TclMPI

1.2

Generated by Doxygen 1.9.1

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.

1.1 Homepage

The main author of this package is Axel Kohlmeyer and you can reach him at akohlmey@gmail.com. The online documentation for this project is at https://akohlmey.github.io/tclmpi/, a PDF version of the documentation is also available, and development is hosted on GitHub.

For basic compilation and installation instructions, please see the file INSTALL. More detailed documentation is available online from the User's Guide.

Information about the implementation and design of the package are in the Developer's Guide.

1.2 Precompiled Binaries

Precompiled binary packages of TclMPI are available for the following operating systems and distributions.

1.2.1 Microsoft Windows

A precompiled installer package for 64-bit Windows 10 (Version 21H1) is available from the TclMPI GitHub Releases Page.

To use this package, the MS-MPI package version 10.x and ActiveTcl version 8.6 from ActiveState must be downloaded and installed first. The installer will check for them and refuse to install TclMPI without.

MS-MPI is available here. You only need the "msmpisetup.exe" file and ActiveTcl is available here

1.2.2 Ubuntu Linux

PPA repositories for Ubuntu Linux are hosted on Launchpad at: https://launchpad.net/~akohlmey/+archive/ubu

To access the PPA and install the package, use the commands:
```
sudo add-apt-repository ppa:akohlmey/tclmpi
sudo apt-get update
sudo apt-get install tcl-tclmpi
```

Currently binaries are created for Ubuntu 20.04LTS and later.

1.2.3 Fedora Linux

Repositories with TclMPI packages for Fedora Linux are hosted on Copr at https://copr.↩ fedorainfracloud.org/coprs/akohlmey/TclMPI/

To access the Copr repository use the command:
```
sudo dnf copr enable akohlmey/TclMPI
```

To install TclMPI with support for the OpenMPI MPI library, use the command:
```
sudo dnf install tclmpi-openmpi
```

To install TclMPI with support for MPICH MPI library, use the command:
```
sudo dnf install tclmpi-mpich
```

Please note that to use any of the MPI libraries the corresponding environment module must be loaded first, e.g. with:
```
module load mpi
```

1.3 Test Status

1.4 Citing

You can cite TclMPI as:

Axel Kohlmeyer. (2021). TclMPI: Release 1.2 [Data set]. Zenodo.

1.5 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 an API that is convenient to people that already know how to program parallel programs with MPI in C, C++ or Fortran. It has since been adopted to provide an MPI wrapper for the OpenSees software: https://github.com/ambaker1/OpenSeesMPI

3.1 Pre-compiled Binary Packages

While it is usually expected that MPI based parallel applications are compiled from source code using the target machine's local MPI implementation, that is not always convenient or necessary. This applies for example to the Windows platform or Linux distributions where mechanisms are in place to check that are pre-requisites are installed and binaries are compatible. The TclMPI homepage has links to available binaries and information about how to install them as they become available.

3.2 Compilation

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 using a variety of C compilers (GNU, Clang, Intel, PGI, MSVC). You need to have both, Tcl and MPI installed including their respective development support packages (sometimes called SDK). The MPI library has to be at least MPI-2 standard compliant and the Tcl version should be 8.6 or later. 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.

To configure and build TclMPI you need to run CMake the usual way, in a console window with with:

```
cmake -B  build-folder -S .
cmake   --build build-folder
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)
- CMAKE_INSTALL_PREFIX Path to installation location prefix (default: (platform specific))
- BUILD_TESTING Enable unit testing (default: on)
- DOWNLOAD_MPICH4WIN Download MPICH2-1.4.1 headers and link library (default: off, only supported when cross-compiling on Linux for Windows)

To change settings from the defaults append `-D<SETTING>=<VALUE>` to the `cmake` command line and replace `<SETTING>` and `<VALUE>` accordingly or you may use the `ccmake` text mode UI or `cmake-gui`.

3.3 Building the Documentation

Documentation in HTML and PDF format is extracted from the sources using doxygen, if available. The build of the HTML format documentation is requested with

```
cmake --build build-folder --target html
```

The documentation will be in folder `build-folder/html`. To generate the PDF documentation, PDFLaTeX and several LaTeX style packages need to be installed. This is requested using

```
cmake --build build-folder --target pdf
```

and the resulting documentation will be in `build-folder/tclmpi_docs.pdf`.

3.4 Installation

To install the TclMPI package you can use

```
cmake --build build-folder --target install
```

which should by default install the compiled shared object and the associated two Tcl files into a subfolder of `<CMAKE_INSTALL_PREFIX>/tcl8.6`. The default value of CMAKE_INSTALL_PREFIX is system specific, but it can changed with `-D CMAKE_INSTALL_PREFIX=/some/path` when configuring with CMake, then the installation will be into the corresponding location.

To tell Tcl where to find the package, you need to either set or expand the TCLLIBPATH environment variable to the folder into which you have installed the files or place `auto_path [concat /usr/local/tcl8.6/ $auto_path]` at the beginning of your Tcl script or in your `.tclshrc` file (or .vmdrc or similar). Then you should be able to load the TclMPI wrappers on demand by using the command `package require tclmpi`.

For the extended Tcl shell `tclmpish`, the `_tclmpi.so` file is not used and instead `tclmpish` already includes the coresponding code and needs to be 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`.

3.5 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, clone the repository to follow development, or work on your own branches after 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.

3.6 Example Programs

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

3.6.1 Hello World

This is the TclMPI version of "hello world".

```
#!/usr/bin/env tclsh
package require tclmpi 1.2
# 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
```

3.6.2 Computation of Pi

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

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

```
#!/usr/bin/env tclsh
package require tclmpi 1.2
# 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
```

3.6.3 Distributed Sum

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

