode(code)

Get predefined MPI datatype from character code or type string.

Parameters

code (*str*) –

Return type

Datatype

classmethod fromhandle(handle)

Create object from MPI handle.

Parameters

handle (*int*) –

Return type

Datatype

py2f()

Return type

int

tocode()

Get character code or type string from predefined MPI datatype.

Return type

str

Attributes Documentation

combiner

Combiner.

contents

Contents.

envelope

Envelope.

extent

Extent.

handle

MPI handle.

is_named

Is a named datatype.

is_predefined

Is a predefined datatype.

lb

Lower bound.

name

Print name.

size

Size (in bytes).

true_extent

True extent.

true_lb

True lower bound.

true_ub

True upper bound.

typechar

Character code.

typestr

Type string.

ub

Upper bound.

mpi4py.MPI.Distgraphcomm

```
class mpi4py.MPI.Distgraphcomm
```

Bases: *Topocomm*

Distributed graph topology intracommunicator.

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

Parameters

`comm` (`Distgraphcomm` / `None`) –

Return type

Self

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], tuple[list[int], list[int]] | None]`

Get_dist_neighbors_count()

Return adjacency information for a distributed graph topology.

Return type

`int`

mpi4py.MPI.Errhandler

```
class mpi4py.MPI.Errhandler
```

Bases: `object`

Error handler.

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

Parameters

`errhandler` (`Errhandler` / `None`) –

Return type

Self

Methods Summary

<code>Free()</code>	Free an error handler.
<code>f2py(arg)</code>	
<code>fromhandle(handle)</code>	Create object from MPI handle.
<code>py2f()</code>	

Attributes Summary

<code>handle</code>	MPI handle.
---------------------	-------------

Methods Documentation

`Free()`

Free an error handler.

Return type

None

`classmethod f2py(arg)`

Parameters

`arg (int)` –

Return type

Errhandler

`classmethod fromhandle(handle)`

Create object from MPI handle.

Parameters

`handle (int)` –

Return type

Errhandler

`py2f()`

Return type

int

Attributes Documentation

handle

MPI handle.

mpi4py.MPI.File

`class mpi4py.MPI.File`

Bases: `object`

File I/O context.

`static __new__(cls, file=None)`

Parameters

`file (File / None) –`

Return type

`Self`

Methods Summary

<code>Call_errhandler(errorcode)</code>	Call the error handler installed on a file.
<code>Close()</code>	Close a file.
<code>Create_errhandler(errhandler_fn)</code>	Create a new error handler for files.
<code>Delete(filename[, info])</code>	Delete a file.
<code>Get_amode()</code>	Return the file access mode.
<code>Get_atomicity()</code>	Return the atomicity mode.
<code>Get_byte_offset(offset)</code>	Return the absolute byte position in the file.
<code>Get_errhandler()</code>	Get the error handler for a file.
<code>Get_group()</code>	Access the group of processes that opened the file.
<code>Get_info()</code>	Return the current hints for a file.
<code>Get_position()</code>	Return the current position of the individual file pointer.
<code>Get_position_shared()</code>	Return the current position of the shared file pointer.
<code>Get_size()</code>	Return the file size.
<code>Get_type_extent(datatype)</code>	Return the extent of datatype in the file.
<code>Get_view()</code>	Return the file view.
<code>Iread(buf)</code>	Nonblocking read using individual file pointer.
<code>Iread_all(buf)</code>	Nonblocking collective read using individual file pointer.
<code>Iread_at(offset, buf)</code>	Nonblocking read using explicit offset.
<code>Iread_at_all(offset, buf)</code>	Nonblocking collective read using explicit offset.
<code>Iread_shared(buf)</code>	Nonblocking read using shared file pointer.
<code>Iwrite(buf)</code>	Nonblocking write using individual file pointer.
<code>Iwrite_all(buf)</code>	Nonblocking collective write using individual file pointer.
<code>Iwrite_at(offset, buf)</code>	Nonblocking write using explicit offset.
<code>Iwrite_at_all(offset, buf)</code>	Nonblocking collective write using explicit offset.
<code>Iwrite_shared(buf)</code>	Nonblocking write using shared file pointer.
<code>Open(comm, filename[, amode, info])</code>	Open a file.
<code>Preallocate(size)</code>	Preallocate storage space for a file.

continues on next page

Table 4 – continued from previous page

<code>Read(buf[, status])</code>	Read using individual file pointer.
<code>Read_all(buf[, status])</code>	Collective read using individual file pointer.
<code>Read_all_begin(buf)</code>	Start a split collective read using individual file pointer.
<code>Read_all_end(buf[, status])</code>	Complete a split collective read using individual file pointer.
<code>Read_at(offset, buf[, status])</code>	Read using explicit offset.
<code>Read_at_all(offset, buf[, status])</code>	Collective read using explicit offset.
<code>Read_at_all_begin(offset, buf)</code>	Start a split collective read using explicit offset.
<code>Read_at_all_end(buf[, status])</code>	Complete a split collective read using explicit offset.
<code>Read_ordered(buf[, status])</code>	Collective read using shared file pointer.
<code>Read_ordered_begin(buf)</code>	Start a split collective read using shared file pointer.
<code>Read_ordered_end(buf[, status])</code>	Complete a split collective read using shared file pointer.
<code>Read_shared(buf[, status])</code>	Read using shared file pointer.
<code>Seek(offset[, whence])</code>	Update the individual file pointer.
<code>Seek_shared(offset[, whence])</code>	Update the shared file pointer.
<code>Set_atomicity(flag)</code>	Set the atomicity mode.
<code>Set_errhandler(errhandler)</code>	Set the error handler for a file.
<code>Set_info(info)</code>	Set new values for the hints associated with a file.
<code>Set_size(size)</code>	Set the file size.
<code>Set_view([disp, etype, filetype, datarep, info])</code>	Set the file view.
<code>Sync()</code>	Causes all previous writes to be transferred to the storage device.
<code>Write(buf[, status])</code>	Write using individual file pointer.
<code>Write_all(buf[, status])</code>	Collective write using individual file pointer.
<code>Write_all_begin(buf)</code>	Start a split collective write using individual file pointer.
<code>Write_all_end(buf[, status])</code>	Complete a split collective write using individual file pointer.
<code>Write_at(offset, buf[, status])</code>	Write using explicit offset.
<code>Write_at_all(offset, buf[, status])</code>	Collective write using explicit offset.
<code>Write_at_all_begin(offset, buf)</code>	Start a split collective write using explicit offset.
<code>Write_at_all_end(buf[, status])</code>	Complete a split collective write using explicit offset.
<code>Write_ordered(buf[, status])</code>	Collective write using shared file pointer.
<code>Write_ordered_begin(buf)</code>	Start a split collective write using shared file pointer.
<code>Write_ordered_end(buf[, status])</code>	Complete a split collective write using shared file pointer.
<code>Write_shared(buf[, status])</code>	Write using shared file pointer.
<code>f2py(arg)</code>	
<code>fromhandle(handle)</code>	Create object from MPI handle.
<code>py2f()</code>	

Attributes Summary

<code>amode</code>	Access mode.
<code>atomicity</code>	Atomicity mode.
<code>group</code>	Group.
<code>handle</code>	MPI handle.
<code>info</code>	Info hints.
<code>size</code>	Size (in bytes).

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 Create_errhandler(errhandler_fn)`

Create a new error handler for files.

Parameters

`errhandler_fn` (`Callable[[File, int], None]`) –

Return type

`Errhandler`

`classmethod Delete(filename, info=INFO_NULL)`

Delete a file.

Parameters

- `filename` (`PathLike` / `str` / `bytes`) –
- `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.

Note: Input *offset* is measured in etype units relative to the current file view.

Parameters

offset ([int](#)) –

Return type

[int](#)

Get_errhandler()

Get the error handler for a file.

Return type

[Errhandler](#)

Get_group()

Access the group of processes that opened the file.

Return type

[Group](#)

Get_info()

Return the current hints for a file.

Return type

[Info](#)

Get_position()

Return the current position of the individual file pointer.

Note: Position is measured in etype units relative to the current file view.

Return type

[int](#)

Get_position_shared()

Return the current position of the shared file pointer.

Note: Position is measured 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** (Intracom) –
- **filename** (PathLike | str | bytes) –
- **amode** (*int*) –

- **info** ([Info](#)) –

Return type

Self

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** ([Status](#) / *None*) –

Return type

None

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

Collective read using individual file pointer.

Parameters

- **buf** ([BufSpec](#)) –
- **status** ([Status](#) / *None*) –

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** ([Status](#) / *None*) –

Return type

None

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

Read using explicit offset.

Parameters

- **offset** ([int](#)) –

- **buf** ([BufSpec](#)) –
- **status** ([Status](#) / [None](#)) –

Return type

None

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

Collective read using explicit offset.

Parameters

- **offset** ([int](#)) –
- **buf** ([BufSpec](#)) –
- **status** ([Status](#) / [None](#)) –

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** ([Status](#) / [None](#)) –

Return type

None

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

Collective read using shared file pointer.

Parameters

- **buf** ([BufSpec](#)) –
- **status** ([Status](#) / [None](#)) –

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** ([Status](#) / [None](#)) –

Return type

None

Read_shared(buf, status=None)

Read using shared file pointer.

Parameters

- **buf** ([BufSpec](#)) –
- **status** ([Status](#) / [None](#)) –

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)

Set 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** ([Datatype](#) / [None](#)) –
- **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** ([Status](#) / [None](#)) –

Return type

[None](#)

Write_all(buf, status=None)

Collective write using individual file pointer.

Parameters

- **buf** ([BufSpec](#)) –
- **status** ([Status](#) / [None](#)) –

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 (Status / None) –

Return type

None

Write_at(offset, buf, status=None)

Write using explicit offset.

Parameters

- offset (int) –
- buf (BufSpec) –
- status (Status / None) –

Return type

None

Write_at_all(offset, buf, status=None)

Collective write using explicit offset.

Parameters

- offset (int) –
- buf (BufSpec) –
- status (Status / None) –

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** (`Status` / `None`) –

Return type

`None`

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

Collective write using shared file pointer.

Parameters

- **buf** (`BufSpec`) –
- **status** (`Status` / `None`) –

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** (`Status` / `None`) –

Return type

`None`

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

Write using shared file pointer.

Parameters

- **buf** (`BufSpec`) –
- **status** (`Status` / `None`) –

Return type

`None`

classmethod f2py(*arg*)

Parameters

arg (`int`) –

Return type

`File`

classmethod fromhandle(*handle*)

Create object from MPI handle.

Parameters

handle (`int`) –

Return type
File

`py2f()`

Return type
int

Attributes Documentation

amode

Access mode.

atomicity

Atomicity mode.

group

Group.

handle

MPI handle.

info

Info hints.

size

Size (in bytes).

mpi4py.MPI.Graphcomm

`class mpi4py.MPI.Graphcomm`

Bases: *Topocomm*

General graph topology intracommunicator.

`static __new__(cls, comm=None)`

Parameters

`comm (Graphcomm / None) –`

Return type

Self

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>	Edges.
<code>index</code>	Index.
<code>nedges</code>	Number of edges.
<code>neighbors</code>	Neighbors.
<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`

Edges.

`index`

Index.

nedges
Number of edges.

neighbors
Neighbors.

nneighbors
Number of neighbors.

nnodes
Number of nodes.

topo
Topology information.

mpi4py.MPI.Grequest

```
class mpi4py.MPI.Grequest
    Bases: Request
    Generalized request handler.

    static __new__(cls, request=None)

    Parameters
        request (Grequest / None) –
    Return type
        Self
```

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, kwargs=None)

Create and return a user-defined request.

Parameters

- `query_fn (Callable[[...], None])` –
- `free_fn (Callable[[...], None])` –
- `cancel_fn (Callable[[...], None])` –
- `args (tuple[Any] / None)` –

- **kwargs** (`dict[str, Any]` / `None`) –

Return type

`Grequest`

mpi4py.MPI.Group

`class mpi4py.MPI.Group`

Bases: `object`

Group of processes.

`static __new__(cls, group=None)`

Parameters

`group` (`Group` / `None`) –

Return type

`Self`

Methods Summary

<code>Compare(group)</code>	Compare two groups.
<code>Create_from_session_pset(session, pset_name)</code>	Create a new group from session and process set.
<code>Difference(group1, group2)</code>	Create a new group from the difference of two existing groups.
<code>Dup()</code>	Duplicate a group.
<code>Excl(ranks)</code>	Create a new group by excluding listed members.
<code>Free()</code>	Free a group.
<code>Get_rank()</code>	Return the rank of this process in a group.
<code>Get_size()</code>	Return the number of processes in a group.
<code>Incl(ranks)</code>	Create a new group by including listed members.
<code>Intersection(group1, group2)</code>	Create a new group from the intersection of two existing groups.
<code>Range_excl(ranks)</code>	Create a new group by excluding ranges of members.
<code>Range_incl(ranks)</code>	Create a new group by including ranges of members.
<code>Translate_ranks([ranks, group])</code>	Translate ranks in a group to those in another group.
<code>Union(group1, group2)</code>	Create a new group from the union of two existing groups.
<code>f2py(arg)</code>	
<code>fromhandle(handle)</code>	Create object from MPI handle.
<code>py2f()</code>	

Attributes Summary

<code>handle</code>	MPI handle.
<code>rank</code>	Rank of this process.
<code>size</code>	Number of processes.

Methods Documentation

Compare(*group*)

Compare two groups.

Parameters

`group` ([Group](#)) –

Return type

`int`

classmethod Create_from_session_pset(*session, pset_name*)

Create a new group from session and process set.

Parameters

- `session` ([Session](#)) –
- `pset_name` ([str](#)) –

Return type

`Self`

classmethod Difference(*group1, group2*)

Create a new group from the difference of two existing groups.

Parameters

- `group1` ([Group](#)) –
- `group2` ([Group](#)) –

Return type

`Self`

Dup()

Duplicate a group.

Return type

`Self`

Excl(*ranks*)

Create a new group by excluding listed members.

Parameters

`ranks` ([Sequence\[int\]](#)) –

Return type

`Self`

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 number of processes in a group.

Return type

int

Incl(ranks)

Create a new group by including listed members.

Parameters**ranks** (*Sequence[int]*) –**Return type***Self***classmethod Intersection(group1, group2)**

Create a new group from the intersection of two existing groups.

Parameters

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

Return type*Self***Range_excl(ranks)**

Create a new group by excluding ranges of members.

Parameters**ranks** (*Sequence[tuple[int, int, int]]*) –**Return type***Self***Range_incl(ranks)**

Create a new group by including ranges of members.

Parameters**ranks** (*Sequence[tuple[int, int, int]]*) –**Return type***Self***Translate_ranks(ranks=None, group=None)**

Translate ranks in a group to those in another group.

Parameters

- **ranks** (*Sequence[int] / None*) –
- **group** (*Group / None*) –

Return type

`list[int]`

classmethod `Union(group1, group2)`

Create a new group from the union of two existing groups.

Parameters

- `group1` (`Group`) –
- `group2` (`Group`) –

Return type

`Self`

classmethod `f2py(arg)`

Parameters

`arg` (`int`) –

Return type

`Group`

classmethod `fromhandle(handle)`

Create object from MPI handle.

Parameters

`handle` (`int`) –

Return type

`Group`

`py2f()`

Return type

`int`

Attributes Documentation

handle

MPI handle.

rank

Rank of this process.

size

Number of processes.

mpi4py.MPI.InPlaceType

class `mpi4py.MPI.InPlaceType`

Bases: `int`

Type of `IN_PLACE`.

static `__new__(cls)`

Return type

`Self`

mpi4py.MPI.Info

```
class mpi4py.MPI.Info
    Bases: object

    Info object.

    static __new__(cls, info=None)

    Parameters
        info (Info / None) –

    Return type
        Self
```

Methods Summary

Create([items])	Create a new info object.
Create_env([args])	Create a new environment info object.
Delete(key)	Remove a (key, value) pair from info.
Dup()	Duplicate an existing info object.
Free()	Free an info object.
Get(key)	Retrieve the value associated with a key.
Get_nkeys()	Return the number of currently defined keys in info.
Get_nthkey(n)	Return the <i>n</i> -th defined key in info.
Set(key, value)	Store a value associated with a key.
clear()	Clear contents.
copy()	Copy contents.
f2py(arg)	
fromhandle(handle)	Create object from MPI handle.
get(key[, default])	Retrieve value by key.
items()	Return list of items.
keys()	Return list of keys.
pop(key, *default)	Pop value by key.
popitem()	Pop first item.
update([items])	Update contents.
values()	Return list of values.

Attributes Summary

handle	MPI handle.
--------	-------------

Methods Documentation

classmethod Create(items=None)

Create a new info object.

Parameters

items (`Info` / `Mapping[str, str]` / `Iterable[tuple[str, str]]` / `None`) –

Return type

`Self`

classmethod Create_env(args=None)

Create a new environment info object.

Parameters

args (`Sequence[str]` / `None`) –

Return type

`Self`

Delete(key)

Remove a (key, value) pair from info.

Parameters

key (`str`) –

Return type

`None`

Dup()

Duplicate an existing info object.

Return type

`Self`

Free()

Free an info object.

Return type

`None`

Get(key)

Retrieve the value associated with a key.

Parameters

key (`str`) –

Return type

`str | None`

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.

Parameters

n (`int`) –

Return type

`str`

Set(*key*, *value*)

Store a value associated with a key.

Parameters

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

Return type

`None`

clear()

Clear contents.

Return type

`None`

copy()

Copy contents.

Return type

`Self`

classmethod f2py(*arg*)**Parameters**

arg (`int`) –

Return type

`Info`

classmethod fromhandle(*handle*)

Create object from MPI handle.

Parameters

handle (`int`) –

Return type

`Info`

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

Retrieve value by key.

Parameters

- **key** (`str`) –
- **default** (`str` / `None`) –

Return type

`str | None`

items()

Return list of items.

Return type

`list[tuple[str, str]]`

keys()

Return list of keys.

Return type
list[str]

pop(key, *default)

Pop value by key.

Parameters

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

Return type

str

popitem()

Pop first item.

Return type
tuple[str, str]

py2f()

Return type
int

update(items=(), **kwds)

Update contents.

Parameters

- **items** (Info / Mapping[str, str] / Iterable[tuple[str, str]]) –
- **kwds** (str) –

Return type

None

values()

Return list of values.

Return type
list[str]

Attributes Documentation**handle**

MPI handle.

mpi4py.MPI.Intercomm

```
class mpi4py.MPI.Intercomm
```

Bases: *Comm*

Intercommunicator.

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

Parameters

`comm` (`Intercomm` / `None`) –

Return type

Self

Methods Summary

<code>Create_from_groups</code> (local_group, ...[, ...])	Create communicator from group.
<code>Get_remote_group()</code>	Access the remote group associated with the intercommunicator.
<code>Get_remote_size()</code>	Intercommunicator remote size.
<code>Merge</code> ([high])	Merge intercommunicator into an intracommunicator.

Attributes Summary

<code>remote_group</code>	Remote group.
<code>remote_size</code>	Number of remote processes.

Methods Documentation

```
classmethod Create_from_groups(local_group, local_leader, remote_group, remote_leader,
                               stringtag='org.mpi4py', info=INFO_NULL, errhandler=None)
```

Create communicator from group.

Parameters

- `local_group` (`Group`) –
- `local_leader` (`int`) –
- `remote_group` (`Group`) –
- `remote_leader` (`int`) –
- `stringtag` (`str`) –
- `info` (`Info`) –
- `errhandler` (`Errhandler` / `None`) –

Return type

`Intracomm`

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 into an intracommunicator.

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

Bases: [Comm](#)

Intracommunicator.

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

comm ([Intracomm](#) / [None](#)) –

Return type

[Self](#)

Methods Summary

<code>Accept(port_name[, info, root])</code>	Accept a request to form a new intercommunicator.
<code>Cart_map(dims[, periods])</code>	Determine optimal process placement on a Cartesian topology.
<code>Connect(port_name[, info, root])</code>	Make a request to form a new intercommunicator.
<code>Create_cart(dims[, periods, reorder])</code>	Create cartesian communicator.
<code>Create_dist_graph(sources, degrees, destinations)</code>	Create distributed graph communicator.
<code>Create_dist_graph_adjacent(sources, destinations)</code>	Create distributed graph communicator.
<code>Create_from_group(group[, stringtag, info, ...])</code>	Create communicator from group.
<code>Create_graph(index, edges[, reorder])</code>	Create graph communicator.
<code>Create_group(group[, tag])</code>	Create communicator from group.
<code>Create_intercomm(local_leader, peer_comm, ...)</code>	Create intercommunicator.
<code>Exscan(sendbuf, recvbuf[, op])</code>	Exclusive Scan.
<code>Exscan_init(sendbuf, recvbuf[, op, info])</code>	Persistent Exclusive Scan.
<code>Graph_map(index, edges)</code>	Determine optimal process placement on a graph topology.
<code>Iexscan(sendbuf, recvbuf[, op])</code>	Inclusive Scan.
<code>Iscan(sendbuf, recvbuf[, op])</code>	Inclusive Scan.
<code>Scan(sendbuf, recvbuf[, op])</code>	Inclusive Scan.
<code>Scan_init(sendbuf, recvbuf[, op, info])</code>	Persistent Inclusive Scan.
<code>Spawn(command[, args, maxprocs, info, root, ...])</code>	Spawn instances of a single MPI application.
<code>Spawn_multiple(command[, args, maxprocs, ...])</code>	Spawn instances of multiple MPI applications.
<code>exscan(sendobj[, op])</code>	Exclusive Scan.
<code>scan(sendobj[, op])</code>	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)`

Determine optimal process placement on a Cartesian topology.

Parameters

- `dims (Sequence[int])` –
- `periods (Sequence[bool] / None)` –

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** (*Sequence[bool]* / *None*) –
- **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** (*Sequence[int]* / *None*) –
- **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** (*Sequence[int]* / *None*) –
- **destweights** (*Sequence[int]* / *None*) –
- **info** (*Info*) –
- **reorder** (*bool*) –

Return type

Distgraphcomm

classmethod Create_from_group(group, stringtag='org.mpi4py', info=INFO_NULL, errhandler=None)

Create communicator from group.

Parameters

- **group** ([Group](#)) –
- **stringtag** ([str](#)) –
- **info** ([Info](#)) –
- **errhandler** ([Errhandler](#) / [None](#)) –

Return type

Intracomm

Create_graph(index, edges, reorder=False)

Create graph communicator.

Parameters

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

Return type

Graphcomm

Create_group(group, tag=0)

Create communicator from group.

Parameters

- **group** ([Group](#)) –
- **tag** ([int](#)) –

Return type

Intracomm

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** ([BufSpec](#) / [InPlace](#)) –

- **recvbuf** (`BufSpec`) –

- **op** (`Op`) –

Return type

`None`

Exscan_init(*sendbuf*, *recvbuf*, *op*=*SUM*, *info*=*INFO_NULL*)

Persistent Exclusive Scan.

Parameters

- **sendbuf** (`BufSpec / InPlace`) –
- **recvbuf** (`BufSpec`) –
- **op** (`Op`) –
- **info** (`Info`) –

Return type

`Request`

Graph_map(*index*, *edges*)

Determine optimal process placement on a graph topology.

Parameters

- **index** (`Sequence[int]`) –
- **edges** (`Sequence[int]`) –

Return type

`int`

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

Inclusive Scan.

Parameters

- **sendbuf** (`BufSpec / InPlace`) –
- **recvbuf** (`BufSpec`) –
- **op** (`Op`) –

Return type

`Request`

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

Inclusive Scan.

Parameters

- **sendbuf** (`BufSpec / InPlace`) –
- **recvbuf** (`BufSpec`) –
- **op** (`Op`) –

Return type

`Request`

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

Inclusive Scan.

Parameters

- **sendbuf** (*BufSpec* / *InPlace*) –
- **recvbuf** (*BufSpec*) –
- **op** (*Op*) –

Return type

None

Scan_init(*sendbuf*, *recvbuf*, *op*=SUM, *info*=INFO_NULL)

Persistent Inclusive Scan.

Parameters

- **sendbuf** (*BufSpec* / *InPlace*) –
- **recvbuf** (*BufSpec*) –
- **op** (*Op*) –
- **info** (*Info*) –

Return type

Request

Spawn(*command*, *args*=None, *maxprocs*=1, *info*=INFO_NULL, *root*=0, *errcodes*=None)

Spawn instances of a single MPI application.

Parameters

- **command** (*str*) –
- **args** (*Sequence*[*str*] / *None*) –
- **maxprocs** (*int*) –
- **info** (*Info*) –
- **root** (*int*) –
- **errcodes** (*list*[*int*] / *None*) –

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** (*Sequence*[*Sequence*[*str*]] / *None*) –
- **maxprocs** (*Sequence*[*int*] / *None*) –
- **info** (*Sequence*[*Info*] / *Info*) –
- **root** (*int*) –
- **errcodes** (*list*[*list*[*int*]] / *None*) –

Return type*Intercomm***exscan**(*sendobj*, *op*=*SUM*)

Exclusive Scan.

Parameters

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

Return type*Any***scan**(*sendobj*, *op*=*SUM*)

Inclusive Scan.

Parameters

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

Return type*Any***mpi4py.MPI.Message****class mpi4py.MPI.Message**Bases: *object*

Matched message.

static __new__(*cls*, *message*=*None*)**Parameters****message** (*Message* / *None*) –**Return type***Self***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>fromhandle</i> (<i>handle</i>)	Create object from MPI handle.
<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.

Attributes Summary

<code>handle</code>	MPI handle.
---------------------	-------------

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` ([Status](#) / [None](#)) –

Return type

[Self](#) | [None](#)

`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` ([Status](#) / [None](#)) –

Return type

[Self](#)

`Recv(buf, status=None)`

Blocking receive of matched message.

Parameters

- `buf` ([BufSpec](#)) –
- `status` ([Status](#) / [None](#)) –

Return type

[None](#)

classmethod f2py(arg)

Parameters
arg (*int*) –

Return type
Message

classmethod fromhandle(handle)

Create object from MPI handle.

Parameters
handle (*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** (*Status* / *None*) –

Return type
Self | None

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** (*Status* / *None*) –

Return type
Self

py2f()

Return type
int

recv(status=None)

Blocking receive of matched message.

Parameters
status (*Status* / *None*) –

Return type*Any***Attributes Documentation****handle**

MPI handle.

mpi4py.MPI.Op**class mpi4py.MPI.Op**Bases: `object`

Reduction operation.

static __new__(cls, op=None)**Parameters**`op (Op / None) –`**Return type***Self***Methods Summary**

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

Attributes Summary

<code>handle</code>	MPI handle.
<code>is_commutative</code>	Is a commutative operation.
<code>is_predefined</code>	Is a predefined operation.

Methods Documentation

classmethod Create(function, commute=False)

Create a user-defined reduction operation.

Parameters

- **function** (`Callable[[Buffer, Buffer, Datatype], None]`) –
- **commute** (`bool`) –

Return type

`Self`

Free()

Free a user-defined reduction operation.

Return type

`None`

Is_commutative()

Query reduction operations for their commutativity.

Return type

`bool`

Reduce_local(inbuf, inoutbuf)

Apply a reduction operation to local data.

Parameters

- **inbuf** (`BufSpec`) –
- **inoutbuf** (`BufSpec`) –

Return type

`None`

classmethod f2py(arg)

Parameters

`arg (int)` –

Return type

`Op`

classmethod fromhandle(handle)

Create object from MPI handle.

Parameters

`handle (int)` –

Return type

`Op`

py2f()

Return type

`int`

Attributes Documentation

handle

MPI handle.

is_commutative

Is a commutative operation.

is_predefined

Is a predefined operation.

mpi4py.MPI.Pickle

class mpi4py.MPI.Pickle

Bases: `object`

Pickle/unpickle Python objects.

static `__new__(cls, pickle=None)`

Parameters

`pickle (Pickle / None) –`

Return type

`Self`

Methods Summary

<code>dumps(obj)</code>	Serialize object to pickle data stream.
<code>dumps_oob(obj)</code>	Serialize object to pickle data stream and out-of-band buffers.
<code>loads(data)</code>	Deserialize object from pickle data stream.
<code>loads_oob(data, buffers)</code>	Deserialize object from pickle data stream and out-of-band buffers.

Attributes Summary

<code>PROTOCOL</code>	Protocol version.
<code>THRESHOLD</code>	Out-of-band threshold.

Methods Documentation

`dumps(obj)`

Serialize object to pickle data stream.

Parameters

`obj (Any) –`

Return type

`bytes`

umps_oob(*obj*)

Serialize object to pickle data stream and out-of-band buffers.

Parameters

obj (*Any*) –

Return type

tuple[bytes, list[buffer]]

loads(*data*)

Deserialize object from pickle data stream.

Parameters

data (*Buffer*) –

Return type

Any

loads_oob(*data, buffers*)

Deserialize object from pickle data stream and out-of-band buffers.

Parameters

- **data** (*Buffer*) –

- **buffers** (*Iterable[Buffer]*) –

Return type

Any

Attributes Documentation

PROTOCOL

Protocol version.

THRESHOLD

Out-of-band threshold.

mpi4py.MPI.Prequest

class mpi4py.MPI.Prequest

Bases: *Request*

Persistent request handler.

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

request (*Prequest* / *None*) –

Return type

Self

Methods Summary

<code>Parrived(partition)</code>	Test partial completion of a partitioned receive operation.
<code>Pready(partition)</code>	Mark a given partition as ready.
<code>Pready_list(partitions)</code>	Mark a sequence of partitions as ready.
<code>Pready_range(partition_low, partition_high)</code>	Mark a range of partitions as ready.
<code>Start()</code>	Initiate a communication with a persistent request.
<code>Startall(requests)</code>	Start a collection of persistent requests.

Methods Documentation

`Parrived(partition)`

Test partial completion of a partitioned receive operation.

Parameters

`partition (int)` –

Return type

`bool`

`Pready(partition)`

Mark a given partition as ready.

Parameters

`partition (int)` –

Return type

`None`

`Pready_list(partitions)`

Mark a sequence of partitions as ready.

Parameters

`partitions (Sequence[int])` –

Return type

`None`

`Pready_range(partition_low, partition_high)`

Mark a range of partitions as ready.

Parameters

- `partition_low (int)` –
- `partition_high (int)` –

Return type

`None`

`Start()`

Initiate a communication with a persistent request.

