TclMPI

1.1

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.

1.0.1 Homepage:

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

For 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.

1.0.2 Test Status:

1.0.3 Citing:

You can cite TclMPI as:

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

1.0.4 Acknowledgements:

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

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

Chapter 2

Copyright and License for TclMPI

Copyright (c) 2012,2016,2017,2018,2019,2020,2021 Axel Kohlmeyer akohlmey@gmail.com All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the author of this software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL Axel Kohlmeyer BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Chapter 3

TclMPI User's Guide

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

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

3.1 Compilation and Installation

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

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

To configure and build TclMPI you need to run CMake the usual way, for example 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)
- BUILD_TESTING Enable unit testing (default: on)
- DOWNLOAD_MPICH4WIN Download MPICH2-1.4.1 headers and link library (default: off, Windows only)

- CMAKE_INSTALL_PREFIX Path to installation location prefix (default: (platform specific))

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

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

3.2 Software Development and Bug Reports

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

3.3 Example Programs

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

3.3.1 Hello World

This is the TclMPI version of "hello world".

```
#!/bin/sh \
exec tclsh "$0" "$@"
package require tclmpi 1.1
# initialize MPI
::tclmpi::init
# get size of communicator and rank of process
set comm tclmpi::comm_world
set size [::tclmpi::comm_size $comm]
set rank [::tclmpi::comm_rank $comm]
puts "hello world, this is rank $rank of $size"
# shut down MPI
::tclmpi::finalize
exit 0
```

3.3.2 Computation of Pi

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

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

```
#!/bin/sh \
exec tclsh "$0" "$@"
package require tclmpi 1.1
# initialize MPI
::tclmpi::init
set comm tclmpi::comm_world
set size [::tclmpi::comm_size $comm]
```

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

3.3.3 Distributed Sum

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

```
#!/bin/sh \
exec tclsh "$0" "$@"
package require tclmpi 1.1
# data summation helper function
proc sum {data} {
    set sum 0
    foreach d $data {
        set sum [expr {$sum + $d}]
    }
    return $sum
}
::tclmpi::init
set comm       $tclmpi::comm_world
set mpi_sum    $tclmpi::sum
set mpi_double $tclmpi::double
set mpi_int    $tclmpi::int
set size [::tclmpi::comm_size $comm]
set rank [::tclmpi::comm_rank $comm]
set master 0
# The master creates the list of data
#
set dataSize 1000000
set data {}
if { $comm == $master } {
    set mysum 0
    for { set i 0 } { $i < $dataSize } { incr i } {
        lappend data $i
    }
}
# add padding, so the number of data elements is divisible
# by the number of processors as required by tclmpi::scatter
set needpad [expr {$dataSize % $size}]
set numpad [expr {$needpad ? ($size - $needpad) : 0}]
if { [comm_rank $comm] == $master } {
    for {set i 0} {$i < $numpad} {incr i} {
        lappend data 0
    }
}
set blocksz [expr {($dataSize + $numpad)/ $size}]
# distribute data and do the summation on each node
# the sum the result across all nodes. Note: the data
# is integer, but we need to do the full sum in double
# precison to avoid overflows.
set mydata [::tclmpi::scatter $data $mpi_int $master $comm]
set sum [::tclmpi::allreduce [sum $mydata] $mpi_double $mpi_sum $comm]
if { $comm == $master } {
    puts "Distributed sum: $sum"
}
::tclmpi::finalize
```

Chapter 4

TclMPI Developer's Guide

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

4.1 Overall Design and Differences to the MPI C-bindings

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

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

4.2 Naming Conventions

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

Chapter 5

Namespace Index

5.1 Namespace List

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

Chapter 6

Class Index

6.1 Class List

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

Chapter 7

Namespace Documentation

7.1 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 = "1.1"

  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

7.1.1 Detailed Description

TclMPI package Tcl namespace

7.1.2 Function Documentation

7.1.2.1 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().

7.1.2.2 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().

7.1.2.3 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().

7.1.2.4 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().

7.1.2.5 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().

7.1.2.6 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().

7.1.2.7 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().

7.1.2.8 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().

7.1.2.9 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().

7.1.2.10 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().

7.1.2.11 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().

7.1.2.12 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().

7.1.2.13 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().

7.1.2.14 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().

7.1.2.15 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().

7.1.2.16 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().

7.1.2.17 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().

7.1.2.18 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().

7.1.2.19 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().

7.1.2.20 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().

7.1.2.21 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().

7.1.2.22 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().

7.1.2.23 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().

7.1.2.24 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().

7.1.2.25 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 for array with list of statuses (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 call (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 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) and the results will be stored as lists with the information in the same order as the list of requests.

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

7.2 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

7.2.1 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.

7.2.2 Function Documentation

7.2.2.1 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.

7.2.2.2 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.

7.2.2.3 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.

7.2.2.4 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.

7.2.2.5 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.

7.2.2.6 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.

7.2.2.7 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

7.2.2.8 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.

7.2.2.9 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 8

Class Documentation

8.1 tclmpi_comm Struct Reference

Collaboration diagram for tclmpi_comm:

Public Attributes

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

8.1.1 Detailed Description

Linked list entry to map MPI communicators to strings.

8.1.2 Member Data Documentation

8.1.2.1 comm

`MPI_Comm tclmpi_comm::comm`

MPI communicator corresponding of this entry

8.1.2.2 label

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

String representing the communicator in Tcl

8.1.2.3 next

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

Pointer to next element in linked list

8.1.2.4 valid

```
int tclmpi_comm::valid
```

Non-zero if communicator is valid

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

- _tclmpi.c

8.2 tclmpi_dblint Struct Reference

Public Attributes

- double d
- int i

8.2.1 Detailed Description

Represent a double/integer pair

8.2.2 Member Data Documentation

8.2.2.1 d

```
double tclmpi_dblint::d
```

double data value

8.2.2.2 i

```
int tclmpi_dblint::i
```

location data

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

- _tclmpi.c

8.3 tclmpi_intint Struct Reference

Public Attributes

- int i1
- int i2

8.3.1 Detailed Description

Represent an integer/integer pair

8.3.2 Member Data Documentation

8.3.2.1 i1

```
int tclmpi_intint::i1
```

integer data value

8.3.2.2 i2

```
int tclmpi_intint::i2
```

location data

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

- _tclmpi.c

8.4 tclmpi_req Struct Reference

Collaboration diagram for tclmpi_req:

Public Attributes

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

8.4.1 Detailed Description

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

8.4.2 Member Data Documentation

8.4.2.1 comm

```
MPI_Comm tclmpi_req::comm
```

communicator for non-blocking receive

8.4.2.2 data

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

pointer to send or receive data buffer

8.4.2.3 label

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

identifier of this request

8.4.2.4 len

```
int tclmpi_req::len
```

size of data block

8.4.2.5 next

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

pointer to next struct

8.4.2.6 req

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

pointer MPI request handle generated by MPI

8.4.2.7 source

```
int tclmpi_req::source
```

source rank of non-blocking receive

8.4.2.8 tag

```
int tclmpi_req::tag
```

tag selector of non-blocking receive

8.4.2.9 type

```
int tclmpi_req::type
```

data type of send data

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

- _tclmpi.c

Index