```
#!/usr/bin/env tclsh
package require tclmpi 1.2
# 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
```

3.7 TclMPI Tcl command reference

All TclMPI Tcl commands are placed into the tclmpi namespace.

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 functions are mapped to Tcl and what design choices were made in the process.

4.1 Overall Design and Differences to the MPI C-bindings

To be consistent with typical Tcl conventions, all commands and constants are in lower case and prefixed with tclmpi, so that clashes with existing programs are reduced.

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 in C/C++ or Fortran and not in 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 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 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_↩ COMM_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.

4.4 TclMPI Tcl Test Harness command reference

TclMPI includes a simple unit test harness written in (of course) Tcl. The corresponding commands are placed into the tclmpi_test namespace. Check out the files in the `tests` folders for examples.

Chapter 5

Module Index

5.1 Modules

Here is a list of all modules:

Chapter 6

Namespace Index

6.1 Namespace List

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

Chapter 7

Data Structure Index

7.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 8

File Index

8.1 File List

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

Chapter 9

Module Documentation

9.1 Support functions and data structures

Data Structures

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

Macros

- #define TCLMPI_LABEL_SIZE 32
- #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

- static const char * mpi2tcl_comm (MPI_Comm comm)
- static MPI_Comm tcl2mpi_comm (const char *label)
- static const char * tclmpi_add_comm (MPI_Comm comm)
- static int tclmpi_del_comm (const char *label)
- static int tclmpi_get_op (const char *opstr, MPI_Op *op)
- static const char * tclmpi_add_req ()
- static tclmpi_req_t * tclmpi_find_req (const char *label)
- static int tclmpi_del_req (tclmpi_req_t *req)
- static int tclmpi_datatype (const char *type)
- static int tclmpi_errcheck (Tcl_Interp *interp, int ierr, Tcl_Obj *obj)
- static int tclmpi_commcheck (Tcl_Interp *interp, MPI_Comm comm, Tcl_Obj *obj0, Tcl_Obj *obj1)
- static int tclmpi_typecheck (Tcl_Interp *interp, int type, Tcl_Obj *obj0, Tcl_Obj *obj1)

Variables

- static tclmpi_comm_t * first_comm = NULL
- static tclmpi_comm_t * last_comm = NULL
- static int tclmpi_comm_cntr = 0
- static MPI_Comm MPI_COMM_INVALID
- static tclmpi_req_t * first_req = NULL
- static int tclmpi_req_cntr = 0
- static int tclmpi_conv_handler = TCLMPI_ERROR
- static char tclmpi_errmsg [MPI_MAX_ERROR_STRING]

9.1.1 Detailed Description

9.1.2 Macro Definition Documentation

9.1.2.1 TCLMPI_ABORT

```
#define TCLMPI_ABORT -3
```

abort on problems

9.1.2.2 TCLMPI_AUTO

```
#define TCLMPI_AUTO 1
```

the tcl native data type (string)

9.1.2.3 TCLMPI_CONV_CHECK

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

Value:

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

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.

9.1.2.4 TCLMPI_DOUBLE

```
#define TCLMPI_DOUBLE 4
```

floating point data type

9.1.2.5 TCLMPI_DOUBLE_INT

```
#define TCLMPI_DOUBLE_INT 5
```

data type for double/integer pair

9.1.2.6 TCLMPI_ERROR

```
#define TCLMPI_ERROR -2
```

flag problems as Tcl errors

9.1.2.7 TCLMPI_INT

```
#define TCLMPI_INT 2
```

data type for integers

9.1.2.8 TCLMPI_INT_INT

```
#define TCLMPI_INT_INT 3
```

data type for pairs of integers

9.1.2.9 TCLMPI_INVALID

```
#define TCLMPI_INVALID -1
```

not ready to handle data

9.1.2.10 TCLMPI_LABEL_SIZE

```
#define TCLMPI_LABEL_SIZE 32
```

Size of stringbuffer for tclmpi labels

9.1.2.11 TCLMPI_NONE

```
#define TCLMPI_NONE 0
```

no data type assigned

9.1.2.12 TCLMPI_TOZERO

```
#define TCLMPI_TOZERO -4
```

convert problematic data items to zero

9.1.3 Typedef Documentation

9.1.3.1 tclmpi_comm_t

```
typedef struct tclmpi_comm tclmpi_comm_t
```

Linked list entry type for managing MPI communicators

9.1.3.2 tclmpi_dblint_t

```
typedef struct tclmpi_dblint tclmpi_dblint_t
```

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

9.1.3.3 tclmpi_intint_t

```
typedef struct tclmpi_intint tclmpi_intint_t
```

Data type for maxloc/minloc reductions with two integers

9.1.3.4 tclmpi_req_t

```
typedef struct tclmpi_req tclmpi_req_t
```

Linked list entry type for managing MPI requests

9.1.4 Function Documentation

9.1.4.1 mpi2tcl_comm()

```
static const char* mpi2tcl_comm (
            MPI_Comm comm )  [static]
```

Translate an MPI communicator to its Tcl label.

Parameters

comm	an MPI communicator

Returns

the corresponding string label or NULL.

This function will search through the linked list of known communicators until it finds the (first) match and then returns the string label to the calling function. If a NULL is returned, the communicator does not yet exist in the linked list.

9.1.4.2 tcl2mpi_comm()

```
static MPI_Comm tcl2mpi_comm (
            const char * label )  [static]
```

Translate a Tcl communicator label into the MPI communicator it represents.

Parameters

label	the Tcl name for the communicator

Returns

the matching MPI communicator or MPI_COMM_INVALID

This function will search through the linked list of known communicators until it finds the (first) match and then returns the string label to the calling function. If a NULL is returned, the communicator does not yet exist in the linked list.

9.1.4.3 tclmpi_add_comm()

```
static const char* tclmpi_add_comm (
            MPI_Comm comm )  [static]
```

Add an MPI communicator to the linked list of communicators, if needed.

Parameters

comm	an MPI communicator

Returns

the corresponding string label or NULL.

This function will first call mpi2tcl_comm in order to see, if the communicator handed in, is already listed and return that communicators Tcl label string. If it is not yet lists, a new entry is added to the linked list and a new label of the format "tclmpi::comm%d" assigned. The (global/static) variable tclmpi_comm_cntr is incremented every time to make the communicator label unique.

9.1.4.4 tclmpi_add_req()

```
static const char* tclmpi_add_req ( )  [static]
```

Allocate and add an entry to the request map linked list

Returns

the corresponding string label or NULL.

This function will allocate and initialize a new linked list entry for the translation between MPI requests and their string representation passed to Tcl scripts. The assigned label of the for "tclmpi::req%d" will be returned. The (global/static) variable tclmpi_req_cntr is incremented every time to make the communicator label unique.

9.1.4.5 tclmpi_commcheck()

```
static int tclmpi_commcheck (
            Tcl_Interp * interp,
            MPI_Comm comm,
            Tcl_Obj * obj0,
            Tcl_Obj * obj1 )  [static]
```

convenience function to report an unknown communicator as Tcl error

Parameters

interp	current Tcl interpreter
comm	MPI communicator
obj0	Tcl object representing the current command name
obj1	Tcl object representing the communicator as Tcl name

Returns

TCL_ERROR if the communicator is MPI_COMM_INVALID or TCL_OK

9.1.4.6 tclmpi_datatype()

```
static int tclmpi_datatype (
            const char * type )  [static]
```

convert a string describing a data type to a numeric representation

9.1.4.7 tclmpi_del_comm()