Return type

`None`

```
classmethod Startall(requests)
    Start a collection of persistent requests.

    Parameters
        requests (list[Request]) –
```

Return type

None

mpi4py.MPI.Request

```
class mpi4py.MPI.Request
    Bases: object

    Request handler.

    static __new__(cls, request=None)

    Parameters
        request (Request / None) –
```

Return type

Self

Methods Summary

<code>Cancel()</code>	Cancel a request.
<code>Free()</code>	Free a communication request.
<code>Get_status([status])</code>	Non-destructive test for the completion of a request.
<code>Get_status_all(requests[, statuses])</code>	Non-destructive test for the completion of all requests.
<code>Get_status_any(requests[, status])</code>	Non-destructive test for the completion of any requests.
<code>Get_status_some(requests[, statuses])</code>	Non-destructive test for completion of some requests.
<code>Test([status])</code>	Test for the completion of a non-blocking operation.
<code>Testall(requests[, statuses])</code>	Test for completion of all previously initiated requests.
<code>Testany(requests[, status])</code>	Test for completion of any previously initiated request.
<code>Testsome(requests[, statuses])</code>	Test for completion of some previously initiated requests.
<code>Wait([status])</code>	Wait for a non-blocking operation to complete.
<code>Waitall(requests[, statuses])</code>	Wait for all previously initiated requests to complete.
<code>Waitany(requests[, status])</code>	Wait for any previously initiated request to complete.
<code>Waitsome(requests[, statuses])</code>	Wait for some previously initiated requests to complete.
<code>cancel()</code>	Cancel a request.
<code>f2py(arg)</code>	
<code>fromhandle(handle)</code>	Create object from MPI handle.
<code>get_status([status])</code>	Non-destructive test for the completion of a request.
<code>get_status_all(requests[, statuses])</code>	Non-destructive test for the completion of all requests.
<code>get_status_any(requests[, status])</code>	Non-destructive test for the completion of any requests.
<code>get_status_some(requests[, statuses])</code>	Non-destructive test for completion of some requests.
<code>py2f()</code>	
<code>test([status])</code>	Test for the completion of a non-blocking operation.
<code>testall(requests[, statuses])</code>	Test for completion of all previously initiated requests.
<code>testany(requests[, status])</code>	Test for completion of any previously initiated request.
<code>testsome(requests[, statuses])</code>	Test for completion of some previously initiated requests.
<code>wait([status])</code>	Wait for a non-blocking operation to complete.
<code>waitall(requests[, statuses])</code>	Wait for all previously initiated requests to complete.
<code>waitany(requests[, status])</code>	Wait for any previously initiated request to complete.
<code>waitsome(requests[, statuses])</code>	Wait for some previously initiated requests to complete.

Attributes Summary

<code>handle</code>	MPI handle.
---------------------	-------------

Methods Documentation

`Cancel()`

Cancel a 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 (Status / None) –`

Return type

`bool`

`classmethod Get_status_all(requests, statuses=None)`

Non-destructive test for the completion of all requests.

Parameters

- `requests (Sequence[Request]) –`
- `statuses (list[Status] / None) –`

Return type

`bool`

`classmethod Get_status_any(requests, status=None)`

Non-destructive test for the completion of any requests.

Parameters

- `requests (Sequence[Request]) –`
- `status (Status / None) –`

Return type

`tuple[int, bool]`

`classmethod Get_status_some(requests, statuses=None)`

Non-destructive test for completion of some requests.

Parameters

- `requests (Sequence[Request]) –`
- `statuses (list[Status] / None) –`

Return type`list[int] | None`**Test**(*status=None*)

Test for the completion of a non-blocking operation.

Parameters`status (Status / None) –`**Return type**`bool`**classmethod Testall**(*requests, statuses=None*)

Test for completion of all previously initiated requests.

Parameters

- `requests (Sequence[Request]) –`
- `statuses (list[Status] / None) –`

Return type`bool`**classmethod Testany**(*requests, status=None*)

Test for completion of any previously initiated request.

Parameters

- `requests (Sequence[Request]) –`
- `status (Status / None) –`

Return type`tuple[int, bool]`**classmethod Testsome**(*requests, statuses=None*)

Test for completion of some previously initiated requests.

Parameters

- `requests (Sequence[Request]) –`
- `statuses (list[Status] / None) –`

Return type`list[int] | None`**Wait**(*status=None*)

Wait for a non-blocking operation to complete.

Parameters`status (Status / None) –`**Return type**`Literal[True]`**classmethod Waitall**(*requests, statuses=None*)

Wait for all previously initiated requests to complete.

Parameters

- `requests (Sequence[Request]) –`
- `statuses (list[Status] / None) –`

Return type

Literal[True]

classmethod **Waitany**(*requests*, *status*=None)

Wait for any previously initiated request to complete.

Parameters

- **requests** (*Sequence*[Request]) –
- **status** (*Status* / *None*) –

Return type

int

classmethod **Waitsome**(*requests*, *statuses*=None)

Wait for some previously initiated requests to complete.

Parameters

- **requests** (*Sequence*[Request]) –
- **statuses** (*list*[*Status*] / *None*) –

Return type

list[int] | *None*

cancel()

Cancel a request.

Return type

None

classmethod **f2py**(*arg*)

Parameters

arg (*int*) –

Return type

Request

classmethod **fromhandle**(*handle*)

Create object from MPI handle.

Parameters

handle (*int*) –

Return type

Request

get_status(*status*=None)

Non-destructive test for the completion of a request.

Parameters

status (*Status* / *None*) –

Return type

bool

classmethod **get_status_all**(*requests*, *statuses*=None)

Non-destructive test for the completion of all requests.

Parameters

- **requests** (*Sequence[Request]*) –
- **statuses** (*list[Status]* / *None*) –

Return type
`bool`

classmethod `get_status_any`(*requests*, *status=None*)

Non-destructive test for the completion of any requests.

Parameters

- **requests** (*Sequence[Request]*) –
- **status** (*Status* / *None*) –

Return type
`tuple[int, bool]`

classmethod `get_status_some`(*requests*, *statuses=None*)

Non-destructive test for completion of some requests.

Parameters

- **requests** (*Sequence[Request]*) –
- **statuses** (*list[Status]* / *None*) –

Return type
`list[int] | None`

py2f()

Return type
`int`

test(*status=None*)

Test for the completion of a non-blocking operation.

Parameters

- **status** (*Status* / *None*) –

Return type
`tuple[bool, Any | None]`

classmethod `testall`(*requests*, *statuses=None*)

Test for completion of all previously initiated requests.

Parameters

- **requests** (*Sequence[Request]*) –
- **statuses** (*list[Status]* / *None*) –

Return type
`tuple[bool, list[Any] | None]`

classmethod `testany`(*requests*, *status=None*)

Test for completion of any previously initiated request.

Parameters

- **requests** (*Sequence[Request]*) –
- **status** (*Status* / *None*) –

Return type

tuple[int, bool, Any | None]

classmethod testsome(*requests*, *statuses*=None)

Test for completion of some previously initiated requests.

Parameters

- **requests** (*Sequence[Request]*) –
- **statuses** (*list[Status] / None*) –

Return type

tuple[list[int] | None, list[Any] | None]

wait(*status*=None)

Wait for a non-blocking operation to complete.

Parameters

status (*Status / None*) –

Return type

Any

classmethod waitall(*requests*, *statuses*=None)

Wait for all previously initiated requests to complete.

Parameters

- **requests** (*Sequence[Request]*) –
- **statuses** (*list[Status] / None*) –

Return type

list[Any]

classmethod waitany(*requests*, *status*=None)

Wait for any previously initiated request to complete.

Parameters

- **requests** (*Sequence[Request]*) –
- **status** (*Status / None*) –

Return type

tuple[int, Any]

classmethod waitsome(*requests*, *statuses*=None)

Wait for some previously initiated requests to complete.

Parameters

- **requests** (*Sequence[Request]*) –
- **statuses** (*list[Status] / None*) –

Return type

tuple[list[int] | None, list[Any] | None]

Attributes Documentation

handle

MPI handle.

mpi4py.MPI.Session

class mpi4py.MPI.Session

Bases: `object`

Session context.

static `__new__(cls, session=None)`

Parameters

`session(Session / None) –`

Return type

`Self`

Methods Summary

<code>Attach_buffer(buf)</code>	Attach a user-provided buffer for sending in buffered mode.
<code>Call_errhandler(errorcode)</code>	Call the error handler installed on a session.
<code>Create_errhandler(errhandler_fn)</code>	Create a new error handler for sessions.
<code>Create_group(pset_name)</code>	Create a new group from session and process set.
<code>Detach_buffer()</code>	Remove an existing attached buffer.
<code>Finalize()</code>	Finalize a session.
<code>Flush_buffer()</code>	Block until all buffered messages have been transmitted.
<code>Get_errhandler()</code>	Get the error handler for a session.
<code>Get_info()</code>	Return the current hints for a session.
<code>Get_nth_pset(n[, info])</code>	Name of the n -th process set.
<code>Get_num_psets([info])</code>	Number of available process sets.
<code>Get_pset_info(pset_name)</code>	Return the current hints for a session and process set.
<code>Iflush_buffer()</code>	Nonblocking flush for buffered messages.
<code>Init([info, errhandler])</code>	Create a new session.
<code>Set_errhandler(errhandler)</code>	Set the error handler for a session.
<code>f2py(arg)</code>	
<code>fromhandle(handle)</code>	Create object from MPI handle.
<code>py2f()</code>	

Attributes Summary

<code>handle</code>	MPI handle.
---------------------	-------------

Methods Documentation

`Attach_buffer(buf)`

Attach a user-provided buffer for sending in buffered mode.

Parameters

`buf (Buffer / None) –`

Return type

`None`

`Call_errhandler(errorcode)`

Call the error handler installed on a session.

Parameters

`errorcode (int) –`

Return type

`None`

`classmethod Create_errhandler(errhandler_fn)`

Create a new error handler for sessions.

Parameters

`errhandler_fn (Callable[[Session, int], None]) –`

Return type

`Errhandler`

`Create_group(pset_name)`

Create a new group from session and process set.

Parameters

`pset_name (str) –`

Return type

`Group`

`Detach_buffer()`

Remove an existing attached buffer.

Return type

`Buffer | None`

`Finalize()`

Finalize a session.

Return type

`None`

`Flush_buffer()`

Block until all buffered messages have been transmitted.

Return type

`None`

Get_errhandler()

Get the error handler for a session.

Return type

Errhandler

Get_info()

Return the current hints for a session.

Return type

Info

Get_nth_pset(*n*, info=INFO_NULL)

Name of the *n*-th process set.

Parameters

- **n** (*int*) –
- **info** (*Info*) –

Return type

str

Get_num_psets(info=INFO_NULL)

Number of available process sets.

Parameters

info (*Info*) –

Return type

int

Get_pset_info(*pset_name*)

Return the current hints for a session and process set.

Parameters

pset_name (*str*) –

Return type

Info

Iflush_buffer()

Nonblocking flush for buffered messages.

Return type

Request

classmethod Init(info=INFO_NULL, errhandler=None)

Create a new session.

Parameters

- **info** (*Info*) –
- **errhandler** (*Errhandler* / *None*) –

Return type

Self

Set_errhandler(*errhandler*)

Set the error handler for a session.

Parameters
`errhandler` (`Errhandler`) –

Return type
`None`

classmethod `f2py(arg)`

Parameters
`arg` (`int`) –

Return type
`Session`

classmethod `fromhandle(handle)`

Create object from MPI handle.

Parameters
`handle` (`int`) –

Return type
`Session`

`py2f()`

Return type
`int`

Attributes Documentation

handle

MPI handle.

mpi4py.MPI.Status

class `mpi4py.MPI.Status`

Bases: `object`

Status object.

static `__new__(cls, status=None)`

Parameters
`status` (`Status` / `None`) –

Return type
`Self`

Methods Summary

<code>Get_count([datatype])</code>	Get the number of <i>top level</i> elements.
<code>Get_elements(datatype)</code>	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(flag)</code>	Set the cancelled state associated with a status.
<code>Set_elements(datatype, count)</code>	Set the number of elements in a status.
<code>Set_error(error)</code>	Set message error.
<code>Set_source(source)</code>	Set message source.
<code>Set_tag(tag)</code>	Set message tag.
<code>f2py(arg)</code>	
<code>py2f()</code>	

Attributes Summary

<code>cancelled</code>	Cancelled state.
<code>count</code>	Byte count.
<code>error</code>	Message error.
<code>source</code>	Message source.
<code>tag</code>	Message tag.

Methods Documentation

`Get_count(datatype=BYTE)`

Get the number of *top level* elements.

Parameters

`datatype` ([Datatype](#)) –

Return type

`int`

`Get_elements(datatype)`

Get the number of basic elements in a datatype.

Parameters

`datatype` ([Datatype](#)) –

Return type

`int`

`Get_error()`

Get message error.

Return type

`int`

Get_source()

Get message source.

Return type

`int`

Get_tag()

Get message tag.

Return type

`int`

Is_cancelled()

Test to see if a request was cancelled.

Return type

`bool`

Set_cancelled(*flag*)

Set the cancelled state associated with a status.

Note: This method should be used only 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 method 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
        Self

py2f()

    Return type
        list[int]
```

Attributes Documentation

cancelled
Cancelled state.

count
Byte count.

error
Message error.

source
Message source.

tag
Message tag.

mpi4py.MPI.Topocomm

