TclMPI

1.1

Generated by Doxygen 1.8.20

1 Main Page 1

1.0.1 Homepage: . . . 1
1.0.2 Test Status: . . . 1
1.0.3 Citing: . . . 1
1.0.4 Acknowledgements: . . . 1

2 Copyright and License for TclMPI 3

3 TclMPI User's Guide 5

3.1 Compilation and Installation . . . 5
3.2 Software Development and Bug Reports . . . 6
3.3 Example Programs . . . 6
3.3.1 Hello World . . . 6
3.3.2 Computation of Pi . . . 6
3.3.3 Distributed Sum . . . 7

4 TclMPI Developer's Guide 9

4.1 Overall Design and Differences to the MPI C-bindings . . . 9
4.2 Naming Conventions . . . 9
4.3 TclMPI Support Functions . . . 10
4.3.1 Mapping MPI Communicators . . . 10
4.3.2 Mapping MPI Requests . . . 10
4.3.3 Mapping Data Types . . . 10
4.3.4 Common Error Message Processing . . . 10

5 Class Index 11

5.1 Class List . . . 11

6 File Index 13

6.1 File List . . . 13

7 Class Documentation 15

7.1 tclmpi_comm Struct Reference . . . 15
7.1.1 Detailed Description . . . 15
7.1.2 Member Data Documentation . . . 15
7.1.2.1 comm . . . 15
7.1.2.2 label . . . 16
7.1.2.3 next . . . 16
7.1.2.4 valid . . . 16
7.2 tclmpi_dblint Struct Reference . . . 16
7.2.1 Detailed Description . . . 16
7.2.2 Member Data Documentation . . . 16
7.2.2.1 d . . . 16
7.2.2.2 i . . . 17

7.3 tclmpi_intint Struct Reference . . . 17
7.3.1 Detailed Description . . . 17
7.3.2 Member Data Documentation . . . 17
7.3.2.1 i1 . . . 17
7.3.2.2 i2 . . . 17
7.4 tclmpi_req Struct Reference . . . 18
7.4.1 Detailed Description . . . 18
7.4.2 Member Data Documentation . . . 18
7.4.2.1 comm . . . 18
7.4.2.2 data . . . 18
7.4.2.3 label . . . 19
7.4.2.4 len . . . 19
7.4.2.5 next . . . 19
7.4.2.6 req . . . 19
7.4.2.7 source . . . 19
7.4.2.8 tag . . . 19
7.4.2.9 type . . . 19

8 File Documentation **21**

8.1 _tclmpi.c File Reference . . . 21
8.1.1 Macro Definition Documentation . . . 22
8.1.1.1 MPI_VERSION . . . 23
8.1.1.2 TCLMPI_ABORT . . . 23
8.1.1.3 TCLMPI_AUTO . . . 23
8.1.1.4 TCLMPI_CONV_CHECK . . . 23
8.1.1.5 TCLMPI_DOUBLE . . . 24
8.1.1.6 TCLMPI_DOUBLE_INT . . . 24
8.1.1.7 TCLMPI_ERROR . . . 24
8.1.1.8 TCLMPI_INT . . . 24
8.1.1.9 TCLMPI_INT_INT . . . 24
8.1.1.10 TCLMPI_INVALID . . . 24
8.1.1.11 TCLMPI_NONE . . . 24
8.1.1.12 TCLMPI_TOZERO . . . 25
8.1.2 Typedef Documentation . . . 25
8.1.2.1 tclmpi_comm_t . . . 25
8.1.2.2 tclmpi_dblint_t . . . 25
8.1.2.3 tclmpi_intint_t . . . 25
8.1.2.4 tclmpi_req_t . . . 25
8.1.3 Function Documentation . . . 25
8.1.3.1 _tclmpi_Init() . . . 25
8.1.3.2 TclMPI_Abort() . . . 26
8.1.3.3 TclMPI_Allgather() . . . 26

8.1.3.4 TclMPI_Allreduce() . . . 27
8.1.3.5 TclMPI_Barrier() . . . 27
8.1.3.6 TclMPI_Bcast() . . . 28
8.1.3.7 TclMPI_Comm_free() . . . 29
8.1.3.8 TclMPI_Comm_rank() . . . 29
8.1.3.9 TclMPI_Comm_size() . . . 30
8.1.3.10 TclMPI_Comm_split() . . . 30
8.1.3.11 TclMPI_Conv_get() . . . 31
8.1.3.12 TclMPI_Conv_set() . . . 31
8.1.3.13 TclMPI_Finalize() . . . 32
8.1.3.14 TclMPI_Gather() . . . 32
8.1.3.15 TclMPI_Init() . . . 33
8.1.3.16 TclMPI_Iprobe() . . . 33
8.1.3.17 TclMPI_Irecv() . . . 34
8.1.3.18 TclMPI_Isend() . . . 34
8.1.3.19 TclMPI_Probe() . . . 36
8.1.3.20 TclMPI_Recv() . . . 37
8.1.3.21 TclMPI_Reduce() . . . 37
8.1.3.22 TclMPI_Scatter() . . . 38
8.1.3.23 TclMPI_Send() . . . 38
8.1.3.24 TclMPI_Wait() . . . 39

Index **41**

Chapter 1

Main Page

The TclMPI package contains software that wraps an MPI library for Tcl and allows MPI calls to be used from Tcl scripts. This code can be compiled as a shared object to be loaded into an existing Tcl interpreter or as a standalone TclMPI interpreter. In combination with some additional bundled Tcl script code, additional commands are provided that allow to run Tcl scripts in parallel via "mpirun" or "mpiexec" similar to C, C++ or Fortran programs.

1.0.1 Homepage:

The main author of this package is Axel Kohlmeyer and you can reach him at akohlmey@gmail.com. The official homepage for this project is https://akohlmey.github.io/tclmpi/ and development is hosted on GitHub.

For compilation and installation instructions, please see the file INSTALL. Detailed documentation is available online from the project home page and as a PDF file in the source package.

1.0.2 Test Status:

1.0.3 Citing:

You can cite TclMPI as:

Axel Kohlmeyer. (2021). TclMPI: Release 1.1 [Data set]. Zenodo.