```
static int tclmpi_del_comm (
            const char * label )  [static]
```

Remove an MPI communicator from the linked list of communicators

Parameters

label	the Tcl name for the communicator

Returns

TCL_OK if deletion was successful, else TCL_ERROR

This function will find the entry in the linked list that matches the Tcl label string, remove it and free the associated resources.

9.1.4.8 tclmpi_del_req()

```
static int tclmpi_del_req (
            tclmpi_req_t * req )  [static]
```

remove tclmpi_req_t entry from the request linked list

Parameters

req	a pointer to the request in question

Returns

TCL_OK on succes, TCL_ERROR on failure

This function will search through the linked list of known MPI requests until it finds the (first) match and then will remove it from the linked and free the allocated storage. If TCL_ERROR is returned, the request did not exist in the linked list.

9.1.4.9 tclmpi_errcheck()

```
static int tclmpi_errcheck (
             Tcl_Interp * interp,
             int ierr,
             Tcl_Obj * obj )  [static]
```

convert MPI error code to Tcl error error message and append to result

Parameters

interp	current Tcl interpreter
ierr	MPI error number. return value of an MPI call.
obj	Tcl object representing the current command name

Returns

TCL_OK if the "error" is MPI_SUCCESS or TCL_ERROR

This is a convenience wrapper that will use MPI_Error_string() to convert any error code returned from MPI function calls to the respective error class and that into a string. This string is appended to the Tcl result vector of the current command. Should be called after each MPI call. Since we change error handlers on all communicators to not result in fatal errors, we have to generate Tcl errors instead (which can be caught).

9.1.4.10 tclmpi_find_req()

```
static tclmpi_req_t* tclmpi_find_req (
             const char * label )  [static]
```

translate Tcl representation of an MPI request to request itself.

Parameters

label	the Tcl name for the communicator

Returns

a pointer to the matching tclmpi_req_t structure

This function will search through the linked list of known MPI requests until it finds the (first) match and then returns a pointer to this data. If NULL is returned, the communicator does not exist in the linked list.

9.1.4.11 tclmpi_get_op()

```
static int tclmpi_get_op (
             const char * opstr,
             MPI_Op * op )  [static]
```

Translate TclMPI strings to MPI constants for reductions

Parameters

opstr	string constant describing the operator
op	pointer to location for storing the MPI constant

Returns

TCL_OK if the string was recognized else TCL_ERROR

This is a convenience function to consistently convert TclMPI string constants representing reduction operators to their corresponding MPI counterparts.

9.1.4.12 tclmpi_typecheck()

```
static int tclmpi_typecheck (
             Tcl_Interp * interp,
             int type,
             Tcl_Obj * obj0,
             Tcl_Obj * obj1 )  [static]
```

convenience function to report an unknown data type as Tcl error

Parameters

interp	current Tcl interpreter
type	TclMPI data type
obj0	Tcl object representing the current command name
obj1	Tcl object representing the data type as Tcl name

Returns

TCL_ERROR if the communicator is TCLMPI_NONE or TCL_OK

9.1.5 Variable Documentation

9.1.5.1 first_comm

```
tclmpi_comm_t* first_comm = NULL  [static]
```

First element of the communicator map list

9.1.5.2 first_req

```
tclmpi_req_t* first_req = NULL  [static]
```

First element of the list of generated requests

9.1.5.3 last_comm

```
tclmpi_comm_t* last_comm = NULL  [static]
```

Last element of the communicator map list

9.1.5.4 MPI_COMM_INVALID

```
MPI_Comm MPI_COMM_INVALID  [static]
```

Additional global communicator to detect unlisted communicators

9.1.5.5 tclmpi_comm_cntr

```
int tclmpi_comm_cntr = 0  [static]
```

Communicator counter. Incremented to get unique strings

9.1.5.6 tclmpi_conv_handler

```
int tclmpi_conv_handler = TCLMPI_ERROR  [static]
```

Selects what to do when a data element in a list cannot be successfully converted to the desired data type. Default is to throw a Tcl error.

9.1.5.7 tclmpi_errmsg

```
char tclmpi_errmsg[MPI_MAX_ERROR_STRING]  [static]
```

buffer for error messages.

9.1.5.8 tclmpi_req_cntr

```
int tclmpi_req_cntr = 0  [static]
```

Request counter. Incremented to get unique strings

9.2 TclMPI wrapper functions

Functions

- int TclMPI_Initialized (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Finalized (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- 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[])

9.2.1 Detailed Description

9.2.2 Function Documentation

9.2.2.1 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().

9.2.2.2 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.

9.2.2.3 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.

9.2.2.4 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.

9.2.2.5 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.

9.2.2.6 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.

9.2.2.7 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.

9.2.2.8 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.

9.2.2.9 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.

9.2.2.10 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++.

9.2.2.11 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++.

9.2.2.12 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.

9.2.2.13 TclMPI_Finalized()

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

wrapper for MPI_Finalized()

Returns

TCL_OK or TCL_ERROR

This function checks whether the MPI environment has been shut down.

9.2.2.14 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.

9.2.2.15 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.

9.2.2.16 TclMPI_Initialized()

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

wrapper for MPI_Initialized()

Returns

TCL_OK or TCL_ERROR

This function checks whether the MPI environment has been initialized.

9.2.2.17 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.

9.2.2.18 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.

9.2.2.19 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.

9.2.2.20 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).

9.2.2.21 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.

9.2.2.22 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.

9.2.2.23 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.

9.2.2.24 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.

9.2.2.25 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.

Chapter 10

Namespace Documentation

10.1 tclmpi Namespace Reference

Functions

- proc init ()
- proc initialized ()
- proc conv_set (handler)
- proc conv_get (handler)
- proc finalize ()
- proc finalized ()
- proc abort (comm, errorcode)
- proc comm_size (comm)
- proc comm_rank (comm)
- proc comm_split (comm, color, key)
- proc comm_free (comm)
- proc barrier (comm)
- proc bcast (data, type, root, comm)
- proc scatter (data, type, root, comm)
- proc allgather (data, type, comm)
- proc gather (data, type, root, comm)
- proc allreduce (data, type, op, comm)
- proc reduce (data, type, op, root, comm)
- proc send (data, type, dest, tag, comm)
- proc isend (data, type, dest, tag, comm)
- proc recv (type, source, tag, comm, status={})
- proc irecv (type, source, tag, comm)
- proc probe (source, tag, comm, status={})
- proc wait (request, status={})
- proc waitall (requests, status={})

Variables

- variable version = "1.2"

  version number of this package
- variable auto = tclmpi::auto

  constant for automatic data type
- variable int = tclmpi::int

  constant for integer data type
- variable intint = tclmpi::intint

  constant for integer pair data type
- variable double = tclmpi::double

  constant for double data type
- variable dblint = tclmpi::dblint

  constant for double/int pair data type
- variable comm_world = tclmpi::comm_world

  constant for world communicator
- variable comm_self = tclmpi::comm_self

  constant for self communicator
- variable comm_null = tclmpi::comm_null

  constant empty communicator
- variable any_source = tclmpi::any_source

  constant to accept messages from any source rank
- variable any_tag = tclmpi::any_tag

  constant to accept messages with any tag
- variable sum = tclmpi::sum

  summation operation
- variable prod = tclmpi::prod

  product operation
- variable max = tclmpi::max

  maximum operation
- variable min = tclmpi::min

  minimum operation
- variable land = tclmpi::land

  logical and operation
- variable band = tclmpi::band

  bitwise and operation
- variable lor = tclmpi::lor

  logical or operation
- variable bor = tclmpi::bor

  bitwise or operation
- variable lxor = tclmpi::lxor

  logical xor operation
- variable bxor = tclmpi::bxor

  bitwise xor operation
- variable maxloc = tclmpi::maxloc

  maximum and location operation
- variable minloc = tclmpi::minloc

  minimum and location operation
- variable error = tclmpi::error

  throw a Tcl error when a data conversion fails
- variable abort = tclmpi::abort

call MPI_Abort() when a data conversion fails

- variable tozero = tclmpi::tozero

  silently assign zero for failed data conversions

- variable undefined = tclmpi::undefined

  constant to indicate an undefined number

10.1.1 Detailed Description

TclMPI package Tcl namespace

10.1.2 Function Documentation

10.1.2.1 abort()

