MPI datatype	C datatype
MPI_CHAR	`char` (treated as printable character)
MPI_SHORT	`signed short int`
MPI_INT	`signed int`
MPI_LONG	`signed long int`
MPI_LONG_LONG_INT	`signed long long int`
MPI_LONG_LONG (as a synonym)	`signed long long int`
MPI_SIGNED_CHAR	`signed char` (treated as integral value)
MPI_UNSIGNED_CHAR	`unsigned char` (treated as integral value)
MPI_UNSIGNED_SHORT	`unsigned short int`
MPI_UNSIGNED	`unsigned int`
MPI_UNSIGNED_LONG	`unsigned long int`
MPI_UNSIGNED_LONG_LONG	`unsigned long long int`
MPI_SHORT_FLOAT	`short float`
MPI_FLOAT	`float`
MPI_DOUBLE	`double`
MPI_LONG_DOUBLE	`long double`
MPI_FLOAT16	`float16_t`
MPI_FLOAT32	`float32_t`
MPI_FLOAT64	`float64_t`
MPI_FLOAT128	`float128_t`
MPI_WCHAR	`wchar_t` (defined in `<stddef.h>`) (treated as printable character)
MPI_C_BOOL	`_Bool`
MPI_INT8_T	`int8_t`
MPI_INT16_T	`int16_t`
MPI_INT32_T	`int32_t`
MPI_INT64_T	`int64_t`
MPI_UINT8_T	`uint8_t`
MPI_UINT16_T	`uint16_t`
MPI_UINT32_T	`uint32_t`
MPI_UINT64_T	`uint64_t`
MPI_C_COMPLEX	`float _Complex`
MPI_C_FLOAT_COMPLEX (as a synonym)	`float _Complex`
MPI_C_SHORT_FLOAT_COMPLEX	`short float _Complex`
MPI_C_DOUBLE_COMPLEX	`double _Complex`
MPI_C_LONG_DOUBLE_COMPLEX	`long double _Complex`
MPI_BYTE	
MPI_PACKED	

Table 3.2: Predefined MPI datatypes corresponding to C datatypes

MPI datatype	C datatype	Fortran datatype
MPI_AINT	MPI_Aint	INTEGER (KIND=MPI_ADDRESS_KIND)
MPI_OFFSET	MPI_Offset	INTEGER (KIND=MPI_OFFSET_KIND)
MPI_COUNT	MPI_Count	INTEGER (KIND=MPI_COUNT_KIND)

Table 3.3: Predefined MPI datatypes corresponding to both C and Fortran datatypes

INTEGER (KIND=MPI_ADDRESS_KIND), INTEGER (KIND=MPI_OFFSET_KIND), and INTEGER (KIND=MPI_COUNT_KIND). This is described in Table 3.3. All predefined datatype handles are available in all language bindings. See Sections 17.2.6 and 17.2.10 on page 662 and 670 for information on interlanguage communication with these types.

If there is an accompanying C++ compiler then the datatypes in Table 3.4 are also supported in C and Fortran.

MPI datatype	C++ datatype
MPI_CXX_BOOL	bool
MPI_CXX_SHORT_FLOAT_COMPLEX	std::complex<short float>
MPI_CXX_FLOAT_COMPLEX	std::complex<float>
MPI_CXX_DOUBLE_COMPLEX	std::complex<double>
MPI_CXX_LONG_DOUBLE_COMPLEX	std::complex<long double>

Table 3.4: Predefined MPI datatypes corresponding to C++ datatypes

3.2.3 Message Envelope

In addition to the data part, messages carry information that can be used to distinguish messages and selectively receive them. This information consists of a fixed number of fields, which we collectively call the **message envelope**. These fields are

source
destination
tag
communicator

The message source is implicitly determined by the identity of the message sender. The other fields are specified by arguments in the send operation.

The message destination is specified by the dest argument.

The integer-valued message tag is specified by the tag argument. This integer can be used by the program to distinguish different types of messages. The range of valid tag values is $0, \ldots, \mathsf{UB}$, where the value of UB is implementation dependent. It can be found by querying the value of the attribute MPI_TAG_UB, as described in Chapter 8. MPI requires that UB be no less than 32767.

The comm argument specifies the **communicator** that is used for the send operation. Communicators are explained in Chapter 6; below is a brief summary of their usage.