1.0.4 Acknowledgements:

Thanks to Arjen Markus and Chris MacDermaid for encouragement and (lots of) constructive criticism, that has helped enourmously to develop the package from a crazy idea to its current level. Thanks to Alex Baker for motivating me to convert to using CMake as build system which makes building TclMPI natively on Windows much easier.

A special thanks also goes to Karolina Sarnowska-Upton and Andrew Grimshaw that allowed me to use TclMPI as an example in their MPI portability study, which helped to find quite a few bugs and resolve several portability issues before the code was hitting the real world.

Chapter 2

Copyright and License for TclMPI

Chapter 3

TclMPI User's Guide

This page describes Tcl bindings for MPI. This package provides a shared object that can be loaded into a Tcl interpreter to provide additional commands that act as an interface to an underlying MPI implementation. This allows to run Tcl scripts in parallel via mpirun or mpiexec similar to C, C++ or Fortran programs and communicate via wrappers to MPI function call.

The original motivation for writing this package was to complement a Tcl wrapper for the LAMMPS molecular dynamics simulation software, but also allow using the VMD molecular visualization and analysis package in parallel without having to recompile VMD and using a convenient API to people that already know how to program parallel programs with MPI in C, C++ or Fortran.

3.1 Compilation and Installation

The package currently consist of a single C source file which usually will be compiled for dynamic linkage, but can also be compiled into a new Tcl interpreter with TclMPI included (required on some platforms that require static linkage) and a Tcl script file. In addition the package contains some examples, a simple unit test harness (implemented in Tcl) and a set of tests to be run with either one MPI rank (test01, test02) or two MPI ranks (test03, test04).

The build system uses CMake (version 3.16 or later) and has been confirmed to work on Linux macOS and Windows. The MPI library has to be at least MPI-2 standard compliant and the Tcl version should be 8.6 or later (it may work with 8.5, too). When compiled for a dynamically loaded shared object (DSO) or DLL file, the MPI library has to be compiled and linked with support for building shared libraries as well (this is the default for OpenMPI on Linux, but your mileage may vary).

You need to run CMake the usual way, for example with `cmake -B build-folder -S .`, then `cmake --build build-folder`, followed by `cmake --install build-folder`. There are a few settings that can be used to adjust what is compiled and installed and where. The following settings are supported:

- BUILD_TCLMPI_SHELL Build a `tclmpish` executable as extended Tcl shell (default: on)
- ENABLE_TCL_STUBS Use the Tcl stubs mechanism (default: on, requires Tcl 8.6 or later)
- BUILD_TESTING Enable unit testing (default: on)
- DOWNLOAD_MPICH4WIN Download MPICH2-1.4.1 headers and link library (default: off, Windows only)
- CMAKE_INSTALL_PREFIX Path to installation location prefix (default: (platform specific))

To change settings from the defaults append -D<SETTING>=<VALUE> to the cmake command line and replace <SETTING> and <VALUE> accordingly.

To enable the new TclMPI package you can use the command set auto_path [concat /usr/local/tcl8.6/ $auto_path] in your .tclshrc (or .vmdrc or similar) file and then you can load the TclMPI wrappers on demand simply by using the command package require tclmpi. For the extended shell, the _tclmpi.so file is not use and instead tclmpish needs to run instead of tclsh. For that you may append the bin folder of the installation tree to your PATH environment variable. In case of using the custom Tcl shell, the startup script would be called .tclmpishrc instead of .tclshrc.

3.2 Software Development and Bug Reports

The TclMPI code is maintained using git for source code management, and the project is hosted on github at https://github.com/akohlmey/tclmpi From there you can download snapshots of the development and releases, clode the repository to follow development, or work on your own branch through forking it. Bug reports and feature requests should also be filed on github at through the issue tracker at: https://github.com/akohlmey/tclmpi/issues.

3.3 Example Programs

The following section provides some simple examples using TclMPI to recreate some common MPI example programs in Tcl.

3.3.1 Hello World

This is the TclMPI version of "hello world".

```
#!/bin/sh \
exec tclsh "$0" "$@"
package require tclmpi 0.9
# initialize MPI
::tclmpi::init
# get size of communicator and rank of process
set comm tclmpi::comm_world
set size [::tclmpi::comm_size $comm]
set rank [::tclmpi::comm_rank $comm]
puts "hello world, this is rank $rank of $size"
# shut down MPI
::tclmpi::finalize
exit 0
```

3.3.2 Computation of Pi

This script uses TclMPI to compute the value of Pi from numerical quadrature of the integral:

$$\pi = \int_0^1 \frac{4}{1+x^2} dx$$

```
#!/bin/sh \
exec tclsh "$0" "$@"
package require tclmpi 0.9
# initialize MPI
::tclmpi::init
set comm tclmpi::comm_world
set size [::tclmpi::comm_size $comm]
set rank [::tclmpi::comm_rank $comm]
set master 0
set num [lindex $argv 0]
```

```
# make sure all processes have the same interval parameter
set num [::tclmpi::bcast $num ::tclmpi::int $master $comm]
# run parallel calculation
set h [expr {1.0/$num}]
set sum 0.0
for {set i $rank} {$i < $num} {incr i $size} {
    set sum [expr {$sum + 4.0/(1.0 + ($h*($i+0.5))**2)}]
}
set mypi [expr {$h * $sum}]
# combine and print results
set mypi [::tclmpi::allreduce $mypi tclmpi::double \
            tclmpi::sum $comm]
if {$rank == $master} {
    set rel [expr {abs(($mypi - 3.14159265358979)/3.14159265358979)}]
    puts "result: $mypi. relative error: $rel"
}
# shut down MPI
::tclmpi::finalize
exit 0
```