```
proc tclmpi::abort (
            comm ,
            errorcode  )
```

Terminates the MPI environment from Tcl

Parameters

comm	Tcl representation of an MPI communicator
errorcode	an integer that will be returned as exit code to the OS

This command makes a best attempt to abort all tasks sharing the communicator and exit with the provided error code. Only one task needs to call tclmpi::abort. This command terminates the program, so there can be no return value.

For implementation details see TclMPI_Abort().

10.1.2.2 allgather()

```
proc tclmpi::allgather (
            data ,
            type ,
            comm  )
```

Collects data from all processes on the communicator

Parameters

data	data to be distributed (Tcl data object)
type	data type to be used (string constant)
comm	Tcl representation of an MPI communicator

Returns

data that was collected or empty

This command collects data the provided list from all processes sharing the communicator. The data argument has to be present on all processes and has to be of the same length. The data resulting from the gather will be stored in the return value of the command for all processes. This function call is an implicit synchronization.

For implementation details see TclMPI_Allgather().

10.1.2.3 allreduce()

```
proc tclmpi::allreduce (
             data ,
             type ,
             op ,
             comm  )
```

Combines data from all processes and distributes the result back to them

Parameters

data	data to be reduced (Tcl data object)
type	data type to be used (string constant)
op	reduction operation (string constant)
comm	Tcl representation of an MPI communicator

Returns

data resulting from the reduction operation

This command performs a global reduction operation op on the provided data object across all processes participating in the communicator comm. If data is a list, then the reduction will be done across each respective entry of the same list index. The result is distributed to all processes and used as return value of the command. This command only supports the data types tclmpi::int and tclmpi::double and tclmpi::intint for operations tclmpi::maxloc and tclmpi::minloc. The following reduction operations are supported: tclmpi::max (maximum), tclmpi::min (minimum), tclmpi::sum (sum), tclmpi::prod (product), tclmpi::land (logical and), tclmpi::band (bitwise and), tclmpi::lor (logical or), tclmpi::bor (bitwise or), tclmpi::lxor (logical exclusive or), tclmpi::bxor (bitwise exclusive or), tclmpi::maxloc (max value and location), tclmpi::minloc (min value and location). This function call is an implicit synchronization.

For implementation details see TclMPI_Allreduce().

10.1.2.4 barrier()

```
proc tclmpi::barrier (
             comm  )
```

Synchronize MPI processes

Parameters

comm	Tcl representation of an MPI communicator

Blocks the caller until all processes sharing the communicator have called it; the call returns at any process only after **all** processes have entered the call and thus effectively synchronizes the processes. This function has no return value.

For implementation details see TclMPI_Barrier().

10.1.2.5 bcast()

```
proc tclmpi::bcast (
             data ,
             type ,
             root ,
             comm  )
```

Broadcasts data from one process to all processes on the communicator

Parameters

data	data to be broadcast (Tcl data object)
type	data type to be used (string constant)
root	rank of process that is providing the data (integer)
comm	Tcl representation of an MPI communicator

Returns

data that was broadcast

This command broadcasts the provided data object (list or single number or string) from the process with rank root on the communicator comm to all processes sharing the communicator. The data argument has to be present on all processes but will be ignored on all but the root process. The data resulting from the broadcast will be stored in the return value of the command on **all** processes. This is important when the data type is not tclmpi::auto, since using other data types may incur an irreversible conversion of the data elements. This function call is an implicit synchronization.

For implementation details see TclMPI_Bcast().

10.1.2.6 comm_free()

```
proc tclmpi::comm_free (
             comm  )
```

Deletes a dynamically created communicator and frees its resources

Parameters

comm	Tcl representation of an MPI communicator

This operation marks the MPI communicator associated with it Tcl representation comm for deallocation by the underlying MPI library. Any pending communications using this communicator will still complete normally.

For implementation details see TclMPI_Comm_free().

10.1.2.7 comm_rank()

```
proc tclmpi::comm_rank (
             comm  )
```

Returns the rank of the current process in an MPI communicator

Parameters

comm	Tcl representation of an MPI communicator

Returns

rank on the communicator (integer between 0 and size-1)

This function gives the rank of the process in the particular communicator. Many programs will be written with a manager-worker model, where one process (such as the rank-zero process) will play a supervisory role, and the other processes will serve as compute nodes. In this framework, tclmpi::comm_size and tclmpi::comm_rank are useful for determining the roles of the various processes of a communicator.

For implementation details see TclMPI_Comm_rank().

10.1.2.8 comm_size()

