# A Draft Java Binding for MPI

Bryan Carpenter, Geoffrey Fox,
Sung-Hoon Ko, Xinying Li, Guansong Zhang

*Northeast Parallel Architectures Centre,*
*Syracuse University,*
*111 College Place,*
*Syracuse, New York 13244-410*
*{dbc,gcf,shko,xli,zgs}npac.syr.edu*

August 20, 1998

# Contents

# 1 Introduction

This document proposes a Java language binding for MPI, version 1.1. The document is meant to be read in conjunction with the that standard, and its principal sections are laid out in the same way as the standard to allow cross-referencing. The appendices collect together complete Java class interfaces. Where practical the current proposals follow the MPI C++ interface defined in the MPI standard version 2.0.

The current document should be seen as a specification for software under development at NPAC. The interface described makes no claim as a general standard for Java MPI binding, and it has not been endorsed by the MPI forum.

A protoype Java MPI wrapper has been available from NPAC for some time[1]. We aim to make an initial implementation of the full interface described here available by the end of December, 1997.

# 2 MPI Terms and Conventions

## 2.1 Document Notation

No special issues for Java binding.

## 2.2 Procedure Specification

In general we use *italicized* names to refer to entities in the MPI language independent procedure definitions, and `typewriter` font for concrete Java entities.

Unless otherwise stated, Java argument names are identical to the corresponding language independent names. Unless otherwise stated, in non-static member functions of `Comm`, `Status`, `Request`, `Datatype`, `Op` or `Group` (and subclasses), the class instance stands for the *comm*, *status*, *request*, *datatype*, *op* or *group* argument, respectively in the corresponding the language independent procedure definition.

## 2.3 Semantic terms

No special issues for Java binding.

## 2.4 Data types

Opaque objects will be presented as Java objects. This introduces the option of simplifying the user's task in managing these objects. MPI destructors can be called by the Java object destructors, which are called automatically by the Java garbage collector.

[1] `http://www.npac.syr.edu/users/yjchang/javaMPI/`

In contrast to the MPI C++ binding, we adopt this strategy as the general rule. Explicit calls to MPI destructor functions are typically omitted from the Java user interface. Exceptions are made for the `Comm` and `Request` classes. *MPI_COMM_FREE* is a collective operation, so the user must ensure that calls are made consistently across processors. Similarly *MPI_REQUEST_FREE* has a specific semantic effect which is better left under user control.

## 2.5 Language Binding

Special issues for Java binding:

- Some options allowed for derived data types in the C and Fortran binding are deleted in the Java binding. The Java VM does not provide any concept of a global linear address space. Passing physical addresses to data type definitions is not allowed, and use of the *MPI_TYPE_STRUCT* datatype constructor is restricted in a way that makes it impossible to send mixed primitive types in a single message. In an initial implementation it will not be possible to directly send a general Java object in a message. We assume that Java programmers who want to achieve this goal will explicitly serialize their objects before transmitting them. Later we hope to treat `MPI.JAVA_OBJECT` as a new MPI *primitive type*, with serialization handled automatically in the wrapper code.

- Message buffers in C or Fortran can be multidimensional arrays, so it may seem natural to allow Java message buffers to be multidimensional arrays. But a Java multi-dimensional array does not necessarily map to a contiguous chunk of memory (it is implemented as a vector of separately allocated arrays), so there is no obvious way to support this facility in a "wrapper" interface. Hence Java message buffers are always one dimensional arrays with primitive element type. To allow a little extra flexibility, an `offset` parameter is associated with any buffer argument. This defines the position of the first actual buffer element in the Java array.

- Unlike the C and Fortran interfaces, the Java interfaces to MPI calls will not return explicit error codes. The Java exception mechanism will be used to report errors. This mechanism is very widely used by Java libraries. It is inconvenient to use up the single return value of a Java function with an error code (Java doesn't allow function arguments to be passed by reference, so returning multiple values tends to be more awkward than in normal languages). A few functions in the MPI interface return multiple values, even after the error code is eliminated. In these cases the Java binding returns a single object of some artificial class introduced to support the function concerned.

- To permit a straightforward wrapper implementation it seems to be necessary to impose some non-interference rules for concurrent read and write operations on arrays. When an array is passed to an MPI member such as a send or receive operation, the wrapper code will extract a pointer to the contents of the array using a JNI `Get<PrimitiveType>ArrayElements` routine. If the garbage collector does not support "pinning" (temporarily disabling run-time relocation of data for specific arrays), the pointer returned by this `Get` function may be to a temporary copy of the elements. The copy will be written back to the true Java array when a subsequent call to `Release<PrimitiveType>ArrayElements` is made. If two operations involving the same array are active concurrently, this copy back may result in failure to save modifications made by one or more of the concurrent calls.

For safety, the rule suggested is that when several MPI send or receive (etc) operations are active concurrently, if any one of those operations writes to a particular array, none of the other operations must read or write *any portion* of that array.

Operations may occur concurrently through explicit use of threads, *or through use of non-blocking communication operations.* So far as use of non-blocking communication operations is concerned, the restriction could be relaxed at the price of having the wrapper always explicitly copy buffers using *MPI_PACK* and *MPI_UNPACK*. Of course this is an undesirable overhead. A final decision on whether this overhead is acceptable will be defered until the implementation is more complete.

### 2.6 Processes

No special issues for Java binding.

### 2.7 Error Handling

As explained in section 2.5, the Java members do not return error codes. The Java exceptions thrown instead are defined in section 7.3.

## 3 Point-to-Point Communication

### 3.1 Introduction

In general the Java binding of point-to-point communication operations will realize the MPI functions as members of the `Comm` class.

### 3.2 Blocking Send and Receive operations

```
void Comm.Send(Object buf, int offset, int count,
```

```
                    Datatype datatype, int dest, int tag)
```

Java binding of the MPI operation *MPI_SEND*. The actual argument associated with `buf` must be a one-dimensional array. The array elements must have primitive type, and that type must agree with `datatype`, or be the *base type* of `datatype` (see section 3.12). The value `offset` is a subscript in the array, defining the position of the first item of the message.

