TclMPI - Tcl Bindings for MPI

0.6

Generated by Doxygen 1.8.1

Sun May 20 2012 22:30:16

Contents

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/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.

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 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

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

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

Chapter 5

Data Structure Documentation

5.1 tclmpi_comm_t Struct Reference

Data Fields

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

5.1.1 Detailed Description

Linked list entry to map MPI communicators to strings.

Linked list entry type for managing MPI communicators

5.1.2 Field Documentation

5.1.2.1 MPI_Comm tclmpi_comm_t::comm

MPI communicator corresponding of this entry

5.1.2.2 const char* tclmpi_comm_t::label

String representing the communicator in Tcl

5.1.2.3 tclmpi_comm_t* tclmpi_comm_t::next

Pointer to next element in linked list

5.1.2.4 int tclmpi_comm_t::valid

Non-zero if communicator is valid

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

- tcl_mpi.c

5.2 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

5.2.1 Detailed Description

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

Linked list entry type for managing MPI requests

5.2.2 Field Documentation

5.2.2.1 MPI_Comm tclmpi_req_t::comm

communicator for non-blocking receive

5.2.2.2 void* tclmpi_req_t::data

pointer to send or receive data buffer

5.2.2.3 const char* tclmpi_req_t::label

identifier of this request

5.2.2.4 int tclmpi_req_t::len

size of data block

5.2.2.5 tclmpi_req_t* tclmpi_req_t::next

pointer to next struct

5.2.2.6 MPI_Request* tclmpi_req_t::req

pointer MPI request handle generated by MPI

5.2.2.7 int tclmpi_req_t::source

source rank of non-blocking receive

5.2.2.8 int tclmpi_req_t::tag

tag selector of non-blocking receive

5.2.2.9 int tclmpi_req_t::type

data type of send data

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

- tcl_mpi.c

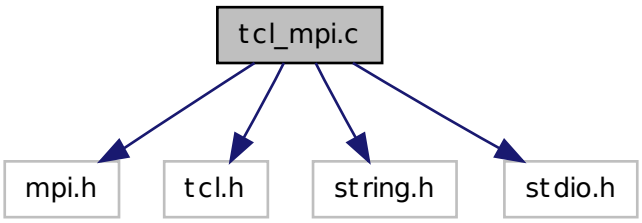
Chapter 6

File Documentation

6.1 tcl_mpi.c File Reference

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

Include dependency graph for tcl_mpi.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_commcheck (Tcl_Interp *interp, MPI_Comm comm, Tcl_Obj *obj0, Tcl_Obj *obj1)
- static int tclmpi_typecheck (Tcl_Interp *interp, int type, Tcl_Obj *obj0, Tcl_Obj *obj1)
- int TclMPI_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

6.1.1 Detailed Description

6.1.2 Macro Definition Documentation

6.1.2.1 #define TCLMPI_AUTO 1

the tcl native data type (string)

6.1.2.2 #define TCLMPI_DOUBLE 4

floating point data type

6.1.2.3 #define TCLMPI_DOUBLE_INT 5

data type for double/integer pair

6.1.2.4 #define TCLMPI_INT 2

data type for integers

6.1.2.5 #define TCLMPI_INT_INT 3

data type for pairs of integers

6.1.2.6 #define TCLMPI_INVALID -1

not ready to handle data

6.1.2.7 #define TCLMPI_NONE 0

no data type assigned

6.1.3 Function Documentation

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

Translate an MPI communicator to its Tcl label.

Parameters

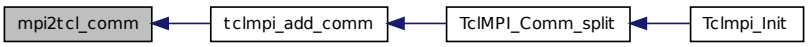
comm	an MPI communicator

Returns

the corresponding string label or NULL.

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

Here is the caller graph for this function:



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

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

Parameters

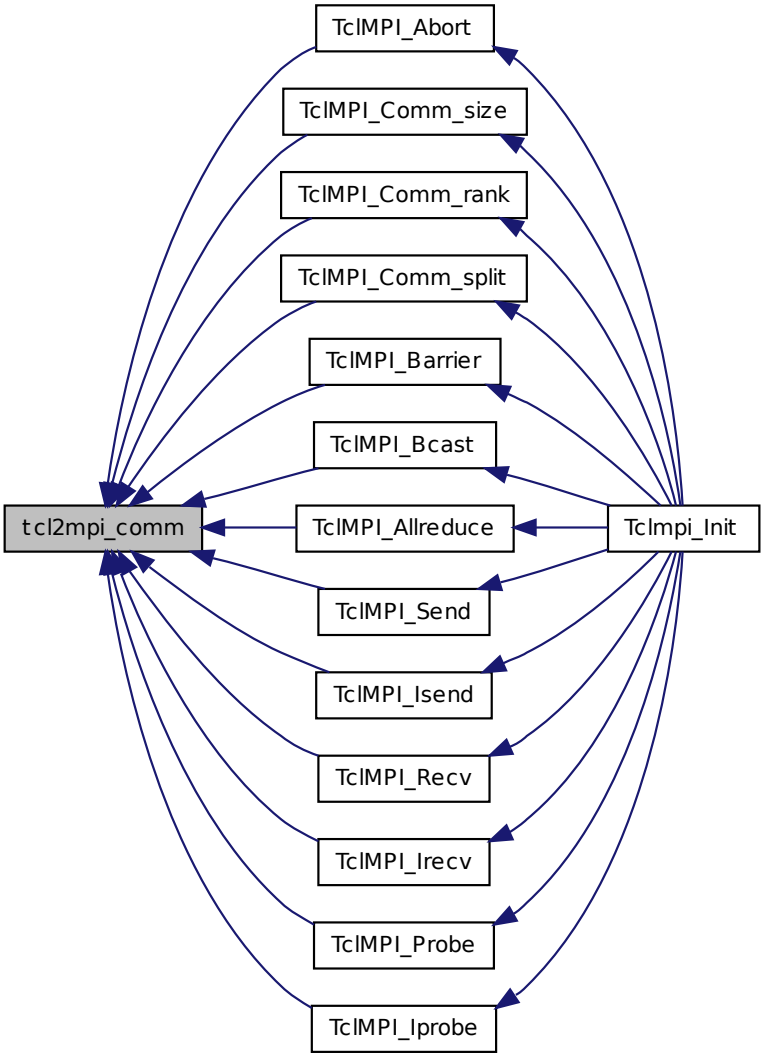
label	the Tcl name for the communicator

Returns

the matching MPI communicator or MPI_COMM_INVALID

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

Here is the caller graph for this function:



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

wrapper for MPI_Abort()

Parameters

nodata	ignored
interp	current Tcl interpreter

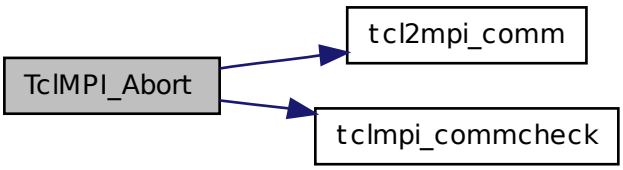
objc	number of argument objects
objv	list of argument object

Returns

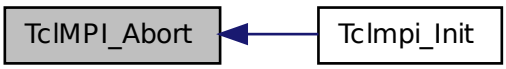
TCL_OK or TCL_ERROR

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

Parameters

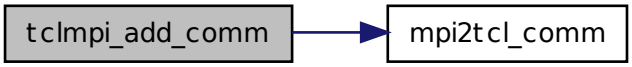
comm	an MPI communicator

Returns

the corresponding string label or NULL.

This function will first call mpi2tcl_comm in order to see, if the communicator handed 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:

6.1.3.5 static const char* tclmpi_add_req ( ) [static]

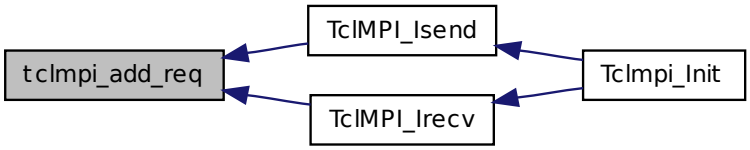
Allocate and add an entry to the request map linked list

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:



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

wrapper for MPI_Allreduce()

Parameters

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

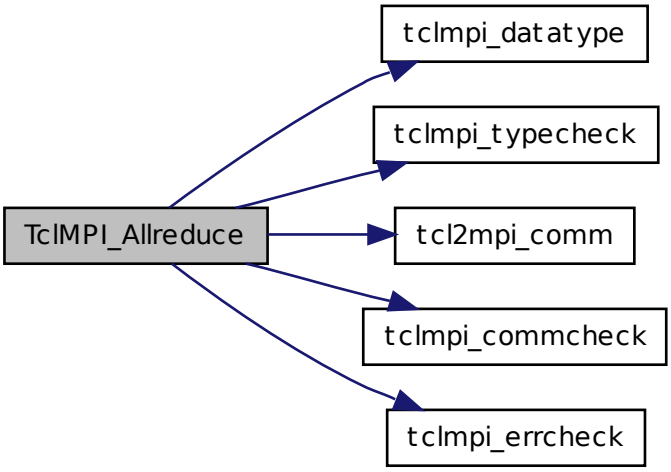
Returns

TCL_OK or TCL_ERROR

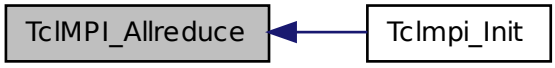
This function implements a reduction plus broadcast function for TclMPI. This operation does not accept the ::tclmpi-::auto data type, also support for types outside of ::tclmpi::int and ::tclmpi::double is incomplete. The length of the data is inferred from the data object passed to this function and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

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.3.7 int TclMPI_Barrier (ClientData *nodata*, Tcl_Interp * *interp*, int *objc*, Tcl_Obj *const *objv[]*)

wrapper for MPI_Barrier()