3.3.3 Distributed Sum

This is a small example version that distributes a data set and computes the sum across all elements in parallel.

```
#!/bin/sh \
exec tclsh "$0" "$@"
package require tclmpi 0.9
# data summation helper function
proc sum {data} {
    set sum 0
    foreach d $data {
        set sum [expr {$sum + $d}]
    }
    return $sum
}
::tclmpi::init
set comm       $tclmpi::comm_world
set mpi_sum    $tclmpi::sum
set mpi_double $tclmpi::double
set mpi_int    $tclmpi::int
set size [::tclmpi::comm_size $comm]
set rank [::tclmpi::comm_rank $comm]
set master 0
# The master creates the list of data
#
set dataSize 1000000
set data {}
if { $comm == $master } {
    set mysum 0
    for { set i 0 } { $i < $dataSize } { incr i } {
        lappend data $i
    }
}
# add padding, so the number of data elements is divisible
# by the number of processors as required by tclmpi::scatter
set needpad [expr {$dataSize % $size}]
set numpad [expr {$needpad ? ($size - $needpad) : 0}]
if { [comm_rank $comm] == $master } {
    for {set i 0} {$i < $numpad} {incr i} {
        lappend data 0
    }
}
set blocksz [expr {($dataSize + $numpad)/ $size}]
# distribute data and do the summation on each node
# the sum the result across all nodes. Note: the data
# is integer, but we need to do the full sum in double
# precison to avoid overflows.
set mydata [::tclmpi::scatter $data $mpi_int $master $comm]
set sum [::tclmpi::allreduce [sum $mydata] $mpi_double $mpi_sum $comm]
if { $comm == $master } {
    puts "Distributed sum: $sum"
}
::tclmpi::finalize
```

Chapter 4

TclMPI Developer's Guide

This document explains the implementation of the Tcl bindings for MPI implemented in TclMPI. The following sections will document how and which MPI is mapped to Tcl and what design choices were made.

4.1 Overall Design and Differences to the MPI C-bindings

To be consistent with typical Tcl conventions all commands and constants in lower case and prefixed with tclmpi, so that clashes with existing programs are reduced. This is not yet set up to be a proper namespace, but that may happen at a later point, if the need arises. The overall philosophy of the bindings is to make the API similar to the MPI one (e.g. maintain the order of arguments), but don't stick to it slavishly and do things the Tcl way wherever justified. Convenience and simplicity take precedence over performance. If performance matters that much, one would write the entire code C/C++ or Fortran and not Tcl. The biggest visible change is that for sending data around, receive buffers will be automatically set up to handle the entire message. Thus the typical "count" arguments of the C/C++ or Fortran bindings for MPI is not required, and the received data will be the return value of the corresponding command. This is consistent with the automatic memory management in Tcl, but this convenience and consistency will affect performance and the semantics. For example calls to tclmpi::bcast will be converted into *two* calls to MPI_Bcast(); the first will broadcast the size of the data set being sent (so that a sufficiently sized buffers can be allocated) and then the second call will finally send the data for real. Similarly, tclmpi::recv will be converted into calling MPI_Probe() and then MPI_Recv() for the purpose of determining the amount of temporary storage required. The second call will also use the MPI_SOURCE and MPI_TAG flags from the MPI_Status object created for MPI_Probe() to make certain, the correct data is received.

Things get even more complicated with with non-blocking receives. Since we need to know the size of the message to receive, a non-blocking receive can only be posted, if the corresponding send is already pending. This is being determined by calling MPI_Iprobe() and when this shows no (matching) pending message, the parameters for the receive will be cached and the then MPI_Probe() followed by MPI_Recv() will be called as part of tclmpi::wait. The blocking/non-blocking behavior of the Tcl script should be very close to the corresponding C bindings, but probably not as efficient.

4.2 Naming Conventions

All functions that are new Tcl commands follow the MPI naming conventions, but using TclMPI_ as prefix instead of MPI_. The corresponding Tcl commands are placed in the tclmpi namespace and all lower case. Example: TclMPI_Init() is the wrapper for MPI_Init() and is provided as command tclmpi::init. Defines and constants from the MPI header file are represented in TclMPI as plain strings, all lowercase and with a tclmpi:: prefix. Thus MPI_C↩ OMM_WORLD becomes tclmpi::comm_world and MPI_INT becomes tclmpi::int.

Functions that are internal to the plugin as well as static variables are prefixed with all lower case, i.e. tclmpi_. Those functions have to be declared static.

All string constants are also declared as namespace variables, e.g. $tclmpi::comm_world, so that shortcut notations are possible as shown in the following example:

```
namespace upvar tclmpi comm_world comm
namespace upvar tclmpi int        mpi_int
```

4.3 TclMPI Support Functions

Several MPI entities like communicators, requests, status objects cannot be represented directly in Tcl. For TclMPI they need to be mapped to something else, for example a string that will uniquely identify this entity and then it will be translated into the real object it represents with the help of the following support functions.

4.3.1 Mapping MPI Communicators

MPI communicators are represented in TclMPI by strings of the form "tclmpi::comm%d", with "%d" being replaced by a unique integer. In addition, a few string constants are mapped to the default communicators that are defined in MPI. These are tclmpi::comm_world, tclmpi::comm_self, and tclmpi::comm_null, which represent MPI_COMM↩ _WORLD, MPI_COMM_SELF, and MPI_COMM_NULL, respectively.

Internally the map is maintained in a simple linked list which is initialized with the three default communicators when the plugin is loaded and where new communicators are added at the end as needed. The functions mpi2tcl_comm and tcl2mpi_comm are then used to translate from one representation to the other while tclmpi_add_comm will append a new structure containing the communicator to the list. Correspondingly tclmpi_del_comm will remove a communicator entry from the lest, based on its Tcl string representation.

4.3.2 Mapping MPI Requests

MPI requests are represented in TclMPI by strings of the form "tclmpi::req%d", with "%d" being replaced by a unique integer. Internally this map is maintained in a simple linked list to which new requests are appended and from which completed requests are removed as needed. The function tclmpi_find_req is used to locate a specific request and its associated data from its string label. In addition, tclmpi_add_req will append a new request to the list, and tclmpi_del_req will remove (completed) requests.

4.3.3 Mapping Data Types

The helper function tclmpi_datatype is used to convert string constants representing specific data types into integer constants for convenient branching. Data types in TclMPI are somewhat different from MPI data types to match better the spirit of Tcl scripting.