The basic datatypes for Java will be

| MPI datatype | Java datatype |
|---|---|
| MPI.BYTE | `byte` |
| MPI.CHAR | `char` |
| MPI.SHORT | `short` |
| MPI.BOOLEAN | `boolean` |
| MPI.INT | `int` |
| MPI.LONG | `long` |
| MPI.FLOAT | `float` |
| MPI.DOUBLE | `double` |
| MPI.PACKED | |

Later we intend to add a new "primitive" type `MPI.JAVA_OBJECT`. If a buffer has "Java object" is its base type, it will be serialized inside the wrapper code.

```
Status Comm.Recv(Object buf, int offset, int count,
                 Datatype datatype, int source, int tag)
```

Java binding of the MPI operation *MPI_RECV*. The actual argument associated with `buf` must be a one-dimensional array. The array elements must have primitive type, and that type must agree with `datatype`, or be the *base type* of `datatype` (see section 3.12). The value `offset` is a subscript in the array, defining the position into which the first item of the incoming message will be copied. The return value is *status*.

The MPI constants *MPI_ANY_SOURCE* and *MPI_ANY_TAG* are available as `MPI.ANY_SOURCE` and `MPI.ANY_TAG`.

```
int Status.Get_count(Datatype datatype)
```

Java binding of the MPI operation *MPI_GET_COUNT*. The return value is *count*.

## 3.3 Data type matching and data conversion

We have no requirement for representation conversion, because all Java implementations are supposed to use the same representation, and the MPI standard does not mandate support for inter-language communication.

## 3.4 Communication Modes

```
void Comm.Bsend(Object buf, int offset, int count,
                Datatype datatype, int dest, int tag)
```

Java binding of the MPI operation *MPI_BSEND*. Arguments as for `send`.

```
void Comm.Ssend(Object buf, int offset, int count,
                Datatype datatype, int dest, int tag)
```

Java binding of the MPI operation *MPI_SSEND*. Arguments as for `send`.

```
void Comm.Rsend(Object buf, int offset, int count,
                Datatype datatype, int dest, int tag)
```

Java binding of the MPI operation *MPI_RSEND*. Arguments as for `send`.

## 3.5 Semantics of point-to-point communication

No special issues for Java binding.

## 3.6 Buffer allocation and usage

```
void MPI.Buffer_attach(byte [] buffer)
```

Java binding of the MPI operation *MPI_BUFFER_ATTACH*. The value of *size* is taken to be size of the `buffer` argument.

```
byte [] MPI.Buffer_detach()
```

Java binding of the MPI operation *MPI_BUFFER_DETACH*. The return value is *buffer*. The size of the returned arrray is *size*.

The constant *MPI_BSEND_OVERHEAD* is available as `MPI.BSEND_OVERHEAD`.

### 3.7 Nonblocking communication

```
Request Comm.Isend(Object buf, int offset, int count,
                   Datatype datatype, int dest, int tag)
```

Java binding of the MPI operation *MPI_ISEND*. Arguments as for `send`. The result value is *request*.

```
Request Comm.Ibsend(Object buf, int offset, int count,
                    Datatype datatype, int dest, int tag)
```

Java binding of the MPI operation *MPI_IBSEND*. Arguments as for `send`. The result value is *request*.

```
Request Comm.Issend(Object buf, int offset, int count,
                    Datatype datatype, int dest, int tag)
```

Java binding of the MPI operation *MPI_ISSEND*. Arguments as for `send`. The result value is *request*.

```
Request Comm.Irsend(Object buf, int offset, int count,
                    Datatype datatype, int dest, int tag)
```

Java binding of the MPI operation *MPI_IRSEND*. Arguments as for `send`. The result value is *request*.

```
Request Comm.Irecv(Object buf, int offset, int count,
                   Datatype datatype, int source, int tag)
```

Java binding of the MPI operation *MPI_IRECV*. Arguments as for `Recv`. The result value is *request*.

```
Status Request.Wait()
```

Java binding of the MPI operation *MPI_WAIT*. Return value is *status*.

```
Status Request.Test()
```

Java binding of the MPI operation *MPI_TEST*. Return value is *status*, or `null` if *flag* was false.

```
void Request.Free()
```

Java binding of the MPI operation *MPI_REQUEST_FREE*. (Note: a *Free* method is required rather than a *finalize* method, because *MPI_REQUEST_FREE* has a specific effect on communications, and we cannot assume that the Java garbage collector would call a *finalize* at an appropriate time.)

```
boolean Request.Is_null()
```

Returns true if the underlying MPI Request object is equal to *MPI_REQUEST_NULL*, false otherwise.

```
static Status Request.Waitany(Request [] array_of_requests)
```

Java binding of the MPI operation *MPI_WAITANY*. The value of *count* is taken to be size of the `array_of_requests` argument. Return value is *status*. It contains *index* in its `index` field.

```
static Status Request.Testany(Request [] array_of_requests)
```

Java binding of the MPI operation *MPI_TESTANY*. The value of *count* is taken to be size of the `array_of_requests` argument. Return value is *status*, containing *index* in its `index` field, or `null` if *flag* was false.

```
static void Request.Waitall(Request [] array_of_requests,
                            Status [] array_of_statuses)
```

Java binding of the MPI operation *MPI_WAITALL*. The value of *count* is taken to be size of the `array_of_requests` argument. The second argument, `array_of_statuses`, should be the same size.

```
static boolean Request.Testall(Request [] array_of_requests,
                               Status [] array_of_statuses)
```

Java binding of the MPI operation *MPI_WAITALL*. Arguments as for `Waitall`. The return value is *flag*.

```
static Status [] Request.Waitsome(Request [] array_of_requests)
```

Java binding of the MPI operation *MPI_WAITALL*. The value of *incount* is taken to be size of `array_of_requests` argument. The return value is an array of size *outcount* containing the statuses and indices from *array_of_statuses* and *array_of_indices*.

```
static Status [] Request.Testsome(Request [] array_of_requests)
```

Java binding of the MPI operation *MPI_TESTALL*. Arguments and return value as for `Waitsome`.

## 3.8 Probe and Cancel

```
Status Comm.Iprobe(int source, int tag)
```

Java binding of the MPI operation *MPI_IPROBE*. Return value is *status*, or `null` if *flag* was false.

```
Status Comm.Probe(int source, int tag)
```

Java binding of the MPI operation *MPI_PROBE*. Return value is *status*.

```
void Request.Cancel()
```

Java binding of the MPI operation *MPI_CANCEL*.

```
boolean Status.Test_cancelled()
```

Java binding of the MPI operation *MPI_TEST_CANCELLED*. The return value is *flag*.