Parameters

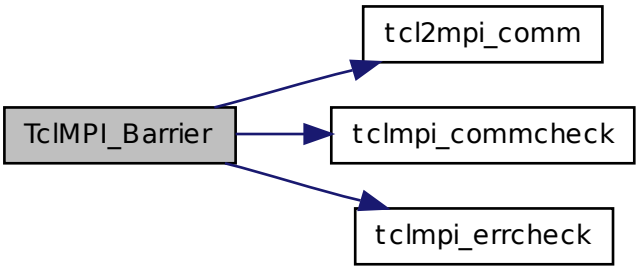
nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

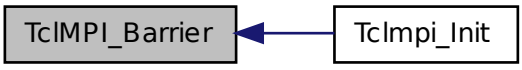
TCL_OK or TCL_ERROR

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

wrapper for MPI_Bcast()

Parameters

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

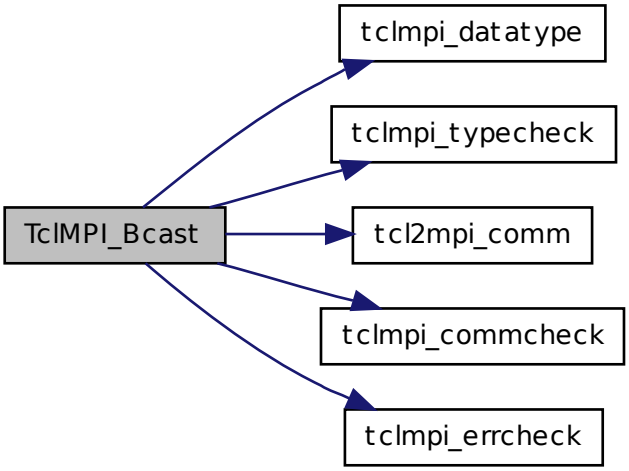
Returns

TCL_OK or TCL_ERROR

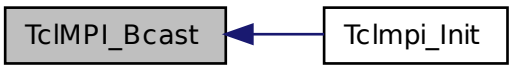
This function implements a broadcast function for TclMPI. Unlike in the C bindings, the length of the data is inferred from the data object passed to this function and thus a 'count' argument is not needed. Only a limited number of data types are currently supported, since Tcl has a limited number of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

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

wrapper for MPI_Comm_rank()

Parameters

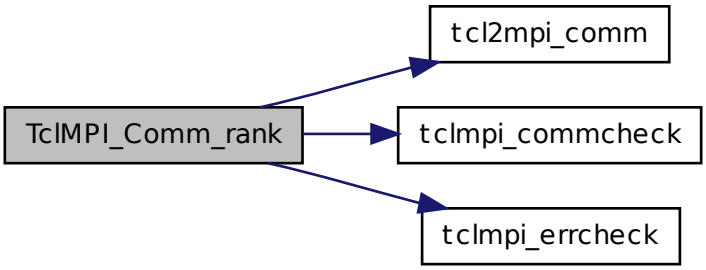
nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

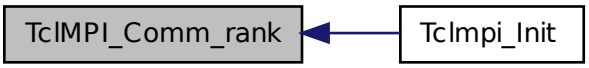
TCL_OK or TCL_ERROR

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

wrapper for MPI_Comm_size()

Parameters

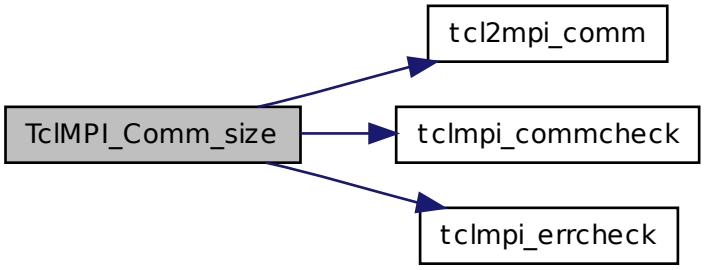
nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

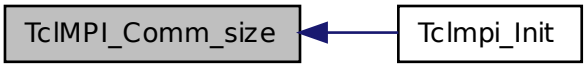
TCL_OK or TCL_ERROR

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

wrapper for MPI_Comm_split()

Parameters

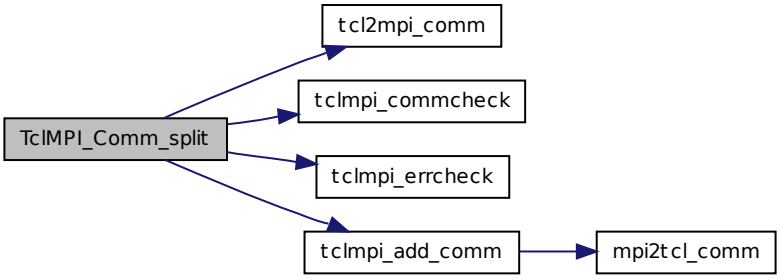
nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

