MATLAB/MEXToolkitforC++/ROS

1.0

Generated by Doxygen 1.7.1

Fri Nov 25 2011 02:00:11

Contents

Chapter 1

MATLAB/MEX Toolkit for C++/ROS

Welcome to the documentation site for the MATLAB/MEX Toolkit for C++/ROS

Author

Edward T. Kaszubski (ekaszubski@gmail.com)

Examples

- CMake Only
- ROS/MATLAB Bridge (MATLAB side)
- ROS/MATLAB Bridge (ROS side)

Chapter 2

ROS/MATLAB Bridge (MATLAB side)

This example set consists of the following examples:

- test_ros.cpp

2.1 test_ros.cpp

This example shows how to use the ros::SharedMemoryPublisher / ros::SharedMemorySubscriber to create a bridge between MATLAB and a fully-functional ROS node (MATLAB side)

See also

ROS/MATLAB Bridge (ROS side)

```
/*********************************************************************
 *  src/test_ros.cpp
 *  --------------------
 *
 *  Copyright (c) 2011, Edward T. Kaszubski ( ekaszubski@gmail.com )
 *  All rights reserved.
 *
 *  Redistribution and use in source and binary forms, with or without
 *  modification, are permitted provided that the following conditions are
 *  met:
 *
 *  * Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  * Redistributions in binary form must reproduce the above
 *    copyright notice, this list of conditions and the following disclaimer
 *    in the documentation and/or other materials provided with the
 *    distribution.
 *  * Neither the name of usc-ros-pkg nor the names of its
 *    contributors may be used to endorse or promote products derived from
 *    this software without specific prior written permission.
 *
 *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 *  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 *  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 *  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 *  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 *  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 *  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 *  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
```

```
 *  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 *  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 *********************************************************************/

#include <iostream>
#include <ros/shared_memory_publisher.h>
#include <ros/shared_memory_subscriber.h>
#include <test_matlab/Vector3.h>

int main( int argc, char ** argv )
{
    if( argc >= 2 )
    {
        //usleep( 1000*1000 );
        ros::SharedMemorySubscriber sub( argv[1] );
        ros::SharedMemoryPublisher pub( sub.getStorage().getName() );

        auto msg = sub.fetch<test_matlab::Vector3>();
        std::cout << msg.x << ", " << msg.y << ", " << msg.z << std::endl;

        msg.x *= 2;
        msg.y *= 2;
        msg.z *= 2;

        pub.publish( msg );
    }

    return 0;
}
```

Chapter 3

ROS/MATLAB Bridge (ROS side)

This example set consists of the following examples:

- test_ros.cpp

3.1 test_ros.cpp

This example shows how to create a fully-functioning ROS node connected to MATLAB via a Shared-Memory bridge