```
class mpi4py.MPI.Topocomm
    Bases: Intracomm

    Topology intracommunicator.
```

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

Parameters

`comm` (`TopoComm` / `None`) –

Return type

`Self`

Methods Summary

<code>Ineighbor_allgather(sendbuf, recvbuf)</code>	Nonblocking Neighbor Gather to All.
<code>Ineighbor_allgatherv(sendbuf, recvbuf)</code>	Nonblocking Neighbor Gather to All Vector.
<code>Ineighbor_alltoall(sendbuf, recvbuf)</code>	Nonblocking Neighbor All to All.
<code>Ineighbor_alltoallv(sendbuf, recvbuf)</code>	Nonblocking Neighbor All to All Vector.
<code>Ineighbor_alltoallw(sendbuf, recvbuf)</code>	Nonblocking Neighbor All to All General.
<code>Neighbor_allgather(sendbuf, recvbuf)</code>	Neighbor Gather to All.
<code>Neighbor_allgather_init(sendbuf, recvbuf[, info])</code>	Persistent Neighbor Gather to All.
<code>Neighbor_allgatherv(sendbuf, recvbuf)</code>	Neighbor Gather to All Vector.
<code>Neighbor_allgatherv_init(sendbuf, recvbuf[, ...])</code>	Persistent Neighbor Gather to All Vector.
<code>Neighbor_alltoall(sendbuf, recvbuf)</code>	Neighbor All to All.
<code>Neighbor_alltoall_init(sendbuf, recvbuf[, info])</code>	Persistent Neighbor All to All.
<code>Neighbor_alltoallv(sendbuf, recvbuf)</code>	Neighbor All to All Vector.
<code>Neighbor_alltoallv_init(sendbuf, recvbuf[, info])</code>	Persistent Neighbor All to All Vector.
<code>Neighbor_alltoallw(sendbuf, recvbuf)</code>	Neighbor All to All General.
<code>Neighbor_alltoallw_init(sendbuf, recvbuf[, info])</code>	Persistent Neighbor All to All General.
<code>neighbor_allgather(sendobj)</code>	Neighbor Gather to All.
<code>neighbor_alltoall(sendobj)</code>	Neighbor All to All.

Attributes Summary

<code>degrees</code>	Number of incoming and outgoing neighbors.
<code>indegree</code>	Number of incoming neighbors.
<code>inedges</code>	Incoming neighbors.
<code>inoutedges</code>	Incoming and outgoing neighbors.
<code>outdegree</code>	Number of outgoing neighbors.
<code>outedges</code>	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 General.

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_allgather_init(*sendbuf*, *recvbuf*, *info*=*INFO_NULL*)

Persistent Neighbor Gather to All.

Parameters

- **sendbuf** ([BufSpec](#)) –
- **recvbuf** ([BufSpecB](#)) –
- **info** ([Info](#)) –

Return type

[Prequest](#)

Neighbor_allgatherv(*sendbuf*, *recvbuf*)

Neighbor Gather to All Vector.

Parameters

- **sendbuf** ([BufSpec](#)) –
- **recvbuf** ([BufSpecV](#)) –

Return type

[None](#)

Neighbor_allgatherv_init(*sendbuf*, *recvbuf*, *info*=*INFO_NULL*)

Persistent Neighbor Gather to All Vector.

Parameters

- **sendbuf** ([BufSpec](#)) –
- **recvbuf** ([BufSpecV](#)) –
- **info** ([Info](#)) –

Return type

[Prequest](#)

Neighbor_alltoall(*sendbuf*, *recvbuf*)

Neighbor All to All.

Parameters

- **sendbuf** ([BufSpecB](#)) –
- **recvbuf** ([BufSpecB](#)) –

Return type

[None](#)

Neighbor_alltoall_init(*sendbuf*, *recvbuf*, *info*=*INFO_NULL*)

Persistent Neighbor All to All.

Parameters

- **sendbuf** ([BufSpecB](#)) –
- **recvbuf** ([BufSpecB](#)) –
- **info** ([Info](#)) –

Return type

[Prequest](#)

Neighbor_alltoallv(*sendbuf*, *recvbuf*)

Neighbor All to All Vector.

Parameters

- **sendbuf** ([BufSpecV](#)) –
- **recvbuf** ([BufSpecV](#)) –

Return type

[None](#)

Neighbor_alltoallv_init(*sendbuf*, *recvbuf*, *info*=*INFO_NULL*)

Persistent Neighbor All to All Vector.

Parameters

- **sendbuf** ([BufSpecV](#)) –
- **recvbuf** ([BufSpecV](#)) –
- **info** ([Info](#)) –

Return type

[Prequest](#)

Neighbor_alltoallw(*sendbuf*, *recvbuf*)

Neighbor All to All General.

Parameters

- **sendbuf** ([BufSpecW](#)) –
- **recvbuf** ([BufSpecW](#)) –

Return type

[None](#)

Neighbor_alltoallw_init(*sendbuf*, *recvbuf*, *info*=*INFO_NULL*)

Persistent Neighbor All to All General.

Parameters

- **sendbuf** ([BufSpecW](#)) –
- **recvbuf** ([BufSpecW](#)) –
- **info** ([Info](#)) –

Return type

[Prequest](#)

neighbor_allgather(*sendobj*)

Neighbor Gather to All.

Parameters

sendobj (*Any*) –

Return type

list[Any]

neighbor_alltoall(*sendobj*)

Neighbor All to All.

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

Bases: *object*

Remote memory access context.

static __new__(*cls, win=None*)

Parameters

win (*Win* / *None*) –

Return type

Self

Methods Summary

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

continues on next page

Table 5 – continued from previous page

<code>Shared_query(rank)</code>	Query the process-local address for remote memory segments.
<code>Start(group[, assertion])</code>	Start an RMA access epoch for MPI.
<code>Sync()</code>	Synchronize public and private copies of the window.
<code>Test()</code>	Test whether an RMA exposure epoch has completed.
<code>Unlock(rank)</code>	Complete an RMA access epoch at the target process.
<code>Unlock_all()</code>	Complete an RMA access epoch at all processes.
<code>Wait()</code>	Complete an RMA exposure epoch begun with <code>Post</code> .
<code>f2py(arg)</code>	
<code>fromhandle(handle)</code>	Create object from MPI handle.
<code>py2f()</code>	
<code>tomemory()</code>	Return window memory buffer.

Attributes Summary

<code>attrs</code>	Attributes.
<code>flavor</code>	Create flavor.
<code>group</code>	Group.
<code>handle</code>	MPI handle.
<code>info</code>	Info hints.
<code>model</code>	Memory model.
<code>name</code>	Print name.

Methods Documentation

Accumulate(*origin*, *target_rank*, *target=None*, *op=SUM*)

Accumulate data into the target process.

Parameters

- **origin** (`BufSpec`) –
- **target_rank** (`int`) –
- **target** (`TargetSpec` / `None`) –
- **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*Self***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** (*Intracom*) –

Return type*Self***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()Complete an RMA operation begun after an *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** (`Buffer` / `Bottom`) –
- **disp_unit** (`int`) –
- **info** (`Info`) –
- **comm** (`Intracomm`) –

Return type

`Self`

classmethod `Create_dynamic`(`info=INFO_NULL, comm=COMM_SELF`)

Create an window object for one-sided communication.

Parameters

- **info** (`Info`) –
- **comm** (`Intracomm`) –

Return type

`Self`

classmethod `Create_errhandler`(`errhandler_fn`)

Create a new error handler for windows.

Parameters

`errhandler_fn` (`Callable[[Win, int], None]`) –

Return type

`Errhandler`

classmethod `Create_keyval`(`copy_fn=None, delete_fn=None, nopython=False`)

Create a new attribute key for windows.

Parameters

- **copy_fn** (`Callable[[Win, int, Any], Any] / None`) –
- **delete_fn** (`Callable[[Win, int, Any], None] / 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 a 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 a 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** (*TargetSpec* / *None*) –

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** (*TargetSpec* / *None*) –
- **op** (*Op*) –

Return type

None

Get_attr(*keyval*)

Retrieve attribute value by key.

Parameters

keyval (*int*) –

Return type

int | *Any* | *None*

Get_errhandler()

Get the error handler for a window.

Return type

Errhandler

Get_group()

Access the group of processes that created the window.

Return type

Group

Get_info()

Return the current hints for a window.

Return type

Info

Get_name()

Get the print name for this 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** (*TargetSpec* / *None*) –

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** (`TargetSpec` / `None`) –
- **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** (`TargetSpec` / `None`) –

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** (`TargetSpec` / `None`) –
- **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** (`TargetSpec` / `None`) –

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 for this window.

Parameters

name ([str](#)) –

Return type

None

Shared_query(*rank*)

Query the process-local address for remote memory segments.

Parameters

rank ([int](#)) –

Return type

[tuple](#)[[buffer](#), [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 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 *Post*.

Return type

Literal[True]

classmethod f2py(*arg*)**Parameters**

arg (*int*) –

Return type

Win

classmethod fromhandle(*handle*)

Create object from MPI handle.

Parameters

handle (*int*) –

Return type

Win

py2f()**Return type**

int

tomemory()

Return window memory buffer.

Return type

buffer

Attributes Documentation

attrs

Attributes.

flavor

Create flavor.

group

Group.

handle

MPI handle.

info

Info hints.

model

Memory model.

name

Print name.

mpi4py.MPI.buffer

class mpi4py.MPI.buffer

Bases: `object`

Buffer.

static __new__(cls, buf)

Parameters

`buf (Buffer)` –

Return type

`Self`

Methods Summary

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

Attributes Summary

<code>address</code>	Buffer address.
<code>format</code>	Format of each element.
<code>itemsize</code>	Size (in bytes) of each element.
<code>nbytes</code>	Buffer size (in bytes).
<code>obj</code>	Object exposing buffer.
<code>readonly</code>	Buffer is read-only.

Methods Documentation

static allocate(*nbytes*, *clear=False*)

Buffer allocation.

Parameters

- **nbytes** (*int*) –
- **clear** (*bool*) –

Return type

buffer

static fromaddress(*address*, *nbytes*, *readonly=False*)

Buffer from address and size in bytes.

Parameters

- **address** (*int*) –
- **nbytes** (*int*) –
- **readonly** (*bool*) –

Return type

buffer

static frombuffer(*obj*, *readonly=False*)

Buffer from buffer-like object.

Parameters

- **obj** (*Buffer*) –
- **readonly** (*bool*) –

Return type

buffer

release()

Release the underlying buffer exposed by the buffer object.

Return type

None

tobytes(*order=None*)

Return the data in the buffer as a byte string.

Parameters

- **order** (*str* / *None*) –

Return type

bytes

toreadonly()

Return a readonly version of the buffer object.

Return type

buffer

Attributes Documentation

address

Buffer address.

format

Format of each element.

itemsize

Size (in bytes) of each element.

nbytes

Buffer size (in bytes).

obj

Object exposing buffer.

readonly

Buffer is read-only.

mpi4py.MPI.memory

mpi4py.MPI.memory

alias of *buffer*

Exceptions

<i>Exception</i>	Exception class.
------------------	------------------

mpi4py.MPI.Exception

exception mpi4py.MPI.Exception

Bases: `RuntimeError`

Exception class.

static `__new__(cls, ierr=SUCCESS)`

Parameters

`ierr (int)` –

Return type

Self

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 remote memory access.
<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>Flush_buffer()</code>	Block until all buffered messages have been transmitted.
<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_hw_resource_info()</code>	Obtain information about the hardware platform of the calling processor.
<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.
<code>Iflush_buffer()</code>	Nonblocking flush for buffered messages.
<code>Init()</code>	Initialize the MPI execution environment.
<code>Init_thread([required])</code>	Initialize the MPI execution environment.
<code>Is_finalized()</code>	Indicate whether <code>Finalize</code> has completed.
<code>Is_initialized()</code>	Indicate 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 used to connect group of 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>Remove_error_class(errorclass)</code>	Remove an <i>error class</i> from the known error classes.
<code>Remove_error_code(errorcode)</code>	Remove an <i>error code</i> from the known error codes.
<code>Remove_error_string(errorcode)</code>	Remove <i>error string</i> association from <i>error class</i> or <i>error code</i> .
<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 *error code*.

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 remote memory access.

Parameters

- **size** (`int`) –
- **info** (`Info`) –

Return type

`buffer`

mpi4py.MPI.Attach_buffer

`mpi4py.MPI.Attach_buffer(buf)`

Attach a user-provided buffer for sending in buffered mode.

Parameters

buf (`Buffer` / `None`) –

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(nnode, dim)`

Return a balanced distribution of processes per coordinate direction.

Parameters

- `nnode` (`int`) –
- `dim` (`int` / `Sequence[int]`) –

Return type

`list[int]`

`mpi4py.MPI.Detach_buffer`

`mpi4py.MPI.Detach_buffer()`

Remove an existing attached buffer.

Return type

`Buffer | None`

`mpi4py.MPI.Finalize`

`mpi4py.MPI.Finalize()`

Terminate the MPI execution environment.

Return type

`None`

`mpi4py.MPI.Flush_buffer`

`mpi4py.MPI.Flush_buffer()`

Block until all buffered messages have been transmitted.

Return type

`None`

`mpi4py.MPI.Free_mem`

`mpi4py.MPI.Free_mem(mem)`

Free memory allocated with `Alloc_mem`.

Parameters

`mem` (`buffer`) –

Return type

`None`

mpi4py.MPI.Get_address

`mpi4py.MPI.Get_address(location)`

Get the address of a location in memory.

Parameters

`location` (`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_hw_resource_info

`mpi4py.MPI.Get_hw_resource_info()`

Obtain information about the hardware platform of the calling processor.

Return type

`Info`

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.

Return type

`tuple[int, int]`

mpi4py.MPI.Iflush_buffer

`mpi4py.MPI.Iflush_buffer()`

Nonblocking flush for buffered messages.

Return type

`Request`

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()`

Indicate whether `Finalize` has completed.

Return type

`bool`

`mpi4py.MPI.Is_initialized`

`mpi4py.MPI.Is_initialized()`

Indicate 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 used to connect group of 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.Remove_error_class`

`mpi4py.MPI.Remove_error_class(errorclass)`

Remove an *error class* from the known error classes.

Parameters

`errorclass (int)` –

Return type

`None`

mpi4py.MPI.Remove_error_code

`mpi4py.MPI.Remove_error_code(errorcode)`

Remove an *error code* from the known error codes.

Parameters

`errorcode (int)` –

Return type

`None`

mpi4py.MPI.Remove_error_string

`mpi4py.MPI.Remove_error_string(errorcode)`

Remove *error string* association from *error class* or *error code*.

Parameters

`errorcode (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

- string with the name of the MPI implementation.
- integer 3-tuple version number (`major`, `minor`, `micro`).

Return type

tuple[str, tuple[int, int, int]]

Attributes

<code>UNDEFINED</code>	Constant UNDEFINED of type int
<code>ANY_SOURCE</code>	Constant ANY_SOURCE of type int
<code>ANY_TAG</code>	Constant ANY_TAG of type int
<code>PROC_NULL</code>	Constant PROC_NULL of type int
<code>ROOT</code>	Constant ROOT of type int
<code>BOTTOM</code>	Constant BOTTOM of type BottomType
<code>IN_PLACE</code>	Constant IN_PLACE of type InPlaceType
<code>KEYVAL_INVALID</code>	Constant KEYVAL_INVALID of type int
<code>TAG_UB</code>	Constant TAG_UB of type int
<code>HOST</code>	Constant HOST of type int
<code>IO</code>	Constant IO of type int
<code>WTIME_IS_GLOBAL</code>	Constant WTIME_IS_GLOBAL of type int
<code>UNIVERSE_SIZE</code>	Constant UNIVERSE_SIZE of type int
<code>APPNUM</code>	Constant APPNUM of type int
<code>LASTUSEDCODE</code>	Constant LASTUSEDCODE of type int
<code>WIN_BASE</code>	Constant WIN_BASE of type int
<code>WIN_SIZE</code>	Constant WIN_SIZE of type int
<code>WIN_DISP_UNIT</code>	Constant WIN_DISP_UNIT of type int
<code>WIN_CREATE_FLAVOR</code>	Constant WIN_CREATE_FLAVOR of type int
<code>WIN_FLAVOR</code>	Constant WIN_FLAVOR of type int
<code>WIN_MODEL</code>	Constant WIN_MODEL of type int
<code>SUCCESS</code>	Constant SUCCESS of type int
<code>ERR_LASTCODE</code>	Constant ERR_LASTCODE of type int
<code>ERR_TYPE</code>	Constant ERR_TYPE of type int
<code>ERR_REQUEST</code>	Constant ERR_REQUEST of type int
<code>ERR_OP</code>	Constant ERR_OP of type int
<code>ERR_GROUP</code>	Constant ERR_GROUP of type int
<code>ERR_INFO</code>	Constant ERR_INFO of type int
<code>ERR_ERRHANDLER</code>	Constant ERR_ERRHANDLER of type int

continues on next page

Table 7 – continued from previous page

<i>ERR_SESSION</i>	Constant <i>ERR_SESSION</i> of type <i>int</i>
<i>ERR_COMM</i>	Constant <i>ERR_COMM</i> of type <i>int</i>
<i>ERR_WIN</i>	Constant <i>ERR_WIN</i> of type <i>int</i>
<i>ERR_FILE</i>	Constant <i>ERR_FILE</i> of type <i>int</i>
<i>ERR_BUFFER</i>	Constant <i>ERR_BUFFER</i> of type <i>int</i>
<i>ERR_COUNT</i>	Constant <i>ERR_COUNT</i> of type <i>int</i>
<i>ERR_TAG</i>	Constant <i>ERR_TAG</i> of type <i>int</i>
<i>ERR_RANK</i>	Constant <i>ERR_RANK</i> of type <i>int</i>
<i>ERR_ROOT</i>	Constant <i>ERR_ROOT</i> of type <i>int</i>
<i>ERR_TRUNCATE</i>	Constant <i>ERR_TRUNCATE</i> of type <i>int</i>
<i>ERR_IN_STATUS</i>	Constant <i>ERR_IN_STATUS</i> of type <i>int</i>
<i>ERR_PENDING</i>	Constant <i>ERR_PENDING</i> of type <i>int</i>
<i>ERR_TOPOLOGY</i>	Constant <i>ERR_TOPOLOGY</i> of type <i>int</i>
<i>ERR_DIMS</i>	Constant <i>ERR_DIMS</i> of type <i>int</i>
<i>ERR_ARG</i>	Constant <i>ERR_ARG</i> of type <i>int</i>
<i>ERR_OTHER</i>	Constant <i>ERR_OTHER</i> of type <i>int</i>
<i>ERR_UNKNOWN</i>	Constant <i>ERR_UNKNOWN</i> of type <i>int</i>
<i>ERR_INTERN</i>	Constant <i>ERR_INTERN</i> of type <i>int</i>
<i>ERR_KEYVAL</i>	Constant <i>ERR_KEYVAL</i> of type <i>int</i>
<i>ERR_NO_MEM</i>	Constant <i>ERR_NO_MEM</i> of type <i>int</i>
<i>ERR_INFO_KEY</i>	Constant <i>ERR_INFO_KEY</i> of type <i>int</i>
<i>ERR_INFO_VALUE</i>	Constant <i>ERR_INFO_VALUE</i> of type <i>int</i>
<i>ERR_INFO_NOKEY</i>	Constant <i>ERR_INFO_NOKEY</i> of type <i>int</i>
<i>ERR_SPAWN</i>	Constant <i>ERR_SPAWN</i> of type <i>int</i>
<i>ERR_PORT</i>	Constant <i>ERR_PORT</i> of type <i>int</i>
<i>ERR_SERVICE</i>	Constant <i>ERR_SERVICE</i> of type <i>int</i>
<i>ERR_NAME</i>	Constant <i>ERR_NAME</i> of type <i>int</i>
<i>ERR_PROC_ABORTED</i>	Constant <i>ERR_PROC_ABORTED</i> of type <i>int</i>
<i>ERR_BASE</i>	Constant <i>ERR_BASE</i> of type <i>int</i>
<i>ERR_SIZE</i>	Constant <i>ERR_SIZE</i> of type <i>int</i>
<i>ERR_DISP</i>	Constant <i>ERR_DISP</i> of type <i>int</i>
<i>ERR_ASSERT</i>	Constant <i>ERR_ASSERT</i> of type <i>int</i>
<i>ERR_LOCKTYPE</i>	Constant <i>ERR_LOCKTYPE</i> of type <i>int</i>
<i>ERR_RMA_CONFLICT</i>	Constant <i>ERR_RMA_CONFLICT</i> of type <i>int</i>
<i>ERR_RMA_SYNC</i>	Constant <i>ERR_RMA_SYNC</i> of type <i>int</i>
<i>ERR_RMA_RANGE</i>	Constant <i>ERR_RMA_RANGE</i> of type <i>int</i>
<i>ERR_RMA_ATTACH</i>	Constant <i>ERR_RMA_ATTACH</i> of type <i>int</i>
<i>ERR_RMA_SHARED</i>	Constant <i>ERR_RMA_SHARED</i> of type <i>int</i>
<i>ERR_RMA_FLAVOR</i>	Constant <i>ERR_RMA_FLAVOR</i> of type <i>int</i>
<i>ERR_BAD_FILE</i>	Constant <i>ERR_BAD_FILE</i> of type <i>int</i>
<i>ERR_NO SUCH FILE</i>	Constant <i>ERR_NO SUCH FILE</i> of type <i>int</i>
<i>ERR_FILE_EXISTS</i>	Constant <i>ERR_FILE_EXISTS</i> of type <i>int</i>
<i>ERR_FILE_IN_USE</i>	Constant <i>ERR_FILE_IN_USE</i> of type <i>int</i>
<i>ERR_AMODE</i>	Constant <i>ERR_AMODE</i> of type <i>int</i>
<i>ERR_ACCESS</i>	Constant <i>ERR_ACCESS</i> of type <i>int</i>
<i>ERR_READ_ONLY</i>	Constant <i>ERR_READ_ONLY</i> of type <i>int</i>
<i>ERR_NO_SPACE</i>	Constant <i>ERR_NO_SPACE</i> of type <i>int</i>
<i>ERR_QUOTA</i>	Constant <i>ERR_QUOTA</i> of type <i>int</i>
<i>ERR_NOT_SAME</i>	Constant <i>ERR_NOT_SAME</i> of type <i>int</i>
<i>ERR_IO</i>	Constant <i>ERR_IO</i> of type <i>int</i>
<i>ERR_UNSUPPORTED_OPERATION</i>	Constant <i>ERR_UNSUPPORTED_OPERATION</i> of type <i>int</i>
<i>ERR_UNSUPPORTED_DATAREP</i>	Constant <i>ERR_UNSUPPORTED_DATAREP</i> of type <i>int</i>

continues on next page

Table 7 – continued from previous page

<code>ERR_CONVERSION</code>	Constant <code>ERR_CONVERSION</code> of type <code>int</code>
<code>ERR_DUP_DATAREP</code>	Constant <code>ERR_DUP_DATAREP</code> of type <code>int</code>
<code>ERR_VALUE_TOO_LARGE</code>	Constant <code>ERR_VALUE_TOO_LARGE</code> of type <code>int</code>
<code>ERR_REVOKED</code>	Constant <code>ERR_REVOKED</code> of type <code>int</code>
<code>ERR_PROC_FAILED</code>	Constant <code>ERR_PROC_FAILED</code> of type <code>int</code>
<code>ERR_PROC_FAILED_PENDING</code>	Constant <code>ERR_PROC_FAILED_PENDING</code> of type <code>int</code>
<code>ORDER_C</code>	Constant <code>ORDER_C</code> of type <code>int</code>
<code>ORDER_FORTRAN</code>	Constant <code>ORDER_FORTRAN</code> of type <code>int</code>
<code>ORDER_F</code>	Constant <code>ORDER_F</code> of type <code>int</code>
<code>TYPECLASS_INTEGER</code>	Constant <code>TYPECLASS_INTEGER</code> of type <code>int</code>
<code>TYPECLASS_REAL</code>	Constant <code>TYPECLASS_REAL</code> of type <code>int</code>
<code>TYPECLASS_COMPLEX</code>	Constant <code>TYPECLASS_COMPLEX</code> of type <code>int</code>
<code>DISTRIBUTE_NONE</code>	Constant <code>DISTRIBUTE_NONE</code> of type <code>int</code>
<code>DISTRIBUTE_BLOCK</code>	Constant <code>DISTRIBUTE_BLOCK</code> of type <code>int</code>
<code>DISTRIBUTE_CYCLIC</code>	Constant <code>DISTRIBUTE_CYCLIC</code> of type <code>int</code>
<code>DISTRIBUTE_DFLT_DARG</code>	Constant <code>DISTRIBUTE_DFLT_DARG</code> of type <code>int</code>
<code>COMBINER_NAMED</code>	Constant <code>COMBINER_NAMED</code> of type <code>int</code>
<code>COMBINER_DUP</code>	Constant <code>COMBINER_DUP</code> of type <code>int</code>
<code>COMBINER_CONTIGUOUS</code>	Constant <code>COMBINER_CONTIGUOUS</code> of type <code>int</code>
<code>COMBINER_VECTOR</code>	Constant <code>COMBINER_VECTOR</code> of type <code>int</code>
<code>COMBINER_HVECTOR</code>	Constant <code>COMBINER_HVECTOR</code> of type <code>int</code>
<code>COMBINER_INDEXED</code>	Constant <code>COMBINER_INDEXED</code> of type <code>int</code>
<code>COMBINER_HINDEXED</code>	Constant <code>COMBINER_HINDEXED</code> of type <code>int</code>
<code>COMBINER_INDEXED_BLOCK</code>	Constant <code>COMBINER_INDEXED_BLOCK</code> of type <code>int</code>
<code>COMBINER_HINDEXED_BLOCK</code>	Constant <code>COMBINER_HINDEXED_BLOCK</code> of type <code>int</code>
<code>COMBINER_STRUCT</code>	Constant <code>COMBINER_STRUCT</code> of type <code>int</code>
<code>COMBINER_SUBARRAY</code>	Constant <code>COMBINER_SUBARRAY</code> of type <code>int</code>
<code>COMBINER_DARRAY</code>	Constant <code>COMBINER_DARRAY</code> of type <code>int</code>
<code>COMBINER_RESIZED</code>	Constant <code>COMBINER_RESIZED</code> of type <code>int</code>
<code>COMBINER_VALUE_INDEX</code>	Constant <code>COMBINER_VALUE_INDEX</code> of type <code>int</code>
<code>COMBINER_F90_INTEGER</code>	Constant <code>COMBINER_F90_INTEGER</code> of type <code>int</code>
<code>COMBINER_F90_REAL</code>	Constant <code>COMBINER_F90_REAL</code> of type <code>int</code>
<code>COMBINER_F90_COMPLEX</code>	Constant <code>COMBINER_F90_COMPLEX</code> of type <code>int</code>
<code>F_SOURCE</code>	Constant <code>F_SOURCE</code> of type <code>int</code>
<code>F_TAG</code>	Constant <code>F_TAG</code> of type <code>int</code>
<code>F_ERROR</code>	Constant <code>F_ERROR</code> of type <code>int</code>
<code>F_STATUS_SIZE</code>	Constant <code>F_STATUS_SIZE</code> of type <code>int</code>
<code>IDENT</code>	Constant <code>IDENT</code> of type <code>int</code>
<code>CONGRUENT</code>	Constant <code>CONGRUENT</code> of type <code>int</code>
<code>SIMILAR</code>	Constant <code>SIMILAR</code> of type <code>int</code>
<code>UNEQUAL</code>	Constant <code>UNEQUAL</code> of type <code>int</code>
<code>CART</code>	Constant <code>CART</code> of type <code>int</code>
<code>GRAPH</code>	Constant <code>GRAPH</code> of type <code>int</code>
<code>DIST_GRAPH</code>	Constant <code>DIST_GRAPH</code> of type <code>int</code>
<code>UNWEIGHTED</code>	Constant <code>UNWEIGHTED</code> of type <code>int</code>
<code>WEIGHTS_EMPTY</code>	Constant <code>WEIGHTS_EMPTY</code> of type <code>int</code>
<code>COMM_TYPE_SHARED</code>	Constant <code>COMM_TYPE_SHARED</code> of type <code>int</code>
<code>COMM_TYPE_HW_GUIDED</code>	Constant <code>COMM_TYPE_HW_GUIDED</code> of type <code>int</code>
<code>COMM_TYPE_HW_UNGUIDED</code>	Constant <code>COMM_TYPE_HW_UNGUIDED</code> of type <code>int</code>
<code>COMM_TYPE_RESOURCE_GUIDED</code>	Constant <code>COMM_TYPE_RESOURCE_GUIDED</code> of type <code>int</code>
<code>BSEND_OVERHEAD</code>	Constant <code>BSEND_OVERHEAD</code> of type <code>int</code>

continues on next page

Table 7 – continued from previous page

<i>BUFFER_AUTOMATIC</i>	Constant <i>BUFFER_AUTOMATIC</i> of type <i>BufferAutomaticType</i>
<i>WIN_FLAVOR_CREATE</i>	Constant <i>WIN_FLAVOR_CREATE</i> of type <i>int</i>
<i>WIN_FLAVOR_ALLOCATE</i>	Constant <i>WIN_FLAVOR_ALLOCATE</i> of type <i>int</i>
<i>WIN_FLAVOR_DYNAMIC</i>	Constant <i>WIN_FLAVOR_DYNAMIC</i> of type <i>int</i>
<i>WIN_FLAVOR_SHARED</i>	Constant <i>WIN_FLAVOR_SHARED</i> of type <i>int</i>
<i>WIN_SEPARATE</i>	Constant <i>WIN_SEPARATE</i> of type <i>int</i>
<i>WIN_UNIFIED</i>	Constant <i>WIN_UNIFIED</i> of type <i>int</i>
<i>MODE_NOCHECK</i>	Constant <i>MODE_NOCHECK</i> of type <i>int</i>
<i>MODE_NOSTORE</i>	Constant <i>MODE_NOSTORE</i> of type <i>int</i>
<i>MODE_NOPUT</i>	Constant <i>MODE_NOPUT</i> of type <i>int</i>
<i>MODE_NOPRECEDE</i>	Constant <i>MODE_NOPRECEDE</i> of type <i>int</i>
<i>MODE_NOSUCCEED</i>	Constant <i>MODE_NOSUCCEED</i> of type <i>int</i>
<i>LOCK_EXCLUSIVE</i>	Constant <i>LOCK_EXCLUSIVE</i> of type <i>int</i>
<i>LOCK_SHARED</i>	Constant <i>LOCK_SHARED</i> of type <i>int</i>
<i>MODE_RDONLY</i>	Constant <i>MODE_RDONLY</i> of type <i>int</i>
<i>MODE_WRONLY</i>	Constant <i>MODE_WRONLY</i> of type <i>int</i>
<i>MODE_RDWR</i>	Constant <i>MODE_RDWR</i> of type <i>int</i>
<i>MODE_CREATE</i>	Constant <i>MODE_CREATE</i> of type <i>int</i>
<i>MODE_EXCL</i>	Constant <i>MODE_EXCL</i> of type <i>int</i>
<i>MODE_DELETE_ON_CLOSE</i>	Constant <i>MODE_DELETE_ON_CLOSE</i> of type <i>int</i>
<i>MODE_UNIQUE_OPEN</i>	Constant <i>MODE_UNIQUE_OPEN</i> of type <i>int</i>
<i>MODE_SEQUENTIAL</i>	Constant <i>MODE_SEQUENTIAL</i> of type <i>int</i>
<i>MODE_APPEND</i>	Constant <i>MODE_APPEND</i> of type <i>int</i>
<i>SEEK_SET</i>	Constant <i>SEEK_SET</i> of type <i>int</i>
<i>SEEK_CUR</i>	Constant <i>SEEK_CUR</i> of type <i>int</i>
<i>SEEK_END</i>	Constant <i>SEEK_END</i> of type <i>int</i>
<i>DISPLACEMENT_CURRENT</i>	Constant <i>DISPLACEMENT_CURRENT</i> of type <i>int</i>
<i>DISP_CUR</i>	Constant <i>DISP_CUR</i> of type <i>int</i>
<i>THREAD_SINGLE</i>	Constant <i>THREAD_SINGLE</i> of type <i>int</i>
<i>THREAD_FUNNELED</i>	Constant <i>THREAD_FUNNELED</i> of type <i>int</i>
<i>THREAD_SERIALIZED</i>	Constant <i>THREAD_SERIALIZED</i> of type <i>int</i>
<i>THREAD_MULTIPLE</i>	Constant <i>THREAD_MULTIPLE</i> of type <i>int</i>
<i>VERSION</i>	Constant <i>VERSION</i> of type <i>int</i>
<i>SUBVERSION</i>	Constant <i>SUBVERSION</i> of type <i>int</i>
<i>MAX_PROCESSOR_NAME</i>	Constant <i>MAX_PROCESSOR_NAME</i> of type <i>int</i>
<i>MAX_ERROR_STRING</i>	Constant <i>MAX_ERROR_STRING</i> of type <i>int</i>
<i>MAX_PORT_NAME</i>	Constant <i>MAX_PORT_NAME</i> of type <i>int</i>
<i>MAX_INFO_KEY</i>	Constant <i>MAX_INFO_KEY</i> of type <i>int</i>
<i>MAX_INFO_VAL</i>	Constant <i>MAX_INFO_VAL</i> of type <i>int</i>
<i>MAX_OBJECT_NAME</i>	Constant <i>MAX_OBJECT_NAME</i> of type <i>int</i>
<i>MAX_DATAREP_STRING</i>	Constant <i>MAX_DATAREP_STRING</i> of type <i>int</i>
<i>MAX_LIBRARY_VERSION_STRING</i>	Constant <i>MAX_LIBRARY_VERSION_STRING</i> of type <i>int</i>
<i>MAX_PSET_NAME_LEN</i>	Constant <i>MAX_PSET_NAME_LEN</i> of type <i>int</i>
<i>MAX_STRINGTAG_LEN</i>	Constant <i>MAX_STRINGTAG_LEN</i> of type <i>int</i>
<i>DATATYPE_NULL</i>	Object <i>DATATYPE_NULL</i> of type <i>Datatype</i>
<i>PACKED</i>	Object <i>PACKED</i> of type <i>Datatype</i>
<i>BYTE</i>	Object <i>BYTE</i> of type <i>Datatype</i>
<i>AINT</i>	Object <i>AINT</i> of type <i>Datatype</i>
<i>OFFSET</i>	Object <i>OFFSET</i> of type <i>Datatype</i>
<i>COUNT</i>	Object <i>COUNT</i> of type <i>Datatype</i>
<i>CHAR</i>	Object <i>CHAR</i> of type <i>Datatype</i>

continues on next page

Table 7 – continued from previous page

<i>WCHAR</i>	Object <i>WCHAR</i> of type <i>Datatype</i>
<i>SIGNED_CHAR</i>	Object <i>SIGNED_CHAR</i> of type <i>Datatype</i>
<i>SHORT</i>	Object <i>SHORT</i> of type <i>Datatype</i>
<i>INT</i>	Object <i>INT</i> of type <i>Datatype</i>
<i>LONG</i>	Object <i>LONG</i> of type <i>Datatype</i>
<i>LONG_LONG</i>	Object <i>LONG_LONG</i> of type <i>Datatype</i>
<i>UNSIGNED_CHAR</i>	Object <i>UNSIGNED_CHAR</i> of type <i>Datatype</i>
<i>UNSIGNED_SHORT</i>	Object <i>UNSIGNED_SHORT</i> of type <i>Datatype</i>
<i>UNSIGNED</i>	Object <i>UNSIGNED</i> of type <i>Datatype</i>
<i>UNSIGNED_LONG</i>	Object <i>UNSIGNED_LONG</i> of type <i>Datatype</i>
<i>UNSIGNED_LONG_LONG</i>	Object <i>UNSIGNED_LONG_LONG</i> of type <i>Datatype</i>
<i>FLOAT</i>	Object <i>FLOAT</i> of type <i>Datatype</i>
<i>DOUBLE</i>	Object <i>DOUBLE</i> of type <i>Datatype</i>
<i>LONG_DOUBLE</i>	Object <i>LONG_DOUBLE</i> of type <i>Datatype</i>
<i>C_BOOL</i>	Object <i>C_BOOL</i> of type <i>Datatype</i>
<i>INT8_T</i>	Object <i>INT8_T</i> of type <i>Datatype</i>
<i>INT16_T</i>	Object <i>INT16_T</i> of type <i>Datatype</i>
<i>INT32_T</i>	Object <i>INT32_T</i> of type <i>Datatype</i>
<i>INT64_T</i>	Object <i>INT64_T</i> of type <i>Datatype</i>
<i>UINT8_T</i>	Object <i>UINT8_T</i> of type <i>Datatype</i>
<i>UINT16_T</i>	Object <i>UINT16_T</i> of type <i>Datatype</i>
<i>UINT32_T</i>	Object <i>UINT32_T</i> of type <i>Datatype</i>
<i>UINT64_T</i>	Object <i>UINT64_T</i> of type <i>Datatype</i>
<i>C_COMPLEX</i>	Object <i>C_COMPLEX</i> of type <i>Datatype</i>
<i>C_FLOAT_COMPLEX</i>	Object <i>C_FLOAT_COMPLEX</i> of type <i>Datatype</i>
<i>C_DOUBLE_COMPLEX</i>	Object <i>C_DOUBLE_COMPLEX</i> of type <i>Datatype</i>
<i>C_LONG_DOUBLE_COMPLEX</i>	Object <i>C_LONG_DOUBLE_COMPLEX</i> of type <i>Datatype</i>
<i>CXX_BOOL</i>	Object <i>CXX_BOOL</i> of type <i>Datatype</i>
<i>CXX_FLOAT_COMPLEX</i>	Object <i>CXX_FLOAT_COMPLEX</i> of type <i>Datatype</i>
<i>CXX_DOUBLE_COMPLEX</i>	Object <i>CXX_DOUBLE_COMPLEX</i> of type <i>Datatype</i>
<i>CXX_LONG_DOUBLE_COMPLEX</i>	Object <i>CXX_LONG_DOUBLE_COMPLEX</i> of type <i>Datatype</i>
<i>SHORT_INT</i>	Object <i>SHORT_INT</i> of type <i>Datatype</i>
<i>INT_INT</i>	Object <i>INT_INT</i> of type <i>Datatype</i>
<i>TWOINT</i>	Object <i>TWOINT</i> of type <i>Datatype</i>
<i>LONG_INT</i>	Object <i>LONG_INT</i> of type <i>Datatype</i>
<i>FLOAT_INT</i>	Object <i>FLOAT_INT</i> of type <i>Datatype</i>
<i>DOUBLE_INT</i>	Object <i>DOUBLE_INT</i> of type <i>Datatype</i>
<i>LONG_DOUBLE_INT</i>	Object <i>LONG_DOUBLE_INT</i> of type <i>Datatype</i>
<i>CHARACTER</i>	Object <i>CHARACTER</i> of type <i>Datatype</i>
<i>LOGICAL</i>	Object <i>LOGICAL</i> of type <i>Datatype</i>
<i>INTEGER</i>	Object <i>INTEGER</i> of type <i>Datatype</i>
<i>REAL</i>	Object <i>REAL</i> of type <i>Datatype</i>
<i>DOUBLE_PRECISION</i>	Object <i>DOUBLE_PRECISION</i> of type <i>Datatype</i>
<i>COMPLEX</i>	Object <i>COMPLEX</i> of type <i>Datatype</i>
<i>DOUBLE_COMPLEX</i>	Object <i>DOUBLE_COMPLEX</i> of type <i>Datatype</i>
<i>LOGICAL1</i>	Object <i>LOGICAL1</i> of type <i>Datatype</i>
<i>LOGICAL2</i>	Object <i>LOGICAL2</i> of type <i>Datatype</i>
<i>LOGICAL4</i>	Object <i>LOGICAL4</i> of type <i>Datatype</i>
<i>LOGICAL8</i>	Object <i>LOGICAL8</i> of type <i>Datatype</i>
<i>INTEGER1</i>	Object <i>INTEGER1</i> of type <i>Datatype</i>
<i>INTEGER2</i>	Object <i>INTEGER2</i> of type <i>Datatype</i>
<i>INTEGER4</i>	Object <i>INTEGER4</i> of type <i>Datatype</i>

continues on next page

Table 7 – continued from previous page

INTEGER8	Object INTEGER8 of type <i>Datatype</i>
INTEGER16	Object INTEGER16 of type <i>Datatype</i>
REAL2	Object REAL2 of type <i>Datatype</i>
REAL4	Object REAL4 of type <i>Datatype</i>
REAL8	Object REAL8 of type <i>Datatype</i>
REAL16	Object REAL16 of type <i>Datatype</i>
COMPLEX4	Object COMPLEX4 of type <i>Datatype</i>
COMPLEX8	Object COMPLEX8 of type <i>Datatype</i>
COMPLEX16	Object COMPLEX16 of type <i>Datatype</i>
COMPLEX32	Object COMPLEX32 of type <i>Datatype</i>
UNSIGNED_INT	Object UNSIGNED_INT of type <i>Datatype</i>
SIGNED_SHORT	Object SIGNED_SHORT of type <i>Datatype</i>
SIGNED_INT	Object SIGNED_INT of type <i>Datatype</i>
SIGNED_LONG	Object SIGNED_LONG of type <i>Datatype</i>
SIGNED_LONG_LONG	Object SIGNED_LONG_LONG of type <i>Datatype</i>
BOOL	Object BOOL of type <i>Datatype</i>
SINT8_T	Object SINT8_T of type <i>Datatype</i>
SINT16_T	Object SINT16_T of type <i>Datatype</i>
SINT32_T	Object SINT32_T of type <i>Datatype</i>
SINT64_T	Object SINT64_T of type <i>Datatype</i>
F_BOOL	Object F_BOOL of type <i>Datatype</i>
F_INT	Object F_INT of type <i>Datatype</i>
F_FLOAT	Object F_FLOAT of type <i>Datatype</i>
F_DOUBLE	Object F_DOUBLE of type <i>Datatype</i>
F_COMPLEX	Object F_COMPLEX of type <i>Datatype</i>
F_FLOAT_COMPLEX	Object F_FLOAT_COMPLEX of type <i>Datatype</i>
F_DOUBLE_COMPLEX	Object F_DOUBLE_COMPLEX of type <i>Datatype</i>
REQUEST_NULL	Object REQUEST_NULL of type <i>Request</i>
MESSAGE_NULL	Object MESSAGE_NULL of type <i>Message</i>
MESSAGE_NO_PROC	Object MESSAGE_NO_PROC of type <i>Message</i>
OP_NULL	Object OP_NULL of type <i>Op</i>
MAX	Object MAX of type <i>Op</i>
MIN	Object MIN of type <i>Op</i>
SUM	Object SUM of type <i>Op</i>
PROD	Object PROD of type <i>Op</i>
LAND	Object LAND of type <i>Op</i>
BAND	Object BAND of type <i>Op</i>
LOR	Object LOR of type <i>Op</i>
BOR	Object BOR of type <i>Op</i>
LXOR	Object LXOR of type <i>Op</i>
BXOR	Object BXOR of type <i>Op</i>
MAXLOC	Object MAXLOC of type <i>Op</i>
MINLOC	Object MINLOC of type <i>Op</i>
REPLACE	Object REPLACE of type <i>Op</i>
NO_OP	Object NO_OP of type <i>Op</i>
GROUP_NULL	Object GROUP_NULL of type <i>Group</i>
GROUP_EMPTY	Object GROUP_EMPTY of type <i>Group</i>
INFO_NULL	Object INFO_NULL of type <i>Info</i>
INFO_ENV	Object INFO_ENV of type <i>Info</i>
ERRHANDLER_NULL	Object ERRHANDLER_NULL of type <i>Errhandler</i>
ERRORS_RETURN	Object ERRORS_RETURN of type <i>Errhandler</i>
ERRORS_ABORT	Object ERRORS_ABORT of type <i>Errhandler</i>

continues on next page

Table 7 – continued from previous page

<code>ERRORS_ARE_FATAL</code>	Object <code>ERRORS_ARE_FATAL</code> of type <code>Errhandler</code>
<code>SESSION_NULL</code>	Object <code>SESSION_NULL</code> of type <code>Session</code>
<code>COMM_NULL</code>	Object <code>COMM_NULL</code> of type <code>Comm</code>
<code>COMM_SELF</code>	Object <code>COMM_SELF</code> of type <code>Intracomm</code>
<code>COMM_WORLD</code>	Object <code>COMM_WORLD</code> of type <code>Intracomm</code>
<code>WIN_NULL</code>	Object <code>WIN_NULL</code> of type <code>Win</code>
<code>FILE_NULL</code>	Object <code>FILE_NULL</code> of type <code>File</code>
<code>pickle</code>	Object <code>pickle</code> of type <code>Pickle</code>

mpi4py.MPI.UNDEFINED

`mpi4py.MPI.UNDEFINED: int = UNDEFINED`

Constant `UNDEFINED` of type `int`

mpi4py.MPI.ANY_SOURCE

`mpi4py.MPI.ANY_SOURCE: int = ANY_SOURCE`

Constant `ANY_SOURCE` of type `int`

mpi4py.MPI.ANY_TAG

`mpi4py.MPI.ANY_TAG: int = ANY_TAG`

Constant `ANY_TAG` of type `int`

mpi4py.MPI.PROC_NULL

`mpi4py.MPI.PROC_NULL: int = PROC_NULL`

Constant `PROC_NULL` of type `int`

mpi4py.MPI.ROOT

`mpi4py.MPI.ROOT: int = ROOT`

Constant `ROOT` of type `int`

mpi4py.MPI.BOTTOM

`mpi4py.MPI.BOTTOM: BottomType = BOTTOM`

Constant `BOTTOM` of type `BottomType`

mpi4py.MPI.IN_PLACE

mpi4py.MPI.IN_PLACE: `InPlaceType` = IN_PLACE

Constant IN_PLACE of type `InPlaceType`

mpi4py.MPI.KEYVAL_INVALID

mpi4py.MPI.KEYVAL_INVALID: `int` = KEYVAL_INVALID

Constant KEYVAL_INVALID of type `int`

mpi4py.MPI.TAG_UB

mpi4py.MPI.TAG_UB: `int` = TAG_UB

Constant TAG_UB of type `int`

mpi4py.MPI.HOST

mpi4py.MPI.HOST: `int` = HOST

Constant HOST of type `int`

mpi4py.MPI.IO

mpi4py.MPI.IO: `int` = IO

Constant IO of type `int`

mpi4py.MPI.WTIME_IS_GLOBAL

mpi4py.MPI.WTIME_IS_GLOBAL: `int` = WTIME_IS_GLOBAL

Constant WTIME_IS_GLOBAL of type `int`

mpi4py.MPI.UNIVERSE_SIZE

mpi4py.MPI.UNIVERSE_SIZE: `int` = UNIVERSE_SIZE

Constant UNIVERSE_SIZE of type `int`

mpi4py.MPI.APPNUM

mpi4py.MPI.APPNUM: `int` = APPNUM

Constant APPNUM of type `int`

mpi4py.MPI.LASTUSEDCODE

`mpi4py.MPI.LASTUSEDCODE: int = LASTUSEDCODE`
Constant LASTUSEDCODE of type int

mpi4py.MPI.WIN_BASE

`mpi4py.MPI.WIN_BASE: int = WIN_BASE`
Constant WIN_BASE of type int

mpi4py.MPI.WIN_SIZE

`mpi4py.MPI.WIN_SIZE: int = WIN_SIZE`
Constant WIN_SIZE of type int

mpi4py.MPI.WIN_DISP_UNIT

`mpi4py.MPI.WIN_DISP_UNIT: int = WIN_DISP_UNIT`
Constant WIN_DISP_UNIT of type int

mpi4py.MPI.WIN_CREATE_FLAVOR

`mpi4py.MPI.WIN_CREATE_FLAVOR: int = WIN_CREATE_FLAVOR`
Constant WIN_CREATE_FLAVOR of type int

mpi4py.MPI.WIN_FLAVOR

`mpi4py.MPI.WIN_FLAVOR: int = WIN_FLAVOR`
Constant WIN_FLAVOR of type int

mpi4py.MPI.WIN_MODEL

`mpi4py.MPI.WIN_MODEL: int = WIN_MODEL`
Constant WIN_MODEL of type int

mpi4py.MPI.SUCCESS

`mpi4py.MPI.SUCCESS: int = SUCCESS`
Constant SUCCESS of type int

mpi4py.MPI.ERR_LASTCODE

`mpi4py.MPI.ERR_LASTCODE: int = ERR_LASTCODE`
Constant ERR_LASTCODE of type `int`

mpi4py.MPI.ERR_TYPE

`mpi4py.MPI.ERR_TYPE: int = ERR_TYPE`
Constant ERR_TYPE of type `int`

mpi4py.MPI.ERR_REQUEST

`mpi4py.MPI.ERR_REQUEST: int = ERR_REQUEST`
Constant ERR_REQUEST of type `int`

mpi4py.MPI.ERR_OP

`mpi4py.MPI.ERR_OP: int = ERR_OP`
Constant ERR_OP of type `int`

mpi4py.MPI.ERR_GROUP

`mpi4py.MPI.ERR_GROUP: int = ERR_GROUP`
Constant ERR_GROUP of type `int`

mpi4py.MPI.ERR_INFO

`mpi4py.MPI.ERR_INFO: int = ERR_INFO`
Constant ERR_INFO of type `int`

mpi4py.MPI.ERR_ERRHANDLER

`mpi4py.MPI.ERR_ERRHANDLER: int = ERR_ERRHANDLER`
Constant ERR_ERRHANDLER of type `int`

mpi4py.MPI.ERR_SESSION

`mpi4py.MPI.ERR_SESSION: int = ERR_SESSION`
Constant ERR_SESSION of type `int`

mpi4py.MPI.ERR_COMM

`mpi4py.MPI.ERR_COMM: int = ERR_COMM`
Constant ERR_COMM of type int

mpi4py.MPI.ERR_WIN

`mpi4py.MPI.ERR_WIN: int = ERR_WIN`
Constant ERR_WIN of type int

mpi4py.MPI.ERR_FILE

`mpi4py.MPI.ERR_FILE: int = ERR_FILE`
Constant ERR_FILE of type int

mpi4py.MPI.ERR_BUFFER

`mpi4py.MPI.ERR_BUFFER: int = ERR_BUFFER`
Constant ERR_BUFFER of type int

mpi4py.MPI.ERR_COUNT

`mpi4py.MPI.ERR_COUNT: int = ERR_COUNT`
Constant ERR_COUNT of type int

mpi4py.MPI.ERR_TAG

`mpi4py.MPI.ERR_TAG: int = ERR_TAG`
Constant ERR_TAG of type int

mpi4py.MPI.ERR_RANK

`mpi4py.MPI.ERR_RANK: int = ERR_RANK`
Constant ERR_RANK of type int

mpi4py.MPI.ERR_ROOT

`mpi4py.MPI.ERR_ROOT: int = ERR_ROOT`
Constant ERR_ROOT of type int

mpi4py.MPI.ERR_TRUNCATE

`mpi4py.MPI.ERR_TRUNCATE: int = ERR_TRUNCATE`
Constant ERR_TRUNCATE of type `int`

mpi4py.MPI.ERR_IN_STATUS

`mpi4py.MPI.ERR_IN_STATUS: int = ERR_IN_STATUS`
Constant ERR_IN_STATUS of type `int`

mpi4py.MPI.ERR_PENDING

`mpi4py.MPI.ERR_PENDING: int = ERR_PENDING`
Constant ERR_PENDING of type `int`

mpi4py.MPI.ERR_TOPOLOGY

`mpi4py.MPI.ERR_TOPOLOGY: int = ERR_TOPOLOGY`
Constant ERR_TOPOLOGY of type `int`

mpi4py.MPI.ERR_DIMS

`mpi4py.MPI.ERR_DIMS: int = ERR_DIMS`
Constant ERR_DIMS of type `int`

mpi4py.MPI.ERR_ARG

`mpi4py.MPI.ERR_ARG: int = ERR_ARG`
Constant ERR_ARG of type `int`

mpi4py.MPI.ERR_OTHER

`mpi4py.MPI.ERR_OTHER: int = ERR_OTHER`
Constant ERR_OTHER of type `int`

mpi4py.MPI.ERR_UNKNOWN

`mpi4py.MPI.ERR_UNKNOWN: int = ERR_UNKNOWN`
Constant ERR_UNKNOWN of type `int`

mpi4py.MPI.ERR_INTERN

`mpi4py.MPI.ERR_INTERN: int = ERR_INTERN`
Constant ERR_INTERN of type int

mpi4py.MPI.ERR_KEYVAL

`mpi4py.MPI.ERR_KEYVAL: int = ERR_KEYVAL`
Constant ERR_KEYVAL of type int

mpi4py.MPI.ERR_NO_MEM

`mpi4py.MPI.ERR_NO_MEM: int = ERR_NO_MEM`
Constant ERR_NO_MEM of type int

mpi4py.MPI.ERR_INFO_KEY

`mpi4py.MPI.ERR_INFO_KEY: int = ERR_INFO_KEY`
Constant ERR_INFO_KEY of type int

mpi4py.MPI.ERR_INFO_VALUE

`mpi4py.MPI.ERR_INFO_VALUE: int = ERR_INFO_VALUE`
Constant ERR_INFO_VALUE of type int

mpi4py.MPI.ERR_INFO_NOKEY

`mpi4py.MPI.ERR_INFO_NOKEY: int = ERR_INFO_NOKEY`
Constant ERR_INFO_NOKEY of type int

mpi4py.MPI.ERR_SPAWN

`mpi4py.MPI.ERR_SPAWN: int = ERR_SPAWN`
Constant ERR_SPAWN of type int

mpi4py.MPI.ERR_PORT

`mpi4py.MPI.ERR_PORT: int = ERR_PORT`
Constant ERR_PORT of type int

mpi4py.MPI.ERR_SERVICE

`mpi4py.MPI.ERR_SERVICE: int = ERR_SERVICE`
Constant ERR_SERVICE of type int

mpi4py.MPI.ERR_NAME

`mpi4py.MPI.ERR_NAME: int = ERR_NAME`
Constant ERR_NAME of type int

mpi4py.MPI.ERR_PROC_ABORTED

`mpi4py.MPI.ERR_PROC_ABORTED: int = ERR_PROC_ABORTED`
Constant ERR_PROC_ABORTED of type int

mpi4py.MPI.ERR_BASE

`mpi4py.MPI.ERR_BASE: int = ERR_BASE`
Constant ERR_BASE of type int

mpi4py.MPI.ERR_SIZE

`mpi4py.MPI.ERR_SIZE: int = ERR_SIZE`
Constant ERR_SIZE of type int

mpi4py.MPI.ERR_DISP

`mpi4py.MPI.ERR_DISP: int = ERR_DISP`
Constant ERR_DISP of type int

mpi4py.MPI.ERR_ASSERT

`mpi4py.MPI.ERR_ASSERT: int = ERR_ASSERT`
Constant ERR_ASSERT of type int

mpi4py.MPI.ERR_LOCKTYPE

`mpi4py.MPI.ERR_LOCKTYPE: int = ERR_LOCKTYPE`
Constant ERR_LOCKTYPE of type int

mpi4py.MPI.ERR_RMA_CONFLICT

`mpi4py.MPI.ERR_RMA_CONFLICT: int = ERR_RMA_CONFLICT`
Constant ERR_RMA_CONFLICT of type int

mpi4py.MPI.ERR_RMA_SYNC

`mpi4py.MPI.ERR_RMA_SYNC: int = ERR_RMA_SYNC`
Constant ERR_RMA_SYNC of type int

mpi4py.MPI.ERR_RMA_RANGE

`mpi4py.MPI.ERR_RMA_RANGE: int = ERR_RMA_RANGE`
Constant ERR_RMA_RANGE of type int

mpi4py.MPI.ERR_RMA_ATTACH

`mpi4py.MPI.ERR_RMA_ATTACH: int = ERR_RMA_ATTACH`
Constant ERR_RMA_ATTACH of type int

mpi4py.MPI.ERR_RMA_SHARED

`mpi4py.MPI.ERR_RMA_SHARED: int = ERR_RMA_SHARED`
Constant ERR_RMA_SHARED of type int

mpi4py.MPI.ERR_RMA_FLAVOR

`mpi4py.MPI.ERR_RMA_FLAVOR: int = ERR_RMA_FLAVOR`
Constant ERR_RMA_FLAVOR of type int

mpi4py.MPI.ERR_BAD_FILE

`mpi4py.MPI.ERR_BAD_FILE: int = ERR_BAD_FILE`
Constant ERR_BAD_FILE of type int

mpi4py.MPI.ERR_NO_SUCH_FILE

`mpi4py.MPI.ERR_NO_SUCH_FILE: int = ERR_NO_SUCH_FILE`
Constant ERR_NO_SUCH_FILE of type int

mpi4py.MPI.ERR_FILE_EXISTS

`mpi4py.MPI.ERR_FILE_EXISTS: int = ERR_FILE_EXISTS`
Constant ERR_FILE_EXISTS of type `int`

mpi4py.MPI.ERR_FILE_IN_USE

`mpi4py.MPI.ERR_FILE_IN_USE: int = ERR_FILE_IN_USE`
Constant ERR_FILE_IN_USE of type `int`

mpi4py.MPI.ERR_AMODE

`mpi4py.MPI.ERR_AMODE: int = ERR_AMODE`
Constant ERR_AMODE of type `int`

mpi4py.MPI.ERR_ACCESS

`mpi4py.MPI.ERR_ACCESS: int = ERR_ACCESS`
Constant ERR_ACCESS of type `int`

mpi4py.MPI.ERR_READ_ONLY

`mpi4py.MPI.ERR_READ_ONLY: int = ERR_READ_ONLY`
Constant ERR_READ_ONLY of type `int`

mpi4py.MPI.ERR_NO_SPACE

`mpi4py.MPI.ERR_NO_SPACE: int = ERR_NO_SPACE`
Constant ERR_NO_SPACE of type `int`

mpi4py.MPI.ERR_QUOTA

`mpi4py.MPI.ERR_QUOTA: int = ERR_QUOTA`
Constant ERR_QUOTA of type `int`

mpi4py.MPI.ERR_NOT_SAME

`mpi4py.MPI.ERR_NOT_SAME: int = ERR_NOT_SAME`
Constant ERR_NOT_SAME of type `int`

mpi4py.MPI.ERR_IO

`mpi4py.MPI.ERR_IO: int = ERR_IO`

Constant ERR_IO of type `int`

mpi4py.MPI.ERR_UNSUPPORTED_OPERATION

`mpi4py.MPI.ERR_UNSUPPORTED_OPERATION: int = ERR_UNSUPPORTED_OPERATION`

Constant ERR_UNSUPPORTED_OPERATION of type `int`

mpi4py.MPI.ERR_UNSUPPORTED_DATAREP

`mpi4py.MPI.ERR_UNSUPPORTED_DATAREP: int = ERR_UNSUPPORTED_DATAREP`

Constant ERR_UNSUPPORTED_DATAREP of type `int`

mpi4py.MPI.ERR_CONVERSION

`mpi4py.MPI.ERR_CONVERSION: int = ERR_CONVERSION`

Constant ERR_CONVERSION of type `int`

mpi4py.MPI.ERR_DUP_DATAREP

`mpi4py.MPI.ERR_DUP_DATAREP: int = ERR_DUP_DATAREP`

Constant ERR_DUP_DATAREP of type `int`

mpi4py.MPI.ERR_VALUE_TOO_LARGE

`mpi4py.MPI.ERR_VALUE_TOO_LARGE: int = ERR_VALUE_TOO_LARGE`

Constant ERR_VALUE_TOO_LARGE of type `int`

mpi4py.MPI.ERR_REVOKED

`mpi4py.MPI.ERR_REVOKED: int = ERR_REVOKED`

Constant ERR_REVOKED of type `int`

mpi4py.MPI.ERR_PROC_FAILED

`mpi4py.MPI.ERR_PROC_FAILED: int = ERR_PROC_FAILED`

Constant ERR_PROC_FAILED of type `int`

mpi4py.MPI.ERR_PROC_FAILED_PENDING