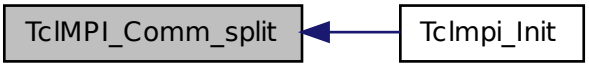
TCL_OK or TCL_ERROR

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

convenience function to report an unknown communicator as Tcl error

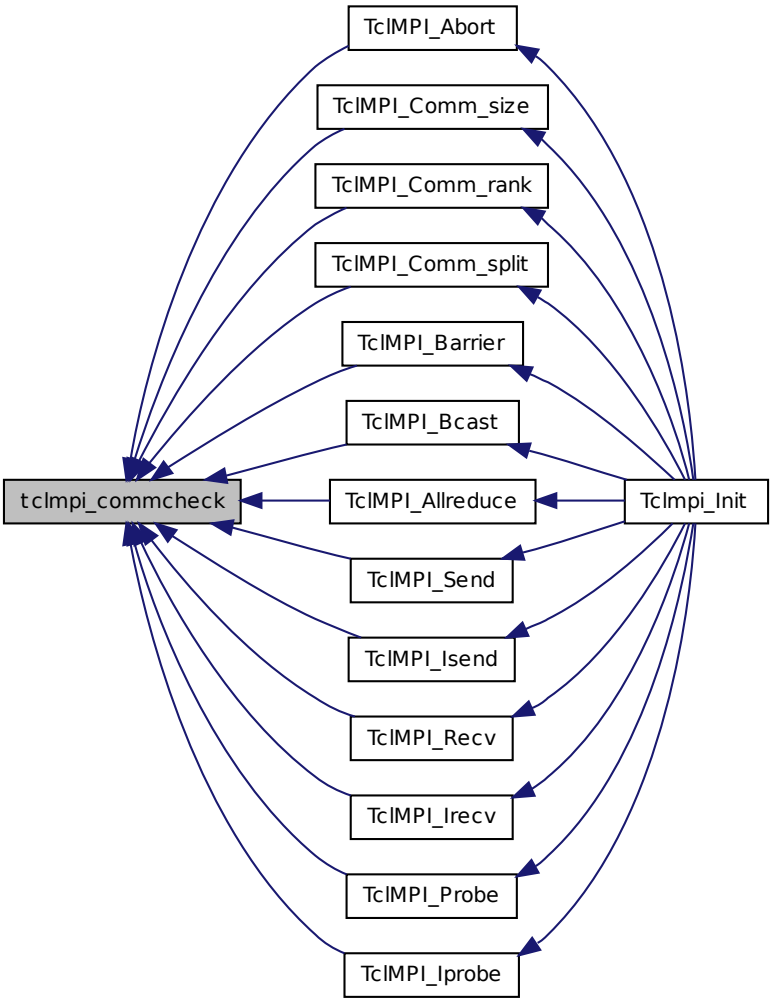
Parameters

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

Returns

TCL_ERROR if the communicator is MPI_COMM_INVALID or TCL_OK

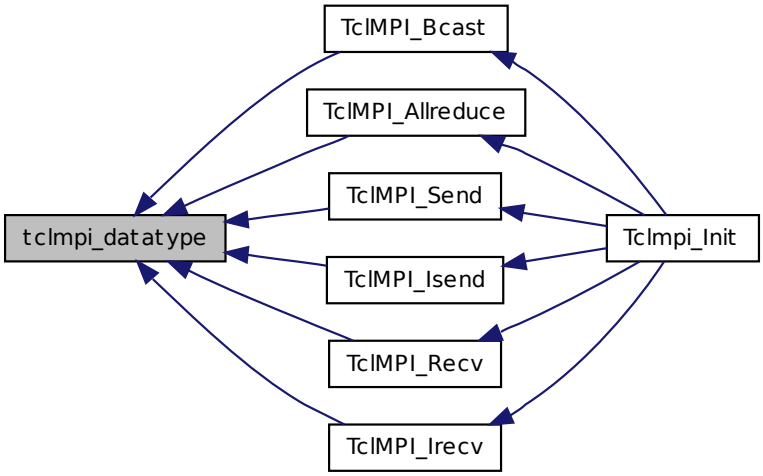
Here is the caller graph for this function:



6.1.3.13 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:



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

remove tclmpi_req_t entry from the request linked list

Parameters

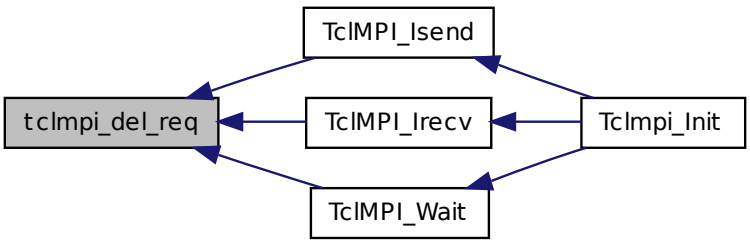
req	a pointer to the request in question

Returns

TCL_OK on succes, TCL_ERROR on failure

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

Here is the caller graph for this function:



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

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

Parameters

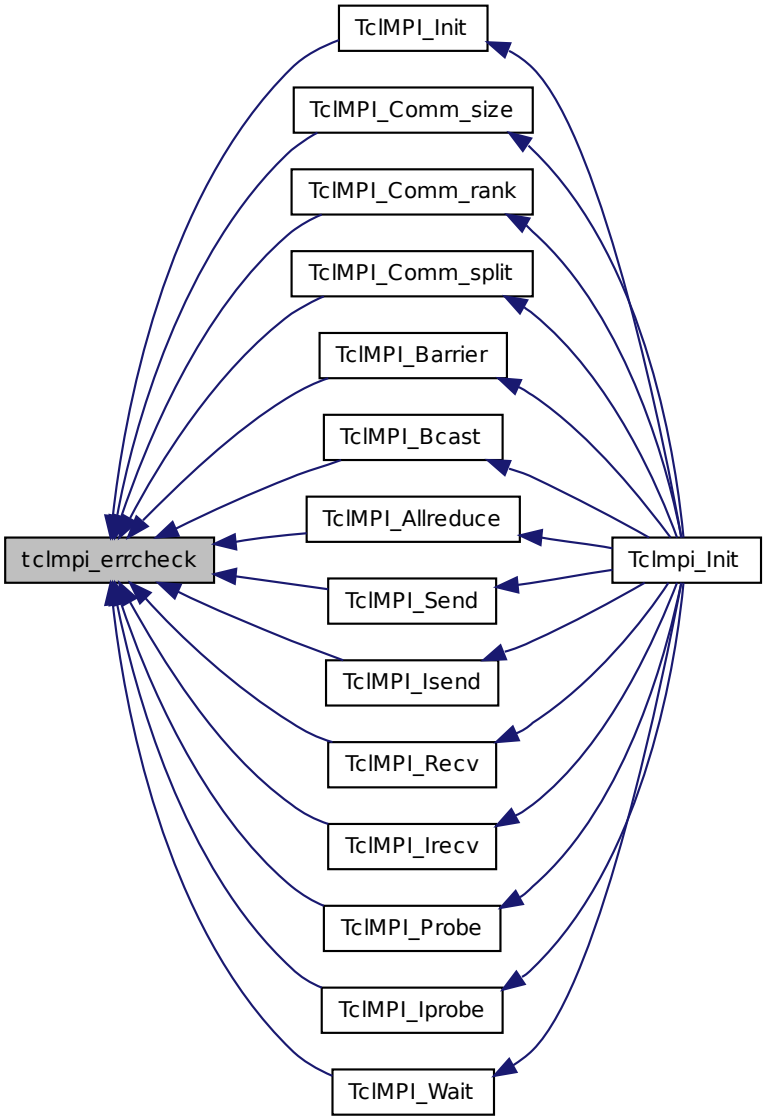
interp	current Tcl interpreter
ierr	MPI error number. return value of an MPI call.
obj	Tcl object representing the current command name

Returns

TCL_OK if the "error" is MPI_SUCCESS or TCL_ERROR

This is a simple convenience wrapper that will use MPI_Error_strin() 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:



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

wrapper for MPI_Finalize()

Parameters

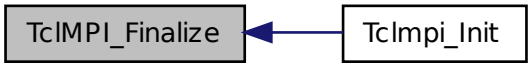
nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

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

Here is the caller graph for this function:



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

translate Tcl representation of an MPI request to request itself.

Parameters

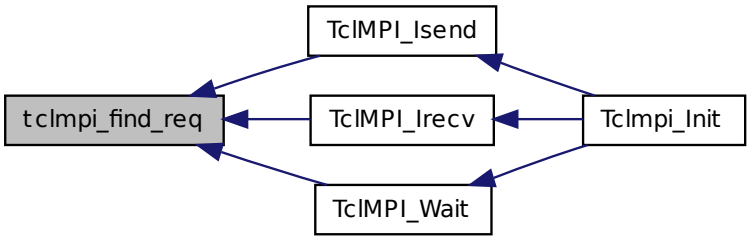
label	the Tcl name for the communicator

Returns

a pointer to the matching tclmpi_req_t structure

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

Here is the caller graph for this function:



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

wrapper for MPI_Init()

Parameters

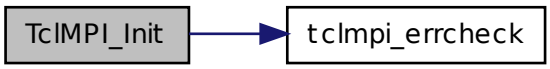
nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

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

Here is the call graph for this function:



Here is the caller graph for this function:

6.1.3.19 int Tclmpi_Init (Tcl_Interp * *interp*)