A communicator specifies the communication context for a communication operation. Each communication context provides a separate "communication universe": messages are

	MPI_TYPE_CREATE_F90_REAL, and if available: MPI_SHORT_FLOAT, MPI_REAL2, MPI_REAL4, MPI_REAL8, MPI_REAL16
Logical:	MPI_LOGICAL, MPI_C_BOOL, MPI_CXX_BOOL
Complex:	MPI_COMPLEX, MPI_C_COMPLEX, MPI_C_FLOAT_COMPLEX (as synonym), MPI_C_DOUBLE_COMPLEX, MPI_C_LONG_DOUBLE_COMPLEX, MPI_CXX_FLOAT_COMPLEX, MPI_CXX_DOUBLE_COMPLEX, MPI_CXX_LONG_DOUBLE_COMPLEX, and handles returned from MPI_TYPE_CREATE_F90_COMPLEX, and if available: MPI_DOUBLE_COMPLEX, MPI_C_SHORT_FLOAT_COMPLEX, MPI_CXX_SHORT_FLOAT_COMPLEX, MPI_COMPLEX4, MPI_COMPLEX8, MPI_COMPLEX16, MPI_COMPLEX32
Byte:	MPI_BYTE
Multi-language types:	MPI_AINT, MPI_OFFSET, MPI_COUNT

Now, the valid datatypes for each operation are specified below.

Op	Allowed Types
MPI_MAX, MPI_MIN	C integer, Fortran integer, Floating point, Multi-language types
MPI_SUM, MPI_PROD	C integer, Fortran integer, Floating point, Complex, Multi-language types
MPI_LAND, MPI_LOR, MPI_LXOR	C integer, Logical
MPI_BAND, MPI_BOR, MPI_BXOR	C integer, Fortran integer, Byte, Multi-language types

These operations together with all listed datatypes are valid in all supported programming languages, see also Reduce Operations in Section 17.2.6.

The following examples use intracommunicators.

Example 5.15

A routine that computes the dot product of two vectors that are distributed across a group of processes and returns the answer at node zero.

according to the first component of each pair, and ties are resolved according to the second component.

The reduce operation is defined to operate on arguments that consist of a pair: value and index. For both Fortran and C, types are provided to describe the pair. The potentially mixed-type nature of such arguments is a problem in Fortran. The problem is circumvented, for Fortran, by having the MPI-provided type consist of a pair of the same type as value, and coercing the index to this type also. In C, the MPI-provided pair type has distinct types and the index is an `int`.

In order to use MPI_MINLOC and MPI_MAXLOC in a reduce operation, one must provide a datatype argument that represents a pair (value and index). MPI provides nine such predefined datatypes. The operations MPI_MAXLOC and MPI_MINLOC can be used with each of the following datatypes.

Fortran:

Name	Description
MPI_2REAL	pair of `REAL`s
MPI_2DOUBLE_PRECISION	pair of `DOUBLE PRECISION` variables
MPI_2INTEGER	pair of `INTEGER`s

C:

Name	Description
MPI_SHORT_FLOAT_INT	`short float` and `int`
MPI_FLOAT_INT	`float` and `int`
MPI_DOUBLE_INT	`double` and `int`
MPI_LONG_INT	`long` and `int`
MPI_2INT	pair of `int`
MPI_SHORT_INT	`short` and `int`
MPI_LONG_DOUBLE_INT	`long double` and `int`

The datatype MPI_2REAL is *as if* defined by the following (see Section 4.1).

```
MPI_Type_contiguous(2, MPI_REAL, MPI_2REAL);
```

Similar statements apply for MPI_2INTEGER, MPI_2DOUBLE_PRECISION, and MPI_2INT.

The datatype MPI_SHORT_INT is *as if* defined by the following sequence of instructions.

```
struct mystruct {
    short val;
    int rank;
};
type[0] = MPI_SHORT;
type[1] = MPI_INT;
disp[0] = 0;
disp[1] = offsetof(struct mystruct, rank);
block[0] = 1;
block[1] = 1;
MPI_Type_create_struct(2, block, disp, type, MPI_SHORT_INT);
```

Type	Length
MPI_PACKED	1
MPI_BYTE	1
MPI_CHAR	1
MPI_UNSIGNED_CHAR	1
MPI_SIGNED_CHAR	1
MPI_WCHAR	2
MPI_SHORT	2
MPI_UNSIGNED_SHORT	2
MPI_INT	4
MPI_UNSIGNED	4
MPI_LONG	4
MPI_UNSIGNED_LONG	4
MPI_LONG_LONG_INT	8
MPI_UNSIGNED_LONG_LONG	8
MPI_SHORT_FLOAT	2
MPI_FLOAT	4
MPI_DOUBLE	8
MPI_LONG_DOUBLE	16
MPI_C_BOOL	1
MPI_INT8_T	1
MPI_INT16_T	2
MPI_INT32_T	4
MPI_INT64_T	8
MPI_UINT8_T	1
MPI_UINT16_T	2
MPI_UINT32_T	4
MPI_UINT64_T	8
MPI_AINT	8
MPI_COUNT	8
MPI_OFFSET	8
MPI_FLOAT16	2
MPI_FLOAT32	4
MPI_FLOAT64	8
MPI_FLOAT128	16
MPI_C_SHORT_FLOAT_COMPLEX	2*2
MPI_C_COMPLEX	2*4
MPI_C_FLOAT_COMPLEX	2*4
MPI_C_DOUBLE_COMPLEX	2*8
MPI_C_LONG_DOUBLE_COMPLEX	2*16