```
mpi4py.MPI.ERR_PROC_FAILED_PENDING: int = ERR_PROC_FAILED_PENDING
Constant ERR_PROC_FAILED_PENDING of type int
```

mpi4py.MPI.ORDER_C

```
mpi4py.MPI.ORDER_C: int = ORDER_C
Constant ORDER_C of type int
```

mpi4py.MPI.ORDER_FORTRAN

```
mpi4py.MPI.ORDER_FORTRAN: int = ORDER_FORTRAN
Constant ORDER_FORTRAN of type int
```

mpi4py.MPI.ORDER_F

```
mpi4py.MPI.ORDER_F: int = ORDER_F
Constant ORDER_F of type int
```

mpi4py.MPI.TYPECLASS_INTEGER

```
mpi4py.MPI.TYPECLASS_INTEGER: int = TYPECLASS_INTEGER
Constant TYPECLASS_INTEGER of type int
```

mpi4py.MPI.TYPECLASS_REAL

```
mpi4py.MPI.TYPECLASS_REAL: int = TYPECLASS_REAL
Constant TYPECLASS_REAL of type int
```

mpi4py.MPI.TYPECLASS_COMPLEX

```
mpi4py.MPI.TYPECLASS_COMPLEX: int = TYPECLASS_COMPLEX
Constant TYPECLASS_COMPLEX of type int
```

mpi4py.MPI.DISTRIBUTE_NONE

```
mpi4py.MPI.DISTRIBUTE_NONE: int = DISTRIBUTE_NONE
Constant DISTRIBUTE_NONE of type int
```

mpi4py.MPI.DISTRIBUTE_BLOCK

`mpi4py.MPI.DISTRIBUTE_BLOCK: int = DISTRIBUTE_BLOCK`
Constant DISTRIBUTE_BLOCK of type int

mpi4py.MPI.DISTRIBUTE_CYCLIC

`mpi4py.MPI.DISTRIBUTE_CYCLIC: int = DISTRIBUTE_CYCLIC`
Constant DISTRIBUTE_CYCLIC of type int

mpi4py.MPI.DISTRIBUTE_DFLT_DARG

`mpi4py.MPI.DISTRIBUTE_DFLT_DARG: int = DISTRIBUTE_DFLT_DARG`
Constant DISTRIBUTE_DFLT_DARG of type int

mpi4py.MPI.COMBINER_NAMED

`mpi4py.MPI.COMBINER_NAMED: int = COMBINER_NAMED`
Constant COMBINER_NAMED of type int

mpi4py.MPI.COMBINER_DUP

`mpi4py.MPI.COMBINER_DUP: int = COMBINER_DUP`
Constant COMBINER_DUP of type int

mpi4py.MPI.COMBINER_CONTIGUOUS

`mpi4py.MPI.COMBINER_CONTIGUOUS: int = COMBINER_CONTIGUOUS`
Constant COMBINER_CONTIGUOUS of type int

mpi4py.MPI.COMBINER_VECTOR

`mpi4py.MPI.COMBINER_VECTOR: int = COMBINER_VECTOR`
Constant COMBINER_VECTOR of type int

mpi4py.MPI.COMBINER_HVECTOR

`mpi4py.MPI.COMBINER_HVECTOR: int = COMBINER_HVECTOR`
Constant COMBINER_HVECTOR of type int

mpi4py.MPI.COMBINER_INDEXED

`mpi4py.MPI.COMBINER_INDEXED: int = COMBINER_INDEXED`

Constant COMBINER_INDEXED of type int

mpi4py.MPI.COMBINER_HINDEXED

`mpi4py.MPI.COMBINER_HINDEXED: int = COMBINER_HINDEXED`

Constant COMBINER_HINDEXED of type int

mpi4py.MPI.COMBINER_INDEXED_BLOCK

`mpi4py.MPI.COMBINER_INDEXED_BLOCK: int = COMBINER_INDEXED_BLOCK`

Constant COMBINER_INDEXED_BLOCK of type int

mpi4py.MPI.COMBINER_HINDEXED_BLOCK

`mpi4py.MPI.COMBINER_HINDEXED_BLOCK: int = COMBINER_HINDEXED_BLOCK`

Constant COMBINER_HINDEXED_BLOCK of type int

mpi4py.MPI.COMBINER_STRUCT

`mpi4py.MPI.COMBINER_STRUCT: int = COMBINER_STRUCT`

Constant COMBINER_STRUCT of type int

mpi4py.MPI.COMBINER_SUBARRAY

`mpi4py.MPI.COMBINER_SUBARRAY: int = COMBINER_SUBARRAY`

Constant COMBINER_SUBARRAY of type int

mpi4py.MPI.COMBINER_DARRAY

`mpi4py.MPI.COMBINER_DARRAY: int = COMBINER_DARRAY`

Constant COMBINER_DARRAY of type int

mpi4py.MPI.COMBINER_RESIZED

`mpi4py.MPI.COMBINER_RESIZED: int = COMBINER_RESIZED`

Constant COMBINER_RESIZED of type int

mpi4py.MPI.COMBINER_VALUE_INDEX

`mpi4py.MPI.COMBINER_VALUE_INDEX: int = COMBINER_VALUE_INDEX`
Constant COMBINER_VALUE_INDEX of type `int`

mpi4py.MPI.COMBINER_F90_INTEGER

`mpi4py.MPI.COMBINER_F90_INTEGER: int = COMBINER_F90_INTEGER`
Constant COMBINER_F90_INTEGER of type `int`

mpi4py.MPI.COMBINER_F90_REAL

`mpi4py.MPI.COMBINER_F90_REAL: int = COMBINER_F90_REAL`
Constant COMBINER_F90_REAL of type `int`

mpi4py.MPI.COMBINER_F90_COMPLEX

`mpi4py.MPI.COMBINER_F90_COMPLEX: int = COMBINER_F90_COMPLEX`
Constant COMBINER_F90_COMPLEX of type `int`

mpi4py.MPI.F_SOURCE

`mpi4py.MPI.F_SOURCE: int = F_SOURCE`
Constant F_SOURCE of type `int`

mpi4py.MPI.F_TAG

`mpi4py.MPI.F_TAG: int = F_TAG`
Constant F_TAG of type `int`

mpi4py.MPI.F_ERROR

`mpi4py.MPI.F_ERROR: int = F_ERROR`
Constant F_ERROR of type `int`

mpi4py.MPI.F_STATUS_SIZE

`mpi4py.MPI.F_STATUS_SIZE: int = F_STATUS_SIZE`
Constant F_STATUS_SIZE of type `int`

mpi4py.MPI.IDENT

`mpi4py.MPI.IDENT: int = IDENT`

Constant IDENT of type `int`

mpi4py.MPI.CONGRUENT

`mpi4py.MPI.CONGRUENT: int = CONGRUENT`

Constant CONGRUENT of type `int`

mpi4py.MPI.SIMILAR

`mpi4py.MPI.SIMILAR: int = SIMILAR`

Constant SIMILAR of type `int`

mpi4py.MPI.UNEQUAL

`mpi4py.MPI.UNEQUAL: int = UNEQUAL`

Constant UNEQUAL of type `int`

mpi4py.MPI.CART

`mpi4py.MPI.CART: int = CART`

Constant CART of type `int`

mpi4py.MPI.GRAPH

`mpi4py.MPI.GRAPH: int = GRAPH`

Constant GRAPH of type `int`

mpi4py.MPI.DIST_GRAPH

`mpi4py.MPI.DIST_GRAPH: int = DIST_GRAPH`

Constant DIST_GRAPH of type `int`

mpi4py.MPI.UNWEIGHTED

`mpi4py.MPI.UNWEIGHTED: int = UNWEIGHTED`

Constant UNWEIGHTED of type `int`

mpi4py.MPI.WEIGHTS_EMPTY

`mpi4py.MPI.WEIGHTS_EMPTY: int = WEIGHTS_EMPTY`
Constant WEIGHTS_EMPTY of type `int`

mpi4py.MPI.COMM_TYPE_SHARED

`mpi4py.MPI.COMM_TYPE_SHARED: int = COMM_TYPE_SHARED`
Constant COMM_TYPE_SHARED of type `int`

mpi4py.MPI.COMM_TYPE_HW_GUIDED

`mpi4py.MPI.COMM_TYPE_HW_GUIDED: int = COMM_TYPE_HW_GUIDED`
Constant COMM_TYPE_HW_GUIDED of type `int`

mpi4py.MPI.COMM_TYPE_HW_UNGUIDED

`mpi4py.MPI.COMM_TYPE_HW_UNGUIDED: int = COMM_TYPE_HW_UNGUIDED`
Constant COMM_TYPE_HW_UNGUIDED of type `int`

mpi4py.MPI.COMM_TYPE_RESOURCE_GUIDED

`mpi4py.MPI.COMM_TYPE_RESOURCE_GUIDED: int = COMM_TYPE_RESOURCE_GUIDED`
Constant COMM_TYPE_RESOURCE_GUIDED of type `int`

mpi4py.MPI.BSEND_OVERHEAD

`mpi4py.MPI.BSEND_OVERHEAD: int = BSEND_OVERHEAD`
Constant BSEND_OVERHEAD of type `int`

mpi4py.MPI.BUFFER_AUTOMATIC

`mpi4py.MPI.BUFFER_AUTOMATIC: BufferAutomaticType = BUFFER_AUTOMATIC`
Constant BUFFER_AUTOMATIC of type `BufferAutomaticType`

mpi4py.MPI.WIN_FLAVOR_CREATE

`mpi4py.MPI.WIN_FLAVOR_CREATE: int = WIN_FLAVOR_CREATE`
Constant WIN_FLAVOR_CREATE of type `int`

mpi4py.MPI.WIN_FLAVOR_ALLOCATE

`mpi4py.MPI.WIN_FLAVOR_ALLOCATE: int = WIN_FLAVOR_ALLOCATE`
Constant WIN_FLAVOR_ALLOCATE of type int

mpi4py.MPI.WIN_FLAVOR_DYNAMIC

`mpi4py.MPI.WIN_FLAVOR_DYNAMIC: int = WIN_FLAVOR_DYNAMIC`
Constant WIN_FLAVOR_DYNAMIC of type int

mpi4py.MPI.WIN_FLAVOR_SHARED

`mpi4py.MPI.WIN_FLAVOR_SHARED: int = WIN_FLAVOR_SHARED`
Constant WIN_FLAVOR_SHARED of type int

mpi4py.MPI.WIN_SEPARATE

`mpi4py.MPI.WIN_SEPARATE: int = WIN_SEPARATE`
Constant WIN_SEPARATE of type int

mpi4py.MPI.WIN_UNIFIED

`mpi4py.MPI.WIN_UNIFIED: int = WIN_UNIFIED`
Constant WIN_UNIFIED of type int

mpi4py.MPI.MODE_NOCHECK

`mpi4py.MPI.MODE_NOCHECK: int = MODE_NOCHECK`
Constant MODE_NOCHECK of type int

mpi4py.MPI.MODE_NOSTORE

`mpi4py.MPI.MODE_NOSTORE: int = MODE_NOSTORE`
Constant MODE_NOSTORE of type int

mpi4py.MPI.MODE_NOPUT

`mpi4py.MPI.MODE_NOPUT: int = MODE_NOPUT`
Constant MODE_NOPUT of type int

mpi4py.MPI.MODE_NOPRECEDE

`mpi4py.MPI.MODE_NOPRECEDE: int = MODE_NOPRECEDE`
Constant MODE_NOPRECEDE of type `int`

mpi4py.MPI.MODE_NOSUCCEED

`mpi4py.MPI.MODE_NOSUCCEED: int = MODE_NOSUCCEED`
Constant MODE_NOSUCCEED of type `int`

mpi4py.MPI.LOCK_EXCLUSIVE

`mpi4py.MPI.LOCK_EXCLUSIVE: int = LOCK_EXCLUSIVE`
Constant LOCK_EXCLUSIVE of type `int`

mpi4py.MPI.LOCK_SHARED

`mpi4py.MPI.LOCK_SHARED: int = LOCK_SHARED`
Constant LOCK_SHARED of type `int`

mpi4py.MPI.MODE_RDONLY

`mpi4py.MPI.MODE_RDONLY: int = MODE_RDONLY`
Constant MODE_RDONLY of type `int`

mpi4py.MPI.MODE_WRONLY

`mpi4py.MPI.MODE_WRONLY: int = MODE_WRONLY`
Constant MODE_WRONLY of type `int`

mpi4py.MPI.MODE_RDWR

`mpi4py.MPI.MODE_RDWR: int = MODE_RDWR`
Constant MODE_RDWR of type `int`

mpi4py.MPI.MODE_CREATE

`mpi4py.MPI.MODE_CREATE: int = MODE_CREATE`
Constant MODE_CREATE of type `int`

mpi4py.MPI.MODE_EXCL

`mpi4py.MPI.MODE_EXCL: int = MODE_EXCL`

Constant MODE_EXCL of type int

mpi4py.MPI.MODE_DELETE_ON_CLOSE

`mpi4py.MPI.MODE_DELETE_ON_CLOSE: int = MODE_DELETE_ON_CLOSE`

Constant MODE_DELETE_ON_CLOSE of type int

mpi4py.MPI.MODE_UNIQUE_OPEN

`mpi4py.MPI.MODE_UNIQUE_OPEN: int = MODE_UNIQUE_OPEN`

Constant MODE_UNIQUE_OPEN of type int

mpi4py.MPI.MODE_SEQUENTIAL

`mpi4py.MPI.MODE_SEQUENTIAL: int = MODE_SEQUENTIAL`

Constant MODE_SEQUENTIAL of type int

mpi4py.MPI.MODE_APPEND

`mpi4py.MPI.MODE_APPEND: int = MODE_APPEND`

Constant MODE_APPEND of type int

mpi4py.MPISEEK_SET

`mpi4py.MPISEEK_SET: int = SEEK_SET`

Constant SEEK_SET of type int

mpi4py.MPISEEK_CUR

`mpi4py.MPISEEK_CUR: int = SEEK_CUR`

Constant SEEK_CUR of type int

mpi4py.MPISEEK_END

`mpi4py.MPISEEK_END: int = SEEK_END`

Constant SEEK_END of type int

mpi4py.MPI.DISPLACEMENT_CURRENT

`mpi4py.MPI.DISPLACEMENT_CURRENT: int = DISPLACEMENT_CURRENT`
Constant DISPLACEMENT_CURRENT of type `int`

mpi4py.MPI.DISP_CUR

`mpi4py.MPI.DISP_CUR: int = DISP_CUR`
Constant DISP_CUR of type `int`

mpi4py.MPI.THREAD_SINGLE

`mpi4py.MPI.THREAD_SINGLE: int = THREAD_SINGLE`
Constant THREAD_SINGLE of type `int`

mpi4py.MPI.THREAD_FUNNELED

`mpi4py.MPI.THREAD_FUNNELED: int = THREAD_FUNNELED`
Constant THREAD_FUNNELED of type `int`

mpi4py.MPI.THREAD_SERIALIZED

`mpi4py.MPI.THREAD_SERIALIZED: int = THREAD_SERIALIZED`
Constant THREAD_SERIALIZED of type `int`

mpi4py.MPI.THREAD_MULTIPLE

`mpi4py.MPI.THREAD_MULTIPLE: int = THREAD_MULTIPLE`
Constant THREAD_MULTIPLE of type `int`

mpi4py.MPI.VERSION

`mpi4py.MPI.VERSION: int = VERSION`
Constant VERSION of type `int`

mpi4py.MPI.SUBVERSION

`mpi4py.MPI.SUBVERSION: int = SUBVERSION`
Constant SUBVERSION of type `int`

mpi4py.MPI.MAX_PROCESSOR_NAME

`mpi4py.MPI.MAX_PROCESSOR_NAME: int = MAX_PROCESSOR_NAME`
Constant MAX_PROCESSOR_NAME of type int

mpi4py.MPI.MAX_ERROR_STRING

`mpi4py.MPI.MAX_ERROR_STRING: int = MAX_ERROR_STRING`
Constant MAX_ERROR_STRING of type int

mpi4py.MPI.MAX_PORT_NAME

`mpi4py.MPI.MAX_PORT_NAME: int = MAX_PORT_NAME`
Constant MAX_PORT_NAME of type int

mpi4py.MPI.MAX_INFO_KEY

`mpi4py.MPI.MAX_INFO_KEY: int = MAX_INFO_KEY`
Constant MAX_INFO_KEY of type int

mpi4py.MPI.MAX_INFO_VAL

`mpi4py.MPI.MAX_INFO_VAL: int = MAX_INFO_VAL`
Constant MAX_INFO_VAL of type int

mpi4py.MPI.MAX_OBJECT_NAME

`mpi4py.MPI.MAX_OBJECT_NAME: int = MAX_OBJECT_NAME`
Constant MAX_OBJECT_NAME of type int

mpi4py.MPI.MAX_DATAREP_STRING

`mpi4py.MPI.MAX_DATAREP_STRING: int = MAX_DATAREP_STRING`
Constant MAX_DATAREP_STRING of type int

mpi4py.MPI.MAX_LIBRARY_VERSION_STRING

`mpi4py.MPI.MAX_LIBRARY_VERSION_STRING: int = MAX_LIBRARY_VERSION_STRING`
Constant MAX_LIBRARY_VERSION_STRING of type int

mpi4py.MPI.MAX_PSET_NAME_LEN

`mpi4py.MPI.MAX_PSET_NAME_LEN: int = MAX_PSET_NAME_LEN`
Constant MAX_PSET_NAME_LEN of type int

mpi4py.MPI.MAX_STRINGTAG_LEN

`mpi4py.MPI.MAX_STRINGTAG_LEN: int = MAX_STRINGTAG_LEN`
Constant MAX_STRINGTAG_LEN of type int

mpi4py.MPI.DATATYPE_NULL

`mpi4py.MPI.DATATYPE_NULL: Datatype = DATATYPE_NULL`
Object DATATYPE_NULL of type *Datatype*

mpi4py.MPI.PACKED

`mpi4py.MPI.PACKED: Datatype = PACKED`
Object PACKED of type *Datatype*

mpi4py.MPI.BYTE

`mpi4py.MPI.BYTE: Datatype = BYTE`
Object BYTE of type *Datatype*

mpi4py.MPI.AINT

`mpi4py.MPI.AINT: Datatype = AINT`
Object AINT of type *Datatype*

mpi4py.MPI.OFFSET

`mpi4py.MPI.OFFSET: Datatype = OFFSET`
Object OFFSET of type *Datatype*

mpi4py.MPI.COUNT

`mpi4py.MPI.COUNT: Datatype = COUNT`
Object COUNT of type *Datatype*

mpi4py.MPI.CHAR

mpi4py.MPI.CHAR: *Datatype* = CHAR

Object CHAR of type *Datatype*

mpi4py.MPI.WCHAR

mpi4py.MPI.WCHAR: *Datatype* = WCHAR

Object WCHAR of type *Datatype*

mpi4py.MPI.SIGNED_CHAR

mpi4py.MPI.SIGNED_CHAR: *Datatype* = SIGNED_CHAR

Object SIGNED_CHAR of type *Datatype*

mpi4py.MPI.SHORT

mpi4py.MPI.SHORT: *Datatype* = SHORT

Object SHORT of type *Datatype*

mpi4py.MPI.INT

mpi4py.MPI.INT: *Datatype* = INT

Object INT of type *Datatype*

mpi4py.MPI.LONG

mpi4py.MPI.LONG: *Datatype* = LONG

Object LONG of type *Datatype*

mpi4py.MPI.LONG_LONG

mpi4py.MPI.LONG_LONG: *Datatype* = LONG_LONG

Object LONG_LONG of type *Datatype*

mpi4py.MPI.UNSIGNED_CHAR

mpi4py.MPI.UNSIGNED_CHAR: *Datatype* = UNSIGNED_CHAR

Object UNSIGNED_CHAR of type *Datatype*

mpi4py.MPI.UNSIGNED_SHORT

`mpi4py.MPI.UNSIGNED_SHORT: Datatype = UNSIGNED_SHORT`
Object UNSIGNED_SHORT of type *Datatype*

mpi4py.MPI.UNSIGNED

`mpi4py.MPI.UNSIGNED: Datatype = UNSIGNED`
Object UNSIGNED of type *Datatype*

mpi4py.MPI.UNSIGNED_LONG

`mpi4py.MPI.UNSIGNED_LONG: Datatype = UNSIGNED_LONG`
Object UNSIGNED_LONG of type *Datatype*

mpi4py.MPI.UNSIGNED_LONG_LONG

`mpi4py.MPI.UNSIGNED_LONG_LONG: Datatype = UNSIGNED_LONG_LONG`
Object UNSIGNED_LONG_LONG of type *Datatype*

mpi4py.MPI.FLOAT

`mpi4py.MPI.FLOAT: Datatype = FLOAT`
Object FLOAT of type *Datatype*

mpi4py.MPI.DOUBLE

`mpi4py.MPI.DOUBLE: Datatype = DOUBLE`
Object DOUBLE of type *Datatype*

mpi4py.MPI.LONG_DOUBLE

`mpi4py.MPI.LONG_DOUBLE: Datatype = LONG_DOUBLE`
Object LONG_DOUBLE of type *Datatype*

mpi4py.MPI.C_BOOL

`mpi4py.MPI.C_BOOL: Datatype = C_BOOL`
Object C_BOOL of type *Datatype*

mpi4py.MPI.INT8_T

`mpi4py.MPI.INT8_T: Datatype = INT8_T`
Object INT8_T of type *Datatype*

mpi4py.MPI.INT16_T

`mpi4py.MPI.INT16_T: Datatype = INT16_T`
Object INT16_T of type *Datatype*

mpi4py.MPI.INT32_T

`mpi4py.MPI.INT32_T: Datatype = INT32_T`
Object INT32_T of type *Datatype*

mpi4py.MPI.INT64_T

`mpi4py.MPI.INT64_T: Datatype = INT64_T`
Object INT64_T of type *Datatype*

mpi4py.MPI.UINT8_T

`mpi4py.MPI.UINT8_T: Datatype = UINT8_T`
Object UINT8_T of type *Datatype*

mpi4py.MPI.UINT16_T

`mpi4py.MPI.UINT16_T: Datatype = UINT16_T`
Object UINT16_T of type *Datatype*

mpi4py.MPI.UINT32_T

`mpi4py.MPI.UINT32_T: Datatype = UINT32_T`
Object UINT32_T of type *Datatype*

mpi4py.MPI.UINT64_T

`mpi4py.MPI.UINT64_T: Datatype = UINT64_T`
Object UINT64_T of type *Datatype*

`mpi4py.MPI.C_COMPLEX`

`mpi4py.MPI.C_COMPLEX: Datatype = C_COMPLEX`
Object C_COMPLEX of type *Datatype*

`mpi4py.MPI.C_FLOAT_COMPLEX`

`mpi4py.MPI.C_FLOAT_COMPLEX: Datatype = C_FLOAT_COMPLEX`
Object C_FLOAT_COMPLEX of type *Datatype*

`mpi4py.MPI.C_DOUBLE_COMPLEX`

`mpi4py.MPI.C_DOUBLE_COMPLEX: Datatype = C_DOUBLE_COMPLEX`
Object C_DOUBLE_COMPLEX of type *Datatype*

`mpi4py.MPI.C_LONG_DOUBLE_COMPLEX`

`mpi4py.MPI.C_LONG_DOUBLE_COMPLEX: Datatype = C_LONG_DOUBLE_COMPLEX`
Object C_LONG_DOUBLE_COMPLEX of type *Datatype*

`mpi4py.MPI.CXX_BOOL`

`mpi4py.MPI.CXX_BOOL: Datatype = CXX_BOOL`
Object CXX_BOOL of type *Datatype*

`mpi4py.MPI.CXX_FLOAT_COMPLEX`

`mpi4py.MPI.CXX_FLOAT_COMPLEX: Datatype = CXX_FLOAT_COMPLEX`
Object CXX_FLOAT_COMPLEX of type *Datatype*

`mpi4py.MPI.CXX_DOUBLE_COMPLEX`

`mpi4py.MPI.CXX_DOUBLE_COMPLEX: Datatype = CXX_DOUBLE_COMPLEX`
Object CXX_DOUBLE_COMPLEX of type *Datatype*

`mpi4py.MPI.CXX_LONG_DOUBLE_COMPLEX`

`mpi4py.MPI.CXX_LONG_DOUBLE_COMPLEX: Datatype = CXX_LONG_DOUBLE_COMPLEX`
Object CXX_LONG_DOUBLE_COMPLEX of type *Datatype*

mpi4py.MPI.SHORT_INT

`mpi4py.MPI.SHORT_INT: Datatype = SHORT_INT`
Object SHORT_INT of type *Datatype*

mpi4py.MPI.INT_INT

`mpi4py.MPI.INT_INT: Datatype = INT_INT`
Object INT_INT of type *Datatype*

mpi4py.MPI.TWOINT

`mpi4py.MPI.TWOINT: Datatype = TWOINT`
Object TWOINT of type *Datatype*

mpi4py.MPI.LONG_INT

`mpi4py.MPI.LONG_INT: Datatype = LONG_INT`
Object LONG_INT of type *Datatype*

mpi4py.MPI.FLOAT_INT

`mpi4py.MPI.FLOAT_INT: Datatype = FLOAT_INT`
Object FLOAT_INT of type *Datatype*

mpi4py.MPI.DOUBLE_INT

`mpi4py.MPI.DOUBLE_INT: Datatype = DOUBLE_INT`
Object DOUBLE_INT of type *Datatype*

mpi4py.MPI.LONG_DOUBLE_INT

`mpi4py.MPI.LONG_DOUBLE_INT: Datatype = LONG_DOUBLE_INT`
Object LONG_DOUBLE_INT of type *Datatype*

mpi4py.MPI.CHARACTER

`mpi4py.MPI.CHARACTER: Datatype = CHARACTER`
Object CHARACTER of type *Datatype*

mpi4py.MPI.LOGICAL

`mpi4py.MPI.LOGICAL: Datatype = LOGICAL`

Object LOGICAL of type *Datatype*

mpi4py.MPI.INTEGER

`mpi4py.MPI.INTEGER: Datatype = INTEGER`

Object INTEGER of type *Datatype*

mpi4py.MPI.REAL

`mpi4py.MPI.REAL: Datatype = REAL`

Object REAL of type *Datatype*

mpi4py.MPI.DOUBLE_PRECISION

`mpi4py.MPI.DOUBLE_PRECISION: Datatype = DOUBLE_PRECISION`

Object DOUBLE_PRECISION of type *Datatype*

mpi4py.MPI.COMPLEX

`mpi4py.MPI.COMPLEX: Datatype = COMPLEX`

Object COMPLEX of type *Datatype*

mpi4py.MPI.DOUBLE_COMPLEX

`mpi4py.MPI.DOUBLE_COMPLEX: Datatype = DOUBLE_COMPLEX`

Object DOUBLE_COMPLEX of type *Datatype*

mpi4py.MPI.LOGICAL1

`mpi4py.MPI.LOGICAL1: Datatype = LOGICAL1`

Object LOGICAL1 of type *Datatype*

mpi4py.MPI.LOGICAL2

`mpi4py.MPI.LOGICAL2: Datatype = LOGICAL2`

Object LOGICAL2 of type *Datatype*

mpi4py.MPI.LOGICAL4

mpi4py.MPI.LOGICAL4: *Datatype* = LOGICAL4
Object LOGICAL4 of type *Datatype*

mpi4py.MPI.LOGICAL8

mpi4py.MPI.LOGICAL8: *Datatype* = LOGICAL8
Object LOGICAL8 of type *Datatype*

mpi4py.MPI.INTEGER1

mpi4py.MPI.INTEGER1: *Datatype* = INTEGER1
Object INTEGER1 of type *Datatype*

mpi4py.MPI.INTEGER2

mpi4py.MPI.INTEGER2: *Datatype* = INTEGER2
Object INTEGER2 of type *Datatype*

mpi4py.MPI.INTEGER4

mpi4py.MPI.INTEGER4: *Datatype* = INTEGER4
Object INTEGER4 of type *Datatype*

mpi4py.MPI.INTEGER8

mpi4py.MPI.INTEGER8: *Datatype* = INTEGER8
Object INTEGER8 of type *Datatype*

mpi4py.MPI.INTEGER16

mpi4py.MPI.INTEGER16: *Datatype* = INTEGER16
Object INTEGER16 of type *Datatype*

mpi4py.MPI.REAL2

mpi4py.MPI.REAL2: *Datatype* = REAL2
Object REAL2 of type *Datatype*

mpi4py.MPI.REAL4