```
proc tclmpi::comm_size (
             comm  )
```

Returns the number of processes involved in an MPI communicator

Parameters

comm	Tcl representation of an MPI communicator

Returns

number of MPI processes on communicator

This function indicates the number of processes involved in a communicator. For tclmpi::comm_world, it indicates the total number of processes available. This call is often used in combination with tclmpi::comm_rank to determine the amount of concurrency available for a specific library or program. tclmpi::comm_rank indicates the rank of the process that calls it in the range from 0...size-1, where size is the return value of tclmpi::comm_size.

For implementation details see TclMPI_Comm_size().

10.1.2.9 comm_split()

```
proc tclmpi::comm_split (
             comm ,
             color ,
             key  )
```

Creates new communicators based on "color" and "key" flags

Parameters

comm	Tcl representation of an MPI communicator
color	subset assignment (non-negative integer or tclmpi::undefined)
key	relative rank assignment (integer)

Returns

Tcl representation of the newly created MPI communicator

This function partitions the group associated with comm into disjoint subgroups, one for each value of color. Each subgroup contains all processes of the same color. Within each subgroup, the processes are ranked in the order defined by the value of the argument key, with ties broken according to their rank in the old group. A new communicator is created for each subgroup and returned in newcomm. A process may supply the color value tclmpi::undefined, in which case the function returns tclmpi::comm_null. This is a collective call, but each process is permitted to provide different values for color and key.

The following example shows how to construct a communicator where the ranks are reversed in comparison to the world communicator.

```
set comm tclmpi::comm_world
set size [::tclmpi::comm_size $comm]
set key -[::tclmpi::comm_rank $comm]
set revcomm [::tclmpi::comm_split $comm 1 $key]
```

For implementation details see TclMPI_Comm_split().

10.1.2.10 conv_get()

```
proc tclmpi::conv_get (
             handler  )
```

Return a string constant naming the error handler for TclMPI data conversions

Returns

string constant for error handler

This function allows to query which error handler is currently active for Tcl data conversions inside TclMPI. For details on the error handlers, see tclmpi::conv_set.

For implementation details see TclMPI_Conv_get().

10.1.2.11 conv_set()

```
proc tclmpi::conv_set (
             handler  )
```

Set the error handler for TclMPI data conversions

Parameters

handler	string constant for error handler

This function sets what action TclMPI should take if a data conversion to tclmpi::int or tclmpi::double fails. When using data types other than tclmpi::auto, the corresponding data needs to be converted from the internal Tcl representatin to the selected native format. However, this does not always succeed for a variety of reasons. With this function TclMPI allows the programmer to define how this is handled. There are currently three handlers available: tclmpi::error (the default setting), tclmpi::abort, and tclmpi::tozero. For tclmpi::error a Tcl error is raised that can be intercepted with catch and TclMPI immediately returns to the calling function. For tclmpi::abort an error message is written directly to the screen 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. This command has no return value.

For implementation details see TclMPI_Conv_set().

10.1.2.12 finalize()

```
proc tclmpi::finalize ( )
```

Shut down the MPI environment from Tcl

This command closes the MPI environment and cleans up all MPI states. All processes much call this routine before exiting. Calling this function before calling tclmpi::init is an error. After calling this function, no more TclMPI commands including tclmpi::finalize and tclmpi::init may be used. This command takes no arguments and has no return value.

For implementation details see TclMPI_Finalize().

10.1.2.13 finalized()

```
proc tclmpi::finalized ( )
```

Check if MPI environment was finalized from Tcl

Returns

boolean value of whether MPI has been shut down

This command checks if tclmpi::finalize has already been called or whether the MPI environment has been shut down otherwise. Since initializing MPI multiple times is an error, you can call this function to determine whether you need to call tclmpi::finalize and whether it is (still) allowed to call tclmpi::init in your Tcl script. This command takes no arguments.

For implementation details see TclMPI_Finalized().

10.1.2.14 gather()

```
proc tclmpi::gather (
             data ,
             type ,
             root ,
             comm  )
```

Collects data from all processes on the communicator

Parameters

data	data to be distributed (Tcl data object)
type	data type to be used (string constant)
root	rank of process that will receive the data (integer)
comm	Tcl representation of an MPI communicator

Returns

data that was collected or empty

This command collects data the provided list from the process with rank root on the communicator comm to all processes sharing the communicator. The data argument has to be present on all processes and has to be of the same length. The data resulting from the gather will be stored in the return value of the command on the root process. This function call is an implicit synchronization. This procedure is the reverse operation of tclmpi::scatter.

For implementation details see TclMPI_Gather().

10.1.2.15 init()

```
proc tclmpi::init ( )
```

Initialize the MPI environment from Tcl

This command initializes the MPI environment. Needs to be called before any other TclMPI commands. MPI can be initialized at most once, so calling tclmpi::init multiple times is an error. Like in the C bindings for MPI, tclmpi::init will scan the argument vector, the global variable $argv, for any MPI implementation specific flags and will remove them. The global variable $argc will be adjusted accordingly. This command takes no arguments and has no return value.

For implementation details see TclMPI_Init().

10.1.2.16 initialized()

```
proc tclmpi::initialized ( )
```

Check if MPI environment is initialized from Tcl

Returns

boolean value of whether MPI has been initialized

This command checks if tclmpi::init has already been called or whether the MPI environment has been set up otherwise. Since initializing MPI multiple times is an error, you can call this function to determine whether you need to call tclmpi::init in your Tcl script. This command takes no arguments.

For implementation details see TclMPI_Initalized().

10.1.2.17 irecv()

```
proc tclmpi::irecv (
            type ,
            source ,
            tag ,
            comm  )
```

Initiate a non-blocking receive

Parameters

type	data type to be used (string constant)
source	rank of sending process or tclmpi::any_source
tag	message identification tag or tclmpi::any_tag
comm	Tcl representation of an MPI communicator

Returns

Tcl representation of generated MPI request

This procedure provides a non-blocking receive operation, i.e. it returns **immediately**. The call does not return any data but a request handle of the form tclmpi::req#, with # being a unique integer number. This request handle is best stored in a variable and needs to be passed to a tclmpi::wait call to wait for completion of the receive and pass the data to the calling code as return value of the wait call. The type argument has to match that of the corresponding send command. Instead of a specific source rank, the constant tclmpi::any_source can be used and similarly tclmpi::any_tag as tag, to not select on source rank or tag, respectively.

For implementation details see TclMPI_Irecv().

10.1.2.18 isend()

