

TcIMPI - TcI Bindings for MPI
0.6

Generated by Doxygen 1.8.1

Mon May 21 2012 19:09:00

Contents

1	TclMPI User's Guide	1
1.1	Compilation and Installation	1
1.2	Software Development and Bug Reports	1
2	TclMPI Developer's Guide	3
2.1	Overall Design and Differences to the MPI C-bindings	3
2.2	TclMPI Support Functions	3
2.2.1	Mapping Communicators	3
3	Namespace Index	5
3.1	Namespace List	5
4	Data Structure Index	7
4.1	Data Structures	7
5	File Index	9
5.1	File List	9
6	Namespace Documentation	11
6.1	tclmpi Namespace Reference	11
6.1.1	Detailed Description	12
6.1.2	Function Documentation	12
6.1.2.1	abort	12
6.1.2.2	allreduce	12
6.1.2.3	barrier	12
6.1.2.4	bcast	12
6.1.2.5	comm_rank	13
6.1.2.6	comm_size	13
6.1.2.7	comm_split	13
6.1.2.8	finalize	13
6.1.2.9	init	13
6.1.2.10	iprobe	13
6.1.2.11	irecv	14

6.1.2.12	isend	14
6.1.2.13	probe	14
6.1.2.14	recv	14
6.1.2.15	send	14
6.1.2.16	wait	15
6.1.2.17	X	15
7	Data Structure Documentation	17
7.1	tclmpi_comm_t Struct Reference	17
7.1.1	Detailed Description	17
7.1.2	Field Documentation	17
7.1.2.1	comm	17
7.1.2.2	label	17
7.1.2.3	next	17
7.1.2.4	valid	17
7.2	tclmpi_req_t Struct Reference	18
7.2.1	Detailed Description	18
7.2.2	Field Documentation	18
7.2.2.1	comm	18
7.2.2.2	data	18
7.2.2.3	label	18
7.2.2.4	len	18
7.2.2.5	next	18
7.2.2.6	req	18
7.2.2.7	source	18
7.2.2.8	tag	19
7.2.2.9	type	19
8	File Documentation	21
8.1	_tclmpi.c File Reference	21
8.1.1	Detailed Description	22
8.1.2	Macro Definition Documentation	22
8.1.2.1	TCLMPI_AUTO	22
8.1.2.2	TCLMPI_DOUBLE	22
8.1.2.3	TCLMPI_DOUBLE_INT	23
8.1.2.4	TCLMPI_INT	23
8.1.2.5	TCLMPI_INT_INT	23
8.1.2.6	TCLMPI_INVALID	23
8.1.2.7	TCLMPI_NONE	23
8.1.3	Function Documentation	23
8.1.3.1	_tclmpi_Init	23

8.1.3.2	mpi2tcl_comm	24
8.1.3.3	tcl2mpi_comm	25
8.1.3.4	TclMPI_Abort	26
8.1.3.5	tclmpi_add_comm	27
8.1.3.6	tclmpi_add_req	28
8.1.3.7	TclMPI_Allreduce	29
8.1.3.8	TclMPI_Barrier	30
8.1.3.9	TclMPI_Bcast	30
8.1.3.10	TclMPI_Comm_rank	31
8.1.3.11	TclMPI_Comm_size	32
8.1.3.12	TclMPI_Comm_split	33
8.1.3.13	tclmpi_commcheck	34
8.1.3.14	tclmpi_datatype	35
8.1.3.15	tclmpi_del_req	36
8.1.3.16	tclmpi_errcheck	37
8.1.3.17	TclMPI_Finalize	38
8.1.3.18	tclmpi_find_req	39
8.1.3.19	TclMPI_Init	39
8.1.3.20	TclMPI_Iprobe	40
8.1.3.21	TclMPI_Irecv	41
8.1.3.22	TclMPI_Isend	43
8.1.3.23	TclMPI_Probe	44
8.1.3.24	TclMPI_Recv	45
8.1.3.25	TclMPI_Send	46
8.1.3.26	tclmpi_typecheck	47
8.1.3.27	TclMPI_Wait	48
8.1.4	Variable Documentation	49
8.1.4.1	first_comm	49
8.1.4.2	first_req	49
8.1.4.3	last_comm	49
8.1.4.4	MPI_COMM_INVALID	49
8.1.4.5	tclmpi_comm_cntr	49
8.1.4.6	tclmpi_errmsg	50
8.1.4.7	tclmpi_init_done	50
8.1.4.8	tclmpi_req_cntr	50
8.2	tclmpi.tcl File Reference	50
8.2.1	Detailed Description	51
8.3	tests/harness.tcl File Reference	51
8.3.1	Detailed Description	51
8.3.2	Function Documentation	51

8.3.2.1	par_error	51
8.3.2.2	par_init	52
8.3.2.3	par_return	52
8.3.2.4	par_set	52
8.3.2.5	run_error	53
8.3.2.6	run_return	53
8.3.2.7	ser_init	53
8.3.2.8	test_format	54
8.3.2.9	test_summary	54

Chapter 1

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.

1.1 Compilation and Installation

The package currently consist of a single C source file which needs to be compiled for dynamic linkage. The corresponding commands for Linux and MacOSX systems are included in the provided makefile. All that is required to compile the package is an installed Tcl development system and a working MPI installation. Since this creates a dynamically loaded shared object (DSO), both Tcl and MPI have to be compiled and linked as shared libraries (this is the default for Tcl and OpenMPI on Linux, but your mileage may vary). As of May 15 2012 the code has only been tested on 32-bit and 64-bit x86 Linux platforms with OpenMPI.

To compile the package adjust the settings in the Makefile according to your platform, MPI and Tcl installation. For most Linux distributions, this requires installing not only an MPI and Tcl package, but also the corresponding development packages, e.g. on Fedora you need `openmpi`, `openmpi-devel`, `tcl`, and `tcl-devel` and their dependencies. Then type `make` to compile the `tclmpi.so` file. With `make check` you can run the integrated unittest package to see, if everything is working as expected.

To install you can create a directory, e.g. `/usr/local/libexec/tclmpi`, and copy the files `tclmpi.so` and `pkgIndex.tcl` into it. If you then use the command `set auto_path [concat /usr/local/libexec/tclmpi $auto_path]` in your `.tclshrc` or `.vmdrc`, you can load the `tclmpi` wrappers on demand simply by using the command `package require tclmpi`.

1.2 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/akohlmeier/tclmpi> From there you can download snapshots of the development and releases, clone 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/akohlmeier/tclmpi/issues>.

Chapter 2

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.

2.1 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 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.

2.2 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.

2.2.1 Mapping 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 communicator to the list.

Chapter 3

Namespace Index

3.1 Namespace List

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

tclmpi	11
----------------------------------	----

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

tclmpi_comm_t	17
tclmpi_req_t	18

Chapter 5

File Index

5.1 File List

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

_tclmpi.c	21
tclmpi.tcl	50
tests/ harness.tcl	51

Chapter 6

Namespace Documentation

6.1 tclmpi Namespace Reference

Functions

- [init](#)
- [finalize](#)
- [abort](#) comm errorcode
- [comm_size](#) comm
- [comm_rank](#) comm
- [comm_split](#) comm color key
- [barrier](#)
- [bcast](#) data type root comm
- [allreduce](#) data type op comm
- [send](#) data type dest tag comm
- [isend](#) data type dest tag comm
- [recv](#) type source tag comm?status?
- [irecv](#) type source tag comm
- [probe](#) source tag comm?status?
- [iprobe](#) source tag comm?status?
- [wait](#) request?status?
- [X](#)

Variables

- [auto](#)
constant for automatic data type
- [int](#)
constant for integer data type
- [intint](#)
constant for integer pair data type
- [double](#)
constant for double data type
- [dblint](#)
constant for double/int pair data type
- [comm_world](#)
constant for world communicator
- [comm_self](#)