```
mpi4py.MPI.REAL4:  Datatype = REAL4
Object REAL4 of type Datatype
```

mpi4py.MPI.REAL8

```
mpi4py.MPI.REAL8:  Datatype = REAL8
Object REAL8 of type Datatype
```

mpi4py.MPI.REAL16

```
mpi4py.MPI.REAL16:  Datatype = REAL16
Object REAL16 of type Datatype
```

mpi4py.MPI.COMPLEX4

```
mpi4py.MPI.COMPLEX4:  Datatype = COMPLEX4
Object COMPLEX4 of type Datatype
```

mpi4py.MPI.COMPLEX8

```
mpi4py.MPI.COMPLEX8:  Datatype = COMPLEX8
Object COMPLEX8 of type Datatype
```

mpi4py.MPI.COMPLEX16

```
mpi4py.MPI.COMPLEX16:  Datatype = COMPLEX16
Object COMPLEX16 of type Datatype
```

mpi4py.MPI.COMPLEX32

```
mpi4py.MPI.COMPLEX32:  Datatype = COMPLEX32
Object COMPLEX32 of type Datatype
```

mpi4py.MPI.UNSIGNED_INT

```
mpi4py.MPI.UNSIGNED_INT:  Datatype = UNSIGNED_INT
Object UNSIGNED_INT of type Datatype
```

mpi4py.MPI.SIGNED_SHORT

`mpi4py.MPI.SIGNED_SHORT: Datatype = SIGNED_SHORT`

Object SIGNED_SHORT of type *Datatype*

mpi4py.MPI.SIGNED_INT

`mpi4py.MPI.SIGNED_INT: Datatype = SIGNED_INT`

Object SIGNED_INT of type *Datatype*

mpi4py.MPI.SIGNED_LONG

`mpi4py.MPI.SIGNED_LONG: Datatype = SIGNED_LONG`

Object SIGNED_LONG of type *Datatype*

mpi4py.MPI.SIGNED_LONG_LONG

`mpi4py.MPI.SIGNED_LONG_LONG: Datatype = SIGNED_LONG_LONG`

Object SIGNED_LONG_LONG of type *Datatype*

mpi4py.MPI.BOOL

`mpi4py.MPI.BOOL: Datatype = BOOL`

Object BOOL of type *Datatype*

mpi4py.MPI.SINT8_T

`mpi4py.MPI.SINT8_T: Datatype = SINT8_T`

Object SINT8_T of type *Datatype*

mpi4py.MPI.SINT16_T

`mpi4py.MPI.SINT16_T: Datatype = SINT16_T`

Object SINT16_T of type *Datatype*

mpi4py.MPI.SINT32_T

`mpi4py.MPI.SINT32_T: Datatype = SINT32_T`

Object SINT32_T of type *Datatype*

mpi4py.MPI.SINT64_T

`mpi4py.MPI.SINT64_T: Datatype = SINT64_T`
Object SINT64_T of type *Datatype*

mpi4py.MPI.F_BOOL

`mpi4py.MPI.F_BOOL: Datatype = F_BOOL`
Object F_BOOL of type *Datatype*

mpi4py.MPI.F_INT

`mpi4py.MPI.F_INT: Datatype = F_INT`
Object F_INT of type *Datatype*

mpi4py.MPI.F_FLOAT

`mpi4py.MPI.F_FLOAT: Datatype = F_FLOAT`
Object F_FLOAT of type *Datatype*

mpi4py.MPI.F_DOUBLE

`mpi4py.MPI.F_DOUBLE: Datatype = F_DOUBLE`
Object F_DOUBLE of type *Datatype*

mpi4py.MPI.F_COMPLEX

`mpi4py.MPI.F_COMPLEX: Datatype = F_COMPLEX`
Object F_COMPLEX of type *Datatype*

mpi4py.MPI.F_FLOAT_COMPLEX

`mpi4py.MPI.F_FLOAT_COMPLEX: Datatype = F_FLOAT_COMPLEX`
Object F_FLOAT_COMPLEX of type *Datatype*

mpi4py.MPI.F_DOUBLE_COMPLEX

`mpi4py.MPI.F_DOUBLE_COMPLEX: Datatype = F_DOUBLE_COMPLEX`
Object F_DOUBLE_COMPLEX of type *Datatype*

mpi4py.MPI.REQUEST_NULL

mpi4py.MPI.REQUEST_NULL: *Request* = REQUEST_NULL

Object REQUEST_NULL of type *Request*

mpi4py.MPI.MESSAGE_NULL

mpi4py.MPI.MESSAGE_NULL: *Message* = MESSAGE_NULL

Object MESSAGE_NULL of type *Message*

mpi4py.MPI.MESSAGE_NO_PROC

mpi4py.MPI.MESSAGE_NO_PROC: *Message* = MESSAGE_NO_PROC

Object MESSAGE_NO_PROC of type *Message*

mpi4py.MPI.OP_NULL

mpi4py.MPI.OP_NULL: *Op* = OP_NULL

Object OP_NULL of type *Op*

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type

Any

mpi4py.MPI.MAX

mpi4py.MPI.MAX: *Op* = MAX

Object MAX of type *Op*

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type

Any

mpi4py.MPI.MIN

mpi4py.MPI.MIN: *Op* = MIN

Object MIN of type *Op*

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type

Any

mpi4py.MPI.SUM`mpi4py.MPI.SUM: Op = SUM`Object SUM of type *Op***Parameters**

- **x** (Any) –
- **y** (Any) –

Return type

Any

mpi4py.MPI.PROD`mpi4py.MPI.PROD: Op = PROD`Object PROD of type *Op***Parameters**

- **x** (Any) –
- **y** (Any) –

Return type

Any

mpi4py.MPI.LAND`mpi4py.MPI.LAND: Op = LAND`Object LAND of type *Op***Parameters**

- **x** (Any) –
- **y** (Any) –

Return type

Any

mpi4py.MPI.BAND`mpi4py.MPI.BAND: Op = BAND`Object BAND of type *Op***Parameters**

- **x** (Any) –
- **y** (Any) –

Return type

Any

mpi4py.MPI.LOR

mpi4py.MPI.LOR: *Op* = LOR

Object LOR of type *Op*

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type

Any

mpi4py.MPI.BOR

mpi4py.MPI.BOR: *Op* = BOR

Object BOR of type *Op*

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type

Any

mpi4py.MPI.LXOR

mpi4py.MPI.LXOR: *Op* = LXOR

Object LXOR of type *Op*

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type

Any

mpi4py.MPI.BXOR

mpi4py.MPI.BXOR: *Op* = BXOR

Object BXOR of type *Op*

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type

Any

mpi4py.MPI.MAXLOC

mpi4py.MPI.MAXLOC: *Op* = MAXLOC

Object MAXLOC of type *Op*

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type

Any

mpi4py.MPI.MINLOC

mpi4py.MPI.MINLOC: *Op* = MINLOC

Object MINLOC of type *Op*

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type

Any

mpi4py.MPI.REPLACE

mpi4py.MPI.REPLACE: *Op* = REPLACE

Object REPLACE of type *Op*

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type

Any

mpi4py.MPI.NO_OP

mpi4py.MPI.NO_OP: *Op* = NO_OP

Object NO_OP of type *Op*

Parameters

- **x** (*Any*) –
- **y** (*Any*) –

Return type

Any

mpi4py.MPI.GROUP_NULL

`mpi4py.MPI.GROUP_NULL: Group = GROUP_NULL`
Object GROUP_NULL of type *Group*

mpi4py.MPI.GROUP_EMPTY

`mpi4py.MPI.GROUP_EMPTY: Group = GROUP_EMPTY`
Object GROUP_EMPTY of type *Group*

mpi4py.MPI.INFO_NULL

`mpi4py.MPI.INFO_NULL: Info = INFO_NULL`
Object INFO_NULL of type *Info*

mpi4py.MPI.INFO_ENV

`mpi4py.MPI.INFO_ENV: Info = INFO_ENV`
Object INFO_ENV of type *Info*

mpi4py.MPI.ERRHANDLER_NULL

`mpi4py.MPI.ERRHANDLER_NULL: Errhandler = ERRHANDLER_NULL`
Object ERRHANDLER_NULL of type *Errhandler*

mpi4py.MPI.ERRORS_RETURN

`mpi4py.MPI.ERRORS_RETURN: Errhandler = ERRORS_RETURN`
Object ERRORS_RETURN of type *Errhandler*

mpi4py.MPI.ERRORS_ABORT

`mpi4py.MPI.ERRORS_ABORT: Errhandler = ERRORS_ABORT`
Object ERRORS_ABORT of type *Errhandler*

mpi4py.MPI.ERRORS_ARE_FATAL

`mpi4py.MPI.ERRORS_ARE_FATAL: Errhandler = ERRORS_ARE_FATAL`
Object ERRORS_ARE_FATAL of type *Errhandler*

mpi4py.MPI.SESSION_NULL

`mpi4py.MPI.SESSION_NULL: Session = SESSION_NULL`

Object SESSION_NULL of type `Session`

mpi4py.MPI.COMM_NULL

`mpi4py.MPI.COMM_NULL: Comm = COMM_NULL`

Object COMM_NULL of type `Comm`

mpi4py.MPI.COMM_SELF

`mpi4py.MPI.COMM_SELF: Intracomm = COMM_SELF`

Object COMM_SELF of type `Intracomm`

mpi4py.MPI.COMM_WORLD

`mpi4py.MPI.COMM_WORLD: Intracomm = COMM_WORLD`

Object COMM_WORLD of type `Intracomm`

mpi4py.MPI.WIN_NULL

`mpi4py.MPI.WIN_NULL: Win = WIN_NULL`

Object WIN_NULL of type `Win`

mpi4py.MPI.FILE_NULL

`mpi4py.MPI.FILE_NULL: File = FILE_NULL`

Object FILE_NULL of type `File`

mpi4py.MPI.pickle

`mpi4py.MPI.pickle: Pickle = <mpi4py.MPI.Pickle object>`

Object pickle of type `Pickle`

12 Citation

If MPI for Python been significant to a project that leads to an academic publication, please acknowledge that fact by citing the project.

- M. Rogowski, S. Aseeri, D. Keyes, and L. Dalcin, *mpi4py.futures: MPI-Based Asynchronous Task Execution for Python*, IEEE Transactions on Parallel and Distributed Systems, 34(2):611-622, 2023. <https://doi.org/10.1109/TPDS.2022.3225481>
- 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>

13 Installation

13.1 Build backends

mpi4py supports two different build backends: `setuptools` (default), `scikit-build-core` (CMake-based), and `meson-python` (Meson-based). The build backend can be selected by setting the `MPI4PY_BUILD_BACKEND` environment variable.

`MPI4PY_BUILD_BACKEND`

Choices

```
"setuptools", "scikit-build-core", "meson-python"
```

Default

```
"setuptools"
```

Request a build backend for building mpi4py from sources.

Using `setuptools`

Tip: Set the `MPI4PY_BUILD_BACKEND` environment variable to "setuptools" to use the `setuptools` build backend.

When using the default `setuptools` build backend, mpi4py relies on the legacy Python distutils framework to build C extension modules. The following environment variables affect the build configuration.

`MPI4PY_BUILD_MPICC`

The `mpicc` compiler wrapper command is searched for in the executable search path (PATH environment variable) and used to compile the `mpi4py.MPI` C extension module. Alternatively, use the `MPI4PY_BUILD_MPICC` environment variable to the full path or command corresponding to the MPI-aware C compiler.

`MPI4PY_BUILD_MPILD`

The `mpicc` compiler wrapper command is also used for linking the `mpi4py.MPI` C extension module. Alternatively, use the `MPI4PY_BUILD_MPILD` environment variable to specify the full path or command corresponding to the MPI-aware C linker.

`MPI4PY_BUILD_MPICFG`

If the MPI implementation does not provide a compiler wrapper, or it is not installed in a default system location, all relevant build information like include/library locations and library lists can be provided in an ini-style configuration file under a [mpi] section. mpi4py can then be asked to use the custom build information by setting the `MPI4PY_BUILD_MPICFG` environment variable to the full path of the configuration file. As an example, see the `mpi.cfg` file located in the top level mpi4py source directory.

MPI4PY_BUILD_CONFIGURE

Some vendor MPI implementations may not provide complete coverage of the MPI standard, or may provide partial features of newer MPI standard versions while advertising support for an older version. Setting the [`MPI4PY_BUILD_CONFIGURE`](#) environment variable to a non-empty string will trigger the run of exhaustive checks for the availability of all MPI constants, predefined handles, and routines.

The following environment variables are aliases for the ones described above. Having shorter names, they are convenient for occasional use in the command line. Its usage is not recommended in automation scenarios like packaging recipes, deployment scripts, and container image creation.

MPICC

Convenience alias for [`MPI4PY_BUILD_MPICC`](#).

MPILD

Convenience alias for [`MPI4PY_BUILD_MPILD`](#).

MPICFG

Convenience alias for [`MPI4PY_BUILD_MPICFG`](#).

Using scikit-build-core

Tip: Set the [`MPI4PY_BUILD_BACKEND`](#) environment variable to "scikit-build-core" to use the `scikit-build-core` build backend.

When using the `scikit-build-core` build backend, mpi4py delegates all of MPI build configuration to CMake's FindMPI module. Besides the obvious advantage of cross-platform support, this delegation to CMake may be convenient in build environments exposing vendor software stacks via intricate module systems. Note however that mpi4py will not be able to look for MPI routines available beyond the MPI standard version the MPI implementation advertises to support (via the `MPI_VERSION` and `MPI_SUBVERSION` macro constants in the `mpi.h` header file), any missing MPI constant or symbol will prevent a successful build.

Using meson-python

Tip: Set the [`MPI4PY_BUILD_BACKEND`](#) environment variable to "meson-python" to use the `meson-python` build backend.

When using the `meson-python` build backend, mpi4py delegates build tasks to the `Meson` build system.

Warning: mpi4py support for the `meson-python` build backend is experimental. For the time being, users must set the `CC` environment variable to the command or path corresponding to the `mpicc` C compiler wrapper.

13.2 Using pip

You can install the latest mpi4py release from its source distribution at PyPI using pip:

```
$ python -m pip install mpi4py
```

You can also install the in-development version with:

```
$ python -m pip install git+https://github.com/mpi4py/mpi4py
```

or:

```
$ python -m pip install https://github.com/mpi4py/mpi4py/tarball/master
```

Note: Installing mpi4py from its source distribution (available at PyPI) or Git source code repository (available at GitHub) requires a C compiler and a working MPI implementation with development headers and libraries.

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
```

13.3 Using conda

The [conda-forge](#) community provides ready-to-use binary packages from an ever growing collection of software libraries built around the multi-platform `conda` package manager. Three MPI implementations are available on conda-forge: Open MPI (Linux and macOS), MPICH (Linux and macOS), and Microsoft MPI (Windows). You can install mpi4py and your preferred MPI implementation using the `conda` package manager:

- to use MPICH do:

```
$ conda install -c conda-forge mpi4py mpich
```

- to use Open MPI do:

```
$ conda install -c conda-forge mpi4py openmpi
```

- to use Microsoft MPI do:

```
$ conda install -c conda-forge mpi4py msmpi
```

MPICH and many of its derivatives are ABI-compatible. You can provide the package specification `mpich=X.Y.*=external_*` (where X and Y are the major and minor version numbers) to request the conda package manager to use system-provided MPICH (or derivative) libraries. Similarly, you can provide the package specification `openmpi=X.Y.*=external_*` to use system-provided Open MPI libraries.

The `openmpi` package on conda-forge has built-in CUDA support, but it is disabled by default. To enable it, follow the instruction outlined during `conda install`. Additionally, UCX support is also available once the `ucx` package is installed.

Warning: Binary conda-forge packages are built with a focus on compatibility. The MPICH and Open MPI packages are build in a constrained environment with relatively dated OS images. Therefore, they may lack support for high-performance features like cross-memory attach (XPMEM/CMA). In production scenarios, it is recommended to use external (either custom-built or system-provided) MPI installations. See the relevant conda-forge documentation about [using external MPI libraries](#).

13.4 Linux

On **Fedora Linux** systems (as well as **RHEL** and their derivatives using the EPEL software repository), you can install binary packages with the system package manager:

- using `dnf` and the `mpich` package:

```
$ sudo dnf install python3-mpi4py-mpich
```

- using `dnf` and the `openmpi` package:

```
$ sudo dnf install python3-mpi4py-openmpi
```

Please remember to load the correct MPI module for your chosen MPI implementation:

- for the `mpich` package do:

```
$ module load mpi/mpich-$(arch)
$ python -c "from mpi4py import MPI"
```

- for the `openmpi` package do:

```
$ module load mpi/openmpi-$(arch)
$ python -c "from mpi4py import MPI"
```

On **Ubuntu Linux** and **Debian Linux** systems, binary packages are available for installation using the system package manager:

```
$ sudo apt install python3-mpi4py
```

Note that on Ubuntu/Debian systems, the `mpi4py` package uses Open MPI. To use MPICH, install the `libmpich-dev` and `python3-dev` packages (and any other required development tools). Afterwards, install `mpi4py` from sources using `pip`.

13.5 macOS

macOS users can install mpi4py using the Homebrew package manager:

```
$ brew install mpi4py
```

Note that the Homebrew mpi4py package uses Open MPI. Alternatively, install the mpich package and next install mpi4py from sources using pip.

13.6 Windows

Windows users can install mpi4py from binary wheels hosted on the Python Package Index (PyPI) using pip:

```
$ python -m pip install mpi4py
```

Windows wheels require a separate, system-wide installation of the Microsoft MPI runtime package.

14 Development

14.1 Prerequisites

You need to have the following software properly installed in order to build *MPI for Python*:

- Python 3.6 or above.
- The Cython compiler.
- A working MPI implementation like MPICH or Open MPI, preferably supporting MPI-4 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.

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.

Optionally, consider installing the following packages:

- NumPy for enabling comprehensive testing of MPI communication.
- CuPy for enabling comprehensive testing with a GPU-aware MPI.
- Sphinx to build documentation.

14.2 Building

MPIfor Python uses **setuptools**-based build system that relies on the `setup.py` file. Some **setuptools** commands (e.g., `build`) accept additional options:

--mpi=

Lets you pass a section with MPI configuration within a special configuration file. Alternatively, you can use the `MPICFG` environment variable.

--mpicc=

Specify the path or name of the `mpicc` C compiler wrapper. Alternatively, use the `MPICC` environment variable.

--mpild=

Specify the full path or name for the MPI-aware C linker. Alternatively, use the `MPILD` environment variable. If not set, the `mpicc` C compiler wrapper is used for linking.

--configure

Runs exhaustive tests for checking about missing MPI types, constants, and functions. This option should be passed in order to build *MPIfor Python* against old MPI-1, MPI-2, or MPI-3 implementations, possibly providing a subset of MPI-4.

If you use a MPI implementation providing a `mpicc` C compiler wrapper (e.g., MPICH or Open MPI), it will be used for compilation and linking. This is the preferred and easiest way to build *MPIfor Python*.

If `mpicc` is found in the executable search path (PATH environment variable), 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, either via the `--mpicc` command option or using the `MPICC` environment variable:

```
$ python setup.py build --mpicc=/path/to/mpicc
$ MPICC=/path/to/mpicc python setup.py build
```

Alternatively, you can provide all the relevant information about your MPI implementation by editing the `mpi.cfg` file located in the top level source directory. 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 specifying you custom configuration section:

```
$ python setup.py build --mpi=other_mpi
$ MPICFG=other_mpi python setup.py build
```

After building, the package is ready for installation in development mode:

```
$ python setup.py develop --user
```

Alternatively, you can generate a binary wheel file in the `dist/` directory with:

```
$ python setup.py bdist_wheel
```

14.3 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.

$ mpiexec -n 5 python -m mpi4py.bench ringtest -l 10 -n 1048576
time for 10 loops = 0.00361614 seconds (5 processes, 1048576 bytes)
```

If you installed from a git clone or the source distribution, issuing at the command line:

```
$ mpiexec -n 5 python 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/main.py
```

or, if you have `nose` unit testing framework installed:

```
$ mpiexec -n 5 nosetests
```

or, if you have `py.test` unit testing framework installed:

```
$ mpiexec -n 5 py.test
```

15 Appendix

15.1 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 3.9) of what is required is shown below:

```
#include <Python.h>
#include <mpi.h>

int main(int argc, char *argv[])
{
    int status, flag;
    MPI_Init(&argc, &argv);
    status = Py_BytesMain(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:

```
$ python3.9-mpi
Python 3.9.6 (default, Jul 16 2021, 00:00:00)
[GCC 11.1.1 20210531 (Red Hat 11.1.1-3)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.executable
'/usr/local/bin/python3.9-mpi'
>>>
```

15.2 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.