## 3.9 Persistent communication requests

```
Prequest Comm.Send_init(Object buf, int offset, int count,
                        Datatype datatype, int dest, int tag)
```

Java binding of the MPI operation *MPI_SEND_INIT*. Return value is *request*.

```
Prequest Comm.Bsend_init(Object buf, int offset, int count,
                         Datatype datatype, int dest, int tag)
```

Java binding of the MPI operation *MPI_BSEND_INIT*. Return value is *request*.

```
Prequest Comm.Ssend_init(Object buf, int offset, int count,
                         Datatype datatype, int dest, int tag)
```

Java binding of the MPI operation *MPI_SSEND_INIT*. Return value is *request*.

```
Prequest Comm.Rsend_init(Object buf, int offset, int count,
                         Datatype datatype, int dest, int tag)
```

Java binding of the MPI operation *MPI_RSEND_INIT*. Return value is *request*.

```
Prequest Comm.Recv_init(Object buf, int offset, int count,
                        Datatype datatype, int source, int tag)
```

Java binding of the MPI operation *MPI_RECV_INIT*. Return value is *request*.

```
void Prequest.Start()
```

Java binding of the MPI operation *MPI_START*.

```
static void Prequest.Startall(Prequest [] array_of_requests)
```

Java binding of the MPI operation *MPI_STARTALL*. The value of *count* is taken to be size of `array_of_requests` argument.

## 3.10 Send-receive

```
Status Comm.Sendrecv(Object sendbuf, int sendoffset,
                     int sendcount, Datatype sendtype,
                     int dest, int sendtag,
                     Object recvbuf, int recvoffset,
                     int recvcount, Datatype recvtype,
                     int source, int recvtag)
```

Java binding of the MPI operation *MPI_SENDRECV*. The value `sendoffset` is a subscript in `sendbuf`, defining the position of the first item of the message to be sent. The value `recvoffset` is a subscript in `recvbuf`, defining the position into which the first item of the incoming message will be copied.

The return value is *status*.

```
Status Comm.Sendrecv_replace(Object buf, int offset,
                     int count, Datatype datatype,
                     int dest, int sendtag,
                     int source, int recvtag)
```

Java binding of the MPI operation *MPI_SENDRECV_REPLACE*. The value `offset` is a subscript in `buf`, defining the position of the first item of the message to be sent, and the position into which the first item of the incoming message will be copied.

The return value is *status*.

### 3.11 Null processes

The constant *MPI_PROC_NULL* is available as `MPI.PROC_NULL`.

### 3.12 Derived datatypes

Every derived data type constructable in the Java binding has a uniquely defined *base type*. This is a Java primitive type. For the system-defined `Datatype` values, the base type is (naturally) the Java primitive type for which the value stands. Derived types inherit their base types from their precursors in a straightforward way.

```
static Datatype Datatype.Contiguous(int count, Datatype oldtype)
```

Java binding of the MPI operation *MPI_TYPE_CONTIGUOUS*. The return value is *newtype*. The base type of *newtype* is the same as the base type of *oldtype*.

```
static Datatype Datatype.Vector(int count, int blocklength,
                                int stride, Datatype oldtype)
```

Java binding of the MPI operation *MPI_TYPE_VECTOR*. The return value is *newtype*. The base type of *newtype* is the same as the base type of *oldtype*.

```
static Datatype Datatype.Hvector(int count, int blocklength,
                                 int stride, Datatype oldtype)
```

Java binding of the MPI operation *MPI_TYPE_HVECTOR*. Arguments as for `Vector`. *Unlike other language bindings*, the value of `stride` is given in units of the base type, *not* bytes.

```
static Datatype Datatype.Indexed(int [] array_of_blocklengths,
                                 int [] array_of_displacements,
                                 Datatype oldtype)
```

Java binding of the MPI operation *MPI_TYPE_INDEXED*. The value of *count* is taken to be size of the `array_of_blocklengths` argument. The second argument, `array_of_displacements`, should be the same size. The return value is *newtype*. The base type of *newtype* is the same as the base type of *oldtype*.

```
static Datatype Datatype.Hindexed(int [] array_of_blocklengths,
                                  int [] array_of_displacements,
                                  Datatype oldtype)
```

Java binding of the MPI operation *MPI_TYPE_HINDEXED*. Arguments as for `Indexed`. *Unlike other language bindings*, the values in `array_of_displacements` are given in units of the base type, *not* bytes.

```
static Datatype Datatype.Struct(int [] array_of_blocklengths,
                                int [] array_of_displacements,
                                Datatype [] array_of_types)
```

Java binding of the MPI operation *MPI_TYPE_STRUCT*. The value of *count* is taken to be size of the `array_of_blocklengths` argument. The second and third arguments, `array_of_displacements` and `array_of_types`, should be the same size. The return value is *newtype*. All elements of `array_of_types` should have the same base type; this will also be the base type of *newtype*. *Unlike other language bindings*, the values in `array_of_displacements` are given in units of the base type, *not* bytes.

```
int Datatype.Extent()
```

Java binding of the MPI operation *MPI_TYPE_EXTENT*. The return value is *extent*.

```
int Datatype.Size()
```

Java binding of the MPI operation *MPI_TYPE_SIZE*. The return value is *size*.

```
int Datatype.Lb()
```

Java binding of the MPI operation *MPI_TYPE_LB*. The return value is *displacement*.

```
int Datatype.Ub()
```

Java binding of the MPI operation *MPI_TYPE_UB*. The return value is *displacement*.

```
void Datatype.Commit()
```

Java binding of the MPI operation *MPI_TYPE_COMMIT*.

```
void Datatype.finalize()
```

The `Datatype` destructor will invoke the MPI operation *MPI_TYPE_FREE*.

```
int Status.Get_elements(Datatype datatype)
```

Java binding of the MPI operation *MPI_GET_ELEMENTS*. The return value is *count*.

## 3.13 Pack and unpack

```
int Comm.Pack(Object inbuf, int offset, int incount,
              Datatype datatype,
              byte [] outbuf, int position)
```

Java binding of the MPI operation *MPI_PACK*. The value of *outcount* is taken to be size of the `outbuf` argument. The return value is the output value of *position*.

```
int Comm.Unpack(byte [] inbuf, int position,
                Object outbuf, int offset, int outcount,
                Datatype datatype)
```

Java binding of the MPI operation *MPI_UNPACK*. The value of *insize* is taken to be size of the `inbuf` argument. The return value is the output value of *position*.

```
int Comm.Pack_size(int incount, Datatype datatype)
```

Java binding of the MPI operation *MPI_PACK_SIZE*. The return value is *size*.