- constant for self communicator*
- [comm_null](#)
constant empty communicator
- [any_source](#)
constant to accept messages from any source rank
- [any_tag](#)
constant to accept messages with any tag
- [version](#)
version number of this package

6.1.1 Detailed Description

TclMPI wrapper

6.1.2 Function Documentation

6.1.2.1 `tclmpi::abort` comm errorcode

Terminates the MPI environment from Tcl

Returns

empty

6.1.2.2 `tclmpi::allreduce` data type op comm

Parameters

<i>comm</i>	Tcl representation of the communicator
-------------	--

Returns

Data for the reduction operation

6.1.2.3 `tclmpi::barrier`

Returns

empty

6.1.2.4 `tclmpi::bcast` data type root comm

Parameters

<i>comm</i>	Tcl representation of the communicator
-------------	--

Returns

Data that was broadcast

6.1.2.5 tclmpi::comm_rank comm

Returns the rank of the current process for the given MPI communicator

Parameters

<i>comm</i>	Tcl representation of the communicator
-------------	--

Returns

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

6.1.2.6 tclmpi::comm_size comm

Returns the number of processes involved in an MPI communicator

Parameters

<i>in</i>	<i>comm</i>	Tcl representation of the communicator
-----------	-------------	--

Returns

number of MPI tasks

6.1.2.7 tclmpi::comm_split comm color key

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

Returns

Tcl representation of the newly created MPI communicator

6.1.2.8 tclmpi::finalize

Shut down the MPI environment from Tcl

Returns

empty

6.1.2.9 tclmpi::init

Initialize the MPI environment from Tcl

Returns

empty

6.1.2.10 tclmpi::iprobe source tag comm ?status?

Parameters

<i>comm</i>	Tcl representation of the communicator
-------------	--

Returns

1 or 0 depending on whether a pending request was detected

6.1.2.11 tclmpi::irecv type source tag comm

Non-blocking receive

Parameters

<i>comm</i>	Tcl representation of the communicator
-------------	--

Returns

Tcl representation of generated MPI request

6.1.2.12 tclmpi::isend data type dest tag comm

Non-blocking send

Parameters

<i>comm</i>	Tcl representation of the communicator
-------------	--

Returns

Tcl representation of generated MPI request

6.1.2.13 tclmpi::probe source tag comm ?status?**Parameters**

<i>comm</i>	Tcl representation of the communicator
-------------	--

Returns

empty

6.1.2.14 tclmpi::recv type source tag comm ?status?

Blocking receive

Parameters

<i>comm</i>	Tcl representation of the communicator
-------------	--

Returns

received data

6.1.2.15 tclmpi::send data type dest tag comm

Blocking send

Parameters

<i>comm</i>	Tcl representation of the communicator
-------------	--

Returns

empty

6.1.2.16 tclmpi::wait request ?status?

Wait for MPI request completion

Parameters

<i>request</i>	Tcl representation of an MPI request
----------------	--------------------------------------

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 released. 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\(\)](#).

6.1.2.17 tclmpi::X

Returns

empty

Chapter 7

Data Structure Documentation

7.1 tclmpi_comm_t Struct Reference

Data Fields

- const char * [label](#)
- MPI_Comm [comm](#)
- int [valid](#)
- tclmpi_comm_t * [next](#)

7.1.1 Detailed Description

Linked list entry to map MPI communicators to strings.

Linked list entry type for managing MPI communicators

7.1.2 Field Documentation

7.1.2.1 MPI_Comm tclmpi_comm_t::comm

MPI communicator corresponding of this entry

7.1.2.2 const char* tclmpi_comm_t::label

String representing the communicator in Tcl

7.1.2.3 tclmpi_comm_t* tclmpi_comm_t::next

Pointer to next element in linked list

7.1.2.4 int tclmpi_comm_t::valid

Non-zero if communicator is valid

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

- [_tclmpi.c](#)

7.2 tclmpi_req_t Struct Reference

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](#)

7.2.1 Detailed Description

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

Linked list entry type for managing MPI requests

7.2.2 Field Documentation

7.2.2.1 MPI_Comm tclmpi_req_t::comm

communicator for non-blocking receive

7.2.2.2 void* tclmpi_req_t::data

pointer to send or receive data buffer

7.2.2.3 const char* tclmpi_req_t::label

identifier of this request

7.2.2.4 int tclmpi_req_t::len

size of data block

7.2.2.5 tclmpi_req_t* tclmpi_req_t::next

pointer to next struct

7.2.2.6 MPI_Request* tclmpi_req_t::req

pointer MPI request handle generated by MPI

7.2.2.7 int tclmpi_req_t::source

source rank of non-blocking receive

7.2.2.8 int tclmpi_req_t::tag

tag selector of non-blocking receive

7.2.2.9 int tclmpi_req_t::type

data type of send data

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

- [_tclmpi.c](#)

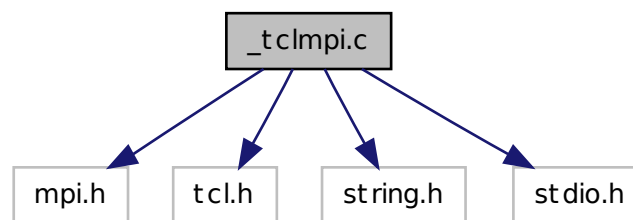
Chapter 8

File Documentation

8.1 _tclmpi.c File Reference

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

Include dependency graph for _tclmpi.c:



Data Structures

- struct `tclmpi_comm_t`
- struct `tclmpi_req_t`

Macros

- `#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`

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 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_comcheck](#) (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_Init](#) (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_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_Allreduce](#) (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int [TclMPI_Send](#) (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int [TclMPI_Isend](#) (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int [TclMPI_Recv](#) (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int [TclMPI_Irecv](#) (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int [TclMPI_Probe](#) (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int [TclMPI_Iprobe](#) (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int [TclMPI_Wait](#) (ClientData nodata, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
- int [_tclmpi_init](#) (Tcl_Interp *interp)

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 char [tclmpi_errmsg](#) [MPI_MAX_ERROR_STRING]
- static int [tclmpi_init_done](#) = 0

8.1.1 Detailed Description

8.1.2 Macro Definition Documentation

8.1.2.1 #define TCLMPI_AUTO 1

the tcl native data type (string)

8.1.2.2 #define TCLMPI_DOUBLE 4

floating point data type

8.1.2.3 #define TCLMPI_DOUBLE_INT 5

data type for double/integer pair

8.1.2.4 #define TCLMPI_INT 2

data type for integers

8.1.2.5 #define TCLMPI_INT_INT 3

data type for pairs of integers

8.1.2.6 #define TCLMPI_INVALID -1

not ready to handle data

8.1.2.7 #define TCLMPI_NONE 0

no data type assigned

8.1.3 Function Documentation

8.1.3.1 int _tclmpi_Init (Tcl_Interp * *interp*)