register this plugin with the Tcl interpreter

Parameters

interp	current Tcl interpreter

Returns

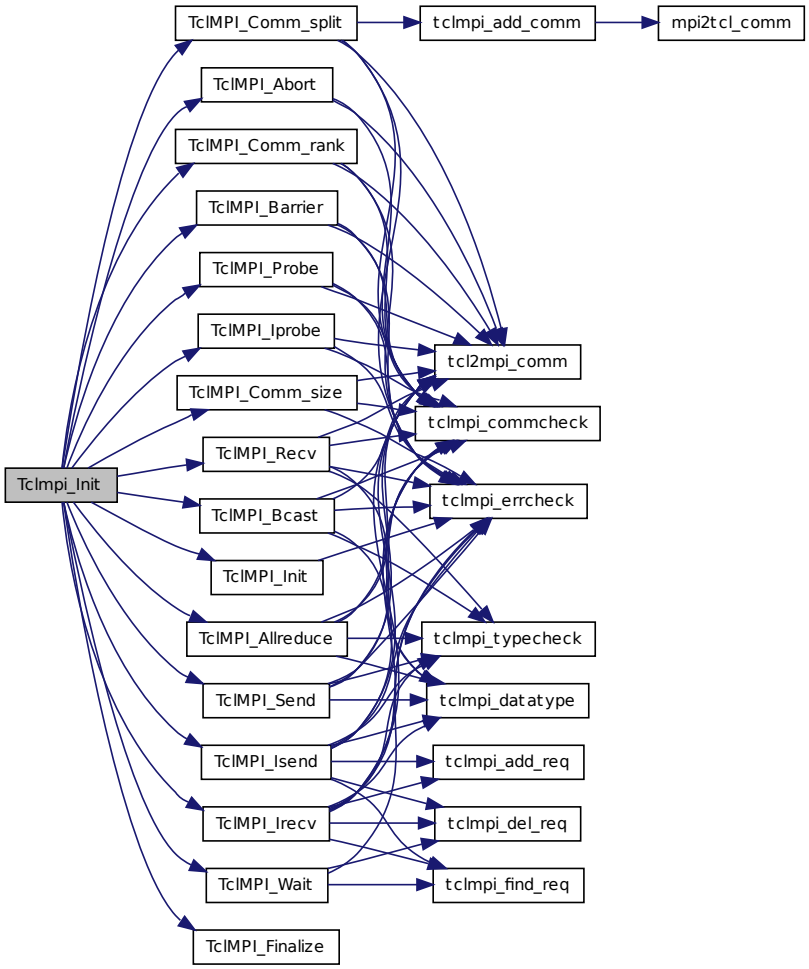
TCL_OK or TCL_ERROR

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

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

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

Here is the call graph for this function:



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

wrapper for MPI_Iprobe()

Parameters

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

Returns

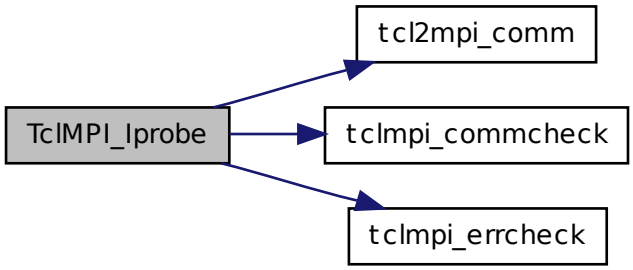
TCL_OK or TCL_ERROR

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

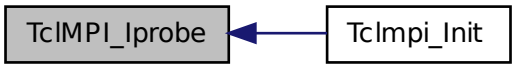
Similar to MPI_Probe, generating a status object to inspect the pending receive is optional. If desired, the argument is taken as a variable name which will then be generated as associative array with several entries similar to what MPI_Status contains. Those are source, tag, error status and count,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

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

wrapper for MPI_Iecv()

Parameters

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

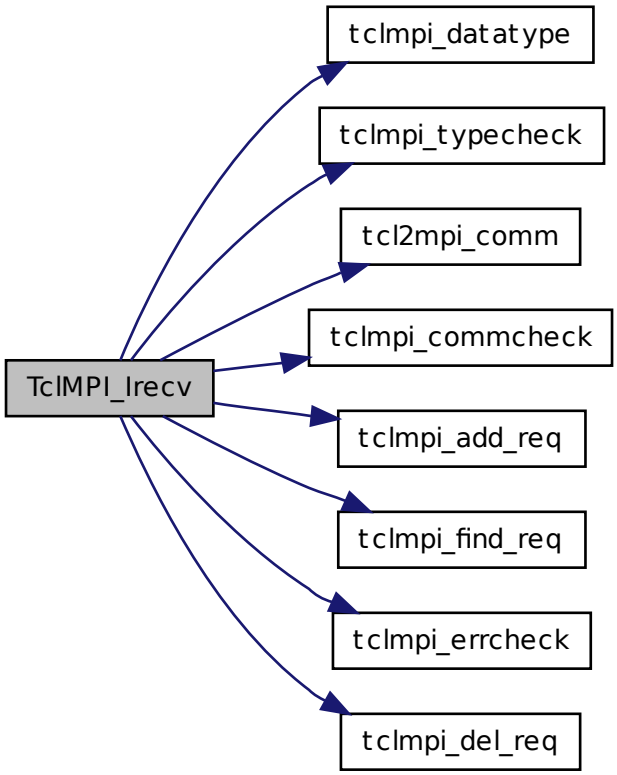
Returns

TCL_OK or TCL_ERROR

This function implements a non-blocking receive operation for TclMPI. Since the length of the data object is supposed to be automatically adjusted to the amount of data being sent, this function needs to be more complex than just a simple wrapper around the corresponding MPI C bindings. It will first call tclmpi_add_req to generate a new

entry to the list of registered MPI requests. It will then call MPI_Iprobe to see if a matching send is already in progress and thus the necessary amount of storage required can be inferred from the MPI_Status object that is populated by MPI_Iprobe. If yes, a temporary receive buffer is allocated and the non-blocking receive is posted and all information is transferred to the tclmpi_req_t object. If not, only the arguments of the receive call are registered in the request object for later use. The command will pass the Tcl string that represents the generated MPI request to the Tcl interpreter as return value. If the MPI call failed, an MPI error message is passed up as result instead and a Tcl error is indicated.