# 4 Collective Communication

## 4.1 Introduction and Overview

In general the Java binding of collective communication operations will realize the MPI functions as members of the `Comm` class.

## 4.2 Communicator argument

No special issues for Java binding.

## 4.3 Barrier synchronization

```
void Intracomm.Barrier()
```

Java binding of the MPI operation *MPI_BARRIER*.

## 4.4 Broadcast

```
void Intracomm.Bcast(Object buffer, int offset, int count,
                     Datatype datatype, int root)
```

Java binding of the MPI operation *MPI_BCAST*. The value `offset` is a subscript in `buffer`, defining the position of the first item to be broadcast or received.

## 4.5 Gather

```
void Intracomm.Gather(Object sendbuf, int sendoffset,
                      int sendcount, Datatype sendtype,
                      Object recvbuf, int recvoffset,
                      int recvcount, Datatype recvtype, int root)
```

Java binding of the MPI operation *MPI_GATHER*. The value `sendoffset` is a subscript in `sendbuf`, defining the position of the first item of the message to be sent. The value `recvoffset` is a subscript in `recvbuf`, defining the position into which the first item of the gathered data will be copied.

```
void Intracomm.Gatherv(Object sendbuf, int sendoffset,
                       int sendcount, Datatype sendtype,
                       Object recvbuf, int recvoffset,
                       int [] recvcount, int [] displs,
                       Datatype recvtype, int root)
```

Java binding of the MPI operation *MPI_GATHERV*. Arguments other than `displs` as for `Gather`. (Note that if `recvtype` is a derived data type, `displs` is in units of the derived data type, unlike `recvoffset`, which is in units of the base type.)

### 4.6 Scatter

```
void Intracomm.Scatter(Object sendbuf, int sendoffset,
                       int sendcount, Datatype sendtype,
                       Object recvbuf, int recvoffset,
                       int recvcount, Datatype recvtype, int root)
```

Java binding of the MPI operation *MPI_SCATTER*. The value `sendoffset` is a subscript in `sendbuf`, defining the position of the first item of the data to be scattered. The value `recvoffset` is a subscript in `recvbuf`, defining the position into which the first item of the incoming message will be copied.

```
void Intracomm.Scatterv(Object sendbuf, int sendoffset,
                        int [] sendcount, int [] displs,
                        Datatype sendtype,
                        Object recvbuf, int recvoffset,
                        int recvcount, Datatype recvtype, int root)
```

Java binding of the MPI operation *MPI_SCATTERV*. Arguments other than `displs` as for `Scatter`.

### 4.7 Gather-to-all

```
void Intracomm.Allgather(Object sendbuf, int sendoffset,
                         int sendcount, Datatype sendtype,
                         Object recvbuf, int recvoffset,
                         int recvcount, Datatype recvtype)
```

Java binding of the MPI operation *MPI_ALLGATHER*. Arguments as for `Gather`, except for the omission of `root`.

```
void Intracomm.Allgatherv(Object sendbuf, int sendoffset,
                          int sendcount, Datatype sendtype,
                          Object recvbuf, int recvoffset,
                          int recvcount, int [] displs,
                          Datatype recvtype)
```

Java binding of the MPI operation *MPI_GATHERV*. Arguments other than `displs` as for `Gather`, except for the omission of `root`.

### 4.8 All-to-All Scatter/Gather

```
void Intracomm.Alltoall(Object sendbuf, int sendoffset,
                        int sendcount, Datatype sendtype,
                        Object recvbuf, int recvoffset,
                        int [] recvcount, Datatype recvtype)
```

Java binding of the MPI operation *MPI_ALLTOALL*. The value `sendoffset` is a subscript in `sendbuf`, defining the position of the first item of the data to be scattered. The value `recvoffset` is a subscript in `recvbuf`, defining the position into which the first item of the gathered data will be copied.

```
void Intracomm.Alltoallv(Object sendbuf, int sendoffset,
                         int [] sendcount, int [] sdispls,
                         Datatype sendtype,
                         Object recvbuf, int recvoffset,
                         int [] recvcount, int [] rdispls,
                         Datatype recvtype)
```

Java binding of the MPI operation *MPI_ALLTOALLV*. Arguments other than `sdispls`, `rdispls` as for `Alltoall`.

### 4.9 Global Reduction Operations

```
void Intracomm.Reduce(Object sendbuf, int sendoffset,
                      Object recvbuf, int recvoffset,
                      int count, Datatype datatype,
                      Op op, int root)
```

Java binding of the MPI operation *MPI_REDUCE*. The value `sendoffset` is a subscript in `sendbuf`, defining the position of the first item input data. The value `recvoffset` is a subscript in `recvbuf`, defining the position into which the first item of result data be copied.

The predefined operations are available in Java as `MPI.MAX`, `MPI.MIN`, `MPI.SUM`, `MPI.PROD`, `MPI.LAND`, `MPI.BAND`, `MPI.LOR`, `MPI.BOR`, `MPI.LXOR`, `MPI.BXOR`, `MPI.MINLOC` and `MPI.MAXLOC`.

The handling of *MINLOC* and *MAXLOC* is modelled on the Fortran binding. The extra predefined types `MPI.SHORT2`, `MPI.INT2`, `MPI.LONG2`, `MPI.FLOAT2`, `MPI.DOUBLE2` describe pairs of Java numeric primitive types.

```
Op.Op(User_function function, boolean commute)
```

Java binding of the MPI operation *MPI_OP_CREATE*.

The abstract base class `User_function` is defined by

```
class User_function {
  public abstract void call(Object invec, Object outvec,
                            Datatype datatype) ;
}
```

There is an implementation problem here: how does the C code inside the wrapper persuade the MPI reduction operation to invoke the Java method, `call`? Some C `MPI_User_function` must dispatch the `call` method of a Java object of type `User_function`. The difficulty is creating a C function at runtime that will do this. A generic C function can reference a Java object through a handle held in some static variable. But if more than one user defined operation is needed, how does a particular invocation find the proper Java function object? One possibility is to assume a fixed bound, $N$, on the number of allowed user-defined operations, and define $N$ different `MPI_User_function` functions, invoking `call` methods through $N$ different static variables. These `MPI_User_function`/static-variable pairs are managed as a pool. Calling the `Op` constructor involves copying a handle to its `function` object argument into one of the static variables in the pool and passing the corresponding `MPI_User_function` to *MPI_OP_CREATE*.

```
void Op.finalize()
```

The `Op` destructor will invoke the MPI operation *MPI_OP_FREE*.

```
void Intracomm.Allreduce(Object sendbuf, int sendoffset,
                         Object recvbuf, int recvoffset,
                         int count, Datatype datatype,
                         Op op)
```

Java binding of the MPI operation *MPI_ALLREDUCE*. Arguments as for `Reduce`, except for the omission of `root`.