4.3.4 Common Error Message Processing

There is a significant redundancy in checking for and reporting error conditions. For this purpose, several support functions exist.

tclmpi_errcheck verifies if calls to the MPI library were successful and if not, generates a formatted error message that is appended to the current result list.

tclmpi_commcheck verifies if a communicator argument was using a valid Tcl representation and if not, generates a formatted error message that is appended to the current result list.

tclmpi_typecheck test if a type argument was using a valid Tcl representation and if not, generates a formatted error message that is appended to the current result list.

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

tclmpi_comm 15
tclmpi_dblint 16
tclmpi_intint 17
tclmpi_req 18

Chapter 6

File Index

6.1 File List

Here is a list of all documented files with brief descriptions:

_tclmpi.c 21

Chapter 7

Class Documentation

7.1 tclmpi_comm Struct Reference

Collaboration diagram for tclmpi_comm:

Public Attributes

- const char * label
- MPI_Comm comm
- int valid
- tclmpi_comm_t * next

7.1.1 Detailed Description

Linked list entry to map MPI communicators to strings.

7.1.2 Member Data Documentation

7.1.2.1 comm

`MPI_Comm tclmpi_comm::comm`

MPI communicator corresponding of this entry

7.1.2.2 label

```
const char* tclmpi_comm::label
```

String representing the communicator in Tcl

7.1.2.3 next

```
tclmpi_comm_t* tclmpi_comm::next
```

Pointer to next element in linked list

7.1.2.4 valid

```
int tclmpi_comm::valid
```

Non-zero if communicator is valid

The documentation for this struct was generated from the following file:

- _tclmpi.c

7.2 tclmpi_dblint Struct Reference

Public Attributes

- double d
- int i

7.2.1 Detailed Description

Represent a double/integer pair

7.2.2 Member Data Documentation

7.2.2.1 d

```
double tclmpi_dblint::d
```

double data value

7.2.2.2 i

```
int tclmpi_dblint::i
```

location data

The documentation for this struct was generated from the following file:

- _tclmpi.c

7.3 tclmpi_intint Struct Reference

Public Attributes

- int i1
- int i2

7.3.1 Detailed Description

Represent an integer/integer pair

7.3.2 Member Data Documentation

7.3.2.1 i1

```
int tclmpi_intint::i1
```

integer data value

7.3.2.2 i2

```
int tclmpi_intint::i2
```

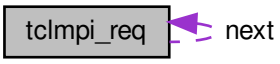
location data

The documentation for this struct was generated from the following file:

- _tclmpi.c

7.4 tclmpi_req Struct Reference

Collaboration diagram for tclmpi_req:



Public Attributes

- const char * label
- void * data
- int len
- int type
- int source
- int tag
- MPI_Request * req
- MPI_Comm comm
- tclmpi_req_t * next

7.4.1 Detailed Description

Linked list entry to map MPI requests to "tclmpi::req%d" strings.

7.4.2 Member Data Documentation

7.4.2.1 comm

```
MPI_Comm tclmpi_req::comm
```

communicator for non-blocking receive

7.4.2.2 data

```
void* tclmpi_req::data
```

pointer to send or receive data buffer

7.4.2.3 label

```
const char* tclmpi_req::label
```

identifier of this request

7.4.2.4 len

```
int tclmpi_req::len
```

size of data block

7.4.2.5 next

```
tclmpi_req_t* tclmpi_req::next
```

pointer to next struct

7.4.2.6 req

```
MPI_Request* tclmpi_req::req
```

pointer MPI request handle generated by MPI

7.4.2.7 source

```
int tclmpi_req::source
```

source rank of non-blocking receive

7.4.2.8 tag

```
int tclmpi_req::tag
```

tag selector of non-blocking receive

7.4.2.9 type

```
int tclmpi_req::type
```

data type of send data

The documentation for this struct was generated from the following file:

- _tclmpi.c

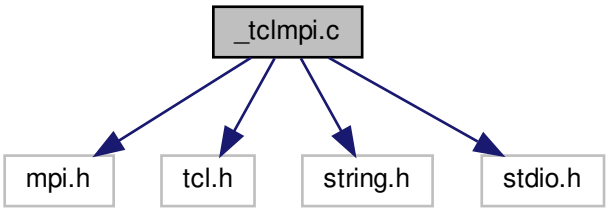
Chapter 8

File Documentation

8.1 _tclmpi.c File Reference

```
#include <mpi.h>
#include <tcl.h>
#include <string.h>
#include <stdio.h>
```

Include dependency graph for _tclmpi.c:



Classes

- struct tclmpi_comm
- struct tclmpi_dblint
- struct tclmpi_intint
- struct tclmpi_req

Macros

- #define MPI_VERSION 1
- #define TCLMPI_TOZERO -4
- #define TCLMPI_ABORT -3
- #define TCLMPI_ERROR -2

- #define TCLMPI_INVALID -1
- #define TCLMPI_NONE 0
- #define TCLMPI_AUTO 1
- #define TCLMPI_INT 2
- #define TCLMPI_INT_INT 3
- #define TCLMPI_DOUBLE 4
- #define TCLMPI_DOUBLE_INT 5
- #define TCLMPI_CONV_CHECK(type, in, out, assign)

Typedefs

- typedef struct tclmpi_comm tclmpi_comm_t
- typedef struct tclmpi_dblint tclmpi_dblint_t
- typedef struct tclmpi_intint tclmpi_intint_t
- typedef struct tclmpi_req tclmpi_req_t

Functions

- int TclMPI_Init (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Conv_set (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Conv_get (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Finalize (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Abort (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Comm_size (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Comm_rank (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Comm_split (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Comm_free (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Barrier (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Bcast (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Scatter (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Allgather (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Gather (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Allreduce (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Reduce (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Send (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Isend (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Recv (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Irecv (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Probe (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Iprobe (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Wait (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int _tclmpi_Init (Tcl_Interp *interp)

8.1.1 Macro Definition Documentation

8.1.1.1 MPI_VERSION

```
#define MPI_VERSION 1
```

Define for backward compatibility with old MPI libraries. We need to be able to detect the API of MPI_VERSION > 1 for clean error handling and making MPI errors catch-able.