```
proc tclmpi::isend (
             data ,
             type ,
             dest ,
             tag ,
             comm  )
```

Perform a non-blocking send

Parameters

data	data to be sent (Tcl data object)
type	data type to be used (string constant)
dest	rank of destination process (non-negative integer)
tag	message identification tag (integer)
comm	Tcl representation of an MPI communicator

Returns

Tcl representation of generated MPI request

This function performs a regular **non-blocking** send to process rank dest on communicator comm. The choice of data type determines how data is being sent and thus unlike in the C-bindings the corresponding receive has to use the same data data type. As a non-blocking call, the function will return immediately. The return value is a string representing the generated MPI request and it can be passed to a call to tclmpi::wait in order to wait for its completion and release all reserved storage associated with the request.

For implementation details see TclMPI_Isend().

10.1.2.19 probe()

```
proc tclmpi::probe (
             source ,
             tag ,
             comm ,
             status  = {} )
```

Blocking test for a message

Parameters

source	rank of sending process or tclmpi::any_source
tag	message identification tag or tclmpi::any_tag
comm	Tcl representation of an MPI communicator
status	variable name for status array (string)

Returns

empty

This function allows to check for an incoming message on the communicator comm without actually receiving it. Nevertheless, this call is blocking, i.e. it will not return unless there is actually a message pending that matches the requirements of source rank and message tag. Instead of a specific source rank, the constant tclmpi::any_source can be used and similarly tclmpi::any_tag as tag, to accept send requests from any rank or tag, respectively. The (optional) status argument would be the name of a variable in which status information about the message will be stored in the form of an array. This associative array has the entries MPI_SOURCE (rank of sender), MPI_TAG (tag of message), COUNT_CHAR (size of message in bytes), COUNT_INT (size of message in tclmpi::int units), COUNT_DOUBLE (size of message in tclmpi::double units).

For implementation details see TclMPI_Probe().

10.1.2.20 recv()

```
proc tclmpi::recv (
             type ,
             source ,
             tag ,
             comm ,
             status  = {} )
```

Perform a blocking receive

Parameters

type	data type to be used (string constant)
source	rank of sending process or tclmpi::any_source
tag	message identification tag or tclmpi::any_tag
comm	Tcl representation of an MPI communicator
status	variable name for status array (string)

Returns

the received data

This procedure provides a blocking receive operation, i.e. it only returns **after** the message is received in full. The received data will be passed as return value. The type argument has to match that of the corresponding send command. Instead of using a specific source rank, the constant tclmpi::any_source can be used and similarly tclmpi::any_tag as tag. This way the receive operation will not select a message based on source rank or tag, respectively. The (optional) status argument would be the name of a variable in which status information about the receive will be stored in the form of an array. The associative array has the entries MPI_SOURCE (rank of sender), MPI_TAG (tag of message), COUNT_CHAR (size of message in bytes), COUNT_INT (size of message in tclmpi::int units), COUNT_DOUBLE (size of message in tclmpi::double units).

For implementation details see TclMPI_Recv().

10.1.2.21 reduce()

```
proc tclmpi::reduce (
             data ,
             type ,
             op ,
             root ,
             comm  )
```

Combines data from all processes on one process

Parameters

data	data to be reduced (Tcl data object)
type	data type to be used (string constant)
op	reduction operation (string constant)
root	rank of process that is receiving the result (integer)
comm	Tcl representation of an MPI communicator

Returns

data resulting from the reduction operation

This command performs a global reduction operation op on the provided data object across all processes participating in the communicator comm. If data is a list, then the reduction will be done across each respective entry of the same list index. The result is collect on the process with rank root and used as return value of the command. For all other processes the return value is empty. This command only supports the data types tclmpi::int and tclmpi::double and tclmpi::intint for operations tclmpi::maxloc and tclmpi::minloc. The following reduction operations are supported: tclmpi::max (maximum), tclmpi::min (minimum), tclmpi::sum (sum), tclmpi::prod (product), tclmpi::land (logical and), tclmpi::band (bitwise and), tclmpi::lor (logical or), tclmpi::bor (bitwise or), tclmpi::lxor (logical exclusive or), tclmpi::bxor (bitwise exclusive or), tclmpi::maxloc (max value and location), tclmpi::minloc (min value and location). This function call is an implicit synchronization.

For implementation details see TclMPI_Reduce().

10.1.2.22 scatter()

```
proc tclmpi::scatter (
             data ,
             type ,
             root ,
             comm  )
```

Distributes data from one process to all processes on the communicator

Parameters

data	data to be distributed (Tcl data object)
type	data type to be used (string constant)
root	rank of process that is providing the data (integer)
comm	Tcl representation of an MPI communicator

Returns

data that was distributed

This command distributes the provided list of data from the process with rank root on the communicator comm to all processes sharing the communicator. The data argument has to be present on all processes but will be ignored on all but the root process. The data resulting from the scatter will be stored in the return value of the command. The data will be distributed evenly, so the length of the list has to be divisible by the number of processes on the communicator. This procedure is the reverse operation of tclmpi::gather. This function call is an implicit synchronization.

For implementation details see TclMPI_Scatter().

10.1.2.23 send()

```
proc tclmpi::send (
             data ,
             type ,
             dest ,
             tag ,
             comm  )
```

Perform a blocking send

Parameters

data	data to be sent (Tcl data object)
type	data type to be used (string constant)
dest	rank of destination process (non-negative integer)
tag	message identification tag (integer)
comm	Tcl representation of an MPI communicator

This function performs a regular **blocking** send to process rank dest on communicator comm. The choice of data type determines how data is being sent and thus unlike in the C-bindings the corresponding receive has to use the

same data data type. As a blocking call, the function will only return when all data is sent. This function has no return value.

For implementation details see TclMPI_Send().

10.1.2.24 wait()

```
proc tclmpi::wait (
             request ,
             status  = {} )
```

Non-blocking test for a message

Parameters

source	rank of sending process or tclmpi::any_source
tag	message identification tag or tclmpi::any_tag
comm	Tcl representation of an MPI communicator
status	variable name for status array (string)

Returns

1 or 0 depending on whether a pending request was detected

This function allows to check for an incoming message on the communicator comm without actually receiving it. Unlike tclmpi::probe, this call is non-blocking, i.e. it will return immediately and report whether there is a message pending or not in its return value (1 or 0, respectively). Instead of a specific source rank, the constant tclmpi::any_source can be used and similarly tclmpi::any_tag as tag, to test for send requests from any rank or tag, respectively. The (optional) status argument would be the name of a variable in which status information about the message will be stored in the form of an array. This associative array has the entries MPI_SOURCE (rank of sender), MPI_TAG (tag of message), COUNT_CHAR (size of message in bytes), COUNT_INT (size of message in tclmpi::int units), COUNT_DOUBLE (size of message in tclmpi::double units).

For implementation details see TclMPI_Iprobe(). Wait for MPI request completion

Parameters

request	Tcl representation of an MPI request
status	variable name for status array (string)

Returns

empty or received data that was associated with the request

This function takes a communication request created by a non-blocking send or receive call (tclmpi::isend or tclmpi::irecv) and waits for its completion. In case of a send, it will merely wait until the matching communication is completed and any resources associated with the request will be releaseed. If the request was generated by a non-blocking receive call, tclmpi::wait will hand the received data to the calling routine in its return value. The (optional) status argument would be the name of a variable in which the resulting status information will be stored in the form of an associative array. The associative array will have the entries MPI_SOURCE (rank of sender), MPI_TAG (tag of message), COUNT_CHAR (size of message in bytes), COUNT_INT (size of message in tclmpi::int units), COUNT_DOUBLE (size of message in tclmpi::double units).

For implementation details see TclMPI_Wait().

10.1.2.25 waitall()