register this plugin with the Tcl interpreter

Parameters

<i>interp</i>	current Tcl interpreter
---------------	-------------------------

Returns

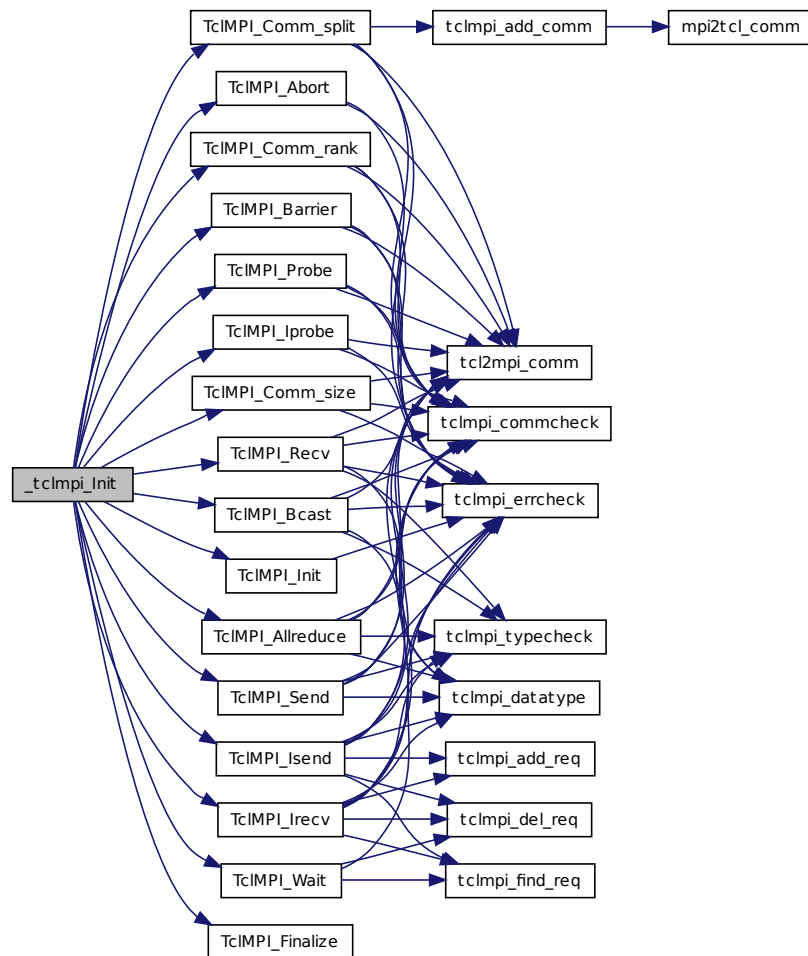
TCL_OK or TCL_ERROR

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.

Here is the call graph for this function:



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

Translate an MPI communicator to its Tcl label.

Parameters

<i>comm</i>	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.

Here is the caller graph for this function:



8.1.3.3 static MPI.Comm tcl2mpi_comm (const char * *label*) [static]

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

Parameters

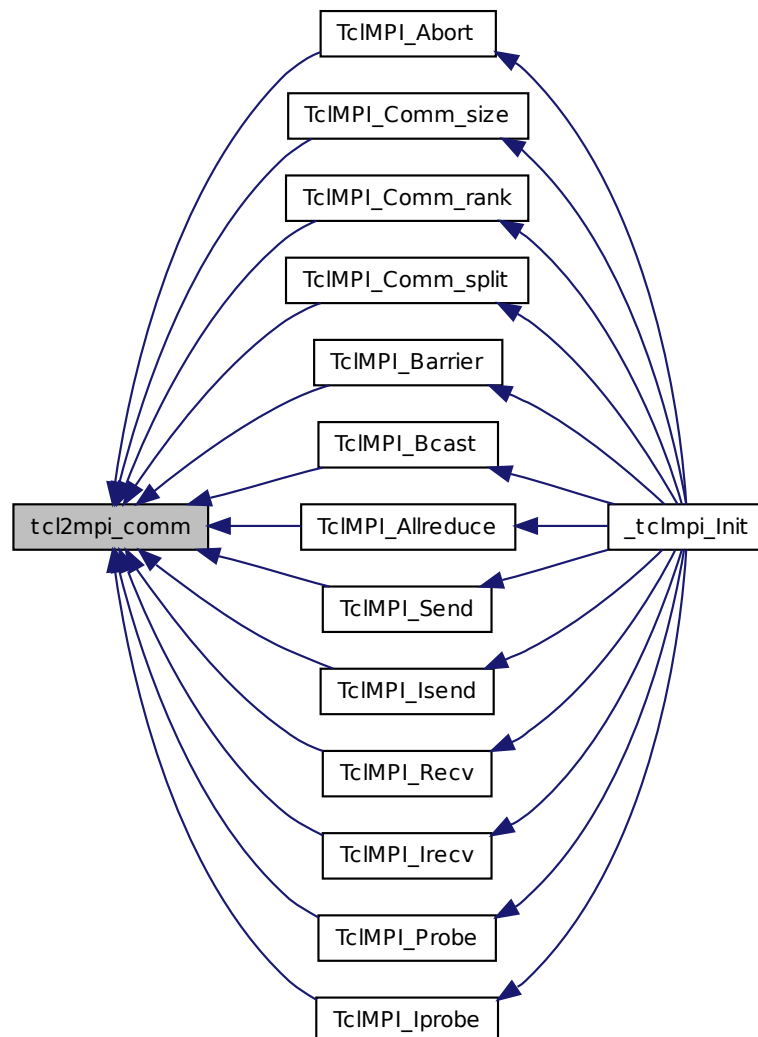
<i>label</i>	the Tcl name for the communicator
--------------	-----------------------------------

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.

Here is the caller graph for this function:



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

wrapper for `MPI_Abort()`

Parameters

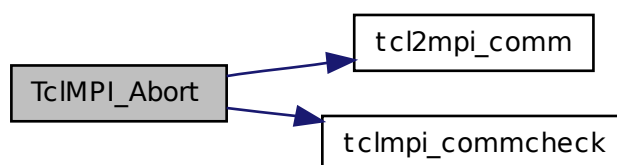
<i>nodata</i>	ignored
<i>interp</i>	current Tcl interpreter
<i>objc</i>	number of argument objects
<i>objv</i>	list of argument object

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().

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

Parameters

<i>comm</i>	an MPI communicator
-------------	---------------------

Returns

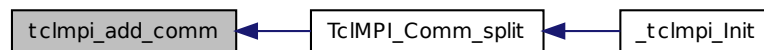
the corresponding string label or NULL.

This function will first call `mpi2tcl_comm` in order to see, if the communicator handed it, 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.

Here is the call graph for this function:



Here is the caller graph for this function:



8.1.3.6 static const char* tclmpi_add_req () [static]

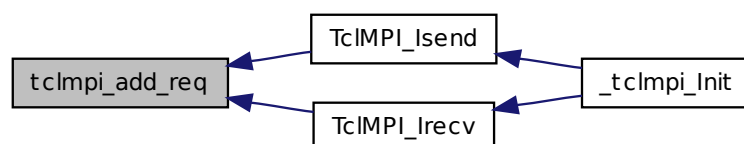
Allocate and add an entry to the request map linked list

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.

Here is the caller graph for this function:



8.1.3.7 int TcIMPI_Allreduce (ClientData *nodata*, Tcl_Interp * *interp*, int *objc*, Tcl_Obj *const *objv*[])

wrapper for MPI_Allreduce()

Parameters

<i>nodata</i>	ignored
<i>interp</i>	current Tcl interpreter
<i>objc</i>	number of argument objects
<i>objv</i>	list of argument object

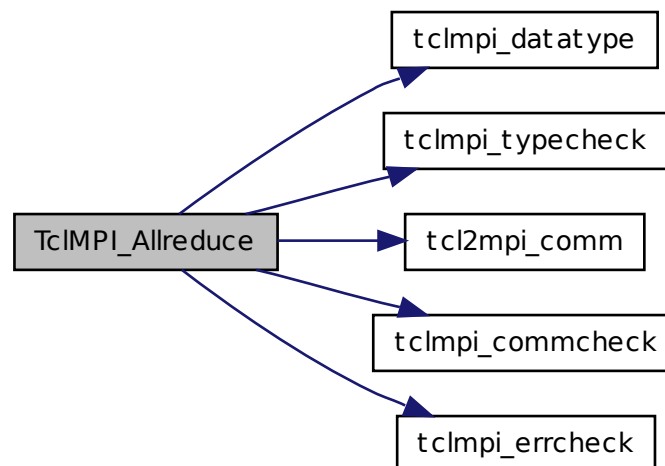
Returns

TCL_OK or TCL_ERROR

This function implements a reduction plus broadcast function for TcIMPI. 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.