Optional Type	Length
MPI_CHARACTER	1
MPI_LOGICAL	4
MPI_INTEGER	4
MPI_SHORT_FLOAT	2
MPI_REAL	4
MPI_DOUBLE_PRECISION	8
MPI_COMPLEX	2*4
MPI_DOUBLE_COMPLEX	2*8
MPI_INTEGER1	1
MPI_INTEGER2	2
MPI_INTEGER4	4
MPI_INTEGER8	8
MPI_INTEGER16	16
MPI_REAL2	2
MPI_REAL4	4
MPI_REAL8	8
MPI_REAL16	16
MPI_COMPLEX2	2*2
MPI_COMPLEX4	2*4
MPI_COMPLEX8	2*8
MPI_COMPLEX16	2*16
MPI_COMPLEX32	2*32
MPI_C_SHORT_FLOAT_COMPLEX	2*2
MPI_C_SHORT_FLOAT	2

C++ Types	Length
MPI_CXX_BOOL	1
MPI_CXX_SHORT_FLOAT_COMPLEX	2*2
MPI_CXX_FLOAT_COMPLEX	2*4
MPI_CXX_DOUBLE_COMPLEX	2*8
MPI_CXX_LONG_DOUBLE_COMPLEX	2*16

Table 13.2: "external32" sizes of predefined datatypes

MPI must provide a named size-specific datatype. The name of this datatype is of the form MPI_<TYPE>n in C and Fortran where <TYPE> is one of REAL, INTEGER and COMPLEX, and **n** is the length in bytes of the machine representation. This datatype locally matches all variables of type (**typeclass**, **n**) in Fortran. The list of names for such types includes:

```
MPI_REAL2
MPI_REAL4
MPI_REAL8
MPI_REAL16
MPI_COMPLEX2
MPI_COMPLEX4
MPI_COMPLEX8
MPI_COMPLEX16
MPI_COMPLEX32
MPI_INTEGER1
MPI_INTEGER2
MPI_INTEGER4
MPI_INTEGER8
MPI_INTEGER16
```

One datatype is required for each representation supported by the Fortran compiler.

> *Rationale.* Particularly for the longer floating-point types, C and Fortran may use different representations. For example, a Fortran compiler may define a 16-byte REAL type with 33 decimal digits of precision while a C compiler may define a 16-byte long double type that implements an 80-bit (10 byte) extended precision floating point value. Both of these types are 16 bytes long, but they are not interoperable. Thus, these types are defined by Fortran, even though C may define types of the same length. (*End of rationale.*)

To be backward compatible with the interpretation of these types in MPI-1, we assume that the nonstandard declarations `REAL*n`, `INTEGER*n`, always create a variable whose representation is of size **n**. These datatypes may also be used for variables declared with `KIND=INT8/16/32/64` or `KIND=REAL32/64/128`, which are defined in the `ISO_FORTRAN_ENV` intrinsic module. Note that the MPI datatypes and the `REAL*n`, `INTEGER*n` declarations count bytes whereas the Fortran `KIND` values count bits. All these datatypes are predefined.

The following functions allow a user to obtain a size-specific MPI datatype for any intrinsic Fortran type.

MPI_SIZEOF(x, size)

IN	x	a Fortran variable of numeric intrinsic type (choice)
OUT	size	size of machine representation of that type (integer)

```
MPI_Sizeof(x, size, ierror)
    TYPE(*), DIMENSION(..)  ::  x
    INTEGER, INTENT(OUT) ::  size
    INTEGER, OPTIONAL, INTENT(OUT) ::  ierror
```