```
proc tclmpi::waitall (
             requests ,
             status  = {} )
```

Wait for multiple MPI request completions

Parameters

requests	List of Tcl representations of an MPI request
status	variable name to store list with deserialization of the status arrays (string)

Returns

empty or list of received data that was associated with the request

This function takes a list communication requests created by non-blocking send or receive calls (tclmpi::isend or tclmpi::irecv) and waits for the completion of all of them. In case of a send, it will merely wait until the matching communication is completed and any resources associated with the request will be releaseed. If the request was generated by a non-blocking receive call, tclmpi::wait will hand the received data to the calling routine in its return value. The (optional) status argument would be the name of a variable in which the resulting status information will be stored in the form of a list of lists, one entry per request. The corresponding associative arrays - like they are created by tclmpi::wait - can reconstituted using the `array set` command.

```
::tclmpi::waitall [list $req1 $req2] status;  * [array set status1 [lindex $status 0]
[array set status2 [lindex $status 1];  *
```

; ∗; ∗ This call is implemented in Tcl as a wrapper around tclmpi::wait

10.2 tclmpi_test Namespace Reference

Functions

- proc test_format (kind, cmd, result)
- proc compare (reflist, result)
- proc ser_init (args)
- proc par_init (args)
- proc run_return (cmd, retval)
- proc run_error (cmd, errormsg)
- proc par_return (cmd, retval, comm=tclmpi::comm_world)
- proc par_error (cmd, retval, comm=tclmpi::comm_world)
- proc test_summary (section)

Variables

- set version

  version of the package
- variable comm = tclmpi::comm_world

  shortcut for world communicator
- variable master = 0

  rank of MPI master process

- variable rank = 0

  rank of this MPI process on $comm
- variable size = 1

  number of processes on $comm
- variable int = tclmpi::int

  shortcut for tclmpi::int data type
- variable intint = tclmpi::intint

  shortcut for tclmpi::intint data type
- variable maxloc = tclmpi::maxloc

  shortcut for tclmpi::maxloc operator
- variable minloc = tclmpi::minloc

  shortcut for tclmpi::minloc operator
- variable pass = 0

  counter for successful tests
- variable fail = 0

  counter for failed tests

10.2.1 Detailed Description

TclMPI test harness implementation namespace

This namespace contains several Tcl procedures that are used to conduct unit tests on the TclMPI package. For simplicity paths are hardcoded, so that this file must not be moved around and stay in the same directory as the individual tests, which in turn have to be in a subdirectory of the directory where the TclMPI shared object and/or the tclmpish extended Tcl shell reside.

10.2.2 Function Documentation

10.2.2.1 compare()

```
proc tclmpi_test::compare (
             reflist ,
             result  )
```

partial result and error message comparison

Parameters

reflist	list of strings that have to appear in the result
result	result string

Returns

1 if all reflist strings were found in result

This function does an inexact comparison of the reference data to the actual result. The reference is a list of strings, each of which has to be matched in a case insensitive string search. The function returns a 1 if all tests did match.

10.2.2.2 par_error()

```
proc tclmpi_test::par_error (
             cmd ,
             retval ,
             comm  = tclmpi::comm_world )
```

run a parallel test that is expected to produce a Tcl error

Parameters

cmd	list of strings or lists with the commands to execute
retval	list of the expected error message(s) or return values
comm	communicator. defaults to world communicator

Returns

empty

This function executes the lists command lines passed in $cmd in parallel each command taken from the list based on the rank of the individual MPI task on the communicator and intercepts its resulting error message or return value using the 'catch' command. It is then checked if one of the commands failed as expected and actual return value are then compared against the expected reference passed in the $retval list with similar assignments to the individual ranks as the commands. If one of the strings does not match or all command unexpectedly succeeded failure is reported otherwise success.

10.2.2.3 par_init()

```
proc tclmpi_test::par_init (
             args  )
```

init for parallel tests

Parameters

args	all parameters are ignored

Returns

empty

This function will perform an initialization of the parallel environment for subsequent parallel tests. It also initializes the global variables $rank and $size.

10.2.2.4 par_return()

```
proc tclmpi_test::par_return (
             cmd ,
             retval ,
             comm  = tclmpi::comm_world )
```

run a parallel test that is expected to succeed

Parameters

cmd	list of strings or lists with the commands to execute
retval	list of the expected return values
comm	communicator. defaults to world communicator

Returns

empty

This function executes the lists command lines passed in $cmd in parallel each command taken from the list based on the rank of the individual MPI task on the communicator and intercepts its resulting return value using the 'catch' command. The actual return value is then compared against the expected reference passed in the $retval list, similarly assigned to the individual ranks as the commands. The result is compared on all ranks and if one of the commands failed or the actual return value is not equal to the expected one, failure is reported and both, expected and and actual results are printed on one of the failing ranks. The error reporting expects that the MPI communicator remains usable after failure.

10.2.2.5 run_error()

```
proc tclmpi_test::run_error (
             cmd ,
             errormsg  )
```

run a serial test that is expected to fail

Parameters

cmd	string or list with the command to execute
errormsg	expected error message contents

Returns

empty

This function executes the command line passed in $cmd and intercepts its resulting error using the 'catch' command. The actual error message is then compared against the expected reference passed in $errormsg. The test is passed if the two strings match, otherwise failure is reported and both the expected and actual error messages are printed. Also an unexpectedly successful execution is considered a failure and its result reported for reference.

10.2.2.6 run_return()

```
proc tclmpi_test::run_return (
             cmd ,
             retval  )
```

run a serial test that is expected to succeed

Parameters

cmd	string or list with the command to execute
retval	expected return value contents

Returns

empty

This function executes the command line passed in $cmd and intercepts its resulting return value using the 'catch' command. The actual return value is then compared against the expected reference passed in $retval. The test is passed if the two strings match, otherwise failure is reported and both the expected and actual results are printed. Also an unexpectedly failure of the command is reported as failure and the resulting error message is reported for debugging.

10.2.2.7 ser_init()