Here is the call graph for this function:



Here is the caller graph for this function:

6.1.3.22 int TclMPI_Isend (ClientData *nodata*, Tcl_Interp * *interp*, int *objc*, Tcl_Obj *const *objv[]*)

wrapper for MPI_Isend()

Parameters

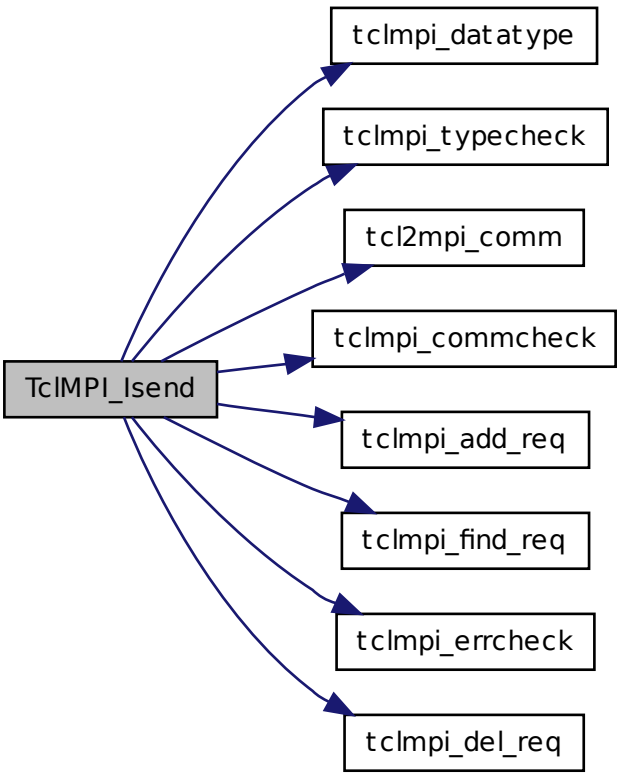
nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

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

Here is the call graph for this function:



Here is the caller graph for this function:

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

wrapper for MPI_Probe()

Parameters

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

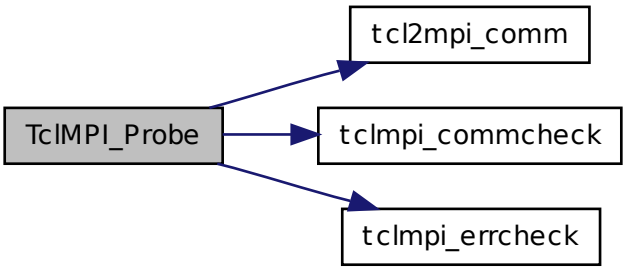
Returns

TCL_OK or TCL_ERROR

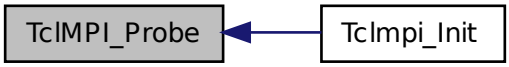
This function implements a blocking probe operation for TclMPI. Argument flags for source, tag, and communicator are translated into their native MPI equivalents and then MPI_Probe called.

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

wrapper for MPI_Recv()

Parameters

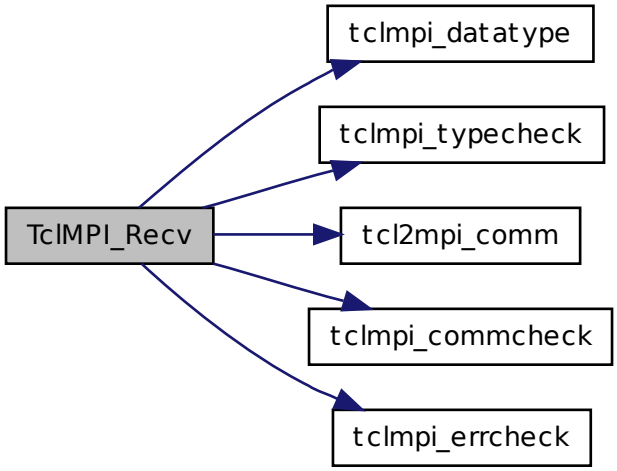
nodata	ignored
interp	current Tcl interpreter
objc	number of argument objects
objv	list of argument object

Returns

TCL_OK or TCL_ERROR

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

Here is the call graph for this function:



Here is the caller graph for this function:

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

wrapper for MPI_Send()