Named Predefined Datatypes	C types
C type: `MPI_Datatype`	
Fortran type: `INTEGER`	
or `TYPE(MPI_Datatype)`	
MPI_CHAR	`char`
	(treated as printable character)
MPI_SHORT	`signed short int`
MPI_INT	`signed int`
MPI_LONG	`signed long`
MPI_LONG_LONG_INT	`signed long long`
MPI_LONG_LONG (as a synonym)	`signed long long`
MPI_SIGNED_CHAR	`signed char`
	(treated as integral value)
MPI_UNSIGNED_CHAR	`unsigned char`
	(treated as integral value)
MPI_UNSIGNED_SHORT	`unsigned short`
MPI_UNSIGNED	`unsigned int`
MPI_UNSIGNED_LONG	`unsigned long`
MPI_UNSIGNED_LONG_LONG	`unsigned long long`
MPI_FLOAT	`float`
MPI_DOUBLE	`double`
MPI_LONG_DOUBLE	`long double`
MPI_WCHAR	`wchar_t`
	(defined in `<stddef.h>`)
	(treated as printable character)
MPI_C_BOOL	`_Bool`
MPI_INT8_T	`int8_t`
MPI_INT16_T	`int16_t`
MPI_INT32_T	`int32_t`
MPI_INT64_T	`int64_t`
MPI_UINT8_T	`uint8_t`
MPI_UINT16_T	`uint16_t`
MPI_UINT32_T	`uint32_t`
MPI_UINT64_T	`uint64_t`
MPI_AINT	`MPI_Aint`
MPI_COUNT	`MPI_Count`
MPI_OFFSET	`MPI_Offset`
MPI_FLOAT32	`float32_t`
MPI_FLOAT64	`float64_t`
MPI_FLOAT128	`float128_t`
MPI_C_COMPLEX	`float _Complex`
MPI_C_FLOAT_COMPLEX	`float _Complex`
MPI_C_DOUBLE_COMPLEX	`double _Complex`
MPI_C_LONG_DOUBLE_COMPLEX	`long double _Complex`
MPI_BYTE	(any C type)
MPI_PACKED	(any C type)

Named Predefined Datatypes	Fortran types
C type: MPI_Datatype Fortran type: INTEGER or TYPE(MPI_Datatype)	
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)
MPI_AINT	INTEGER (KIND=MPI_ADDRESS_KIND)
MPI_COUNT	INTEGER (KIND=MPI_COUNT_KIND)
MPI_OFFSET	INTEGER (KIND=MPI_OFFSET_KIND)
MPI_BYTE	(any Fortran type)
MPI_PACKED	(any Fortran type)

Named Predefined Datatypes[1]	C++ types
C type: MPI_Datatype Fortran type: INTEGER or TYPE(MPI_Datatype)	
MPI_CXX_BOOL	bool
MPI_CXX_FLOAT_COMPLEX	std::complex<float>
MPI_CXX_DOUBLE_COMPLEX	std::complex<double>
MPI_CXX_LONG_DOUBLE_COMPLEX	std::complex<long double>

[1] If an accompanying C++ compiler is missing, then the MPI datatypes in this table are not defined.

Optional datatypes (C)	C types
C type: MPI_Datatype Fortran type: INTEGER or TYPE(MPI_Datatype)	
MPI_SHORT_FLOAT	short float
MPI_FLOAT16	float16_t
MPI_C_SHORT_COMPLEX	short _Complex

Optional datatypes (C++)	C++ types
C type: MPI_Datatype Fortran type: INTEGER or TYPE(MPI_Datatype)	
MPI_CXX_SHORT_FLOAT_COMPLEX	std::complex<short float>

Optional datatypes (Fortran)	Fortran types
C type: `MPI_Datatype`	
Fortran type: `INTEGER`	
or `TYPE(MPI_Datatype)`	
MPI_DOUBLE_COMPLEX	`DOUBLE COMPLEX`
MPI_INTEGER1	`INTEGER*1`
MPI_INTEGER2	`INTEGER*2`
MPI_INTEGER4	`INTEGER*4`
MPI_INTEGER8	`INTEGER*8`
MPI_INTEGER16	`INTEGER*16`
MPI_REAL2	`REAL*2`
MPI_REAL4	`REAL*4`
MPI_REAL8	`REAL*8`
MPI_REAL16	`REAL*16`
MPI_COMPLEX2	`COMPLEX*2`
MPI_COMPLEX4	`COMPLEX*4`
MPI_COMPLEX8	`COMPLEX*8`
MPI_COMPLEX16	`COMPLEX*16`
MPI_COMPLEX32	`COMPLEX*32`

Datatypes for reduction functions (C)
C type: `MPI_Datatype`
Fortran type: `INTEGER` or `TYPE(MPI_Datatype)`
MPI_SHORT_FLOAT_INT (optional)
MPI_FLOAT_INT
MPI_DOUBLE_INT
MPI_LONG_INT
MPI_2INT
MPI_SHORT_INT
MPI_LONG_DOUBLE_INT

Datatypes for reduction functions (Fortran)
C type: `MPI_Datatype`
Fortran type: `INTEGER` or `TYPE(MPI_Datatype)`
MPI_2REAL
MPI_2DOUBLE_PRECISION
MPI_2INTEGER

Reserved communicators
C type: `MPI_Comm`
Fortran type: `INTEGER` or `TYPE(MPI_Comm)`
MPI_COMM_WORLD
MPI_COMM_SELF

Communicator split type constants
C type: `const int` (or unnamed `enum`)
Fortran type: `INTEGER`
MPI_COMM_TYPE_SHARED