16 LICENSE

Copyright (c) 2024, Lisandro Dalcin.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES

OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

17 CHANGES

17.1 Release 4.0.0 [2024-XX-XX]

- New features:
 - Add support for the MPI-4.0 standard.
 - * Use large count MPI-4 routines.
 - * Add persistent collective communication.
 - * Add partitioned point-to-point communication.
 - * Add new communicator constructors.
 - * Add the `Session` class and its methods.
 - Add support for the MPI-4.1 standard.
 - * Add non-destructive completion test for multiple requests.
 - * Add value-index datatype constructor.
 - * Add communicator/session buffer attach/detach/flush.
 - * Support for removal of error classes/codes/strings.
 - * Support for querying hardware resource information.
 - Add preliminary support for the upcoming MPI-5.0 standard.
 - * User-level failure mitigation (ULFM).
 - `mpi4py.util.pool`: New drop-in replacement for `multiprocessing.pool`.
 - Add runtime check for mismatch between `mpiexec` and MPI library.
 - Support `scikit-build-core` as an alternative build backend.
 - Support `meson-python` as an alternative build backend.
- Enhancements:
 - `mpi4py.futures`: Support for parallel tasks.
 - `mpi4py.futures`: Report exception tracebacks in workers.
 - `mpi4py.util.pk15`: Add support for collective communication.
 - Add methods `Datatype.fromcode()`, `Datatype.tocode()` and attributes `Datatype.typestr`, `Datatype.typechar` to simplify NumPy interoperability for simple cases.
 - Add methods `Comm.Create_errhandler()`, `Win.Create_errhandler()`, and `File.Create_errhandler()` to create custom error handlers.

- Add support for pickle serialization of instances of MPI types. All instances of `Datatype`, `Info`, and `Status` can be serialized. Instances of `Op` can be serialized only if created through `mpi4py` by calling `Op.Create()`. Instances of other MPI types can be serialized only if they reference predefined handles.
- Add `handle` attribute and `fromhandle()` class method to MPI classes to ease interoperability with external code. The handle value is a unsigned integer guaranteed to fit on the platform's `uintptr_t` C type.
- Backward-incompatible changes:
 - Python 2 is no longer supported, Python 3.6+ is required, but typing stubs are supported for Python 3.8+.
 - The `Intracomm.Create_group()` method is no longer defined in the base `Comm` class.
 - `Group.Compare()` and `Comm.Compare()` are no longer class methods but instance methods. Existing codes using the former class methods are expected to continue working.
 - `Group.Translate_ranks()` is no longer a class method but a instance method. Existing codes using the former class method are expected to continue working.
 - The LB and UB datatypes are no longer available, use `Datatype.Create_resized()` instead.
 - The `MPI.memory` class has been renamed to `MPI.buffer`. The old name is still available as an alias to the new name.
 - The `mpi4py.dl` module is no longer available.
 - The `mpi4py.get_config` function returns an empty dictionary.

17.2 Release 3.1.5 [2023-10-04]

Warning: This is the last release supporting Python 2.

- Rebuild C sources with Cython 0.29.36 to support Python 3.12.

17.3 Release 3.1.4 [2022-11-02]

Warning: This is the last release supporting Python 2.

- Rebuild C sources with Cython 0.29.32 to support Python 3.11.
- Fix contiguity check for DLPack and CAI buffers.
- Workaround build failures with setuptools v60.

17.4 Release 3.1.3 [2021-11-25]

Warning: This is the last release supporting Python 2.

- Add missing support for MPI.BOTTOM to generalized all-to-all collectives.

17.5 Release 3.1.2 [2021-11-04]

Warning: This is the last release supporting Python 2.

- mpi4py.futures: Add _max_workers property to MPIPoolExecutor.
- mpi4py.util.dtlib: Fix computation of alignment for predefined datatypes.
- mpi4py.util.pk15: Fix deadlock when using ssend() + mprobe().
- mpi4py.util.pk15: Add environment variable MPI4PY_PICKLE_THRESHOLD.
- mpi4py.rc: Interpret "y" and "n" strings as boolean values.
- Fix/add typemap/typestr for MPI.WCHAR/MPI.COUNT datatypes.
- Minor fixes and additions to documentation.
- Minor fixes to typing support.
- Support for local version identifier (PEP-440).

17.6 Release 3.1.1 [2021-08-14]

Warning: This is the last release supporting Python 2.

- Fix typo in Requires-Python package metadata.
- Regenerate C sources with Cython 0.29.24.

17.7 Release 3.1.0 [2021-08-12]

Warning: This is the last release supporting Python 2.

- New features:
 - mpi4py.util: New package collecting miscellaneous utilities.
- Enhancements:
 - Add pickle-based Request.waitsome() and Request.testsome().
 - Add lowercase methods Request.get_status() and Request.cancel().

- Support for passing Python GPU arrays compliant with the [DLPack](#) data interchange mechanism ([link](#)) and the `__cuda_array_interface__` (CAI) standard ([link](#)) to uppercase methods. This support requires that `mpi4py` is built against [CUDA-aware MPI](#) implementations. This feature is currently experimental and subject to future changes.
- `mpi4py.futures`: Add support for initializers and canceling futures at shutdown. Environment variables names now follow the pattern `MPI4PY_FUTURES_*`, the previous `MPI4PY_*` names are deprecated.
- Add type annotations to Cython code. The first line of the docstring of functions and methods displays a signature including type annotations.
- Add companion stub files to support type checkers.
- Support for weak references.
- Miscellaneous:
 - Add a new `mpi4py` publication ([link](#)) to the citation listing.

17.8 Release 3.0.3 [2019-11-04]

- Regenerate Cython wrappers to support Python 3.8.

17.9 Release 3.0.2 [2019-06-11]

- Bug fixes:
 - Fix handling of readonly buffers in support for Python 2 legacy buffer interface. The issue triggers only when using a buffer-like object that is readonly and does not export the new Python 3 buffer interface.
 - Fix build issues with Open MPI 4.0.x series related to removal of many MPI-1 symbols deprecated in MPI-2 and removed in MPI-3.
 - Minor documentation fixes.

17.10 Release 3.0.1 [2019-02-15]

- Bug fixes:
 - Fix `Comm.scatter()` and other collectives corrupting input send list. Add safety measures to prevent related issues in global reduction operations.
 - Fix error-checking code for counts in `Op.Reduce_local()`.
- Enhancements:
 - Map size-specific Python/NumPy typecodes to MPI datatypes.
 - Allow partial specification of target list/tuple arguments in the various `Win` RMA methods.
 - Workaround for removal of `MPI_{LB|UB}` in Open MPI 4.0.
 - Support for Microsoft MPI v10.0.

17.11 Release 3.0.0 [2017-11-08]

- New features:
 - `mpi4py.futures`: Execute computations asynchronously using a pool of MPI processes. This package is based on `concurrent.futures` from the Python standard library.
 - `mpi4py.run`: Run Python code and abort execution in case of unhandled exceptions to prevent deadlocks.
 - `mpi4py.bench`: Run basic MPI benchmarks and tests.
- Enhancements:
 - Lowercase, pickle-based collective communication calls are now thread-safe through the use of fine-grained locking.
 - The MPI module now exposes a `memory` type which is a lightweight variant of the builtin `memoryview` type, but exposes both the legacy Python 2 and the modern Python 3 buffer interface under a Python 2 runtime.
 - The `MPI.Comm.Alltoallw()` method now uses `count=1` and `displ=0` as defaults, assuming that messages are specified through user-defined datatypes.
 - The `Request.Wait[all]()` methods now return `True` to match the interface of `Request.Test[all]()`.
 - The `Win` class now implements the Python buffer interface.
- Backward-incompatible changes:
 - The `buf` argument of the `MPI.Comm.recv()` method is deprecated, passing anything but `None` emits a warning.
 - The `MPI.Win.memory` property was removed, use the `MPI.Win.tomemory()` method instead.
 - Executing `python -m mpi4py` in the command line is now equivalent to `python -m mpi4py.run`. For the former behavior, use `python -m mpi4py.bench`.
 - Python 2.6 and 3.2 are no longer supported. The `mpi4py.MPI` module may still build and partially work, but other pure-Python modules under the `mpi4py` namespace will not.
 - Windows: Remove support for legacy MPICH2, Open MPI, and DeinoMPI.

17.12 Release 2.0.0 [2015-10-18]

- Support for MPI-3 features.
 - Matched probes and receives.
 - Nonblocking collectives.
 - Neighborhood collectives.
 - New communicator constructors.
 - Request-based RMA operations.
 - New RMA communication and synchronisation calls.
 - New window constructors.
 - New datatype constructor.
 - New C++ boolean and floating complex datatypes.
- Support for MPI-2 features not included in previous releases.
 - Generalized All-to-All collective (`Comm.Alltoallw()`)

- User-defined data representations (`Register_datarep()`)
- New scalable implementation of reduction operations for Python objects. This code is based on binomial tree algorithms using point-to-point communication and duplicated communicator contexts. To disable this feature, use `mpi4py.rc.fast_reduce = False`.
- Backward-incompatible changes:
 - Python 2.4, 2.5, 3.0 and 3.1 are no longer supported.
 - Default MPI error handling policies are overridden. After import, mpi4py sets the `ERRORS_RETURN` error handler in `COMM_SELF` and `COMM_WORLD`, as well as any new `Comm`, `Win`, or `File` instance created through mpi4py, thus effectively ignoring the MPI rules about error handler inheritance. This way, MPI errors translate to Python exceptions. To disable this behavior and use the standard MPI error handling rules, use `mpi4py.rc.errors = 'default'`.
 - Change signature of all send methods, `dest` is a required argument.
 - Change signature of all receive and probe methods, `source` defaults to `ANY_SOURCE`, `tag` defaults to `ANY_TAG`.
 - Change signature of send lowercase-spelling methods, `obj` arguments are not mandatory.
 - Change signature of recv lowercase-spelling methods, renamed ‘`obj`’ arguments to ‘`buf`’.
 - Change `Request.Waitsome()` and `Request.Testsome()` to return `None` or `list`.
 - Change signature of all lowercase-spelling collectives, `sendobj` arguments are now mandatory, `recvobj` arguments were removed.
 - Reduction operations `MAXLOC` and `MINLOC` are no longer special-cased in lowercase-spelling methods `Comm.[all]reduce()` and `Comm.[ex]scan()`, the input object must be specified as a tuple `(obj, location)`.
 - Change signature of name publishing functions. The new signatures are `Publish_name(service_name, port_name, info=INFO_NULL)` and `Unpublish_name(service_name, port_name, info=INFO_NULL)`.
 - `Win` instances now cache Python objects exposing memory by keeping references instead of using MPI attribute caching.
 - Change signature of `Win.Lock()`. The new signature is `Win.Lock(rank, lock_type=LOCK_EXCLUSIVE, assertion=0)`.
 - Move `Cartcomm.Map()` to `Intracomm.Cart_map()`.
 - Move `Graphcomm.Map()` to `Intracomm.Graph_map()`.
 - Remove the `mpi4py.MPE` module.
 - Rename the Cython definition file for use with `cimport` statement from `mpi_c.pxd` to `libmpi.pxd`.

17.13 Release 1.3.1 [2013-08-07]

- Regenerate C wrappers with Cython 0.19.1 to support Python 3.3.
- Install *.pxd files in `<site-packages>/mpi4py` to ease the support for Cython’s `cimport` statement in code requiring to access mpi4py internals.
- As a side-effect of using Cython 0.19.1, ancient Python 2.3 is no longer supported. If you really need it, you can install an older Cython and run `python setup.py build_src --force`.

17.14 Release 1.3 [2012-01-20]

- Now `Comm.recv()` accept a buffer to receive the message.
- Add `Comm.irecv()` and `Request.{wait|test}[any|all]()`.
- Add `Intracomm.Spawn_multiple()`.
- Better buffer handling for PEP 3118 and legacy buffer interfaces.
- Add support for attribute attribute caching on communicators, datatypes and windows.
- Install MPI-enabled Python interpreter as `<path>/mpi4py/bin/python-mpi`.
- Windows: Support for building with Open MPI.

17.15 Release 1.2.2 [2010-09-13]

- Add `mpi4py.get_config()` to retrieve information (compiler wrappers, includes, libraries, etc) about the MPI implementation employed to build mpi4py.
- Workaround Python libraries with missing GILState-related API calls in case of non-threaded Python builds.
- Windows: look for MPICH2, DeinoMPI, Microsoft HPC Pack at their default install locations under %Program-Files.
- MPE: fix hacks related to old API's, these hacks are broken when MPE is built with a MPI implementations other than MPICH2.
- HP-MPI: fix for missing Fortran datatypes, use `dllopen()` to load the MPI shared library before `MPI_Init()`
- Many distutils-related fixes, cleanup, and enhancements, better logics to find MPI compiler wrappers.
- Support for `pip install mpi4py`.

17.16 Release 1.2.1 [2010-02-26]

- Fix declaration in Cython include file. This declaration, while valid for Cython, broke the simple-minded parsing used in `conf/mpidistutils.py` to implement configure-tests for availability of MPI symbols.
- Update SWIG support and make it compatible with Python 3. Also generate an warning for SWIG < 1.3.28.
- Fix distutils-related issues in Mac OS X. Now `ARCHFLAGS` environment variable is honored of all Python's `config/Makefile` variables.
- Fix issues with Open MPI < 1.4.2 related to error checking and `MPI_XXX_NULL` handles.

17.17 Release 1.2 [2009-12-29]

- Automatic MPI datatype discovery for NumPy arrays and PEP-3118 buffers. Now buffer-like objects can be messaged directly, it is no longer required to explicitly pass a 2/3-list/tuple like `[data, MPI.DOUBLE]`, or `[data, count, MPI.DOUBLE]`. Only basic types are supported, i.e., all C/C99-native signed/unsigned integral types and single/double precision real/complex floating types. Many thanks to Eilif Muller for the initial feedback.
- Nonblocking send of pickled Python objects. Many thanks to Andreas Kloeckner for the initial patch and enlightening discussion about this enhancement.
- `Request` instances now hold a reference to the Python object exposing the buffer involved in point-to-point communication or parallel I/O. Many thanks to Andreas Kloeckner for the initial feedback.

- Support for logging of user-defined states and events using [MPE](#). Runtime (i.e., without requiring a recompile!) activation of logging of all MPI calls is supported in POSIX platforms implementing `dlopen()`.
- Support for all the new features in MPI-2.2 (new C99 and F90 datatypes, distributed graph topology, local reduction operation, and other minor enhancements).
- Fix the annoying issues related to Open MPI and Python dynamic loading of extension modules in platforms supporting `dlopen()`.
- Fix SLURM dynamic loading issues on SiCortex. Many thanks to Ian Langmore for providing me shell access.

17.18 Release 1.1.0 [2009-06-06]

- Fix bug in `Comm.Iprobe()` that caused segfaults as Python C-API calls were issued with the GIL released (issue #2).
- Add `Comm.bsend()` and `Comm.ssend()` for buffered and synchronous send semantics when communicating general Python objects.
- Now the call `Info.Get(key)` return a *single* value (i.e, instead of a 2-tuple); this value is `None` if `key` is not in the `Info` object, or a string otherwise. Previously, the call redundantly returned `(None, False)` for missing key-value pairs; `None` is enough to signal a missing entry.
- Add support for parametrized Fortran datatypes.
- Add support for decoding user-defined datatypes.
- Add support for user-defined reduction operations on memory buffers. However, at most 16 user-defined reduction operations can be created. Ask the author for more room if you need it.

17.19 Release 1.0.0 [2009-03-20]

This is the fist release of the all-new, Cython-based, implementation of *MPI for Python*. Unfortunately, this implementation is not backward-compatible with the previous one. The list below summarizes the more important changes that can impact user codes.

- Some communication calls had *overloaded* functionality. Now there is a clear distinction between communication of general Python object with *pickle*, and (fast, near C-speed) communication of buffer-like objects (e.g., NumPy arrays).
 - for communicating general Python objects, you have to use all-lowercase methods, like `send()`, `recv()`, `bcast()`, etc.
 - for communicating array data, you have to use `Send()`, `Recv()`, `Bcast()`, etc. methods. 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 the `count`).
- Indexing a communicator with an integer returned a special object associating the communication with a target rank, alleviating you from specifying source/destination/root arguments in point-to-point and collective communications. This functionality is no longer available, expressions like:

```
MPI.COMM_WORLD[0].Send(....)
MPI.COMM_WORLD[0].Recv(....)
MPI.COMM_WORLD[0].Bcast(....)
```

have to be replaced by:

```

MPI.COMM_WORLD.Send(..., dest=0)
MPI.COMM_WORLD.Recv(..., source=0)
MPI.COMM_WORLD.Bcast(..., root=0)

```

- Automatic MPI initialization (i.e., at import time) requests the maximum level of MPI thread support (i.e., it is done by calling `MPI_Init_thread()` and passing `MPI_THREAD_MULTIPLE`). In case you need to change this behavior, you can tweak the contents of the `mpi4py.rc` module.
- In order to obtain the values of predefined attributes attached to the world communicator, now you have to use the `Get_attr()` method on the `MPI.COMM_WORLD` instance:

```
tag_ub = MPI.COMM_WORLD.Get_attr(MPI.TAG_UB)
```

- In the previous implementation, `MPI.COMM_WORLD` and `MPI.COMM_SELF` were associated to **duplicates** of the (C-level) `MPI_COMM_WORLD` and `MPI_COMM_SELF` predefined communicator handles. Now this is no longer the case, `MPI.COMM_WORLD` and `MPI.COMM_SELF` proxies the **actual** `MPI_COMM_WORLD` and `MPI_COMM_SELF` handles.
- Convenience aliases `MPI.WORLD` and `MPI.SELF` were removed. Use instead `MPI.COMM_WORLD` and `MPI.COMM_SELF`.
- Convenience constants `MPI.WORLD_SIZE` and `MPI.WORLD_RANK` were removed. Use instead `MPI.COMM_WORLD.Get_size()` and `MPI.COMM_WORLD.Get_rank()`.

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.
- [mpi4py-futures] M. Rogowski, S. Aseeri, D. Keyes, and L. Dalcin, *mpi4py.futures: MPI-Based Asynchronous Task Execution for Python*, IEEE Transactions on Parallel and Distributed Systems, 34(2):611-622, 2023. <https://doi.org/10.1109/TPDS.2022.3225481>

Python Module Index

M

`mpi4py`, 20
`mpi4py.bench`, 59
`mpi4py.futures`, 38
`mpi4py.MPI`, 59
`mpi4py.run`, 57
`mpi4py.typing`, 35
`mpi4py.util`, 48
`mpi4py.util.dtlib`, 48
`mpi4py.util.pkl5`, 48
`mpi4py.util.pool`, 55

Index

Symbols

`__init__(mpi4py.util.pool.Pool method)`, 55
`__new__(mpi4py.MPI.BottomType static method)`, 60
`__new__(mpi4py.MPI.BufferAutomaticType static method)`, 60
`__new__(mpi4py.MPI.Cartcomm static method)`, 60
`__new__(mpi4py.MPI.Comm static method)`, 62
`__new__(mpi4py.MPI.Datatype static method)`, 93
`__new__(mpi4py.MPI.Distgraphcomm static method)`, 104
`__new__(mpi4py.MPI.Errhandler static method)`, 104
`__new__(mpi4py.MPI.Exception static method)`, 175
`__new__(mpi4py.MPI.File static method)`, 106
`__new__(mpi4py.MPI.Graphcomm static method)`, 118
`__new__(mpi4py.MPI.Grequest static method)`, 120
`__new__(mpi4py.MPI.Group static method)`, 121
`__new__(mpi4py.MPI.InPlaceType static method)`, 124
`__new__(mpi4py.MPI.Info static method)`, 125
`__new__(mpi4py.MPI.Intercomm static method)`, 129
`__new__(mpi4py.MPI.Intracomm static method)`, 130
`__new__(mpi4py.MPI.Message static method)`, 136
`__new__(mpi4py.MPI.Op static method)`, 139
`__new__(mpi4py.MPI.Pickle static method)`, 141
`__new__(mpi4py.MPI.Prequest static method)`, 142
`__new__(mpi4py.MPI.Request static method)`, 144
`__new__(mpi4py.MPI.Session static method)`, 151
`__new__(mpi4py.MPI.Status static method)`, 154
`__new__(mpi4py.MPI.Topocomm static method)`, 157
`__new__(mpi4py.MPI.Win static method)`, 162
`__new__(mpi4py.MPI.buffer static method)`, 173
--configure
 command line option, 236
--mpi
 command line option, 236
--mpicc
 command line option, 236
--mpild
 command line option, 236
-c
 command line option, 58
-m
 command line option, 58

A

`Abort()` (*mpi4py.MPI.Comm method*), 66
`Accept()` (*mpi4py.MPI.Intracomm method*), 131
`Accumulate()` (*mpi4py.MPI.Win method*), 164
`Ack_failed()` (*mpi4py.MPI.Comm method*), 66

`Add_error_class()` (*in module mpi4py.MPI*), 178
`Add_error_code()` (*in module mpi4py.MPI*), 178
`Add_error_string()` (*in module mpi4py.MPI*), 178
`address` (*mpi4py.MPI.buffer attribute*), 175
`Agree()` (*mpi4py.MPI.Comm method*), 66
`AINT` (*in module mpi4py.MPI*), 214
`Aint` (*in module mpi4py.typing*), 36
`Aint_add()` (*in module mpi4py.MPI*), 178
`Aint_diff()` (*in module mpi4py.MPI*), 179
`Allgather()` (*mpi4py.MPI.Comm method*), 66
`allgather()` (*mpi4py.MPI.Comm method*), 88
`allgather()` (*mpi4py.util.pk15.Comm method*), 53
`Allgather_init()` (*mpi4py.MPI.Comm method*), 66
`Allgatherv()` (*mpi4py.MPI.Comm method*), 66
`Allgatherv_init()` (*mpi4py.MPI.Comm method*), 67
`Alloc_mem()` (*in module mpi4py.MPI*), 179
`allocate()` (*mpi4py.MPI.buffer static method*), 174
`Allocate()` (*mpi4py.MPI.Win class method*), 164
`Allocate_shared()` (*mpi4py.MPI.Win class method*), 165
`Allreduce()` (*mpi4py.MPI.Comm method*), 67
`allreduce()` (*mpi4py.MPI.Comm method*), 88
`Allreduce_init()` (*mpi4py.MPI.Comm method*), 67
`Alltoall()` (*mpi4py.MPI.Comm method*), 67
`alltoall()` (*mpi4py.MPI.Comm method*), 88
`alltoall()` (*mpi4py.util.pk15.Comm method*), 54
`Alltoall_init()` (*mpi4py.MPI.Comm method*), 68
`Alltoallv()` (*mpi4py.MPI.Comm method*), 68
`Alltoallv_init()` (*mpi4py.MPI.Comm method*), 68
`Alltoallw()` (*mpi4py.MPI.Comm method*), 68
`Alltoallw_init()` (*mpi4py.MPI.Comm method*), 68
`amode` (*mpi4py.MPI.File attribute*), 118
`ANY_SOURCE` (*in module mpi4py.MPI*), 192
`ANY_TAG` (*in module mpi4py.MPI*), 192
`apply()` (*mpi4py.util.pool.Pool method*), 55
`apply_async()` (*mpi4py.util.pool.Pool method*), 56
`ApplyResult` (*class in mpi4py.util.pool*), 57
`APPNUM` (*in module mpi4py.MPI*), 193
`AsyncResult` (*class in mpi4py.util.pool*), 57
`atomicity` (*mpi4py.MPI.File attribute*), 118
`Attach()` (*mpi4py.MPI.Win method*), 165
`Attach_buffer()` (*in module mpi4py.MPI*), 179
`Attach_buffer()` (*mpi4py.MPI.Comm method*), 69
`Attach_buffer()` (*mpi4py.MPI.Session method*), 152
`attrs` (*mpi4py.MPI.Win attribute*), 172

B

`BAND` (*in module mpi4py.MPI*), 226
`Barrier()` (*mpi4py.MPI.Comm method*), 69
`barrier()` (*mpi4py.MPI.Comm method*), 88

Barrier_init() (*mpi4py.MPI.Comm method*), 69
Bcast() (*mpi4py.MPI.Comm method*), 69
bcast() (*mpi4py.MPI.Comm method*), 88
bcast() (*mpi4py.util.pkl5.Comm method*), 53
Bcast_init() (*mpi4py.MPI.Comm method*), 69
BOOL (*in module mpi4py.MPI*), 223
bootup() (*mpi4py.futures.MPIPoolExecutor method*), 41
BOR (*in module mpi4py.MPI*), 227
BOTTOM (*in module mpi4py.MPI*), 192
Bottom (*in module mpi4py.typing*), 36
BottomType (*class in mpi4py.MPI*), 60
Bsend() (*mpi4py.MPI.Comm method*), 69
bsend() (*mpi4py.MPI.Comm method*), 88
bsend() (*mpi4py.util.pkl5.Comm method*), 50
Bsend_init() (*mpi4py.MPI.Comm method*), 70
BSEND_OVERHEAD (*in module mpi4py.MPI*), 208
buffer (*class in mpi4py.MPI*), 173
Buffer (*in module mpi4py.typing*), 36
BUFFER_AUTOMATIC (*in module mpi4py.MPI*), 208
BufferAutomaticType (*class in mpi4py.MPI*), 60
BufSpec (*in module mpi4py.typing*), 37
BufSpecB (*in module mpi4py.typing*), 37
BufSpecV (*in module mpi4py.typing*), 37
BufSpecW (*in module mpi4py.typing*), 38
BXOR (*in module mpi4py.MPI*), 227
BYTE (*in module mpi4py.MPI*), 214

C

C_BOOL (*in module mpi4py.MPI*), 216
C_COMPLEX (*in module mpi4py.MPI*), 218
C_DOUBLE_COMPLEX (*in module mpi4py.MPI*), 218
C_FLOAT_COMPLEX (*in module mpi4py.MPI*), 218
C_LONG_DOUBLE_COMPLEX (*in module mpi4py.MPI*), 218
Call_errhandler() (*mpi4py.MPI.Comm method*), 70
Call_errhandler() (*mpi4py.MPI.File method*), 108
Call_errhandler() (*mpi4py.MPI.Session method*), 152
Call_errhandler() (*mpi4py.MPI.Win method*), 165
Cancel() (*mpi4py.MPI.Request method*), 146
cancel() (*mpi4py.MPI.Request method*), 148
cancel() (*mpi4py.util.pkl5.Request method*), 49
cancelled (*mpi4py.MPI.Status attribute*), 157
CART (*in module mpi4py.MPI*), 207
Cart_map() (*mpi4py.MPI.Intracomm method*), 131
Cartcomm (*class in mpi4py.MPI*), 60
CC, 232
CHAR (*in module mpi4py.MPI*), 215
CHARACTER (*in module mpi4py.MPI*), 219
clear() (*mpi4py.MPI.Info method*), 127
Clone() (*mpi4py.MPI.Comm method*), 70
Close() (*mpi4py.MPI.File method*), 108
close() (*mpi4py.util.pool.Pool method*), 56
Close_port() (*in module mpi4py.MPI*), 179
collect() (*in module mpi4py.futures*), 44

combiner (*mpi4py.MPI.Datatype attribute*), 103
COMBINER_CONTIGUOUS (*in module mpi4py.MPI*), 204
COMBINER_DARRAY (*in module mpi4py.MPI*), 205
COMBINER_DUP (*in module mpi4py.MPI*), 204
COMBINER_F90_COMPLEX (*in module mpi4py.MPI*), 206
COMBINER_F90_INTEGER (*in module mpi4py.MPI*), 206
COMBINER_F90_REAL (*in module mpi4py.MPI*), 206
COMBINER_HINDEXED (*in module mpi4py.MPI*), 205
COMBINER_HINDEXED_BLOCK (*in module mpi4py.MPI*), 205
COMBINER_HVECTOR (*in module mpi4py.MPI*), 204
COMBINER_INDEXED (*in module mpi4py.MPI*), 205
COMBINER_INDEXED_BLOCK (*in module mpi4py.MPI*), 205
COMBINER_NAMED (*in module mpi4py.MPI*), 204
COMBINER_RESIZED (*in module mpi4py.MPI*), 205
COMBINER_STRUCT (*in module mpi4py.MPI*), 205
COMBINER_SUBARRAY (*in module mpi4py.MPI*), 205
COMBINER_VALUE_INDEX (*in module mpi4py.MPI*), 206
COMBINER_VECTOR (*in module mpi4py.MPI*), 204
Comm (*class in mpi4py.MPI*), 62
Comm (*class in mpi4py.util.pkl5*), 50
COMM_NULL (*in module mpi4py.MPI*), 230
COMM_SELF (*in module mpi4py.MPI*), 230
COMM_TYPE_HW_GUIDED (*in module mpi4py.MPI*), 208
COMM_TYPE_HW_UNGUIDED (*in module mpi4py.MPI*), 208
COMM_TYPE_RESOURCE_GUIDED (*in module mpi4py.MPI*), 208
COMM_TYPE_SHARED (*in module mpi4py.MPI*), 208
COMM_WORLD (*in module mpi4py.MPI*), 230
command line option
 --configure, 236
 --mpi, 236
 --mpicc, 236
 --mpild, 236
 -c, 58
 -m, 58
Commit() (*mpi4py.MPI.Datatype method*), 95
Compare() (*mpi4py.MPI.Comm method*), 70
Compare() (*mpi4py.MPI.Group method*), 122
Compare_and_swap() (*mpi4py.MPI.Win method*), 165
Complete() (*mpi4py.MPI.Grequest method*), 120
Complete() (*mpi4py.MPI.Win method*), 165
COMPLEX (*in module mpi4py.MPI*), 220
COMPLEX16 (*in module mpi4py.MPI*), 222
COMPLEX32 (*in module mpi4py.MPI*), 222
COMPLEX4 (*in module mpi4py.MPI*), 222
COMPLEX8 (*in module mpi4py.MPI*), 222
compose() (*in module mpi4py.futures*), 44
Compute_dims() (*in module mpi4py.MPI*), 180
CONGRUENT (*in module mpi4py.MPI*), 207
Connect() (*mpi4py.MPI.Intracomm method*), 131
contents (*mpi4py.MPI.Datatype attribute*), 103
coords (*mpi4py.MPI.Cartcomm attribute*), 62

copy() (*mpi4py.MPI.Info method*), 127
COUNT (*in module mpi4py.MPI*), 214
Count (*in module mpi4py.typing*), 36
count (*mpi4py.MPI.Status attribute*), 157
Create() (*mpi4py.MPI.Comm method*), 70
Create() (*mpi4py.MPI.Info class method*), 126
Create() (*mpi4py.MPI.Op class method*), 140
Create() (*mpi4py.MPI.Win class method*), 165
Create_cart() (*mpi4py.MPI.Intracomm method*), 132
Create_contiguous() (*mpi4py.MPI.Datatype method*), 95
Create_darray() (*mpi4py.MPI.Datatype method*), 95
Create_dist_graph() (*mpi4py.MPI.Intracomm method*), 132
Create_dist_graph_adjacent() (*mpi4py.MPI.Intracomm method*), 132
Create_dynamic() (*mpi4py.MPI.Win class method*), 166
Create_env() (*mpi4py.MPI.Info class method*), 126
Create_errhandler() (*mpi4py.MPI.Comm class method*), 70
Create_errhandler() (*mpi4py.MPI.File class method*), 108
Create_errhandler() (*mpi4py.MPI.Session class method*), 152
Create_errhandler() (*mpi4py.MPI.Win class method*), 166
Create_f90_complex() (*mpi4py.MPI.Datatype class method*), 95
Create_f90_integer() (*mpi4py.MPI.Datatype class method*), 96
Create_f90_real() (*mpi4py.MPI.Datatype class method*), 96
Create_from_group() (*mpi4py.MPI.Intracomm class method*), 133
Create_from_groups() (*mpi4py.MPI.Intercomm class method*), 129
Create_from_session_pset() (*mpi4py.MPI.Group class method*), 122
Create_graph() (*mpi4py.MPI.Intracomm method*), 133
Create_group() (*mpi4py.MPI.Intracomm method*), 133
Create_group() (*mpi4py.MPI.Session method*), 152
Create_hindexed() (*mpi4py.MPI.Datatype method*), 96
Create_hindexed_block() (*mpi4py.MPI.Datatype method*), 96
Create_hvector() (*mpi4py.MPI.Datatype method*), 97
Create_indexed() (*mpi4py.MPI.Datatype method*), 97
Create_indexed_block() (*mpi4py.MPI.Datatype method*), 97
Create_intercomm() (*mpi4py.MPI.Intracomm method*), 133
Create_keyval() (*mpi4py.MPI.Comm class method*), 71
Create_keyval() (*mpi4py.MPI.Datatype class method*), 97
Create_keyval() (*mpi4py.MPI.Win class method*), 166
Create_resized() (*mpi4py.MPI.Datatype method*), 97
Create_struct() (*mpi4py.MPI.Datatype class method*), 97
Create_subarray() (*mpi4py.MPI.Datatype method*), 98
Create_vector() (*mpi4py.MPI.Datatype method*), 98
CXX_BOOL (*in module mpi4py.MPI*), 218
CXX_DOUBLE_COMPLEX (*in module mpi4py.MPI*), 218
CXX_FLOAT_COMPLEX (*in module mpi4py.MPI*), 218
CXX_LONG_DOUBLE_COMPLEX (*in module mpi4py.MPI*), 218