8.1.1.2 TCLMPI_ABORT

```
#define TCLMPI_ABORT -3
```

abort on problems

8.1.1.3 TCLMPI_AUTO

```
#define TCLMPI_AUTO 1
```

the tcl native data type (string)

8.1.1.4 TCLMPI_CONV_CHECK

```
#define TCLMPI_CONV_CHECK(
               type,
               in,
               out,
               assign )
```

Value:

```
    if (Tcl_Get ## type ## FromObj(interp,in,out) != TCL_OK) {   \
        switch (tclmpi_conv_handler) {                           \
          case TCLMPI_TOZERO:                                    \
              Tcl_ResetResult(interp);                           \
              assign=0;                                          \
              break;                                             \
          case TCLMPI_ABORT:                                     \
              fprintf(stderr,"Error on data element %d: %s\n",   \
                      i, Tcl_GetStringResult(interp));           \
              MPI_Abort(comm,i);                                 \
              break;                                             \
          case TCLMPI_ERROR:                                     \
          default: /* fallthrough */                             \
              return TCL_ERROR;                                  \
              break;                                             \
        }                                                        \
    }
```

Data conversion with with error handling

Parameters

type	Tcl data type for calling Tcl_Get<Type>FromObj()
in	pointer to input object for conversion
out	pointer to output storage for conversion
assign	target to assign a zero to for TCLMPI_TOZERO

This macro enables consistent handling of data conversions. It also queries the tclmpi_conv_handler variable to

jump to the selected conversion error behavior. For TCLMPI_ERROR (the default) a Tcl error is raised and TclMPI returns to the calling function. For TCLMPI_ABORT and error message is written to stderr and parallel execution on the current communicator is terminated via MPI_Abort(). For TCLMPI_TOZERO the error is silently ignored and the data element handed in as assign parameter is set to zero.

8.1.1.5 TCLMPI_DOUBLE

```
#define TCLMPI_DOUBLE 4
```

floating point data type

8.1.1.6 TCLMPI_DOUBLE_INT

```
#define TCLMPI_DOUBLE_INT 5
```

data type for double/integer pair

8.1.1.7 TCLMPI_ERROR

```
#define TCLMPI_ERROR -2
```

flag problems as Tcl errors

8.1.1.8 TCLMPI_INT

```
#define TCLMPI_INT 2
```

data type for integers

8.1.1.9 TCLMPI_INT_INT

```
#define TCLMPI_INT_INT 3
```

data type for pairs of integers

8.1.1.10 TCLMPI_INVALID

```
#define TCLMPI_INVALID -1
```

not ready to handle data

8.1.1.11 TCLMPI_NONE

```
#define TCLMPI_NONE 0
```

no data type assigned

8.1.1.12 TCLMPI_TOZERO

```
#define TCLMPI_TOZERO -4
```

convert problematic data items to zero

8.1.2 Typedef Documentation

8.1.2.1 tclmpi_comm_t

```
typedef struct tclmpi_comm tclmpi_comm_t
```

Linked list entry type for managing MPI communicators

8.1.2.2 tclmpi_dblint_t

```
typedef struct tclmpi_dblint tclmpi_dblint_t
```

Data type for maxloc/minloc reductions with a double and an integer

8.1.2.3 tclmpi_intint_t

```
typedef struct tclmpi_intint tclmpi_intint_t
```

Data type for maxloc/minloc reductions with two integers

8.1.2.4 tclmpi_req_t

```
typedef struct tclmpi_req tclmpi_req_t
```

Linked list entry type for managing MPI requests

8.1.3 Function Documentation

8.1.3.1 _tclmpi_Init()

```
int _tclmpi_Init (
            Tcl_Interp * interp )
```

register the package as a plugin with the Tcl interpreter

Parameters

interp	current Tcl interpreter

Returns

TCL_OK or TCL_ERROR

This function sets up the plugin to register the various MPI wrappers in this package with the Tcl interpreter.

Depending on the USE_TCL_STUBS define being active or not, this is done using the native dynamic loader interface or the Tcl stubs interface, which would allow to load the plugin into static executables and plugins from different Tcl versions.

In addition the linked list for translating MPI communicators is initialized for the predefined communicators tclmpi↩ ::comm_world, tclmpi::comm_self, and tclmpi::comm_null and its corresponding MPI counterparts.

8.1.3.2 TclMPI_Abort()

```
int TclMPI_Abort (
             ClientData nodata,
             Tcl_Interp * interp,
             int objc,
             Tcl_Obj *const objv[] )
```

wrapper for MPI_Abort()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function translates the Tcl string representing a communicator into the corresponding MPI communicator and then calls MPI_Abort().

8.1.3.3 TclMPI_Allgather()

```
int TclMPI_Allgather (
             ClientData nodata,
             Tcl_Interp * interp,
             int objc,
             Tcl_Obj *const objv[] )
```

wrapper for MPI_Allgather()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function implements a gather operation that collects data for TclMPI. This operation does not accept the tclmpi::auto data type, also support for types outside of tclmpi::int and tclmpi::double is incomplete. The length of the data is inferred from the data object passed to this function and thus a 'count' argument is not needed. The number of data items has to be the same on all processes on the communicator.

The result is converted back into Tcl objects and passed up as result value to the calling Tcl code on all processors. If the MPI call failed, an MPI error message is passed up as result instead.

8.1.3.4 TclMPI_Allreduce()

```
int TclMPI_Allreduce (
             ClientData nodata,
             Tcl_Interp * interp,
             int objc,
             Tcl_Obj *const objv[] )
```

wrapper for MPI_Allreduce()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function implements a reduction plus broadcast function for TclMPI. This operation does not accept the tclmpi↩ ::auto data type, also support for types outside of tclmpi::int and tclmpi::double is incomplete. The length of the data is inferred from the data object passed to this function and thus a 'count' argument is not needed.

The result is converted back into Tcl objects and passed up as result value to the calling Tcl code. If the MPI call failed, an MPI error message is passed up as result instead.