Here is the call graph for this function:



Here is the caller graph for this function:



8.1.3.8 int TcIMPI_Barrier (ClientData *nodata*, Tcl_Interp * *interp*, int *objc*, Tcl_Obj *const *objv*[])

wrapper for MPI_Barrier()

Parameters

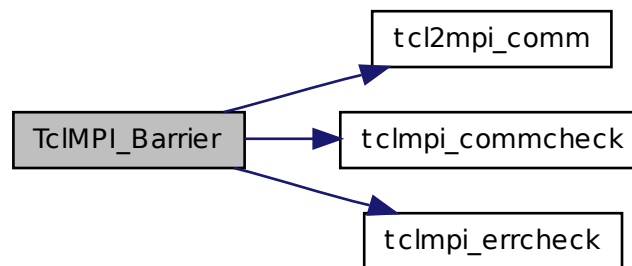
<i>nodata</i>	ignored
<i>interp</i>	current Tcl interpreter
<i>objc</i>	number of argument objects
<i>objv</i>	list of argument object

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.

Here is the call graph for this function:



Here is the caller graph for this function:



8.1.3.9 int TcIMPI_Bcast (ClientData *nodata*, Tcl_Interp * *interp*, int *objc*, Tcl_Obj *const *objv*[])

wrapper for MPI_Bcast()

Parameters

<i>nodata</i>	ignored
<i>interp</i>	current Tcl interpreter
<i>objc</i>	number of argument objects
<i>objv</i>	list of argument object

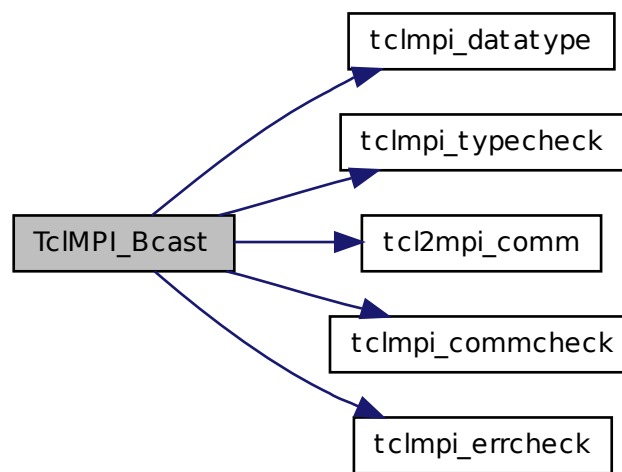
Returns

TCL_OK or TCL_ERROR

This function implements a broadcast function for TcIMPI. 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.

Here is the call graph for this function:



Here is the caller graph for this function:



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

wrapper for `MPI_Comm_rank()`

Parameters

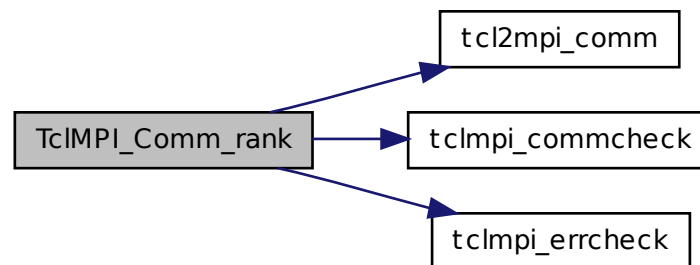
<i>nodata</i>	ignored
<i>interp</i>	current Tcl interpreter
<i>objc</i>	number of argument objects
<i>objv</i>	list of argument object

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.

Here is the call graph for this function:



Here is the caller graph for this function:



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

wrapper for MPI_Comm_size()

Parameters

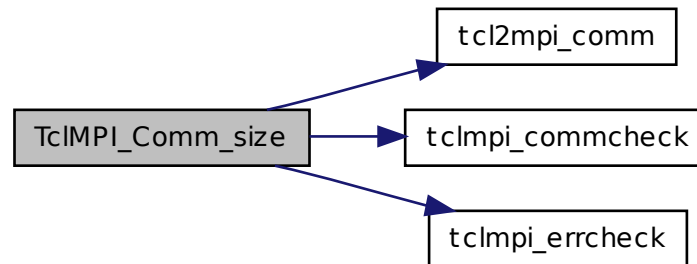
<i>nodata</i>	ignored
<i>interp</i>	current Tcl interpreter
<i>objc</i>	number of argument objects
<i>objv</i>	list of argument object

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.

Here is the call graph for this function:



Here is the caller graph for this function:



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

wrapper for MPI_Comm_split()

Parameters

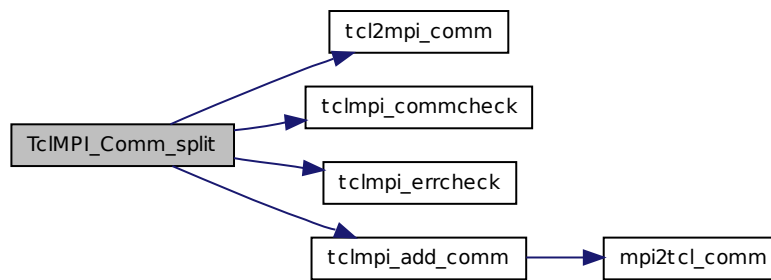
<i>nodata</i>	ignored
<i>interp</i>	current Tcl interpreter
<i>objc</i>	number of argument objects
<i>objv</i>	list of argument object

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.

Here is the call graph for this function:



Here is the caller graph for this function:



8.1.3.13 `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

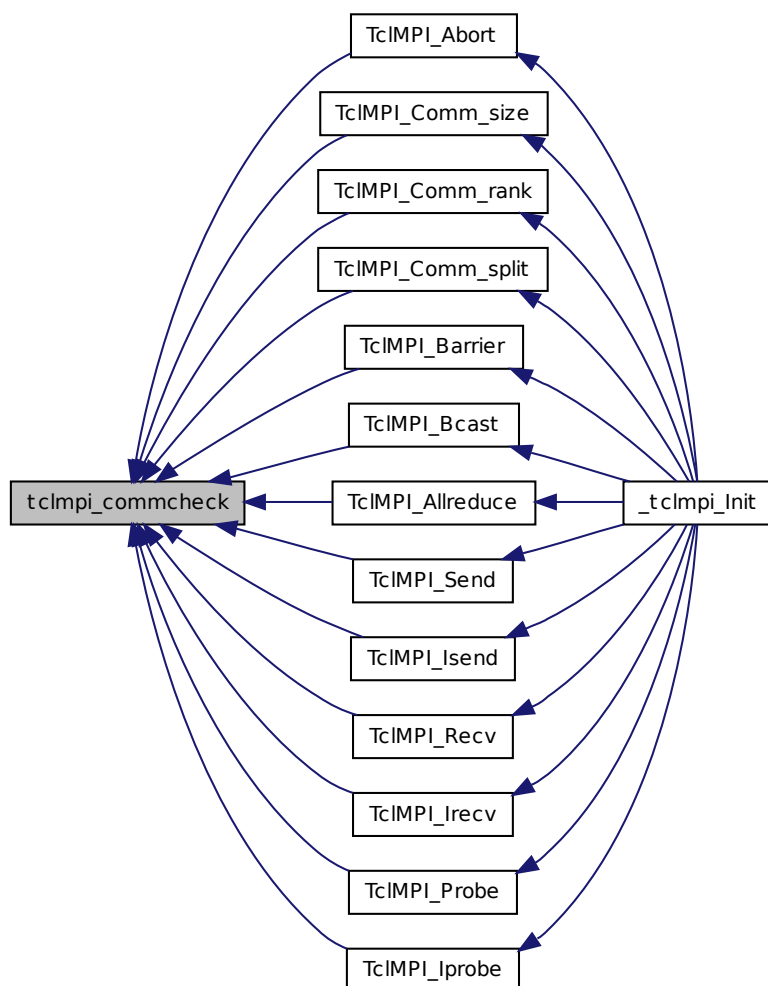
Parameters

<i>interp</i>	current Tcl interpreter
<i>comm</i>	MPI communicator
<i>obj0</i>	Tcl object representing the current command name
<i>obj1</i>	Tcl object representing the communicator as Tcl name