```
proc tclmpi_test::ser_init (
            args  )
```

init for serial tests

Parameters

args	all parameters are ignored

Returns

empty

This function will perform a simple init test requesting the tclmpi package and matching it against the current verison number. It will also initialize some commonly used global variables. If called from a parallel environment, it will only execute and produce output on the master process

10.2.2.8 test_format()

```
proc tclmpi_test::test_format (
            kind ,
            cmd ,
            result  )
```

format output

Parameters

kind	string representing the kind of test (max 11 chars).
cmd	string representing the command. will be truncated as needed.
result	string indicating the result (PASS or FAIL/reason)

Returns

the formatted string

This function will format a test summary message, so that it does not break the output on a regular terminal screen. The first column will be the total number of the test computed from the sum of passed and failed tests, followed by a string describing the test type, the command executed and a result string. The command string in the middle will be truncated as needed to not break the format.

10.2.2.9 test_summary()

```
proc tclmpi_test::test_summary (
             section  )
```

print result summary

Parameters

section	number of the test section

Returns

empty

This function will print a nicely formatted summary of the tests. If executed in parallel only the master rank of the world communicator will produce output.

Chapter 11

Data Structure Documentation

11.1 tclmpi_comm Struct Reference

Collaboration diagram for tclmpi_comm:

Data Fields

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

11.1.1 Detailed Description

Linked list entry to map MPI communicators to strings.

11.1.2 Field Documentation

11.1.2.1 comm

`MPI_Comm tclmpi_comm::comm`

MPI communicator corresponding of this entry

11.1.2.2 label

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

String representing the communicator in Tcl

11.1.2.3 next

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

Pointer to next element in linked list

11.1.2.4 valid

```
int tclmpi_comm::valid
```

Non-zero if communicator is valid

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

- _tclmpi.c

11.2 tclmpi_dblint Struct Reference

Data Fields

- double d
- int i

11.2.1 Detailed Description

Represent a double/integer pair

11.2.2 Field Documentation

11.2.2.1 d

```
double tclmpi_dblint::d
```

double data value

11.2.2.2 i

```
int tclmpi_dblint::i
```

location data

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

- _tclmpi.c

11.3 tclmpi_intint Struct Reference

Data Fields

- int i1
- int i2

11.3.1 Detailed Description

Represent an integer/integer pair

11.3.2 Field Documentation

11.3.2.1 i1

```
int tclmpi_intint::i1
```

integer data value

11.3.2.2 i2

```
int tclmpi_intint::i2
```

location data

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

- _tclmpi.c

11.4 tclmpi_req Struct Reference

Collaboration diagram for tclmpi_req:

Data Fields

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

11.4.1 Detailed Description

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

11.4.2 Field Documentation

11.4.2.1 comm

```
MPI_Comm tclmpi_req::comm
```

communicator for non-blocking receive

11.4.2.2 data

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

pointer to send or receive data buffer

11.4.2.3 label

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

identifier of this request

11.4.2.4 len

```
int tclmpi_req::len
```

size of data block

11.4.2.5 next

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

pointer to next struct

11.4.2.6 req

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

pointer MPI request handle generated by MPI

11.4.2.7 source

```
int tclmpi_req::source
```

source rank of non-blocking receive

11.4.2.8 tag

```
int tclmpi_req::tag
```

tag selector of non-blocking receive

11.4.2.9 type

```
int tclmpi_req::type
```

data type of send data

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

- _tclmpi.c

Chapter 12

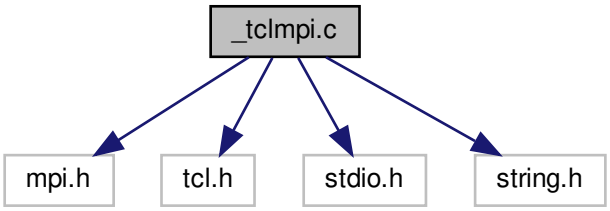
File Documentation

12.1 _tclmpi.c File Reference

This file contains the C code with the Tcl MPI wrappers.

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

Include dependency graph for _tclmpi.c:



Data Structures

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

Macros

- #define TCLMPI_LABEL_SIZE 32
- #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

- static const char * mpi2tcl_comm (MPI_Comm comm)
- static MPI_Comm tcl2mpi_comm (const char *label)
- static const char * tclmpi_add_comm (MPI_Comm comm)
- static int tclmpi_del_comm (const char *label)
- static int tclmpi_get_op (const char *opstr, MPI_Op *op)
- static const char * tclmpi_add_req ()
- static tclmpi_req_t * tclmpi_find_req (const char *label)
- static int tclmpi_del_req (tclmpi_req_t *req)
- static int tclmpi_datatype (const char *type)
- static int tclmpi_errcheck (Tcl_Interp *interp, int ierr, Tcl_Obj *obj)
- static int tclmpi_commcheck (Tcl_Interp *interp, MPI_Comm comm, Tcl_Obj *obj0, Tcl_Obj *obj1)
- static int tclmpi_typecheck (Tcl_Interp *interp, int type, Tcl_Obj *obj0, Tcl_Obj *obj1)
- int TclMPI_Initialized (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int TclMPI_Finalized (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- 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[])
- static void tclmpi_init_api (Tcl_Interp *interp)
- int _tclmpi_Init (Tcl_Interp *interp)

Variables

- static tclmpi_comm_t * first_comm = NULL
- static tclmpi_comm_t * last_comm = NULL
- static int tclmpi_comm_cntr = 0
- static MPI_Comm MPI_COMM_INVALID
- static tclmpi_req_t * first_req = NULL
- static int tclmpi_req_cntr = 0
- static int tclmpi_conv_handler = TCLMPI_ERROR
- static char tclmpi_errmsg [MPI_MAX_ERROR_STRING]

12.1.1 Detailed Description

This file contains the C code with the Tcl MPI wrappers.

12.1.2 Function Documentation

12.1.2.1 _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.

12.1.2.2 tclmpi_init_api()

```
static void tclmpi_init_api (
            Tcl_Interp * interp )  [static]
```

initialize TclMPI extensions and do one time init

Parameters

interp	pointer to current Tcl interpreter

This hooks up the commands provided by TclMPI into the provided interpreter and also initializes the predefined communicators tclmpi::comm_world, tclmpi::comm_self, and tclmpi::comm_null and its corresponding MPI counterparts.

Index