8.1.3.5 TclMPI_Barrier()

```
int TclMPI_Barrier (
             ClientData nodata,
```

```
            Tcl_Interp * interp,
            int objc,
            Tcl_Obj *const objv[] )
```

wrapper for MPI_Barrier()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function translates the Tcl string representing a communicator into the corresponding MPI communicator and then calls MPI_Barrier(). If the MPI call failed, an MPI error message is passed up as result.

8.1.3.6 TclMPI_Bcast()

```
int TclMPI_Bcast (
            ClientData nodata,
            Tcl_Interp * interp,
            int objc,
            Tcl_Obj *const objv[] )
```

wrapper for MPI_Bcast()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function implements a broadcast function for TclMPI. Unlike in the C bindings, the length of the data is inferred from the data object passed to this function and thus a 'count' argument is not needed. Only a limited number of data types are currently supported, since Tcl has a limited number of "native" data types. The tclmpi::auto data type transfers the internal string representation of an object, while the other data types convert data to native data types as needed, with all non-representable data translated into either 0 or 0.0. In all cases, two broadcasts are needed. The first to transmit the amount of data being sent so that a suitable receive buffer can be set up.

The result of the broadcast is converted back into Tcl objects and passed up as result value to the calling Tcl code. If the MPI call failed, an MPI error message is passed up as result instead.

8.1.3.7 TclMPI_Comm_free()

```
int TclMPI_Comm_free (
             ClientData nodata,
             Tcl_Interp * interp,
             int objc,
             Tcl_Obj *const objv[] )
```

wrapper for MPI_Comm_free()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function deletes a defined MPI communicator and removes its Tcl representation from the local translation tables.

8.1.3.8 TclMPI_Comm_rank()

```
int TclMPI_Comm_rank (
             ClientData nodata,
             Tcl_Interp * interp,
             int objc,
             Tcl_Obj *const objv[] )
```

wrapper for MPI_Comm_rank()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function translates the Tcl string representing a communicator into the corresponding MPI communicator and then calls MPI_Comm_rank() on it. The resulting number is passed to Tcl as result or the MPI error message is passed up similarly.

8.1.3.9 TclMPI_Comm_size()

```
int TclMPI_Comm_size (
             ClientData nodata,
             Tcl_Interp * interp,
             int objc,
             Tcl_Obj *const objv[] )
```

wrapper for MPI_Comm_size()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function translates the Tcl string representing a communicator into the corresponding MPI communicator and then calls MPI_Comm_size() on it. The resulting number is passed to Tcl as result or the MPI error message is passed up similarly.

8.1.3.10 TclMPI_Comm_split()

```
int TclMPI_Comm_split (
             ClientData nodata,
             Tcl_Interp * interp,
             int objc,
             Tcl_Obj *const objv[] )
```

wrapper for MPI_Comm_split()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function translates the Tcl string representing a communicator into the corresponding MPI communicator also checks and converts the values for 'color' and 'key' and then calls MPI_Comm_split(). The resulting communicator is added to the internal communicator map linked list and its string representation is passed to Tcl as result. If the MPI call failed, the MPI error message is passed up similarly.

8.1.3.11 TclMPI_Conv_get()

```
int TclMPI_Conv_get (
            ClientData nodata,
            Tcl_Interp * interp,
            int objc,
            Tcl_Obj *const objv[] )
```

Get error handler string for data conversions in TclMPI

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK

This function returns which error handler is currently active for data conversions in TclMPI. For details see TclMPI_Conv_set().

There is no equivalent MPI function for this, since there are no data conversions in C or C++.

8.1.3.12 TclMPI_Conv_set()

```
int TclMPI_Conv_set (
            ClientData nodata,
            Tcl_Interp * interp,
            int objc,
            Tcl_Obj *const objv[] )
```

Set error handler for data conversions in TclMPI

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function sets what action TclMPI should take if a conversion of a data element to the requested integer or double data type fails. There are currently three handlers implemented: TCLMPI_ERROR, TCLMPI_ABORT, and TCLMPI_TOZERO.

For TCLMPI_ERROR a Tcl error is raised and TclMPI returns to the calling function. For TCLMPI_ABORT an error message is written to the error output and parallel execution on the current communicator is terminated via MPI_Abort(). For TCLMPI_TOZERO the error is silently ignored and the data element set to zero.

There is no equivalent MPI function for this, since there are no data conversions in C or C++.

8.1.3.13 TclMPI_Finalize()

```
int TclMPI_Finalize (
             ClientData nodata,
             Tcl_Interp * interp,
             int objc,
             Tcl_Obj *const objv[] )
```

wrapper for MPI_Finalize()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function does a little more than just calling MPI_Finalize(). It also tries to detect whether MPI_Init() or MPI_↩ Finialize() have been called before (from Tcl) and then creates a (catchable) Tcl error instead of an (uncatchable) MPI error.

8.1.3.14 TclMPI_Gather()

```
int TclMPI_Gather (
             ClientData nodata,
             Tcl_Interp * interp,
             int objc,
             Tcl_Obj *const objv[] )
```

wrapper for MPI_Gather()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function implements a gather operation that collects data for TclMPI. This operation does not accept the tclmpi::auto data type, also support for types outside of tclmpi::int and tclmpi::double is incomplete. The length of the data is inferred from the data object passed to this function and thus a 'count' argument is not needed. The number of data items has to be the same on all processes on the communicator.

The result is converted back into Tcl objects and passed up as result value to the calling Tcl code on the root processor. If the MPI call failed, an MPI error message is passed up as result instead.

8.1.3.15 TclMPI_Init()

```
int TclMPI_Init (
            ClientData nodata,
            Tcl_Interp * interp,
            int objc,
            Tcl_Obj *const objv[] )
```

wrapper for MPI_Init()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function does a little more work than just calling MPI_Init(). First of it tries to detect whether MPI_Init() has been called before (from Tcl) and then creates a (catchable) Tcl error instead of an (uncatchable) MPI error. It will also try to pass the argument vector to the script from the Tcl generated 'argv' array to the underlying MPI_Init() call and reset argv as needed.

8.1.3.16 TclMPI_Iprobe()

```
int TclMPI_Iprobe (
            ClientData nodata,
            Tcl_Interp * interp,
            int objc,
            Tcl_Obj *const objv[] )
```

wrapper for MPI_Iprobe()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function implements a non-blocking probe operation for TclMPI. Argument flags for source, tag, and communicator are translated into their native MPI equivalents and then MPI_Iprobe called.