## 4.10 Reduce-Scatter

```
void Intracomm.Reduce_scatter(Object sendbuf, int sendoffset,
                              Object recvbuf, int recvoffset,
                              int [] recvcounts, Datatype datatype,
                              Op op)
```

Java binding of the MPI operation *MPI_REDUCE_SCATTER*. Arguments as for `Reduce`, except for the omission of `root` and replacement of `count` with `recvcounts`.

### 4.11 Scan

```
void Intracomm.Scan(Object sendbuf, int sendoffset,
                    Object recvbuf, int recvoffset,
                    int count, Datatype datatype,
                    Op op)
```

Java binding of the MPI operation *MPI_SCAN*. Arguments as for `Reduce`, except for the omission of `root`.

### 4.12 Correctness

No special issues for Java binding.

## 5 Groups, Contexts and Communicators

### 5.1 Introduction

No special issues for Java binding.

### 5.2 Basic Concepts

The constant *MPI_GROUP_EMPTY* is available as `MPI.GROUP_EMPTY`.

The constants *MPI_COMM_WORLD*, *MPI_COMM_SELF* are available as `MPI.COMM_WORLD`, `MPI.COMM_SELF`.

### 5.3 Group Management

```
int Group.Size()
```

Java binding of the MPI operation *MPI_GROUP_SIZE*. The return value is *size*.

```
int Group.Rank()
```

Java binding of the MPI operation *MPI_GROUP_RANK*. The return value is *Rank*.

```
static int [] Group.Translate_ranks(Group group1, int [] ranks1,
                                    Group group2)
```

Java binding of the MPI operation *MPI_GROUP_TRANSLATE_RANKS*. The value of $n$ is taken to be size of the `ranks1` argument. The return value is *ranks2*.

```
static int Group.Compare(Group group1, Group group2)
```

Java binding of the MPI operation *MPI_GROUP_COMPARE*. The return value is *result*.

```
Group Comm.Group()
```

Java binding of the MPI operation *MPI_COMM_GROUP*. The return value is *group*.

```
static Group Group.Union(Group group1, Group group2)
```

Java binding of the MPI operation *MPI_GROUP_UNION*. The return value is *newgroup*.

```
static Group Group.Intersection(Group group1, Group group2)
```

Java binding of the MPI operation *MPI_GROUP_INTERSECTION*. The return value is *newgroup*.

```
static Group Group.Difference(Group group1, Group group2)
```

Java binding of the MPI operation *MPI_GROUP_DIFFERENCE*. The return value is *newgroup*.

```
Group Group.Incl(int [] ranks)
```

Java binding of the MPI operation *MPI_GROUP_INCL*. The value of $n$ is taken to be size of the `ranks` argument. The return value is *newgroup*.

```
Group Group.Excl(int [] ranks)
```

Java binding of the MPI operation *MPI_GROUP_EXCL*. The value of $n$ is taken to be size of the `ranks` argument. The return value is *newgroup*.

```
Group Group.Range_incl(int [] [] ranges)
```

Java binding of the MPI operation *MPI_GROUP_RANGE_INCL*. The value of $n$ is taken to be size of the `ranges` argument. The return value is *newgroup*.

```
Group Group.Range_excl(int [] [] ranges)
```

Java binding of the MPI operation *MPI_GROUP_RANGE_EXCL*. The value of $n$ is taken to be size of the `ranges` argument. The return value is *newgroup*.

```
void Group.finalize()
```

The `Group` destructor will invoke the MPI operation *MPI_GROUP_FREE*.

## 5.4 Communicator Management

```
int Comm.Size()
```

Java binding of the MPI operation *MPI_COMM_SIZE*. The return value is *size*.

```
int Comm.Rank()
```

Java binding of the MPI operation *MPI_COMM_RANK*. The return value is *rank*.

```
static int Compare(Comm comm1, Comm comm2)
```

Java binding of the MPI operation *MPI_COMM_COMPARE*. The return value is *result*.

The constants *MPI_IDENT*, *MPI_CONGRUENT*, *MPI_SIMILAR*, *MPI_UNEQUAL* are available as `MPI.IDENT`, `MPI.CONGRUENT`, `MPI.SIMILAR`, `MPI.UNEQUAL`.

```
Object Comm.clone()
```

Java binding of the MPI operation *MPI_COMM_DUP*. The return value is *newcomm*.

```
Intracomm Intracomm.Create(Group group)
```

Java binding of the MPI operation *MPI_COMM_CREATE*. The return value is *newcomm*.

```
Intracomm Intracomm.Split(int colour, int key)
```

Java binding of the MPI operation *MPI_COMM_SPLIT*. The return value is *newcomm*.

```
void Comm.Free()
```

Java binding of the MPI operation *MPI_COMM_FREE*. (Note: a *Free* method is required rather than a *finalize* method, because *MPI_COMM_FREE* is, strictly-speaking, a collective operations, and we cannot assume that the Java garbage collector would call a *finalize* method synchronously on all processors.)

## 5.5 Motivating Examples

No special issues for Java binding.

## 5.6 Inter-Communication

```
boolean Comm.Test_inter()
```

Java binding of the MPI operation *MPI_COMM_TEST_INTER*. The return value is *flag*.

```
int Intercomm.Remote_size()
```

Java binding of the MPI operation *MPI_COMM_REMOTE_SIZE*. The return value is *size*.

```
Group Intercomm.Remote_group()
```

Java binding of the MPI operation *MPI_COMM_REMOTE_GROUP*. The return value is *group*.

```
Intercomm Comm.Create_intercomm(Comm local_comm, int local_leader,
                                int remote_leader, int tag)
```

Java binding of the MPI operation *MPI_INTERCOMM_CREATE*. The class instance is *peer_comm*. The return value is *newintercomm*.

```
Intracomm Intercomm.merge(boolean high)
```

Java binding of the MPI operation *MPI_INTERCOMM_MERGE*. The class instance is *intercomm*. The return value is *newintracomm*.