Returns

TCL_ERROR if the communicator is MPI_COMM_INVALID or TCL_OK

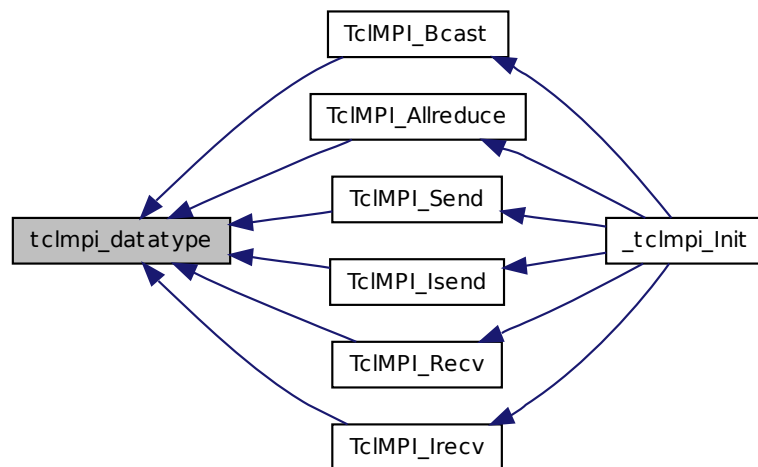
Here is the caller graph for this function:



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

convert a string describing a data type to a numeric representation

Here is the caller graph for this function:



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

remove tclmpi_req_t entry from the request linked list

Parameters

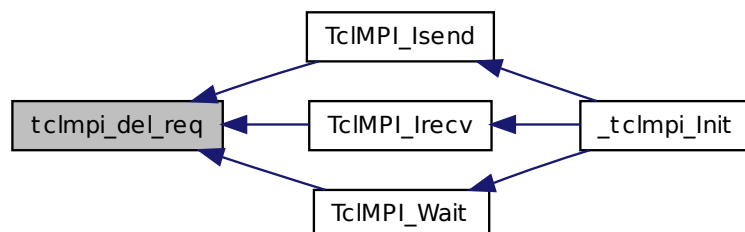
<i>req</i>	a pointer to the request in question
------------	--------------------------------------

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.

Here is the caller graph for this function:



8.1.3.16 `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

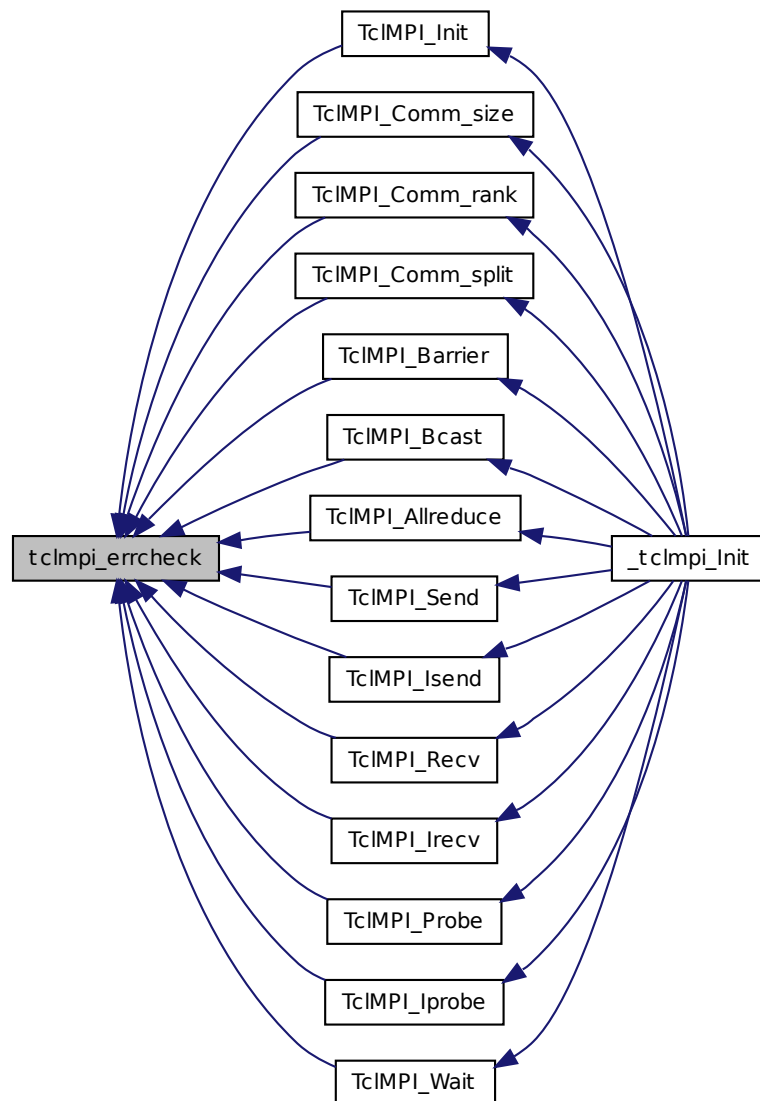
<i>interp</i>	current Tcl interpreter
<i>ierr</i>	MPI error number. return value of an MPI call.
<i>obj</i>	Tcl object representing the current command name

Returns

TCL_OK if the "error" is MPI_SUCCESS or TCL_ERROR

This is a simple convenience wrapper that will use `MPI_Error_strerror()` to convert any error returned from MPI function calls to a Tcl error message appended to the result vector of the current command. Should be called after each MPI call, since we change communicators to not result in fatal errors, so we have to generate Tcl errors instead (which can be caught).

Here is the caller graph for this function:



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

wrapper for `MPI_Finalize()`

Parameters

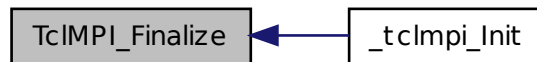
<i>nodata</i>	ignored
<i>interp</i>	current Tcl interpreter
<i>objc</i>	number of argument objects
<i>objv</i>	list of argument object

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_Finalize() have been called before (from Tcl) and then creates a (catchable) Tcl error instead of an (uncatchable) MPI error.

Here is the caller graph for this function:



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

translate Tcl representation of an MPI request to request itself.

Parameters

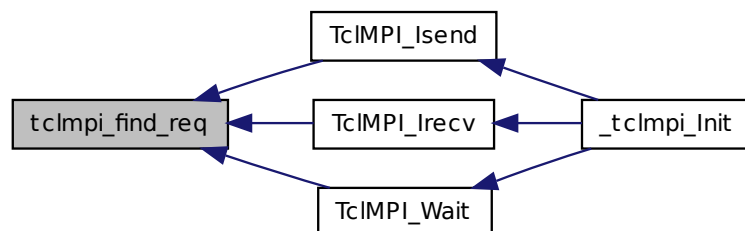
<i>label</i>	the Tcl name for the communicator
--------------	-----------------------------------

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.

Here is the caller graph for this function:



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

wrapper for MPI_Init()

Parameters

<i>nodata</i>	ignored
<i>interp</i>	current Tcl interpreter
<i>objc</i>	number of argument objects
<i>objv</i>	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.