Similar to MPI_Probe, generating a status object to inspect the pending receive is optional. If desired, the argument is taken as a variable name which will then be generated as associative array with several entries similar to what MPI_Status contains. Those are source, tag, error status and count, however this is directly provided as multiple entries translated to char, int and double data types (COUNT_CHAR, COUNT_INT, COUNT_DOUBLE).

The status flag in MPI_Iprobe that returns true if a request is pending will be passed to the calling routine as Tcl result.

8.1.3.17 TclMPI_Irecv()

```
int TclMPI_Irecv (
            ClientData nodata,
            Tcl_Interp * interp,
            int objc,
            Tcl_Obj *const objv[] )
```

wrapper for MPI_Iecv()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function implements a non-blocking receive operation for TclMPI. Since the length of the data object is supposed to be automatically adjusted to the amount of data being sent, this function needs to be more complex than just a simple wrapper around the corresponding MPI C bindings. It will first call tclmpi_add_req to generate a new entry to the list of registered MPI requests. It will then call MPI_Iprobe to see if a matching send is already in progress and thus the necessary amount of storage required can be inferred from the MPI_Status object that is populated by MPI_Iprobe. If yes, a temporary receive buffer is allocated and the non-blocking receive is posted and all information is transferred to the tclmpi_req_t object. If not, only the arguments of the receive call are registered in the request object for later use. The command will pass the Tcl string that represents the generated MPI request to the Tcl interpreter as return value. If the MPI call failed, an MPI error message is passed up as result instead and a Tcl error is indicated.

8.1.3.18 TclMPI_Isend()

```
int TclMPI_Isend (
            ClientData nodata,
            Tcl_Interp * interp,
```

```
            int objc,
            Tcl_Obj *const objv[] )
```

wrapper for MPI_Isend()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function implements a non-blocking send operation for TclMPI. The length of the data is inferred from the data object passed to this function and thus a 'count' argument is not needed. Unlike for the blocking TclMPI_Send, in the case of tclmpi::auto as data a copy has to be made since the string representation of the send data might be invalidated during the send. The command generates a new tclmpi_req_t communication request via tclmpi_add↩ _req and the pointers to the data buffer and the MPI_Request info generated by MPI_Isend is stored in this request list entry for later perusal, see TclMPI_Wait. The generated string label representing this request will be passed on to the calling program as Tcl result. If the MPI call failed, an MPI error message is passed up as result instead and a Tcl error is indicated.

8.1.3.19 TclMPI_Probe()

```
int TclMPI_Probe (
             ClientData nodata,
             Tcl_Interp * interp,
             int objc,
             Tcl_Obj *const objv[] )
```

wrapper for MPI_Probe()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function implements a blocking probe operation for TclMPI. Argument flags for source, tag, and communicator are translated into their native MPI equivalents and then MPI_Probe called.

Similar to MPI_Probe, generating a status object to inspect the pending receive is optional. If desired, the argument is taken as a variable name which will then be generated as associative array with several entries similar to what MPI_Status contains. Those are source, tag, error status and count, however this is directly provided as multiple entries translated to char, int and double data types (COUNT_CHAR, COUNT_INT, COUNT_DOUBLE).

8.1.3.20 TclMPI_Recv()

```
int TclMPI_Recv (
             ClientData nodata,
             Tcl_Interp * interp,
             int objc,
             Tcl_Obj *const objv[] )
```

wrapper for MPI_Recv()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function implements a blocking receive operation for TclMPI. Since the length of the data object is supposed to be automatically adjusted to the amount of data being sent, this function will first call MPI_Probe to identify the amount of storage needed from the MPI_Status object that is populated by MPI_Probe. Then a temporary receive buffer is allocated and then converted back to Tcl objects according to the data type passed to the receive command. Due to this deviation from the MPI C bindings a 'count' argument is not needed. This command returns the received data to the calling procedure. If the MPI call failed, an MPI error message is passed up as result instead and a Tcl error is indicated.

8.1.3.21 TclMPI_Reduce()

```
int TclMPI_Reduce (
             ClientData nodata,
             Tcl_Interp * interp,
             int objc,
             Tcl_Obj *const objv[] )
```

wrapper for MPI_Reduce()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function implements a reduction function for TclMPI. This operation does not accept the tclmpi::auto data type, also support for types outside of tclmpi::int and tclmpi::double is incomplete. The length of the data is inferred from the data object passed to this function and thus a 'count' argument is not needed.

The result is collected on the process with rank root and converted back into Tcl objects and passed up as result value to the calling Tcl code. If the MPI call failed an MPI error message is passed up as result instead.

8.1.3.22 TclMPI_Scatter()

```
int TclMPI_Scatter (
            ClientData nodata,
            Tcl_Interp * interp,
            int objc,
            Tcl_Obj *const objv[] )
```

wrapper for MPI_Scatter()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function implements a scatter operation that distributes data for TclMPI. This operation does not accept the tclmpi::auto data type, also support for types outside of tclmpi::int and tclmpi::double is incomplete. The length of the data is inferred from the data object passed to this function and thus a 'count' argument is not needed. The number of data items has to be divisible by the number of processes on the communicator.

The result is converted back into Tcl objects and passed up as result value to the calling Tcl code. If the MPI call failed an MPI error message is passed up as result instead.

8.1.3.23 TclMPI_Send()

```
int TclMPI_Send (
            ClientData nodata,
            Tcl_Interp * interp,
            int objc,
            Tcl_Obj *const objv[] )
```

wrapper for MPI_Send()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function implements a blocking send operation for TclMPI. The length of the data is inferred from the data object passed to this function and thus a 'count' argument is not needed. In the case of tclmpi::auto, the string representation of the send data is directly passed to MPI_Send() otherwise a copy is made and data converted.

If the MPI call failed, an MPI error message is passed up as result instead and a Tcl error is indicated, otherwise nothing is returned.