### 5.7 Caching

It is assumed that, to achieve the effect of caching attributes in user-customized communicator, programmers will create subclasses of the library-defined communicator classes.

```
void Comm.Attr_put(int keyval, Object attribute_val)
```

Java binding of the MPI operation *MPI_ATTR_PUT*.

```
Object Comm.Attr_get(int keyval)
```

Java binding of the MPI operation *MPI_ATTR_GET*. If *flag* is false, return value is `null`. Otherwise return value is *attribute_val*

```
void Comm.Attr_delete(int keyval)
```

Java binding of the MPI operation *MPI_ATTR_DELETE*.

## 6 Process Topologies

### 6.1 Introduction

Communicators with Cartesian or graph topologies will be realized as instances of the subclasses `Cartcomm` or `Graphcomm`, respectively of `Intracomm`.

### 6.2 Virtual Topologies

No special issues for Java binding.

### 6.3 Embedding in MPI

No special issues for Java binding.

## 6.4 Overview of the Functions

No special issues for Java binding.

## 6.5 Topology Constructors

```
Cartcomm Intracomm.Create_cart(int [] dims, boolean [] periods,
                               boolean reorder)
```

Java binding of the MPI operation *MPI_CART_CREATE*. The class instance is *comm_old*. The value of *ndims* is taken to be size of the `dims` argument. The constructed object is *comm_cart*.

```
static int [] Cartcomm.Dims_create(int nnodes, int ndims)
```

Java binding of the MPI operation *MPI_DIMS_CREATE*. The return value is *dims*.

```
Graphcomm Intracomm.Create_graph(int [] index, int [] edges,
                                 boolean reorder)
```

Java binding of the MPI operation *MPI_GRAPH_CREATE*. The class instance is *comm_old*. The value of *nnodes* is taken to be size of the `index` argument. The constructed object is *comm_graph*.

```
int Comm.Topo_test()
```

Java binding of the MPI operation *MPI_Topo_TEST*. The return value is *status*. It will be one of `MPI.GRAPH`, `MPI.CART` or `MPI.UNDEFINED`.

```
GraphParms Graphcomm.Get()
```

Java binding of the MPI operations *MPI_GRAPHDIMS_GET* and *MPI_GRAPH_GET*. The return value contains *index* and *edges*. The values of *nnodes* and *nedges* can be extracted from the sizes of these arrays.

```
CartParms Cartcomm.Get()
```

Java binding of the MPI operations *MPI_CARTDIM_GET* and *MPI_CART_GET*. The return value contains *dims* and *periods* and *coords*. The value of *ndims* can be extracted from the size of *dims*.

```
int Cartcomm.Rank(int [] coords)
```

Java binding of the MPI operation *MPI_CART_RANK*. The return value is *rank*.

```
int [] Cartcomm.Coords(int rank)
```

Java binding of the MPI operation *MPI_CART_COORDS* and *MPI_CARTDIM_GET* . The return value is *coords*.

```
int [] Graphcomm.Neighbours(int rank)
```

Java binding of the MPI operation *MPI_GRAPH_NEIGHBOURS_COUNT* and *MPI_GRAPH_NEIGHBOURS*. The return value is *neighbours*. The value of *nneighbours* can be extracted from the size *neighbours*.

```
ShiftParms Cartcomm.Shift(int direction, int disp)
```

Java binding of the MPI operation *MPI_CART_SHIFT*. The return value contains *rank_source* and *rank_dest*.

```
Cartcomm Cartcomm.Sub(boolean [] remain_dims)
```

Java binding of the MPI operation *MPI_CART_SUB*. The return value is *newcomm*.

```
int Cartcomm.Map(int [] dims, boolean [] periods)
```

Java binding of the MPI operation *MPI_CART_MAP*. The value of *ndims* is taken to be size of the `dims` argument. The return value is *newrank*.

```
int Graphcomm.Map(int [] index, int [] edges)
```

Java binding of the MPI operation *MPI_GRAPH_MAP*. The value of *nnodes* is taken to be size of the `index` argument. The return value is *newrank*.

# 7 MPI Environmental Management

## 7.1 Implementation information

The constants *MPI_TAG_UB*, *MPI_HOST* and *MPI_IO* are available as `MPI.TAG_UB`, `MPI.HOST`, `MPI.IO`.

```
static String MPI.Get_processor_name()
```

Java binding of the MPI operation *MPI_GET_PROCESSOR_NAME*. The return value is *name*.

## 7.2 Error handling

The constants *MPI_ERRORS_ARE_FATAL*, *MPI_ERRORS_RETURN* are available as `MPI.ERRORS_ARE_FATAL`, `MPI.ERRORS_RETURN`.

If the effective error handler is *MPI_ERRORS_RETURN*, the wrapper codes will throw appropriate Java exceptions (see section 7.3).

For now, we omit a specification of Java interface for creating new MPI error handlers, because the detailed interface of the handler function depends on unstandardized features of the MPI implementation.

```
static void MPI.Errorhandler_set(Errhandler errhandler)
```

Java binding of the MPI operation *MPI_ERRORHANDLER_SET*.

```
static Errhandler MPI.Errorhandler_get()
```

Java binding of the MPI operation *MPI_ERRORHANDLER_GET*. The return value is *errhandler*.

## 7.3 Error codes and classes

The `IOException` subclasses

```
MPIErrBuffer
MPIErrCount
MPIErrType
MPIErrTag
MPIErrComm
MPIErrRank
MPIErrRequest
MPIErrRoot
```

```
MPIErrGroup
MPIErrOp
MPIErrTopology
MPIErrDims
MPIErrArg
MPIErrUnknown
MPIErrTruncate
MPIErrOther
MPIErrIntern
```

correspond to the standard MPI error classes.

## 7.4 Timers

`static double MPI.Wtime()`

Java binding of the MPI operation *MPI_WTIME*.

`static double MPI.Wtick()`

Java binding of the MPI operation *MPI_WTICK*.

## 7.5 Startup

`static void MPI.Init(String[] argv)`

Java binding of the MPI operation *MPI_INIT*. The value of *argc* is taken to be size of the `argv` argument.

`static void MPI.Finalize()`

Java binding of the MPI operation *MPI_FINALIZE*.

`static boolean MPI.Initialized()`

Java binding of the MPI operation *MPI_INITIALIZED*. The return value is *flag*.