Here is the call graph for this function:



Here is the caller graph for this function:



8.1.3.20 int TcIMPI_lprobe (ClientData *nodata*, Tcl_Interp * *interp*, int *objc*, Tcl_Obj *const *objv*[])

wrapper for MPI_lprobe()

Parameters

<i>nodata</i>	ignored
<i>interp</i>	current Tcl interpreter
<i>objc</i>	number of argument objects
<i>objv</i>	list of argument object

Returns

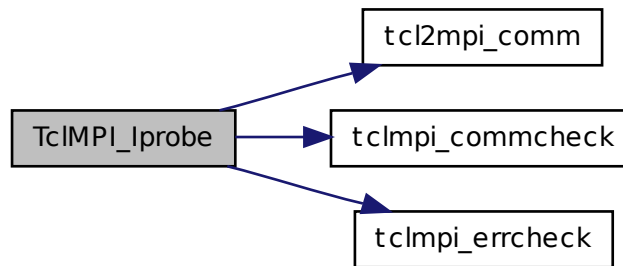
TCL_OK or TCL_ERROR

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.1.3.21 int TcIMPI_Irecv (ClientData *nodata*, Tcl_Interp * *interp*, int *objc*, Tcl_Obj *const *objv*[])

wrapper for MPI_Irecv()

Parameters

<i>nodata</i>	ignored
<i>interp</i>	current Tcl interpreter
<i>objc</i>	number of argument objects
<i>objv</i>	list of argument object

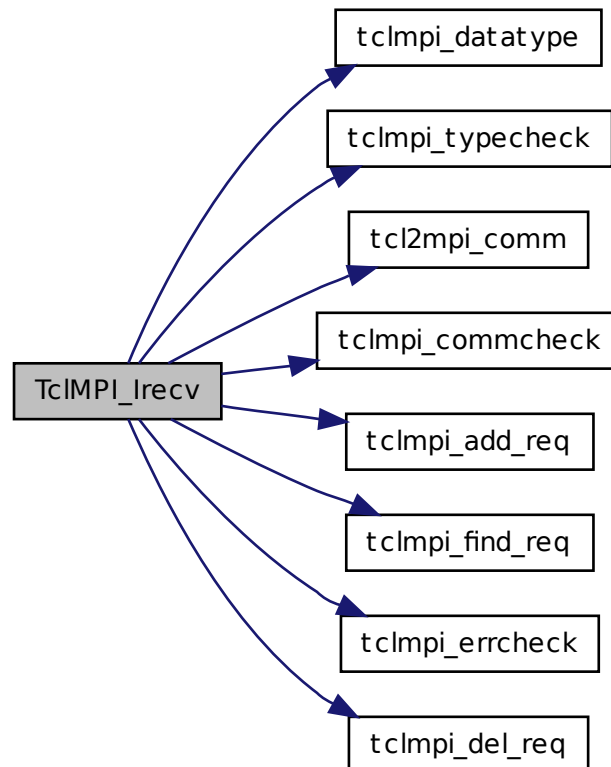
Returns

TCL_OK or TCL_ERROR

This function implements a non-blocking receive operation for TcIMPI. 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.

Here is the call graph for this function:



Here is the caller graph for this function:



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

wrapper for MPI_Isend()

Parameters

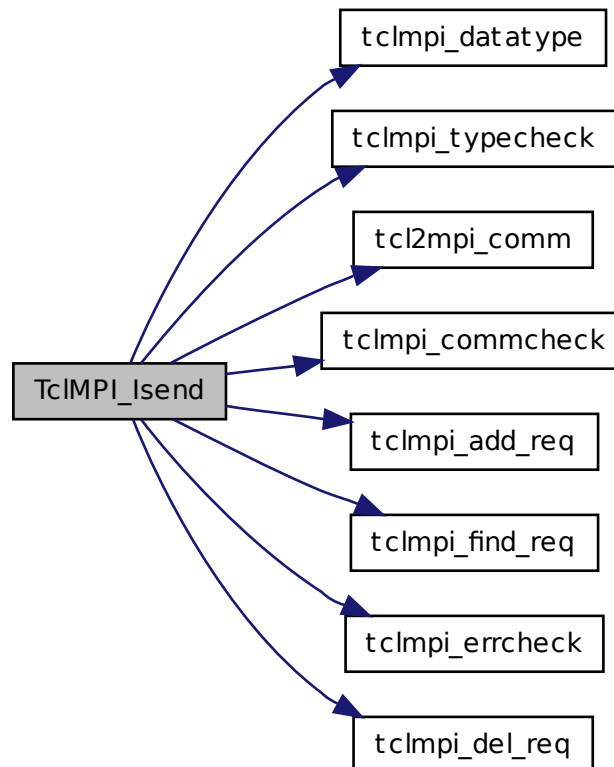
<i>nodata</i>	ignored
<i>interp</i>	current Tcl interpreter
<i>objc</i>	number of argument objects
<i>objv</i>	list of argument object

Returns

TCL_OK or TCL_ERROR

This function implements a non-blocking send operation for TcIMPI. 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 TcIMPI_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 TcIMPI_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.

Here is the call graph for this function:



Here is the caller graph for this function:



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

wrapper for `MPI_Probe()`

Parameters

<i>nodata</i>	ignored
<i>interp</i>	current Tcl interpreter
<i>objc</i>	number of argument objects
<i>objv</i>	list of argument object

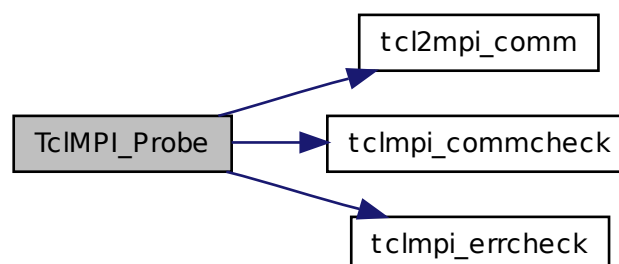
Returns

`TCL_OK` or `TCL_ERROR`

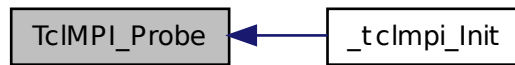
This function implements a blocking probe operation for TcIMPI. Argument flags for source, tag, and communicator are translated into their native MPI equivalents and then `MPI_Probe` called.

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`).

Here is the call graph for this function:



Here is the caller graph for this function:



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

wrapper for `MPI_Recv()`

Parameters

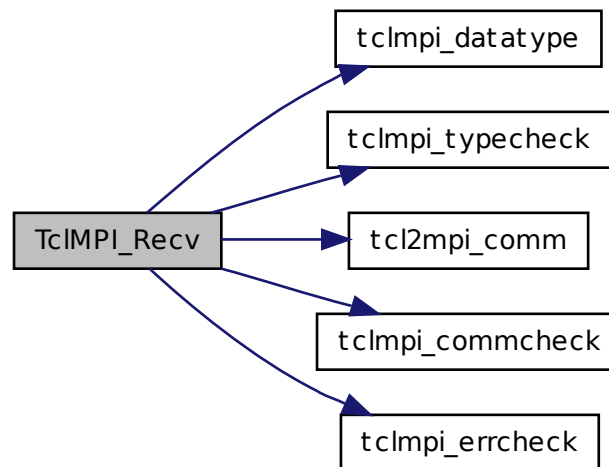
<i>nodata</i>	ignored
<i>interp</i>	current Tcl interpreter
<i>objc</i>	number of argument objects
<i>objv</i>	list of argument object

Returns

`TCL_OK` or `TCL_ERROR`

This function implements a blocking receive operation for TcIMPI. 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.

Here is the call graph for this function:



Here is the caller graph for this function:



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

wrapper for `MPI_Send()`

Parameters

<i>nodata</i>	ignored
<i>interp</i>	current Tcl interpreter
<i>objc</i>	number of argument objects
<i>objv</i>	list of argument object

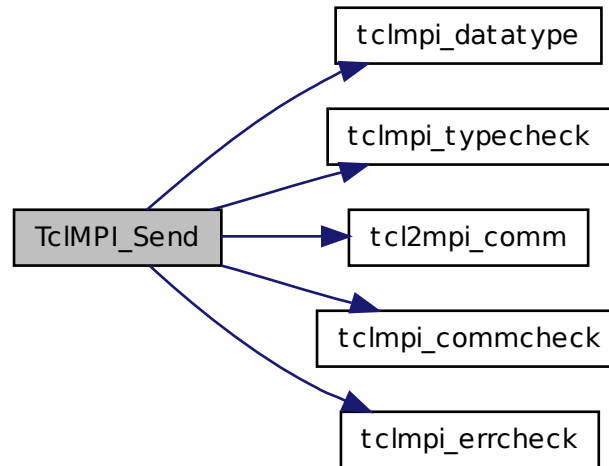
Returns

TCL_OK or TCL_ERROR

This function implements a blocking send operation for TcIMPI. 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.