8.1.3.24 TclMPI_Wait()

```
int TclMPI_Wait (
             ClientData nodata,
             Tcl_Interp * interp,
             int objc,
             Tcl_Obj *const objv[] )
```

wrapper for MPI_Wait()

Parameters

nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

This function implements a wrapper around MPI_Wait for TclMPI. Due to the design decisions in TclMPI, it works a bit different than MPI_Wait, particularly for non-blocking receive requests. As explained in the TclMPI_Irecv documentation, the corresponding MPI_Irecv may not yet have been posted, so we have to first inspect the tclmpi↩ _req_t object, if the receive still needs to be posted. If yes, then we need to do about the same procedure as for a blocking receive, i.e. call MPI_Probe to determine the size of the receive buffer, allocate that buffer and then post a blocking receive. If no, we call MPI_Wait to wait until the non-blocking receive is completed. In both cases, the result needed to be converted to Tcl objects and passed to the calling procedure as Tcl return values. Then the receive buffers can be deleted and the tclmpi_req_t entry removed from it translation table.

For non-blocking send requests, MPI_Wait is called and after completion the send buffer freed and the tclmpi_req_t data released. The MPI spec allows to call MPI_Wait on non-existing MPI_Requests and just return immediately. This is handled directly without calling MPI_Wait, since we cache all generated MPI requests.

Index

_tclmpi.c, 21
    _tclmpi_Init, 25
    MPI_VERSION, 22
    TCLMPI_ABORT, 23
    TclMPI_Abort, 26
    TclMPI_Allgather, 26
    TclMPI_Allreduce, 27
    TCLMPI_AUTO, 23
    TclMPI_Barrier, 27
    TclMPI_Bcast, 28
    TclMPI_Comm_free, 28
    TclMPI_Comm_rank, 29
    TclMPI_Comm_size, 29
    TclMPI_Comm_split, 30
    tclmpi_comm_t, 25
    TCLMPI_CONV_CHECK, 23
    TclMPI_Conv_get, 30
    TclMPI_Conv_set, 31
    tclmpi_dblint_t, 25
    TCLMPI_DOUBLE, 24
    TCLMPI_DOUBLE_INT, 24
    TCLMPI_ERROR, 24
    TclMPI_Finalize, 32
    TclMPI_Gather, 32
    TclMPI_Init, 33
    TCLMPI_INT, 24
    TCLMPI_INT_INT, 24
    tclmpi_intint_t, 25
    TCLMPI_INVALID, 24
    TclMPI_Iprobe, 33
    TclMPI_Irecv, 34
    TclMPI_Isend, 34
    TCLMPI_NONE, 24
    TclMPI_Probe, 36
    TclMPI_Recv, 36
    TclMPI_Reduce, 37
    tclmpi_req_t, 25
    TclMPI_Scatter, 38
    TclMPI_Send, 38
    TCLMPI_TOZERO, 24
    TclMPI_Wait, 39
_tclmpi_Init
    _tclmpi.c, 25

comm
    tclmpi_comm, 15
    tclmpi_req, 18

d
    tclmpi_dblint, 16

data
    tclmpi_req, 18

i
    tclmpi_dblint, 16
i1
    tclmpi_intint, 17
i2
    tclmpi_intint, 17

label
    tclmpi_comm, 15
    tclmpi_req, 18
len
    tclmpi_req, 19

MPI_VERSION
    _tclmpi.c, 22

next
    tclmpi_comm, 16
    tclmpi_req, 19

req
    tclmpi_req, 19

source
    tclmpi_req, 19

tag
    tclmpi_req, 19
TCLMPI_ABORT
    _tclmpi.c, 23
TclMPI_Abort
    _tclmpi.c, 26
TclMPI_Allgather
    _tclmpi.c, 26
TclMPI_Allreduce
    _tclmpi.c, 27
TCLMPI_AUTO
    _tclmpi.c, 23
TclMPI_Barrier
    _tclmpi.c, 27
TclMPI_Bcast
    _tclmpi.c, 28
tclmpi_comm, 15
    comm, 15
    label, 15
    next, 16
    valid, 16
TclMPI_Comm_free

_tclmpi.c, 28
TclMPI_Comm_rank
_tclmpi.c, 29
TclMPI_Comm_size
_tclmpi.c, 29
TclMPI_Comm_split
_tclmpi.c, 30
tclmpi_comm_t
_tclmpi.c, 25
TCLMPI_CONV_CHECK
_tclmpi.c, 23
TclMPI_Conv_get
_tclmpi.c, 30
TclMPI_Conv_set
_tclmpi.c, 31
tclmpi_dblint, 16
d, 16
i, 16
tclmpi_dblint_t
_tclmpi.c, 25
TCLMPI_DOUBLE
_tclmpi.c, 24
TCLMPI_DOUBLE_INT
_tclmpi.c, 24
TCLMPI_ERROR
_tclmpi.c, 24
TclMPI_Finalize
_tclmpi.c, 32
TclMPI_Gather
_tclmpi.c, 32
TclMPI_Init
_tclmpi.c, 33
TCLMPI_INT
_tclmpi.c, 24
TCLMPI_INT_INT
_tclmpi.c, 24
tclmpi_intint, 17
i1, 17
i2, 17
tclmpi_intint_t
_tclmpi.c, 25
TCLMPI_INVALID
_tclmpi.c, 24
TclMPI_Iprobe
_tclmpi.c, 33
TclMPI_Irecv
_tclmpi.c, 34
TclMPI_Isend
_tclmpi.c, 34
TCLMPI_NONE
_tclmpi.c, 24
TclMPI_Probe
_tclmpi.c, 36
TclMPI_Recv
_tclmpi.c, 36
TclMPI_Reduce
_tclmpi.c, 37
tclmpi_req, 18
comm, 18
data, 18
label, 18
len, 19
next, 19
req, 19
source, 19
tag, 19
type, 19
tclmpi_req_t
_tclmpi.c, 25
TclMPI_Scatter
_tclmpi.c, 38
TclMPI_Send
_tclmpi.c, 38
TCLMPI_TOZERO
_tclmpi.c, 24
TclMPI_Wait
_tclmpi.c, 39
type
tclmpi_req, 19

valid
tclmpi_comm, 16