D

Datatype (*class in mpi4py.MPI*), 93
DATATYPE_NULL (*in module mpi4py.MPI*), 214
decode() (*mpi4py.MPI.Datatype method*), 102
degrees (*mpi4py.MPI.Topocomm attribute*), 162
Delete() (*mpi4py.MPI.File class method*), 108
Delete() (*mpi4py.MPI.Info method*), 126
Delete_attr() (*mpi4py.MPI.Comm method*), 71
Delete_attr() (*mpi4py.MPI.Datatype method*), 98
Delete_attr() (*mpi4py.MPI.Win method*), 166
Detach() (*mpi4py.MPI.Win method*), 166
Detach_buffer() (*in module mpi4py.MPI*), 180
Detach_buffer() (*mpi4py.MPI.Comm method*), 71
Detach_buffer() (*mpi4py.MPI.Session method*), 152
Difference() (*mpi4py.MPI.Group class method*), 122
dim (*mpi4py.MPI.Cartcomm attribute*), 62
dims (*mpi4py.MPI.Cartcomm attribute*), 62
dims (*mpi4py.MPI.Graphcomm attribute*), 119
Disconnect() (*mpi4py.MPI.Comm method*), 71
DISP_CUR (*in module mpi4py.MPI*), 212
Displ (*in module mpi4py.typing*), 36
DISPLACEMENT_CURRENT (*in module mpi4py.MPI*), 212
DIST_GRAPH (*in module mpi4py.MPI*), 207
Distgraphcomm (*class in mpi4py.MPI*), 104
DISTRIBUTE_BLOCK (*in module mpi4py.MPI*), 204
DISTRIBUTE_CYCLIC (*in module mpi4py.MPI*), 204
DISTRIBUTE_DFLT_DARG (*in module mpi4py.MPI*), 204
DISTRIBUTE_NONE (*in module mpi4py.MPI*), 203
DOUBLE (*in module mpi4py.MPI*), 216
DOUBLE_COMPLEX (*in module mpi4py.MPI*), 220
DOUBLE_INT (*in module mpi4py.MPI*), 219
DOUBLE_PRECISION (*in module mpi4py.MPI*), 220
dumps() (*mpi4py.MPI.Pickle method*), 141
dumps_oob() (*mpi4py.MPI.Pickle method*), 141
Dup() (*mpi4py.MPI.Comm method*), 71
Dup() (*mpi4py.MPI.Datatype method*), 98
Dup() (*mpi4py.MPI.Group method*), 122
Dup() (*mpi4py.MPI.Info method*), 126
Dup_with_info() (*mpi4py.MPI.Comm method*), 71

E

edges (*mpi4py.MPI.Graphcomm attribute*), 119
envelope (*mpi4py.MPI.Datatype attribute*), 103
environment variable
 CC, 232
 LD_LIBRARY_PATH, 239
 MPI4PY_BUILD_BACKEND, 231, 232
 MPI4PY_BUILD_CONFIGURE, 231, 232
 MPI4PY_BUILD_MPICC, 231, 232
 MPI4PY_BUILD_MPICFG, 231, 232
 MPI4PY_BUILD_MPILD, 231, 232
 MPI4PY_FUTURES_BACKOFF, 40, 41
 MPI4PY_FUTURES_MAX_WORKERS, 39, 41, 46
 MPI4PY_FUTURES_USE_PKLS, 40, 41
 MPI4PY_PICKLE_PROTOCOL, 12, 25
 MPI4PY_PICKLE_THRESHOLD, 25
 MPI4PY_RC_ERRORS, 23, 25
 MPI4PY_RC_FAST_REDUCE, 22, 24
 MPI4PY_RC_FINALIZE, 22, 23
 MPI4PY_RC_INITIALIZE, 21, 23
 MPI4PY_RC_IRecv_BUFSZ, 22, 25
 MPI4PY_RC_RECV_MPROBE, 22, 24
 MPI4PY_RC_THREAD_LEVEL, 22, 24
 MPI4PY_RC_THREADS, 21, 24
 MPICC, 232, 236
 MPICFG, 232, 236
 MPICH_USE_SHLIB, 239
 MPIEXEC_UNIVERSE_SIZE, 46
 MPIILD, 232, 236
 PATH, 231, 236, 238
ERR_ACCESS (*in module mpi4py.MPI*), 201
ERR_AMODE (*in module mpi4py.MPI*), 201
ERR_ARG (*in module mpi4py.MPI*), 197
ERR_ASSERT (*in module mpi4py.MPI*), 199
ERR_BAD_FILE (*in module mpi4py.MPI*), 200
ERR_BASE (*in module mpi4py.MPI*), 199
ERR_BUFFER (*in module mpi4py.MPI*), 196
ERR_COMM (*in module mpi4py.MPI*), 196
ERR_CONVERSION (*in module mpi4py.MPI*), 202
ERR_COUNT (*in module mpi4py.MPI*), 196
ERR_DIMS (*in module mpi4py.MPI*), 197
ERR_DISP (*in module mpi4py.MPI*), 199
ERR_DUP_DATAREP (*in module mpi4py.MPI*), 202
ERR_ERRHANDLER (*in module mpi4py.MPI*), 195
ERR_FILE (*in module mpi4py.MPI*), 196
ERR_FILE_EXISTS (*in module mpi4py.MPI*), 201
ERR_FILE_IN_USE (*in module mpi4py.MPI*), 201
ERR_GROUP (*in module mpi4py.MPI*), 195
ERR_IN_STATUS (*in module mpi4py.MPI*), 197
ERR_INFO (*in module mpi4py.MPI*), 195
ERR_INFO_KEY (*in module mpi4py.MPI*), 198
ERR_INFO_NOKEY (*in module mpi4py.MPI*), 198
ERR_INFO_VALUE (*in module mpi4py.MPI*), 198
ERR_INTERN (*in module mpi4py.MPI*), 198

ERR_IO (*in module mpi4py.MPI*), 202
ERR_KEYVAL (*in module mpi4py.MPI*), 198
ERR_LASTCODE (*in module mpi4py.MPI*), 195
ERR_LOCKTYPE (*in module mpi4py.MPI*), 199
ERR_NAME (*in module mpi4py.MPI*), 199
ERR_NO_MEM (*in module mpi4py.MPI*), 198
ERR_NO_SPACE (*in module mpi4py.MPI*), 201
ERR_NO_SUCH_FILE (*in module mpi4py.MPI*), 200
ERR_NOT_SAME (*in module mpi4py.MPI*), 201
ERR_OP (*in module mpi4py.MPI*), 195
ERR_OTHER (*in module mpi4py.MPI*), 197
ERR_PENDING (*in module mpi4py.MPI*), 197
ERR_PORT (*in module mpi4py.MPI*), 198
ERR_PROC_ABORTED (*in module mpi4py.MPI*), 199
ERR_PROC_FAILED (*in module mpi4py.MPI*), 202
ERR_PROC_FAILED_PENDING (*in module mpi4py.MPI*), 203
ERR_QUOTA (*in module mpi4py.MPI*), 201
ERR_RANK (*in module mpi4py.MPI*), 196
ERR_READ_ONLY (*in module mpi4py.MPI*), 201
ERR_REQUEST (*in module mpi4py.MPI*), 195
ERR_REVOKED (*in module mpi4py.MPI*), 202
ERR_RMA_ATTACH (*in module mpi4py.MPI*), 200
ERR_RMA_CONFLICT (*in module mpi4py.MPI*), 200
ERR_RMA_FLAVOR (*in module mpi4py.MPI*), 200
ERR_RMA_RANGE (*in module mpi4py.MPI*), 200
ERR_RMA_SHARED (*in module mpi4py.MPI*), 200
ERR_RMA_SYNC (*in module mpi4py.MPI*), 200
ERR_ROOT (*in module mpi4py.MPI*), 196
ERR_SERVICE (*in module mpi4py.MPI*), 199
ERR_SESSION (*in module mpi4py.MPI*), 195
ERR_SIZE (*in module mpi4py.MPI*), 199
ERR_SPAWN (*in module mpi4py.MPI*), 198
ERR_TAG (*in module mpi4py.MPI*), 196
ERR_TOPOLOGY (*in module mpi4py.MPI*), 197
ERR_TRUNCATE (*in module mpi4py.MPI*), 197
ERR_TYPE (*in module mpi4py.MPI*), 195
ERR_UNKNOWN (*in module mpi4py.MPI*), 197
ERR_UNSUPPORTED_DATAREP (*in module mpi4py.MPI*), 202
ERR_UNSUPPORTED_OPERATION (*in module mpi4py.MPI*), 202
ERR_VALUE_TOO_LARGE (*in module mpi4py.MPI*), 202
ERR_WIN (*in module mpi4py.MPI*), 196
Errhandler (*class in mpi4py.MPI*), 104
ERRHANDLER_NULL (*in module mpi4py.MPI*), 229
error (*mpi4py.MPI.Status attribute*), 157
error_class (*mpi4py.MPI.Exception attribute*), 176
error_code (*mpi4py.MPI.Exception attribute*), 176
error_string (*mpi4py.MPI.Exception attribute*), 176
errors (*mpi4py.mpi4py.rc attribute*), 22
ERRORS_ABORT (*in module mpi4py.MPI*), 229
ERRORS_ARE_FATAL (*in module mpi4py.MPI*), 229
ERRORS_RETURN (*in module mpi4py.MPI*), 229

`Exception`, 175
`Excl()` (*mpi4py.MPI.Group method*), 122
`Exscan()` (*mpi4py.MPI.Intracomm method*), 133
`exscan()` (*mpi4py.MPI.Intracomm method*), 136
`Exscan_init()` (*mpi4py.MPI.Intracomm method*), 134
`extent` (*mpi4py.MPI.Datatype attribute*), 103

F

`f2py()` (*mpi4py.MPI.Comm class method*), 89
`f2py()` (*mpi4py.MPI.Datatype class method*), 102
`f2py()` (*mpi4py.MPI.Errhandler class method*), 105
`f2py()` (*mpi4py.MPI.File class method*), 117
`f2py()` (*mpi4py.MPI.Group class method*), 124
`f2py()` (*mpi4py.MPI.Info class method*), 127
`f2py()` (*mpi4py.MPI.Message class method*), 137
`f2py()` (*mpi4py.MPI.Op class method*), 140
`f2py()` (*mpi4py.MPI.Request class method*), 148
`f2py()` (*mpi4py.MPI.Session class method*), 154
`f2py()` (*mpi4py.MPI.Status class method*), 157
`f2py()` (*mpi4py.MPI.Win class method*), 172
`F_BOOL` (*in module mpi4py.MPI*), 224
`F_COMPLEX` (*in module mpi4py.MPI*), 224
`F_DOUBLE` (*in module mpi4py.MPI*), 224
`F_DOUBLE_COMPLEX` (*in module mpi4py.MPI*), 224
`F_ERROR` (*in module mpi4py.MPI*), 206
`F_FLOAT` (*in module mpi4py.MPI*), 224
`F_FLOAT_COMPLEX` (*in module mpi4py.MPI*), 224
`F_INT` (*in module mpi4py.MPI*), 224
`F_SOURCE` (*in module mpi4py.MPI*), 206
`F_STATUS_SIZE` (*in module mpi4py.MPI*), 206
`F_TAG` (*in module mpi4py.MPI*), 206
`fast_reduce` (*mpi4py.mpi4py.rc attribute*), 22
`Fence()` (*mpi4py.MPI.Win method*), 166
`Fetch_and_op()` (*mpi4py.MPI.Win method*), 167
`File` (*class in mpi4py.MPI*), 106
`FILE_NULL` (*in module mpi4py.MPI*), 230
`finalize` (*mpi4py.mpi4py.rc attribute*), 22
`Finalize()` (*in module mpi4py.MPI*), 180
`Finalize()` (*mpi4py.MPI.Session method*), 152
`flavor` (*mpi4py.MPI.Win attribute*), 172
`FLOAT` (*in module mpi4py.MPI*), 216
`FLOAT_INT` (*in module mpi4py.MPI*), 219
`Flush()` (*mpi4py.MPI.Win method*), 167
`Flush_all()` (*mpi4py.MPI.Win method*), 167
`Flush_buffer()` (*in module mpi4py.MPI*), 180
`Flush_buffer()` (*mpi4py.MPI.Comm method*), 71
`Flush_buffer()` (*mpi4py.MPI.Session method*), 152
`Flush_local()` (*mpi4py.MPI.Win method*), 167
`Flush_local_all()` (*mpi4py.MPI.Win method*), 167
`format` (*mpi4py.MPI.buffer attribute*), 175
`Free()` (*mpi4py.MPI.Comm method*), 72
`Free()` (*mpi4py.MPI.Datatype method*), 98
`Free()` (*mpi4py.MPI.Errhandler method*), 105
`Free()` (*mpi4py.MPI.Group method*), 122

`Free()` (*mpi4py.MPI.Info method*), 126
`Free()` (*mpi4py.MPI.Op method*), 140
`Free()` (*mpi4py.MPI.Request method*), 146
`Free()` (*mpi4py.MPI.Win method*), 167
`Free()` (*mpi4py.util.pkl5.Request method*), 49
`Free_keyval()` (*mpi4py.MPI.Comm class method*), 72
`Free_keyval()` (*mpi4py.MPI.Datatype class method*), 99
`Free_keyval()` (*mpi4py.MPI.Win class method*), 167
`Free_mem()` (*in module mpi4py.MPI*), 180
`from_numpy_dtype()` (*in module mpi4py.util.dplib*), 48
`fromaddress()` (*mpi4py.MPI.buffer static method*), 174
`frombuffer()` (*mpi4py.MPI.buffer static method*), 174
`fromcode()` (*mpi4py.MPI.Datatype class method*), 102
`fromhandle()` (*mpi4py.MPI.Comm class method*), 89
`fromhandle()` (*mpi4py.MPI.Datatype class method*), 102
`fromhandle()` (*mpi4py.MPI.Errhandler class method*), 105
`fromhandle()` (*mpi4py.MPI.File class method*), 117
`fromhandle()` (*mpi4py.MPI.Group class method*), 124
`fromhandle()` (*mpi4py.MPI.Info class method*), 127
`fromhandle()` (*mpi4py.MPI.Message class method*), 138
`fromhandle()` (*mpi4py.MPI.Op class method*), 140
`fromhandle()` (*mpi4py.MPI.Request class method*), 148
`fromhandle()` (*mpi4py.MPI.Session class method*), 154
`fromhandle()` (*mpi4py.MPI.Win class method*), 172

G

`Gather()` (*mpi4py.MPI.Comm method*), 72
`gather()` (*mpi4py.MPI.Comm method*), 89
`gather()` (*mpi4py.util.pkl5.Comm method*), 53
`Gather_init()` (*mpi4py.MPI.Comm method*), 72
`Gatherv()` (*mpi4py.MPI.Comm method*), 72
`Gatherv_init()` (*mpi4py.MPI.Comm method*), 72
`Get()` (*mpi4py.MPI.Info method*), 126
`get()` (*mpi4py.MPI.Info method*), 127
`Get()` (*mpi4py.MPI.Win method*), 168
`get()` (*mpi4py.util.pool.AsyncResult method*), 57
`Get_accumulate()` (*mpi4py.MPI.Win method*), 168
`Get_address()` (*in module mpi4py.MPI*), 181
`Get_amode()` (*mpi4py.MPI.File method*), 108
`Get_atomicity()` (*mpi4py.MPI.File method*), 108
`Get_attr()` (*mpi4py.MPI.Comm method*), 73
`Get_attr()` (*mpi4py.MPI.Datatype method*), 99
`Get_attr()` (*mpi4py.MPI.Win method*), 168
`Get_byte_offset()` (*mpi4py.MPI.File method*), 108
`Get_cart_rank()` (*mpi4py.MPI.Cartcomm method*), 61
`get_comm_workers()` (*in module mpi4py.futures*), 44
`get_config()` (*in module mpi4py*), 26
`Get_contents()` (*mpi4py.MPI.Datatype method*), 99
`Get_coords()` (*mpi4py.MPI.Cartcomm method*), 61
`Get_count()` (*mpi4py.MPI.Status method*), 155

Get_dim() (*mpi4py.MPI.Cartcomm method*), 61
Get_dims() (*mpi4py.MPI.Graphcomm method*), 119
Get_dist_neighbors() (*mpi4py.MPI.Distgraphcomm method*), 104
Get_dist_neighbors_count()
 (*mpi4py.MPI.Distgraphcomm method*), 104
Get_elements() (*mpi4py.MPI.Status method*), 155
Get_envelope() (*mpi4py.MPI.Datatype method*), 99
Get_errhandler() (*mpi4py.MPI.Comm method*), 73
Get_errhandler() (*mpi4py.MPI.File method*), 109
Get_errhandler() (*mpi4py.MPI.Session method*), 152
Get_errhandler() (*mpi4py.MPI.Win method*), 168
Get_error() (*mpi4py.MPI.Status method*), 155
Get_error_class() (*in module mpi4py.MPI*), 181
Get_error_class() (*mpi4py.MPI.Exception method*), 176
Get_error_code() (*mpi4py.MPI.Exception method*), 176
Get_error_string() (*in module mpi4py.MPI*), 181
Get_error_string() (*mpi4py.MPI.Exception method*), 176
Get_extent() (*mpi4py.MPI.Datatype method*), 99
Get_failed() (*mpi4py.MPI.Comm method*), 73
Get_group() (*mpi4py.MPI.Comm method*), 73
Get_group() (*mpi4py.MPI.File method*), 109
Get_group() (*mpi4py.MPI.Win method*), 168
Get_hw_resource_info() (*in module mpi4py.MPI*), 181
get_include() (*in module mpi4py*), 26
Get_info() (*mpi4py.MPI.Comm method*), 73
Get_info() (*mpi4py.MPI.File method*), 109
Get_info() (*mpi4py.MPI.Session method*), 153
Get_info() (*mpi4py.MPI.Win method*), 168
Get_library_version() (*in module mpi4py.MPI*), 181
Get_name() (*mpi4py.MPI.Comm method*), 73
Get_name() (*mpi4py.MPI.Datatype method*), 99
Get_name() (*mpi4py.MPI.Win method*), 169
Get_neighbors() (*mpi4py.MPI.Graphcomm method*), 119
Get_neighbors_count() (*mpi4py.MPI.Graphcomm method*), 119
Get_nkeys() (*mpi4py.MPI.Info method*), 126
Get_nth_pset() (*mpi4py.MPI.Session method*), 153
Get_nthkey() (*mpi4py.MPI.Info method*), 126
Get_num_pssets() (*mpi4py.MPI.Session method*), 153
Get_parent() (*mpi4py.MPI.Comm class method*), 73
Get_position() (*mpi4py.MPI.File method*), 109
Get_position_shared() (*mpi4py.MPI.File method*), 109
Get_processor_name() (*in module mpi4py.MPI*), 182
Get_pset_info() (*mpi4py.MPI.Session method*), 153
Get_rank() (*mpi4py.MPI.Comm method*), 73
Get_rank() (*mpi4py.MPI.Group method*), 123
Get_remote_group() (*mpi4py.MPI.Intercomm method*), 129
Get_remote_size() (*mpi4py.MPI.Intercomm method*), 130
Get_size() (*mpi4py.MPI.Comm method*), 74
Get_size() (*mpi4py.MPI.Datatype method*), 99
Get_size() (*mpi4py.MPI.File method*), 109
Get_size() (*mpi4py.MPI.Group method*), 123
Get_source() (*mpi4py.MPI.Status method*), 155
Get_status() (*mpi4py.MPI.Request method*), 146
get_status() (*mpi4py.MPI.Request method*), 148
get_status() (*mpi4py.util.pk15.Request method*), 49
Get_status_all() (*mpi4py.MPI.Request class method*), 146
get_status_all() (*mpi4py.MPI.Request class method*), 148
Get_status_any() (*mpi4py.MPI.Request class method*), 146
get_status_any() (*mpi4py.MPI.Request class method*), 149
Get_status_some() (*mpi4py.MPI.Request class method*), 146
get_status_some() (*mpi4py.MPI.Request class method*), 149
Get_tag() (*mpi4py.MPI.Status method*), 156
Get_topo() (*mpi4py.MPI.Cartcomm method*), 61
Get_topo() (*mpi4py.MPI.Graphcomm method*), 119
Get_topology() (*mpi4py.MPI.Comm method*), 74
Get_true_extent() (*mpi4py.MPI.Datatype method*), 99
Get_type_extent() (*mpi4py.MPI.File method*), 109
Get_value_index() (*mpi4py.MPI.Datatype class method*), 99
get_vendor() (*in module mpi4py.MPI*), 186
Get_version() (*in module mpi4py.MPI*), 182
Get_view() (*mpi4py.MPI.File method*), 110
GRAPH (*in module mpi4py.MPI*), 207
Graph_map() (*mpi4py.MPI.Intracomm method*), 134
Graphcomm (*class in mpi4py.MPI*), 118
Grequest (*class in mpi4py.MPI*), 120
Group (*class in mpi4py.MPI*), 121
group (*mpi4py.MPI.Comm attribute*), 92
group (*mpi4py.MPI.File attribute*), 118
group (*mpi4py.MPI.Win attribute*), 172
GROUP_EMPTY (*in module mpi4py.MPI*), 229
GROUP_NULL (*in module mpi4py.MPI*), 229

H

handle (*mpi4py.MPI.Comm attribute*), 92
handle (*mpi4py.MPI.Datatype attribute*), 103
handle (*mpi4py.MPI.Errhandler attribute*), 106
handle (*mpi4py.MPI.File attribute*), 118
handle (*mpi4py.MPI.Group attribute*), 124
handle (*mpi4py.MPI.Info attribute*), 128

handle (*mpi4py.MPI.Message attribute*), 139
handle (*mpi4py.MPI.Op attribute*), 141
handle (*mpi4py.MPI.Request attribute*), 151
handle (*mpi4py.MPI.Session attribute*), 154
handle (*mpi4py.MPI.Win attribute*), 172
HOST (*in module mpi4py.MPI*), 193

|

Iagree () (*mpi4py.MPI.Comm method*), 74
Iallgather () (*mpi4py.MPI.Comm method*), 74
Iallgatherv () (*mpi4py.MPI.Comm method*), 74
Iallreduce () (*mpi4py.MPI.Comm method*), 74
Ialltoall () (*mpi4py.MPI.Comm method*), 74
Ialltoallv () (*mpi4py.MPI.Comm method*), 75
Ialltoallw () (*mpi4py.MPI.Comm method*), 75
Ibarrier () (*mpi4py.MPI.Comm method*), 75
Ibcast () (*mpi4py.MPI.Comm method*), 75
Ibsend () (*mpi4py.MPI.Comm method*), 75
ibsend () (*mpi4py.MPI.Comm method*), 89
ibsend () (*mpi4py.util.pkl5.Comm method*), 51
IDENT (*in module mpi4py.MPI*), 207
Idup () (*mpi4py.MPI.Comm method*), 76
Idup_with_info () (*mpi4py.MPI.Comm method*), 76
Iexscan () (*mpi4py.MPI.Intracomm method*), 134
Iflush_buffer () (*in module mpi4py.MPI*), 182
Iflush_buffer () (*mpi4py.MPI.Comm method*), 76
Iflush_buffer () (*mpi4py.MPI.Session method*), 153
Igather () (*mpi4py.MPI.Comm method*), 76
Igatherv () (*mpi4py.MPI.Comm method*), 76
imap () (*mpi4py.util.pool.Pool method*), 56
imap_unordered () (*mpi4py.util.pool.Pool method*), 56
Improbe () (*mpi4py.MPI.Comm method*), 76
improbe () (*mpi4py.MPI.Comm method*), 89
improbe () (*mpi4py.util.pkl5.Comm method*), 53
IN_PLACE (*in module mpi4py.MPI*), 193
Incl () (*mpi4py.MPI.Group method*), 123
indegree (*mpi4py.MPI.Topocomm attribute*), 162
index (*mpi4py.MPI.Graphcomm attribute*), 119
inedges (*mpi4py.MPI.Topocomm attribute*), 162
Inneighbor_allgather () (*mpi4py.MPI.Topocomm method*), 159
Inneighbor_allgatherv () (*mpi4py.MPI.Topocomm method*), 159
Inneighbor_alltoall () (*mpi4py.MPI.Topocomm method*), 159
Inneighbor_alltoallv () (*mpi4py.MPI.Topocomm method*), 159
Inneighbor_alltoallw () (*mpi4py.MPI.Topocomm method*), 159
Info (*class in mpi4py.MPI*), 125
info (*mpi4py.MPI.Comm attribute*), 92
info (*mpi4py.MPI.File attribute*), 118
info (*mpi4py.MPI.Win attribute*), 173
INFO_ENV (*in module mpi4py.MPI*), 229
INFO_NULL (*in module mpi4py.MPI*), 229
Init () (*in module mpi4py.MPI*), 182
Init () (*mpi4py.MPI.Session class method*), 153
Init_thread () (*in module mpi4py.MPI*), 182
initialize (*mpi4py.mpi4py.rc attribute*), 21
inoutedges (*mpi4py.MPI.Topocomm attribute*), 162
InPlace (*in module mpi4py.typing*), 36
InPlaceType (*class in mpi4py.MPI*), 124
INT (*in module mpi4py.MPI*), 215
INT16_T (*in module mpi4py.MPI*), 217
INT32_T (*in module mpi4py.MPI*), 217
INT64_T (*in module mpi4py.MPI*), 217
INT8_T (*in module mpi4py.MPI*), 217
INT_INT (*in module mpi4py.MPI*), 219
INTEGER (*in module mpi4py.MPI*), 220
INTEGER1 (*in module mpi4py.MPI*), 221
INTEGER16 (*in module mpi4py.MPI*), 221
INTEGER2 (*in module mpi4py.MPI*), 221
INTEGER4 (*in module mpi4py.MPI*), 221
INTEGER8 (*in module mpi4py.MPI*), 221
Intercomm (*class in mpi4py.MPI*), 129
Intercomm (*class in mpi4py.util.pkl5*), 54
Intersection () (*mpi4py.MPI.Group class method*), 123
Intracomm (*class in mpi4py.MPI*), 130
Intracomm (*class in mpi4py.util.pkl5*), 54
I0 (*in module mpi4py.MPI*), 193
Iprobe () (*mpi4py.MPI.Comm method*), 77
iprobe () (*mpi4py.MPI.Comm method*), 89
Iprobe () (*mpi4py.MPI.Message class method*), 137
iprobe () (*mpi4py.MPI.Message class method*), 138
iprobe () (*mpi4py.util.pkl5.Message class method*), 50
Iread () (*mpi4py.MPI.File method*), 110
Iread_all () (*mpi4py.MPI.File method*), 110
Iread_at () (*mpi4py.MPI.File method*), 110
Iread_at_all () (*mpi4py.MPI.File method*), 110
Iread_shared () (*mpi4py.MPI.File method*), 110
Irecv () (*mpi4py.MPI.Comm method*), 77
irecv () (*mpi4py.MPI.Comm method*), 90
Irecv () (*mpi4py.MPI.Message method*), 137
irecv () (*mpi4py.MPI.Message method*), 138
irecv () (*mpi4py.util.pkl5.Comm method*), 52
irecv () (*mpi4py.util.pkl5.Message method*), 50
irecv_bufsz (*mpi4py.mpi4py.rc attribute*), 22
Reduce () (*mpi4py.MPI.Comm method*), 77
Reduce_scatter () (*mpi4py.MPI.Comm method*), 77
Reduce_scatter_block () (*mpi4py.MPI.Comm method*), 77
Isend () (*mpi4py.MPI.Comm method*), 78
Is_cancelled () (*mpi4py.MPI.Status method*), 156
is_commutative (*mpi4py.MPI.Op attribute*), 141
Is_commutative () (*mpi4py.MPI.Op method*), 140
Is_finalized () (*in module mpi4py.MPI*), 182
Is_initialized () (*in module mpi4py.MPI*), 183

is_inter (*mpi4py.MPI.Comm attribute*), 92
Is_inter() (*mpi4py.MPI.Comm method*), 78
is_intra (*mpi4py.MPI.Comm attribute*), 93
Is_intra() (*mpi4py.MPI.Comm method*), 78
is_named (*mpi4py.MPI.Datatype attribute*), 103
is_predefined (*mpi4py.MPI.Datatype attribute*), 103
is_predefined (*mpi4py.MPI.Op attribute*), 141
Is_revoked() (*mpi4py.MPI.Comm method*), 78
Is_thread_main() (*in module mpi4py.MPI*), 183
is_topo (*mpi4py.MPI.Comm attribute*), 93
Iscan() (*mpi4py.MPI.Intracomm method*), 134
Iscatter() (*mpi4py.MPI.Comm method*), 78
Iscatterv() (*mpi4py.MPI.Comm method*), 78
Isend() (*mpi4py.MPI.Comm method*), 79
isend() (*mpi4py.MPI.Comm method*), 90
isend() (*mpi4py.util.pkl5.Comm method*), 51
Isendrecv() (*mpi4py.MPI.Comm method*), 79
Isendrecv_replace() (*mpi4py.MPI.Comm method*), 79
Ishrink() (*mpi4py.MPI.Comm method*), 80
Issend() (*mpi4py.MPI.Comm method*), 80
issend() (*mpi4py.MPI.Comm method*), 90
issend() (*mpi4py.util.pkl5.Comm method*), 51
istarmap() (*mpi4py.util.pool.Pool method*), 56
istarmap_unordered() (*mpi4py.util.pool.Pool method*), 56
items() (*mpi4py.MPI.Info method*), 127
itemsize (*mpi4py.MPI.buffer attribute*), 175
Iwrite() (*mpi4py.MPI.File method*), 111
Iwrite_all() (*mpi4py.MPI.File method*), 111
Iwrite_at() (*mpi4py.MPI.File method*), 111
Iwrite_at_all() (*mpi4py.MPI.File method*), 111
Iwrite_shared() (*mpi4py.MPI.File method*), 111