Here is the call graph for this function:



Here is the caller graph for this function:



8.1.3.26 `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

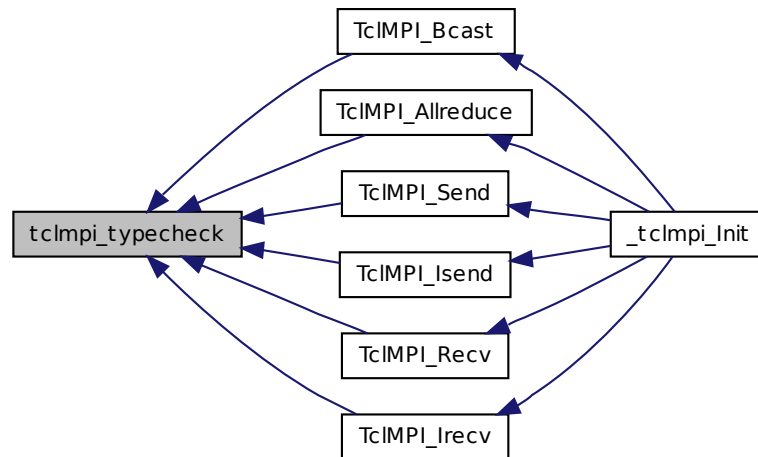
Parameters

<i>interp</i>	current Tcl interpreter
<i>type</i>	TcIMPI data type
<i>obj0</i>	Tcl object representing the current command name
<i>obj1</i>	Tcl object representing the data type as Tcl name

Returns

TCL_ERROR if the communicator is TCLMPI_NONE or TCL_OK

Here is the caller graph for this function:



8.1.3.27 int TcIMPI.Wait (ClientData *nodata*, Tcl_Interp * *interp*, int *objc*, Tcl_Obj *const *objv*[])

wrapper for MPI_Wait()

Parameters

<i>nodata</i>	ignored
<i>interp</i>	current Tcl interpreter
<i>objc</i>	number of argument objects
<i>objv</i>	list of argument object

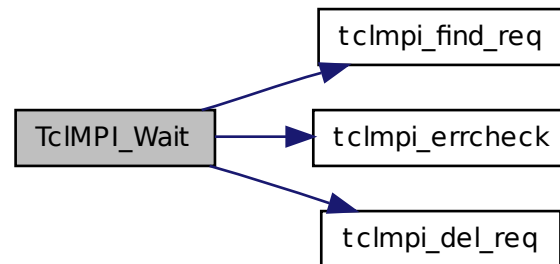
Returns

TCL_OK or TCL_ERROR

This function implements a wrapper around MPI_Wait for TcIMPI. Due to the design decisions in TcIMPI, it works a bit different than MPI_Write, particularly for non-blocking receive requests. As explained in the TcIMPI_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, allocated that buffer and the 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.

Here is the call graph for this function:



Here is the caller graph for this function:



8.1.4 Variable Documentation

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

First element of the communicator map list

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

First element of the list of generated requests

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

Last element of the communicator map list

8.1.4.4 `MPI_Comm MPI_COMM_INVALID` `[static]`

Additional global communicator to detect unlisted communicators

8.1.4.5 `int tclmpi_comm.cnt = 0` `[static]`

Communicator counter. Incremented to get unique strings

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

buffer for error messages.

8.1.4.7 `int tclmpi_init_done = 0` `[static]`

is 1 after `MPI_Init()` and -1 after `MPI_Finalize()`

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

Request counter. Incremented to get unique strings

8.2 tclmpi.tcl File Reference

Namespaces

- namespace `tclmpi`

Functions

- `tclmpi::init`
- `tclmpi::finalize`
- `tclmpi::abort` comm errorcode
- `tclmpi::comm_size` comm
- `tclmpi::comm_rank` comm
- `tclmpi::comm_split` comm color key
- `tclmpi::barrier`
- `tclmpi::bcast` data type root comm
- `tclmpi::allreduce` data type op comm
- `tclmpi::send` data type dest tag comm
- `tclmpi::isend` data type dest tag comm
- `tclmpi::recv` type source tag comm?status?
- `tclmpi::irecv` type source tag comm
- `tclmpi::probe` source tag comm?status?
- `tclmpi::iprobe` source tag comm?status?
- `tclmpi::wait` request?status?
- `tclmpi::X`

Variables

- `tclmpi::auto`
constant for automatic data type
- `tclmpi::int`
constant for integer data type
- `tclmpi::intint`
constant for integer pair data type
- `tclmpi::double`
constant for double data type
- `tclmpi::dblnt`
constant for double/int pair data type
- `tclmpi::comm_world`

- constant for world communicator*
- [tclmpi::comm_self](#)
constant for self communicator
- [tclmpi::comm_null](#)
constant empty communicator
- [tclmpi::any_source](#)
constant to accept messages from any source rank
- [tclmpi::any_tag](#)
constant to accept messages with any tag
- [tclmpi::version](#)
version number of this package

8.2.1 Detailed Description

This is the [tclmpi::](#) namespace and the documentation of the Tcl API of TcIMPI.

8.3 tests/harness.tcl File Reference

Functions

- [test_format](#) kind cmd result
- [ser_init](#) args
- [par_init](#) args
- [run_return](#) cmd retval
- [run_error](#) cmd errmsg
- [par_return](#) cmd retval?comm?
- [par_error](#) cmd retval?comm?
- [par_set](#) name value?comm?
- [test_summary](#) section

8.3.1 Detailed Description

Test harness for TcIMPI

Copyright (c) 2012 Axel Kohlmeyer akohlmey@gmail.com All Rights Reserved.

See the file LICENSE in the top level directory for licensing conditions.

8.3.2 Function Documentation

8.3.2.1 [par_error](#) cmd retval ?comm?

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

Parameters

<i>cmd</i>	list of strings or lists with the commands to execute
<i>retval</i>	list of the expected error message(s) or return values
<i>comm</i>	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.

8.3.2.2 par_init args

init for parallel tests

Parameters

<i>args</i>	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.

8.3.2.3 par_return cmd retval ?comm?

run a parallel test that is expected to succeed

Parameters

<i>cmd</i>	list of strings or lists with the commands to execute
<i>retval</i>	list of the expected return values
<i>comm</i>	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 actual results are printed on one of the failing ranks. The error reporting expects that the MPI communicator remains usable after failure.

8.3.2.4 par_set name value ?comm?

set variable to different values on different ranks

Parameters

<i>name</i>	name of variable
<i>value</i>	list of values, one per rank
<i>comm</i>	communicator

Returns

empty

8.3.2.5 run_error cmd errmsg

run a serial test that is expected to fail

Parameters

<i>cmd</i>	string or list with the command to execute
<i>errmsg</i>	expected error message

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 \$errmsg. 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.

8.3.2.6 run_return cmd retval

run a serial test that is expected to succeed

Parameters

<i>cmd</i>	string or list with the command to execute
<i>retval</i>	expected return value

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.

8.3.2.7 ser_init args

init for serial tests

Parameters

<i>args</i>	all parameters are ignored
-------------	----------------------------

Returns

empty

This function will perform a simple init test requesting the tclmpi package and matching it against the current version number. If called from a parallel environment, it will only execute and produce output on the master process

8.3.2.8 test_format kind cmd result

format output

Parameters

<i>kind</i>	string representing the kind of test (max 11 chars).
<i>cmd</i>	string representing the command. will be truncated as needed.
<i>result</i>	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.

8.3.2.9 test_summary section

print result summary

Parameters

<i>section</i>	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.