Parameters

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

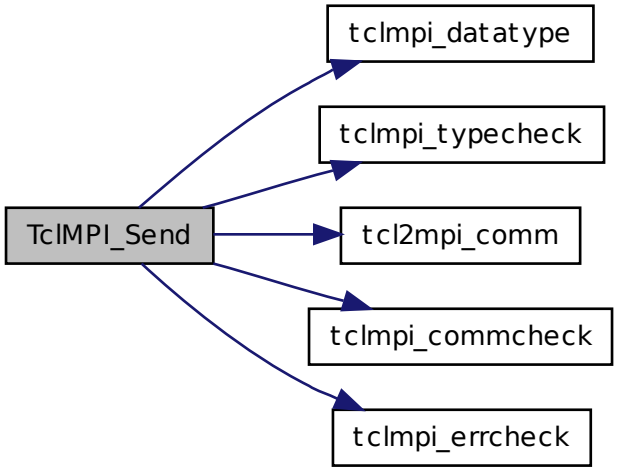
Returns

TCL_OK or TCL_ERROR

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

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

Here is the call graph for this function:



Here is the caller graph for this function:

6.1.3.26 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

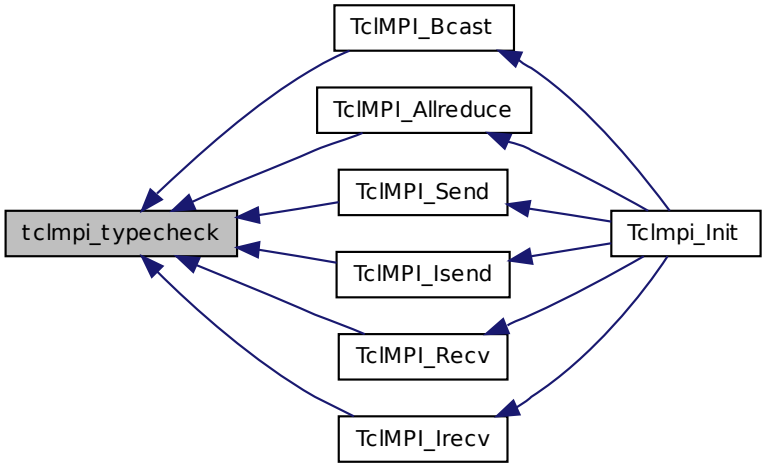
Parameters

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

Returns

TCL_ERROR if the communicator is TCLMPI_NONE or TCL_OK

Here is the caller graph for this function:



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

wrapper for MPI_Wait()

Parameters

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

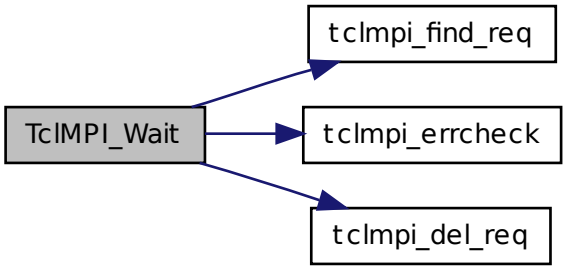
Returns

TCL_OK or TCL_ERROR

This function implements a wrapper around MPI_Wait for TclMPI. Due to the design decisions in TclMPI, it works a bit different than MPI_Write, particularly for non-blocking receive requests. As explained in the TclMPI_Irecv documentation, the corresponding MPI_Irecv may not yet have been posted, so we have to first inspect the tclmpi_req_t object, if the receive still needs to be posted. If yes, then we need to do about the same procedure as for a blocking receive, i.e. call MPI_Probe to determine the size of the receive buffer, 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:

6.1.4 Variable Documentation

6.1.4.1 tclmpi_comm_t* first_comm = NULL [static]

First element of the communicator map list

6.1.4.2 tclmpi_req_t* first_req = NULL [static]

First element of the list of generated requests

6.1.4.3 tclmpi_comm_t* last_comm = NULL [static]

Last element of the communicator map list

6.1.4.4 MPI_Comm MPI_COMM_INVALID [static]

Additional global communicator to detect unlisted communicators

6.1.4.5 int tclmpi_comm_cntr = 0 [static]

Communicator counter. Incremented to get unique strings

6.1.4.6 char tclmpi_errmsg[MPI_MAX_ERROR_STRING] [static]

buffer for error messages.

6.1.4.7 int tclmpi_init_done = 0 [static]

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

6.1.4.8 int tclmpi_req_cntr = 0 [static]

Request counter. Incremented to get unique strings

6.2 tests/harness.tcl File Reference

Functions

- test_format kind cmd result
- ser_init args
- par_init args
- run_return cmd retval
- run_error cmd errormsg
- par_return cmd retval?comm?
- par_error cmd retval?comm?
- par_set name value?comm?
- test_summary section

6.2.1 Detailed Description

Test harness for TclMPI

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

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

6.2.2 Function Documentation

6.2.2.1 par_error cmd retval ?comm?

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.

6.2.2.2 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.

6.2.2.3 par_return cmd retval ?comm?

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.

6.2.2.4 par_set name value ?comm?

set variable to different values on different ranks

Parameters

name	name of variable
value	list of values, one per rank
comm	communicator

Returns

empty

6.2.2.5 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

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.

6.2.2.6 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

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.

6.2.2.7 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. If called from a parallel environment, it will only execute and produce output on the master process

6.2.2.8 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.

6.2.2.9 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.

Index