J

Join() (*mpi4py.MPI.Comm class method*), 80
join() (*mpi4py.util.pool.Pool method*), 56

K

keys() (*mpi4py.MPI.Info method*), 127
KEYVAL_INVALID (*in module mpi4py.MPI*), 193

L

LAND (*in module mpi4py.MPI*), 226
LASTUSEDCODE (*in module mpi4py.MPI*), 194
lb (*mpi4py.MPI.Datatype attribute*), 103
LD_LIBRARY_PATH, 239
loads() (*mpi4py.MPI.Pickle method*), 142
loads_oob() (*mpi4py.MPI.Pickle method*), 142
Lock() (*mpi4py.MPI.Win method*), 169
Lock_all() (*mpi4py.MPI.Win method*), 169
LOCK_EXCLUSIVE (*in module mpi4py.MPI*), 210
LOCK_SHARED (*in module mpi4py.MPI*), 210

LOGICAL (*in module mpi4py.MPI*), 220
LOGICAL1 (*in module mpi4py.MPI*), 220
LOGICAL2 (*in module mpi4py.MPI*), 220
LOGICAL4 (*in module mpi4py.MPI*), 221
LOGICAL8 (*in module mpi4py.MPI*), 221
LONG (*in module mpi4py.MPI*), 215
LONG_DOUBLE (*in module mpi4py.MPI*), 216
LONG_DOUBLE_INT (*in module mpi4py.MPI*), 219
LONG_INT (*in module mpi4py.MPI*), 219
LONG_LONG (*in module mpi4py.MPI*), 215
Lookup_name() (*in module mpi4py.MPI*), 183
LOR (*in module mpi4py.MPI*), 227
LXOR (*in module mpi4py.MPI*), 227

M
map() (*mpi4py.futures.MPIPoolExecutor method*), 40
map() (*mpi4py.util.pool.Pool method*), 56
map_async() (*mpi4py.util.pool.Pool method*), 56
MapResult (*class in mpi4py.util.pool*), 57
Match_size() (*mpi4py.MPI.Datatype class method*), 100
MAX (*in module mpi4py.MPI*), 225
MAX_DATAREP_STRING (*in module mpi4py.MPI*), 213
MAX_ERROR_STRING (*in module mpi4py.MPI*), 213
MAX_INFO_KEY (*in module mpi4py.MPI*), 213
MAX_INFO_VAL (*in module mpi4py.MPI*), 213
MAX_LIBRARY_VERSION_STRING (*in module mpi4py.MPI*), 213
MAX_OBJECT_NAME (*in module mpi4py.MPI*), 213
MAX_PORT_NAME (*in module mpi4py.MPI*), 213
MAX_PROCESSOR_NAME (*in module mpi4py.MPI*), 213
MAX_PSET_NAME_LEN (*in module mpi4py.MPI*), 214
MAX_STRINGTAG_LEN (*in module mpi4py.MPI*), 214
MAXLOC (*in module mpi4py.MPI*), 228
memory (*in module mpi4py.MPI*), 175
Merge() (*mpi4py.MPI.Intercomm method*), 130
Message (*class in mpi4py.MPI*), 136
Message (*class in mpi4py.util.pkl5*), 50
MESSAGE_NO_PROC (*in module mpi4py.MPI*), 225
MESSAGE_NULL (*in module mpi4py.MPI*), 225
MIN (*in module mpi4py.MPI*), 225
MINLOC (*in module mpi4py.MPI*), 228
MODE_APPEND (*in module mpi4py.MPI*), 211
MODE_CREATE (*in module mpi4py.MPI*), 210
MODE_DELETE_ON_CLOSE (*in module mpi4py.MPI*), 211
MODE_EXCL (*in module mpi4py.MPI*), 211
MODE_NOCHECK (*in module mpi4py.MPI*), 209
MODE_NOPRECEDE (*in module mpi4py.MPI*), 210
MODE_NOPUT (*in module mpi4py.MPI*), 209
MODE_NOSTORE (*in module mpi4py.MPI*), 209
MODE_NOSUCCEED (*in module mpi4py.MPI*), 210
MODE_RDONLY (*in module mpi4py.MPI*), 210
MODE_RDWR (*in module mpi4py.MPI*), 210
MODE_SEQUENTIAL (*in module mpi4py.MPI*), 211

MODE_UNIQUE_OPEN (*in module mpi4py.MPI*), 211
MODE_WRONLY (*in module mpi4py.MPI*), 210
model (*mpi4py.MPI.Win attribute*), 173
module

- mpi4py**, 20
- mpi4py.bench**, 59
- mpi4py.futures**, 38
- mpi4py.MPI**, 59
- mpi4py.run**, 57
- mpi4py.typing**, 35
- mpi4py.util**, 48
- mpi4py.util.dplib**, 48
- mpi4py.util.pkl5**, 48
- mpi4py.util.pool**, 55

- mpi4py**
 - module**, 20
- mpi4py.bench**
 - module**, 59
- mpi4py.futures**
 - module**, 38
- mpi4py.MPI**
 - module**, 59

- mpi4py.rc** (*in module mpi4py*), 21
- mpi4py.run**
 - module**, 57
- mpi4py.typing**
 - module**, 35
- mpi4py.util**
 - module**, 48
- mpi4py.util.dplib**
 - module**, 48
- mpi4py.util.pkl5**
 - module**, 48
- mpi4py.util.pool**
 - module**, 55

- MPI4PY_BUILD_BACKEND**, 231, 232
- MPI4PY_BUILD_CONFIGURE**, 232
- MPI4PY_BUILD_MPICC**, 231, 232
- MPI4PY_BUILD_MPICFG**, 231, 232
- MPI4PY_BUILD_MPILD**, 231, 232
- MPI4PY_FUTURES_BACKOFF**, 40, 41
- MPI4PY_FUTURES_MAX_WORKERS**, 39, 41, 46
- MPI4PY_FUTURES_USE_PKL5**, 40, 41
- MPI4PY_PICKLE_PROTOCOL**, 12
- MPI4PY_RC_ERRORS**, 23
- MPI4PY_RC_FAST_REDUCE**, 22
- MPI4PY_RC_FINALIZE**, 22
- MPI4PY_RC_INITIALIZE**, 21
- MPI4PY_RC_IRecv_BUFSZ**, 22
- MPI4PY_RC_RECV_MPROBE**, 22
- MPI4PY_RC_THREAD_LEVEL**, 22
- MPI4PY_RC_THREADS**, 21
- MPICC**, 236
- MPICFG**, 236

MPICH_USE_SHLIB, 239
MPICommExecutor (*class in mpi4py.futures*), 42
MPIEXEC_UNIVERSE_SIZE, 46
MPILD, 236
MPIPoolExecutor (*class in mpi4py.futures*), 39
Mprobe() (*mpi4py.MPI.Comm method*), 80
mprobe() (*mpi4py.MPI.Comm method*), 90
mprobe() (*mpi4py.util.pkl5.Comm method*), 52

N

name (*mpi4py.MPI.Comm attribute*), 93
name (*mpi4py.MPI.Datatype attribute*), 103
name (*mpi4py.MPI.Win attribute*), 173
nbytes (*mpi4py.MPI.buffer attribute*), 175
ndim (*mpi4py.MPI.Cartcomm attribute*), 62
nedges (*mpi4py.MPI.Graphcomm attribute*), 119
Neighbor_allgather() (*mpi4py.MPI.Topocomm method*), 159
neighbor_allgather() (*mpi4py.MPI.Topocomm method*), 161
Neighbor_allgather_init()
(mpi4py.MPI.Topocomm method), 160
Neighbor_allgatherv() (*mpi4py.MPI.Topocomm method*), 160
Neighbor_allgatherv_init()
(mpi4py.MPI.Topocomm method), 160
Neighbor_alltoall() (*mpi4py.MPI.Topocomm method*), 160
neighbor_alltoall() (*mpi4py.MPI.Topocomm method*), 162
Neighbor_alltoall_init() (*mpi4py.MPI.Topocomm method*), 160
Neighbor_alltoallv() (*mpi4py.MPI.Topocomm method*), 161
Neighbor_alltoallv_init()
(mpi4py.MPI.Topocomm method), 161
Neighbor_alltoallw() (*mpi4py.MPI.Topocomm method*), 161
Neighbor_alltoallw_init()
(mpi4py.MPI.Topocomm method), 161
neighbors (*mpi4py.MPI.Graphcomm attribute*), 120
nneighbors (*mpi4py.MPI.Graphcomm attribute*), 120
nnodes (*mpi4py.MPI.Graphcomm attribute*), 120
NO_OP (*in module mpi4py.MPI*), 228
num_workers (*mpi4py.futures.MPIPoolExecutor attribute*), 41

O

obj (*mpi4py.MPI.buffer attribute*), 175
OFFSET (*in module mpi4py.MPI*), 214
Offset (*in module mpi4py.typing*), 36
Op (*class in mpi4py.MPI*), 139
OP_NULL (*in module mpi4py.MPI*), 225
Open() (*mpi4py.MPI.File class method*), 111

`Open_port()` (*in module mpi4py.MPI*), 183
`ORDER_C` (*in module mpi4py.MPI*), 203
`ORDER_F` (*in module mpi4py.MPI*), 203
`ORDER_FORTRAN` (*in module mpi4py.MPI*), 203
`outdegree` (*mpi4py.MPI.Topocomm attribute*), 162
`outedges` (*mpi4py.MPI.Topocomm attribute*), 162

P

`Pack()` (*mpi4py.MPI.Datatype method*), 100
`Pack_external()` (*mpi4py.MPI.Datatype method*), 100
`Pack_external_size()` (*mpi4py.MPI.Datatype method*), 100
`Pack_size()` (*mpi4py.MPI.Datatype method*), 101
`PACKED` (*in module mpi4py.MPI*), 214
`Parrived()` (*mpi4py.MPI.Prequest method*), 143
`PATH`, 231, 236, 238
`Pcontrol()` (*in module mpi4py.MPI*), 183
`periods` (*mpi4py.MPI.Cartcomm attribute*), 62
`Pickle` (*class in mpi4py.MPI*), 141
`pickle` (*in module mpi4py.MPI*), 230
`Pool` (*class in mpi4py.util.pool*), 55
`pop()` (*mpi4py.MPI.Info method*), 128
`popitem()` (*mpi4py.MPI.Info method*), 128
`Post()` (*mpi4py.MPI.Win method*), 169
`Pready()` (*mpi4py.MPI.Prequest method*), 143
`Pready_list()` (*mpi4py.MPI.Prequest method*), 143
`Pready_range()` (*mpi4py.MPI.Prequest method*), 143
`Preallocate()` (*mpi4py.MPI.File method*), 112
`Precv_init()` (*mpi4py.MPI.Comm method*), 80
`Prequest` (*class in mpi4py.MPI*), 142
`Probe()` (*mpi4py.MPI.Comm method*), 80
`probe()` (*mpi4py.MPI.Comm method*), 91
`Probe()` (*mpi4py.MPI.Message class method*), 137
`probe()` (*mpi4py.MPI.Message class method*), 138
`probe()` (*mpi4py.util.pkl5.Message class method*), 50
`PROC_NULL` (*in module mpi4py.MPI*), 192
`PROD` (*in module mpi4py.MPI*), 226
`profile()` (*in module mpi4py*), 26
`PROTOCOL` (*mpi4py.MPI.Pickle attribute*), 142
`Psend_init()` (*mpi4py.MPI.Comm method*), 81
`Publish_name()` (*in module mpi4py.MPI*), 184
`Put()` (*mpi4py.MPI.Win method*), 169
`py2f()` (*mpi4py.MPI.Comm method*), 91
`py2f()` (*mpi4py.MPI.Datatype method*), 102
`py2f()` (*mpi4py.MPI.Errhandler method*), 105
`py2f()` (*mpi4py.MPI.File method*), 118
`py2f()` (*mpi4py.MPI.Group method*), 124
`py2f()` (*mpi4py.MPI.Info method*), 128
`py2f()` (*mpi4py.MPI.Message method*), 138
`py2f()` (*mpi4py.MPI.Op method*), 140
`py2f()` (*mpi4py.MPI.Request method*), 149
`py2f()` (*mpi4py.MPI.Session method*), 154
`py2f()` (*mpi4py.MPI.Status method*), 157
`py2f()` (*mpi4py.MPI.Win method*), 172

Python Enhancement Proposals

PEP 574, 48

Q

`Query_thread()` (*in module mpi4py.MPI*), 184

R

`Raccumulate()` (*mpi4py.MPI.Win method*), 169
`Range_excl()` (*mpi4py.MPI.Group method*), 123
`Range_incl()` (*mpi4py.MPI.Group method*), 123
`rank` (*mpi4py.MPI.Comm attribute*), 93
`rank` (*mpi4py.MPI.Group attribute*), 124
`Read()` (*mpi4py.MPI.File method*), 112
`Read_all()` (*mpi4py.MPI.File method*), 112
`Read_all_begin()` (*mpi4py.MPI.File method*), 112
`Read_all_end()` (*mpi4py.MPI.File method*), 112
`Read_at()` (*mpi4py.MPI.File method*), 112
`Read_at_all()` (*mpi4py.MPI.File method*), 113
`Read_at_all_begin()` (*mpi4py.MPI.File method*), 113
`Read_at_all_end()` (*mpi4py.MPI.File method*), 113
`Read_ordered()` (*mpi4py.MPI.File method*), 113
`Read_ordered_begin()` (*mpi4py.MPI.File method*), 113
`Read_ordered_end()` (*mpi4py.MPI.File method*), 113
`Read_shared()` (*mpi4py.MPI.File method*), 114
`readonly` (*mpi4py.MPI.buffer attribute*), 175
`ready()` (*mpi4py.util.pool.AsyncResult method*), 57
`REAL` (*in module mpi4py.MPI*), 220
`REAL16` (*in module mpi4py.MPI*), 222
`REAL2` (*in module mpi4py.MPI*), 221
`REAL4` (*in module mpi4py.MPI*), 222
`REAL8` (*in module mpi4py.MPI*), 222
`Recv()` (*mpi4py.MPI.Comm method*), 81
`recv()` (*mpi4py.MPI.Comm method*), 91
`Recv()` (*mpi4py.MPI.Message method*), 137
`recv()` (*mpi4py.MPI.Message method*), 138
`recv()` (*mpi4py.util.pkl5.Comm method*), 51
`recv()` (*mpi4py.util.pkl5.Message method*), 50
`Recv_init()` (*mpi4py.MPI.Comm method*), 81
`recv_improbe` (*mpi4py.mpi4py.rc attribute*), 22
`Reduce()` (*mpi4py.MPI.Comm method*), 82
`reduce()` (*mpi4py.MPI.Comm method*), 91
`Reduce_init()` (*mpi4py.MPI.Comm method*), 82
`Reduce_local()` (*mpi4py.MPI.Op method*), 140
`Reduce_scatter()` (*mpi4py.MPI.Comm method*), 82
`Reduce_scatter_block()` (*mpi4py.MPI.Comm method*), 82
`Reduce_scatter_block_init()` (*mpi4py.MPI.Comm method*), 82
`Reduce_scatter_init()` (*mpi4py.MPI.Comm method*), 83
`Register_datarep()` (*in module mpi4py.MPI*), 184
`release()` (*mpi4py.MPI.buffer method*), 174
`remote_group` (*mpi4py.MPI.Intercomm attribute*), 130

remote_size (*mpi4py.MPI.Intercomm attribute*), 130
Remove_error_class() (*in module mpi4py.MPI*), 184
Remove_error_code() (*in module mpi4py.MPI*), 185
Remove_error_string() (*in module mpi4py.MPI*), 185
REPLACE (*in module mpi4py.MPI*), 228
Request (*class in mpi4py.MPI*), 144
Request (*class in mpi4py.util.pkl5*), 49
REQUEST_NULL (*in module mpi4py.MPI*), 225
Revoke() (*mpi4py.MPI.Comm method*), 83
Rget() (*mpi4py.MPI.Win method*), 170
Rget_accumulate() (*mpi4py.MPI.Win method*), 170
ROOT (*in module mpi4py.MPI*), 192
Rput() (*mpi4py.MPI.Win method*), 170
Rsend() (*mpi4py.MPI.Comm method*), 83
Rsend_init() (*mpi4py.MPI.Comm method*), 83

S

Scan() (*mpi4py.MPI.Intracomm method*), 134
scan() (*mpi4py.MPI.Intracomm method*), 136
Scan_init() (*mpi4py.MPI.Intracomm method*), 135
Scatter() (*mpi4py.MPI.Comm method*), 83
scatter() (*mpi4py.MPI.Comm method*), 91
scatter() (*mpi4py.util.pkl5.Comm method*), 53
Scatter_init() (*mpi4py.MPI.Comm method*), 84
Scatterv() (*mpi4py.MPI.Comm method*), 84
Scatterv_init() (*mpi4py.MPI.Comm method*), 84
Seek() (*mpi4py.MPI.File method*), 114
SEEK_CUR (*in module mpi4py.MPI*), 211
SEEK_END (*in module mpi4py.MPI*), 211
SEEK_SET (*in module mpi4py.MPI*), 211
Seek_shared() (*mpi4py.MPI.File method*), 114
Send() (*mpi4py.MPI.Comm method*), 84
send() (*mpi4py.MPI.Comm method*), 91
send() (*mpi4py.util.pkl5.Comm method*), 50
Send_init() (*mpi4py.MPI.Comm method*), 85
Sendrecv() (*mpi4py.MPI.Comm method*), 85
sendrecv() (*mpi4py.MPI.Comm method*), 92
sendrecv() (*mpi4py.util.pkl5.Comm method*), 52
Sendrecv_replace() (*mpi4py.MPI.Comm method*), 86
Session (*class in mpi4py.MPI*), 151
SESSION_NULL (*in module mpi4py.MPI*), 230
Set() (*mpi4py.MPI.Info method*), 127
Set_atomicity() (*mpi4py.MPI.File method*), 114
Set_attr() (*mpi4py.MPI.Comm method*), 86
Set_attr() (*mpi4py.MPI.Datatype method*), 101
Set_attr() (*mpi4py.MPI.Win method*), 170
Set_cancelled() (*mpi4py.MPI.Status method*), 156
Set_elements() (*mpi4py.MPI.Status method*), 156
Set_errhandler() (*mpi4py.MPI.Comm method*), 86
Set_errhandler() (*mpi4py.MPI.File method*), 114
Set_errhandler() (*mpi4py.MPI.Session method*), 153
Set_errhandler() (*mpi4py.MPI.Win method*), 170
Set_error() (*mpi4py.MPI.Status method*), 156
Set_info() (*mpi4py.MPI.Comm method*), 86

Set_info() (*mpi4py.MPI.File method*), 114
Set_info() (*mpi4py.MPI.Win method*), 171
Set_name() (*mpi4py.MPI.Comm method*), 86
Set_name() (*mpi4py.MPI.Datatype method*), 101
Set_name() (*mpi4py.MPI.Win method*), 171
Set_size() (*mpi4py.MPI.File method*), 115
Set_source() (*mpi4py.MPI.Status method*), 156
Set_tag() (*mpi4py.MPI.Status method*), 157
Set_view() (*mpi4py.MPI.File method*), 115
Shared_query() (*mpi4py.MPI.Win method*), 171
Shift() (*mpi4py.MPI.Cartcomm method*), 61
SHORT (*in module mpi4py.MPI*), 215
SHORT_INT (*in module mpi4py.MPI*), 219
Shrink() (*mpi4py.MPI.Comm method*), 87
shutdown() (*mpi4py.futures.MPIPoolExecutor method*), 41

SIGNED_CHAR (*in module mpi4py.MPI*), 215
SIGNED_INT (*in module mpi4py.MPI*), 223
SIGNED_LONG (*in module mpi4py.MPI*), 223
SIGNED_LONG_LONG (*in module mpi4py.MPI*), 223
SIGNED_SHORT (*in module mpi4py.MPI*), 223
SIMILAR (*in module mpi4py.MPI*), 207
SINT16_T (*in module mpi4py.MPI*), 223
SINT32_T (*in module mpi4py.MPI*), 223
SINT64_T (*in module mpi4py.MPI*), 224
SINT8_T (*in module mpi4py.MPI*), 223
size (*mpi4py.MPI.Comm attribute*), 93
size (*mpi4py.MPI.Datatype attribute*), 103
size (*mpi4py.MPI.File attribute*), 118
size (*mpi4py.MPI.Group attribute*), 124
source (*mpi4py.MPI.Status attribute*), 157
Spawn() (*mpi4py.MPI.Intracomm method*), 135
Spawn_multiple() (*mpi4py.MPI.Intracomm method*), 135

Split() (*mpi4py.MPI.Comm method*), 87
Split_type() (*mpi4py.MPI.Comm method*), 87
Ssend() (*mpi4py.MPI.Comm method*), 87
ssend() (*mpi4py.MPI.Comm method*), 92
ssend() (*mpi4py.util.pkl5.Comm method*), 51
Ssend_init() (*mpi4py.MPI.Comm method*), 87
staremap() (*mpi4py.futures.MPIPoolExecutor method*), 40

staremap() (*mpi4py.util.pool.Pool method*), 56
staremap_async() (*mpi4py.util.pool.Pool method*), 56
Start() (*mpi4py.MPI.Qrequest class method*), 120
Start() (*mpi4py.MPI.Prequest method*), 143
Start() (*mpi4py.MPI.Win method*), 171
Startall() (*mpi4py.MPI.Prequest class method*), 143
Status (*class in mpi4py.MPI*), 154
Sub() (*mpi4py.MPI.Cartcomm method*), 61
submit() (*mpi4py.futures.MPIPoolExecutor method*), 40
SUBVERSION (*in module mpi4py.MPI*), 212
SUCCESS (*in module mpi4py.MPI*), 194

successful() (*mpi4py.util.pool.AsyncResult* method), 57
SUM (*in module mpi4py.MPI*), 226
SupportsBuffer (*in module mpi4py.typing*), 36
SupportsCAI (*in module mpi4py.typing*), 36
SupportsDLPack (*in module mpi4py.typing*), 36
Sync() (*mpi4py.MPI.File* method), 115
Sync() (*mpi4py.MPI.Win* method), 171

T

tag (*mpi4py.MPI.Status* attribute), 157
TAG_UB (*in module mpi4py.MPI*), 193
TargetSpec (*in module mpi4py.typing*), 38
terminate() (*mpi4py.util.pool.Pool* method), 56
Test() (*mpi4py.MPI.Request* method), 147
test() (*mpi4py.MPI.Request* method), 149
Test() (*mpi4py.MPI.Win* method), 171
test() (*mpi4py.util.pkl5.Request* method), 49
Testall() (*mpi4py.MPI.Request* class method), 147
testall() (*mpi4py.MPI.Request* class method), 149
testall() (*mpi4py.util.pkl5.Request* class method), 49
Testany() (*mpi4py.MPI.Request* class method), 147
testany() (*mpi4py.MPI.Request* class method), 149
Testsome() (*mpi4py.MPI.Request* class method), 147
testsome() (*mpi4py.MPI.Request* class method), 150
THREAD_FUNNELED (*in module mpi4py.MPI*), 212
thread_level (*mpi4py.mpi4py.rc* attribute), 21
THREAD_MULTIPLE (*in module mpi4py.MPI*), 212
THREAD_SERIALIZED (*in module mpi4py.MPI*), 212
THREAD_SINGLE (*in module mpi4py.MPI*), 212
ThreadPool (*class in mpi4py.util.pool*), 56
threads (*mpi4py.mpi4py.rc* attribute), 21
THRESHOLD (*mpi4py.MPI.Pickle* attribute), 142
to_numpy_dtype() (*in module mpi4py.util.dtype*), 48
tobytes() (*mpi4py.MPI.buffer* method), 174
tocode() (*mpi4py.MPI.Datatype* method), 102
tomemory() (*mpi4py.MPI.Win* method), 172
topo (*mpi4py.MPI.Cartcomm* attribute), 62
topo (*mpi4py.MPI.Graphcomm* attribute), 120
Topocomm (*class in mpi4py.MPI*), 157
topology (*mpi4py.MPI.Comm* attribute), 93
toreadonly() (*mpi4py.MPI.buffer* method), 174
Translate_ranks() (*mpi4py.MPI.Group* method), 123
true_extent (*mpi4py.MPI.Datatype* attribute), 103
true_lb (*mpi4py.MPI.Datatype* attribute), 103
true_ub (*mpi4py.MPI.Datatype* attribute), 103
TWOINT (*in module mpi4py.MPI*), 219
typechar (*mpi4py.MPI.Datatype* attribute), 103
TYPECLASS_COMPLEX (*in module mpi4py.MPI*), 203
TYPECLASS_INTEGER (*in module mpi4py.MPI*), 203
TYPECLASS_REAL (*in module mpi4py.MPI*), 203
TypeSpec (*in module mpi4py.typing*), 36
typestr (*mpi4py.MPI.Datatype* attribute), 103

U

ub (*mpi4py.MPI.Datatype* attribute), 103
UINT16_T (*in module mpi4py.MPI*), 217
UINT32_T (*in module mpi4py.MPI*), 217
UINT64_T (*in module mpi4py.MPI*), 217
UINT8_T (*in module mpi4py.MPI*), 217
UNDEFINED (*in module mpi4py.MPI*), 192
UNEQUAL (*in module mpi4py.MPI*), 207
Union() (*mpi4py.MPI.Group* class method), 124
UNIVERSE_SIZE (*in module mpi4py.MPI*), 193
Unlock() (*mpi4py.MPI.Win* method), 171
Unlock_all() (*mpi4py.MPI.Win* method), 172
Unpack() (*mpi4py.MPI.Datatype* method), 101
Unpack_external() (*mpi4py.MPI.Datatype* method), 101
Unpublish_name() (*in module mpi4py.MPI*), 185
UNSIGNED (*in module mpi4py.MPI*), 216
UNSIGNED_CHAR (*in module mpi4py.MPI*), 215
UNSIGNED_INT (*in module mpi4py.MPI*), 222
UNSIGNED_LONG (*in module mpi4py.MPI*), 216
UNSIGNED_LONG_LONG (*in module mpi4py.MPI*), 216
UNSIGNED_SHORT (*in module mpi4py.MPI*), 216
UNWEIGHTED (*in module mpi4py.MPI*), 207
update() (*mpi4py.MPI.Info* method), 128

V

values() (*mpi4py.MPI.Info* method), 128
VERSION (*in module mpi4py.MPI*), 212

W

Wait() (*mpi4py.MPI.Request* method), 147
wait() (*mpi4py.MPI.Request* method), 150
Wait() (*mpi4py.MPI.Win* method), 172
wait() (*mpi4py.util.pkl5.Request* method), 49
wait() (*mpi4py.util.pool.AsyncResult* method), 57
Waitall() (*mpi4py.MPI.Request* class method), 147
waitall() (*mpi4py.MPI.Request* class method), 150
waitall() (*mpi4py.util.pkl5.Request* class method), 50
Waitany() (*mpi4py.MPI.Request* class method), 148
waitany() (*mpi4py.MPI.Request* class method), 150
Waitsome() (*mpi4py.MPI.Request* class method), 148
waitsome() (*mpi4py.MPI.Request* class method), 150
WCHAR (*in module mpi4py.MPI*), 215
WEIGHTS_EMPTY (*in module mpi4py.MPI*), 208
Win (*class in mpi4py.MPI*), 162
WIN_BASE (*in module mpi4py.MPI*), 194
WIN_CREATE_FLAVOR (*in module mpi4py.MPI*), 194
WIN_DISP_UNIT (*in module mpi4py.MPI*), 194
WIN_FLAVOR (*in module mpi4py.MPI*), 194
WIN_FLAVOR_ALLOCATE (*in module mpi4py.MPI*), 209
WIN_FLAVOR_CREATE (*in module mpi4py.MPI*), 208
WIN_FLAVOR_DYNAMIC (*in module mpi4py.MPI*), 209
WIN_FLAVOR_SHARED (*in module mpi4py.MPI*), 209

`WIN_MODEL` (*in module mpi4py.MPI*), 194
`WIN_NULL` (*in module mpi4py.MPI*), 230
`WIN_SEPARATE` (*in module mpi4py.MPI*), 209
`WIN_SIZE` (*in module mpi4py.MPI*), 194
`WIN_UNIFIED` (*in module mpi4py.MPI*), 209
`Write()` (*mpi4py.MPI.File method*), 115
`Write_all()` (*mpi4py.MPI.File method*), 115
`Write_all_begin()` (*mpi4py.MPI.File method*), 115
`Write_all_end()` (*mpi4py.MPI.File method*), 116
`Write_at()` (*mpi4py.MPI.File method*), 116
`Write_at_all()` (*mpi4py.MPI.File method*), 116
`Write_at_all_begin()` (*mpi4py.MPI.File method*),
 116
`Write_at_all_end()` (*mpi4py.MPI.File method*), 116
`Write_ordered()` (*mpi4py.MPI.File method*), 117
`Write_ordered_begin()` (*mpi4py.MPI.File method*),
 117
`Write_ordered_end()` (*mpi4py.MPI.File method*), 117
`Write_shared()` (*mpi4py.MPI.File method*), 117
`Wtick()` (*in module mpi4py.MPI*), 185
`Wtime()` (*in module mpi4py.MPI*), 186
`WTIME_IS_GLOBAL` (*in module mpi4py.MPI*), 193