`void Comm.Abort(int errorcode)`

Java binding of the MPI operation *MPI_ABORT*.

# A Full public interface of classes

Comments included in the class definitions are generally cross-references to names of sections in the main document, where members following the comment are defined.

## A.1 MPI

```
public class MPI {
  public static Intracomm COMM_WORLD;

  public static Datatype BYTE, CHAR, SHORT, BOOLEAN, INT, LONG,
                         FLOAT, DOUBLE, PACKED, LB, UB ;

  public static int ANY_SOURCE, ANY_TAG ;

  public static int PROC_NULL ;

  public static int BSEND_OVERHEAD ;

  public static int UNDEFINED ;

  public static Op MAX, MIN, SUM, PROD, LAND, BAND,
                   LOR, BOR, LXOR, BXOR, MINLOC, MAXLOC ;

  public static Datatype SHORT2, INT2, LONG2, FLOAT2, DOUBLE2 ;

  public static Group GROUP_EMPTY ;

  public static Comm COMM_SELF ;

  public static int IDENT, CONGRUENT, SIMILAR, UNEQUAL ;

  public static int GRAPH, CART ;

  public static ErrHandler ERRORS_ARE_FATAL, ERRORS_RETURN ;

  public static int TAG_UB, HOST, IO ;

  public static Request REQUEST_NULL;

  // Buffer allocation and usage

  public static void Buffer_attach(byte [] buffer) {...}

  public static void Buffer_detach(byte [] buffer) {...}
```

```
  // Environmental Management

  public static void Init(String[] argv) {...}

  public static void Finalize() {...}

  public static String Get_processor_name() {...}

  public static void Errorhandler_set(Errhandler errhandler) {...}

  public static Errhandler Errorhandler_get() {...}

  public static double Wtime() {...}

  public static double Wtick() {...}

  public static boolean Initialized() {...}

  ...
}

public class Errhandler {
  ...
}
```

## A.2 Comm

```
public class Comm {

  // Communicator Management

  public int Size() {...}

  public int Rank() {...}

  public Group Comm.Group() {...}  // (see ``Group management'')

  public static int Compare(Comm comm1, Comm comm2) {...}

  public Object clone() {...}

  public void Free() {...}


  // Inter-communication

  public boolean Test_inter() {...}

  public Intercomm Create_intercomm(Comm local_comm, int local_leader,
                                   int remote_leader, int tag) {...}


  // Caching

  public void Attr_put(int keyval, Object attribute_val) {...}

  public Object Attr_get(int keyval) {...}

  public void Attr_delete(int keyval) {...}


  // Blocking Send and Receive operations

  public void Send(Object buf, int offset, int count,
                   Datatype datatype, int dest, int tag) {...}

  public Status Recv(Object buf, int offset, int count,
                     Datatype datatype, int source, int tag) {...}


  // Communication Modes

  public void Bsend(Object buf, int offset, int count,
```

```
                  Datatype datatype, int dest, int tag) {...}

public void Ssend(Object buf, int offset, int count,
                  Datatype datatype, int dest, int tag) {...}

public void Rsend(Object buf, int offset, int count,
                  Datatype datatype, int dest, int tag) {...}


// Nonblocking communication

public Request Isend(Object buf, int offset, int count,
                     Datatype datatype, int dest, int tag) {...}

public Request Ibsend(Object buf, int offset, int count,
                      Datatype datatype, int dest, int tag) {...}

public Request Issend(Object buf, int offset, int count,
                      Datatype datatype, int dest, int tag) {...}

public Request Irsend(Object buf, int offset, int count,
                      Datatype datatype, int dest, int tag) {...}

public Request Irecv(Object buf, int offset, int count,
                    Datatype datatype, int source, int tag) {...}


// Probe and cancel

public Status Iprobe(int source, int tag) {...}

public Status Probe(int source, int tag) {...}


// Persistent communication requests

public Prequest Send_init(Object buf, int offset, int count,
                          Datatype datatype, int dest, int tag) {...}

public Prequest Bsend_init(Object buf, int offset, int count,
                           Datatype datatype, int dest, int tag) {...}

public Prequest Ssend_init(Object buf, int offset, int count,
                           Datatype datatype, int dest, int tag) {...}

public Prequest Rsend_init(Object buf, int offset, int count,
                           Datatype datatype, int dest, int tag) {...}
```

```
  public Prequest Recv_init(Object buf, int offset, int count,
                            Datatype datatype, int source, int tag) {...}


  // Send-receive

  public Status Sendrecv(Object sendbuf, int sendoffset,
                         int sendcount, Datatype sendtype,
                         int dest, int sendtag,
                         Object recvbuf, int recvoffset,
                         int recvcount, Datatype recvtype,
                         int source, int recvtag) {...}

  public Status Sendrecv_replace(Object buf, int offset,
                         int count, Datatype datatype,
                         int dest, int sendtag,
                         int source, int recvtag) {...}

  // Pack and unpack

  public int Pack(Object inbuf, int offset, int incount,
                  Datatype datatype,
                  byte [] outbuf, int position) {...}

  public int Unpack(byte [] inbuf, int position,
                    Object outbuf, int offset, int outcount,
                    Datatype datatype) {...}

  public int Pack_size(int incount, Datatype datatype) {...}


  // Process Topologies

  int Topo_test()


  // Environmental Management

  void Abort(int errorcode) {...}

  ...
}
```