Index

- `_tclmpi.c`, [21](#)
 - `_tclmpi_Init`, [23](#)
 - `first_comm`, [49](#)
 - `first_req`, [49](#)
 - `last_comm`, [49](#)
 - `MPI_COMM_INVALID`, [49](#)
 - `mpi2tcl_comm`, [24](#)
 - `TCLMPI_AUTO`, [22](#)
 - `TCLMPI_DOUBLE`, [22](#)
 - `TCLMPI_DOUBLE_INT`, [22](#)
 - `TCLMPI_INT`, [23](#)
 - `TCLMPI_INT_INT`, [23](#)
 - `TCLMPI_INVALID`, [23](#)
 - `TCLMPI_NONE`, [23](#)
 - `tcl2mpi_comm`, [25](#)
 - `TclMPI_Abort`, [26](#)
 - `TclMPI_Allreduce`, [28](#)
 - `TclMPI_Barrier`, [29](#)
 - `TclMPI_Bcast`, [30](#)
 - `TclMPI_Comm_rank`, [31](#)
 - `TclMPI_Comm_size`, [32](#)
 - `TclMPI_Comm_split`, [33](#)
 - `TclMPI_Finalize`, [38](#)
 - `TclMPI_Init`, [39](#)
 - `TclMPI_Iprobe`, [40](#)
 - `TclMPI_Irecv`, [41](#)
 - `TclMPI_Isend`, [42](#)
 - `TclMPI_Probe`, [44](#)
 - `TclMPI_Recv`, [45](#)
 - `TclMPI_Send`, [46](#)
 - `TclMPI_Wait`, [48](#)
 - `tclmpi_add_comm`, [27](#)
 - `tclmpi_add_req`, [28](#)
 - `tclmpi_comm_cntr`, [49](#)
 - `tclmpi_commcheck`, [34](#)
 - `tclmpi_datatype`, [35](#)
 - `tclmpi_del_req`, [36](#)
 - `tclmpi_errcheck`, [36](#)
 - `tclmpi_errmsg`, [49](#)
 - `tclmpi_find_req`, [39](#)
 - `tclmpi_init_done`, [50](#)
 - `tclmpi_req_cntr`, [50](#)
 - `tclmpi_typecheck`, [47](#)
- `_tclmpi_Init`
 - `_tclmpi.c`, [23](#)
- `abort`
 - `tclmpi`, [12](#)
- `allreduce`
 - `tclmpi`, [12](#)
- `barrier`
 - `tclmpi`, [12](#)
- `bcast`
 - `tclmpi`, [12](#)
- `comm`
 - `tclmpi_comm`, [17](#)
 - `tclmpi_req`, [18](#)
- `comm_rank`
 - `tclmpi`, [12](#)
- `comm_size`
 - `tclmpi`, [13](#)
- `comm_split`
 - `tclmpi`, [13](#)
- `data`
 - `tclmpi_req`, [18](#)
- `finalize`
 - `tclmpi`, [13](#)
- `first_comm`
 - `_tclmpi.c`, [49](#)
- `first_req`
 - `_tclmpi.c`, [49](#)
- `harness.tcl`
 - `par_error`, [51](#)
 - `par_init`, [52](#)
 - `par_return`, [52](#)
 - `par_set`, [52](#)
 - `run_error`, [53](#)
 - `run_return`, [53](#)
 - `ser_init`, [53](#)
 - `test_format`, [53](#)
 - `test_summary`, [54](#)
- `init`
 - `tclmpi`, [13](#)
- `iprobe`
 - `tclmpi`, [13](#)
- `irecv`
 - `tclmpi`, [14](#)
- `isend`
 - `tclmpi`, [14](#)
- `label`
 - `tclmpi_comm`, [17](#)
 - `tclmpi_req`, [18](#)
- `last_comm`
 - `_tclmpi.c`, [49](#)
- `len`

- tclmpi_req, 18
- MPI_COMM_INVALID
 - _tclmpi.c, 49
- mpi2tcl_comm
 - _tclmpi.c, 24
- next
 - tclmpi_comm, 17
 - tclmpi_req, 18
- par_error
 - harness.tcl, 51
- par_init
 - harness.tcl, 52
- par_return
 - harness.tcl, 52
- par_set
 - harness.tcl, 52
- probe
 - tclmpi, 14
- recv
 - tclmpi, 14
- req
 - tclmpi_req, 18
- run_error
 - harness.tcl, 53
- run_return
 - harness.tcl, 53
- send
 - tclmpi, 14
- ser_init
 - harness.tcl, 53
- source
 - tclmpi_req, 18
- TCLMPI_AUTO
 - _tclmpi.c, 22
- TCLMPI_DOUBLE
 - _tclmpi.c, 22
- TCLMPI_DOUBLE_INT
 - _tclmpi.c, 22
- TCLMPI_INT
 - _tclmpi.c, 23
- TCLMPI_INT_INT
 - _tclmpi.c, 23
- TCLMPI_INVALID
 - _tclmpi.c, 23
- TCLMPI_NONE
 - _tclmpi.c, 23
- tag
 - tclmpi_req, 18
- tcl2mpi_comm
 - _tclmpi.c, 25
- TclMPI_Abort
 - _tclmpi.c, 26
- TclMPI_Allreduce
 - _tclmpi.c, 28
- TclMPI_Barrier
 - _tclmpi.c, 29
- TclMPI_Bcast
 - _tclmpi.c, 30
- TclMPI_Comm_rank
 - _tclmpi.c, 31
- TclMPI_Comm_size
 - _tclmpi.c, 32
- TclMPI_Comm_split
 - _tclmpi.c, 33
- TclMPI_Finalize
 - _tclmpi.c, 38
- TclMPI_Init
 - _tclmpi.c, 39
- TclMPI_Iprobe
 - _tclmpi.c, 40
- TclMPI_Irecv
 - _tclmpi.c, 41
- TclMPI_Isend
 - _tclmpi.c, 42
- TclMPI_Probe
 - _tclmpi.c, 44
- TclMPI_Recv
 - _tclmpi.c, 45
- TclMPI_Send
 - _tclmpi.c, 46
- TclMPI_Wait
 - _tclmpi.c, 48
- tclmpi, 11
 - abort, 12
 - allreduce, 12
 - barrier, 12
 - bcast, 12
 - comm_rank, 12
 - comm_size, 13
 - comm_split, 13
 - finalize, 13
 - init, 13
 - iprobe, 13
 - irecv, 14
 - isend, 14
 - probe, 14
 - recv, 14
 - send, 14
 - wait, 15
 - X, 15
- tclmpi.tcl, 50
- tclmpi_add_comm
 - _tclmpi.c, 27
- tclmpi_add_req
 - _tclmpi.c, 28
- tclmpi_comm
 - comm, 17
 - label, 17
 - next, 17
 - valid, 17
- tclmpi_comm_cntr
 - _tclmpi.c, 49

- tclmpi_comm_t, [17](#)
- tclmpi_commcheck
 - _tclmpi.c, [34](#)
- tclmpi_datatype
 - _tclmpi.c, [35](#)
- tclmpi_del_req
 - _tclmpi.c, [36](#)
- tclmpi_errcheck
 - _tclmpi.c, [36](#)
- tclmpi_errmsg
 - _tclmpi.c, [49](#)
- tclmpi_find_req
 - _tclmpi.c, [39](#)
- tclmpi_init_done
 - _tclmpi.c, [50](#)
- tclmpi_req
 - comm, [18](#)
 - data, [18](#)
 - label, [18](#)
 - len, [18](#)
 - next, [18](#)
 - req, [18](#)
 - source, [18](#)
 - tag, [18](#)
 - type, [19](#)
- tclmpi_req_cnr
 - _tclmpi.c, [50](#)
- tclmpi_req_t, [18](#)
- tclmpi_typecheck
 - _tclmpi.c, [47](#)
- test_format
 - harness.tcl, [53](#)
- test_summary
 - harness.tcl, [54](#)
- tests/harness.tcl, [51](#)
- type
 - tclmpi_req, [19](#)
- valid
 - tclmpi_comm, [17](#)
- wait
 - tclmpi, [15](#)
- X
 - tclmpi, [15](#)