### A.3 Intracomm and Intercomm

```
public class Intracomm extends Comm {

  public Object clone() { ... }

  public Intracomm Create(Group group) {...}

  public Intracomm Split(int colour, int key) {...}

  // Collective communication

  public void Barrier() {...}

  public void Bcast(Object buffer, int offset, int count,
                    Datatype datatype, int root) {...}

  public void Gather(Object sendbuf, int sendoffset,
                     int sendcount, Datatype sendtype,
                     Object recvbuf, int recvoffset,
                     int recvcount, Datatype recvtype, int root) {...}

  public void Gatherv(Object sendbuf, int sendoffset,
                      int sendcount, Datatype sendtype,
                      Object recvbuf, int recvoffset,
                      int [] recvcount, int [] displs,
                      Datatype recvtype, int root) {...}

  public void Scatter(Object sendbuf, int sendoffset,
                      int sendcount, Datatype sendtype,
                      Object recvbuf, int recvoffset,
                      int recvcount, Datatype recvtype, int root) {...}

  public void Scatterv(Object sendbuf, int sendoffset,
                       int [] sendcount, int [] displs,
                       Datatype sendtype,
                       Object recvbuf, int recvoffset,
                       int recvcount, Datatype recvtype, int root) {...}

  public void Allgather(Object sendbuf, int sendoffset,
                        int sendcount, Datatype sendtype,
                        Object recvbuf, int recvoffset,
                        int recvcount, Datatype recvtype) {...}

  public void Allgatherv(Object sendbuf, int sendoffset,
                         int sendcount, Datatype sendtype,
                         Object recvbuf, int recvoffset,
                         int [] recvcount, int [] displs,
```

```
                       Datatype recvtype) {...}

  public void Alltoall(Object sendbuf, int sendoffset,
                     int sendcount, Datatype sendtype,
                     Object recvbuf, int recvoffset,
                     int recvcount, Datatype recvtype) {...}

  public void Alltoallv(Object sendbuf, int sendoffset,
                      int [] sendcount, int [] sdispls,
                      Datatype sendtype,
                      Object recvbuf, int recvoffset,
                      int [] recvcount, int [] rdispls,
                      Datatype recvtype) {...}

  public void Reduce(Object sendbuf, int sendoffset,
                   Object recvbuf, int recvoffset,
                   int count, Datatype datatype,
                   Op op, int root) {...}

  public void Allreduce(Object sendbuf, int sendoffset,
                      Object recvbuf, int recvoffset,
                      int count, Datatype datatype,
                      Op op) {...}

  public void Reduce_scatter(Object sendbuf, int sendoffset,
                           Object recvbuf, int recvoffset,
                           int [] recvcounts, Datatype datatype,
                           Op op) {...}

  public void Scan(Object sendbuf, int sendoffset,
                 Object recvbuf, int recvoffset,
                 int count, Datatype datatype,
                 Op op) {...}


  // Topology Constructors

  public Graphcomm Create_graph(int [] index, int [] edges,
                               boolean reorder) {...}

  public Cartcomm Create_cart(int [] dims, boolean [] periods,
                                      boolean reorder) {...}

  ...
}

public class Intercomm extends Comm {
```

```
  public Object clone() { ... }

  // Inter-communication

  public int Remote_size() {...}

  public Group Remote_group() {...}

  public Intracomm Merge(boolean high) {...}

  ...
}
```

## A.4 Op

```
public class Op {
  Op(User_function function, boolean commute) {...}

  void finalize() {...}

  ...
}
```

### A.5 Group

```
public class Group {

  // Group Management

  public int Size() {...}

  public int Rank() {...}

  public static int [] Translate_ranks(Group group1, int [] ranks1,
                                       Group group2) {...}

  public static int Compare(Group group1, Group group2) {...}

  public static Group Union(Group group1, Group group2) {...}

  public static Group Intersection(Group group1, Group group2) {...}

  public static Group Difference(Group group1, Group group2) {...}

  public Group Incl(int [] ranks) {...}

  public Group Excl(int [] ranks) {...}

  public Group Range_incl(int [] [] ranges) {...}

  public Group Range_excl(int [] [] ranges) {...}

  public void finalize() {...}

  ...
}
```

## A.6 Status

```
public class Status {

  public int source;
  public int tag;

  public int index ;


  // Blocking Send and Receive operations

  public int Get_count(Datatype datatype) {...}


  // Probe and Cancel

  public boolean Test_cancelled() {...}


  // Derived datatypes

  public int Get_elements(Datatype datatype) {...}

  ...
}
```

## A.7 Request and Prequest

```
public class Request {

  // Nonblocking communication

  public Status Wait() {...}

  public Status Test() {...}

  public void finalize() {...}

  public boolean Is_null() {...}

  public static Status Waitany(Request [] array_of_requests) {...}

  public static Status Testany(Request [] array_of_requests) {...}

  public static Status [] Waitall(Request [] array_of_requests) {...}

  public static Status [] Testall(Request [] array_of_requests) {...}

  public static Status []
               Waitsome(Request [] array_of_requests) {...}

  public static Status []
               Testsome(Request [] array_of_requests) {...}


  // Probe and cancel

  public void Cancel() {...}

  ...
}

public class Prequest extends Request {

  // Persistent communication requests

  public void Start() {...}

  public static void Startall(Request [] array_of_requests) {...}

  ...
}
```

## A.8 Datatype

```
public class Datatype {

  // Derived datatypes

  public static Datatype Contiguous(int count, Datatype oldtype) {...}

  public static Datatype Vector(int count, int blocklength, int stride,
                               Datatype oldtype) {...}

  public static Datatype Hvector(int count, int blocklength, int stride,
                                Datatype oldtype) {...}

  public static Datatype Indexed(int [] array_of_blocklengths,
                                int [] array_of_displacements,
                                Datatype oldtype) {...}

  public static Datatype Hindexed(int [] array_of_blocklengths,
                                 int [] array_of_displacements,
                                 Datatype oldtype) {...}

  public static Datatype Struct(int [] array_of_blocklengths,
                               int [] array_of_displacements,
                               Datatype [] array_of_types) {...}

  public int Extent() {...}

  public int Size() {...}

  public int Lb() {...}

  public int Ub() {...}

  public void Commit() {...}

  public void finalize() {...}

  ...
}
```

### A.9 Classes for virtual topologies

```
public class Cartcomm extends Intracomm {

  public Object clone() { ... }

  // Topology Constructors

  static public int [] Dims_create(int nnodes, int ndims) {...}

  public CartParms Get() {...}

  public int Rank(int [] coords) {...}

  public int [] Coords(int rank) {...}

  public ShiftParms Shift(int direction, int disp) {...}

  public Cartcomm Sub(boolean [] remain_dims) {...}

  public int Map(int [] dims, boolean [] periods) {...}
}

public class CartParms {

  // Return type for Cartcomm.Get()

  public int [] dims ;
  public booleans [] periods ;
  public int [] coords ;
}

public class ShiftParms {

  // Return type for Cartcomm.Shift()

  public int rank_source ;
  public int rank_dest ;
}

public class Graphcomm extends Intracomm {

  public Object clone() { ... }

  // Topology Constructors

  public GraphParms Get() {...}

  public int [] Neighbours(int rank) {...}
```

```
  public int Map(int [] index, int [] edges) {...}
}

public class GraphParms {

  // Return type for Graphcomm.Get()

  public int [] index ;
  public int [] edges ;
}
```