See also

ROS/MATLAB Bridge (MATLAB side)

```
/*********************************************************************
 *  nodes/test_ros.cpp
 *  --------------------
 *
 *  Copyright (c) 2011, Edward T. Kaszubski ( ekaszubski@gmail.com )
 *  All rights reserved.
 *
 *  Redistribution and use in source and binary forms, with or without
 *  modification, are permitted provided that the following conditions are
 *  met:
 *
 *  * Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  * Redistributions in binary form must reproduce the above
 *    copyright notice, this list of conditions and the following disclaimer
 *    in the documentation and/or other materials provided with the
 *    distribution.
 *  * Neither the name of test_matlab_ros nor the names of its
 *    contributors may be used to endorse or promote products derived from
 *    this software without specific prior written permission.
 *
 *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 *  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 *  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 *  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 *  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 *  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 *  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 *  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
```

```
 *  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 *  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 *********************************************************************/

#include <iostream>
#include <ros/shared_memory_publisher.h>
#include <ros/shared_memory_subscriber.h>
#include <test_matlab/Vector3.h>
#include <geometry_msgs/Vector3.h>
#include <ros/ros.h>

ros::SharedMemoryPublisher * pub_ = NULL;

namespace ros_adapters
{

template<class __ToMessage>
struct converter{};

template<>
struct converter<test_matlab::Vector3>
{
    typedef test_matlab::Vector3 _ToMsg;

    static _ToMsg convert( const geometry_msgs::Vector3 & msg )
    {
        _ToMsg result;
        result.x = msg.x;
        result.y = msg.y;
        result.z = msg.z;
        return result;
    }
};

template<>
struct converter<geometry_msgs::Vector3>
{
    typedef geometry_msgs::Vector3 _ToMsg;

    static _ToMsg convert( const test_matlab::Vector3 & msg )
    {
        _ToMsg result;
        result.x = msg.x;
        result.y = msg.y;
        result.z = msg.z;
        return result;
    }
};

} // ros_adapters

void vec3CB( const geometry_msgs::Vector3::ConstPtr & msg )
{
    if( pub_ ) pub_->publish(
      ros_adapters::converter<test_matlab::Vector3>::convert( *msg ) );

    ros::shutdown();
}

int main( int argc, char ** argv )
{
    if( argc >= 2 )
    {
        //usleep( 1000*1000 );
```

```
        pub_ = new ros::SharedMemoryPublisher( argv[1] );

        ros::init( argc, argv, "test_ros" );
        ros::NodeHandle nh_rel( "~" );
        ros::Subscriber vec3_sub = nh_rel.subscribe( "vec3", 1, &vec3CB );

        ros::spin();
    }

    return 0;
}
```

Chapter 4

CMake Only

This example set consists of the following examples:

- test.cpp
- yprime.cpp
- yprime_with_mat.cpp

4.1 test.cpp

This example shows how to use the matlab::Mat wrapper to read/write data to/from MATLAB

```
/*****************************************************************************
 *  src/test.cpp
 *  --------------------
 *
 *  Copyright (c) 2011, Edward T. Kaszubski ( ekaszubski@gmail.com )
 *  All rights reserved.
 *
 *  Redistribution and use in source and binary forms, with or without
 *  modification, are permitted provided that the following conditions are
 *  met:
 *
 *  * Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  * Redistributions in binary form must reproduce the above
 *    copyright notice, this list of conditions and the following disclaimer
 *    in the documentation and/or other materials provided with the
 *    distribution.
 *  * Neither the name of usc-ros-pkg nor the names of its
 *    contributors may be used to endorse or promote products derived from
 *    this software without specific prior written permission.
 *
 *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 *  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 *  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 *  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 *  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 *  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 *  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 *  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 *  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 *  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
```

```
 *  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 ****************************************************************************/

#include "mex.h"
#include <matlab/mat.h>
#include <iostream>

void printUsage()
{
    std::cout << "This function simply scales the given mat: test( scale, mat )\n
      mat must have non-zero dimensions and both arguments must be passed" << std::endl
      ;
}

void mexFunction( int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[] )
{
    // make sure we are reading exactly two arguments and returning no more than
      one value
    if( nrhs != 2 || nlhs > 1 ) return printUsage();

    // construct a matlab::Mat from the first argument; this stores an mxArray *
      and does not do any allocation
    // here, the resuling mat is 1x1
    // we then use the implicit cast operator to read a scalar ( identical to cal
      ling matlab::Mat::first() )
    const double scale = matlab::Mat<double>( prhs[0] );

    // construct a matlab::Mat from the second argument; this stores an mxArray *
       and does not do any allocation
    const matlab::Mat<double> input_mat( prhs[1] );

    // make sure the dimensions of the mat are at least 1x1
    if( input_mat.rows_ * input_mat.cols_ == 0 ) return printUsage();

//  std::cout << input_mat << std::endl;

    // construct a new matlab::Mat with the same dimensions as input_mat; this al
      locates a new mxArray
    // specifically, since we want to use doubles, it calls: mxCreateNumericArray
      ( rows_, cols_, mxDOUBLE_CLASS, mxREAL )
    matlab::Mat<double> output_mat( input_mat.getDim() );

//  example of resizing the mat:
//  std::cout << output_mat << std::endl;
//  output_mat.resize( input_mat.rows_ + 1, input_mat.cols_ + 1 );
//  std::cout << output_mat << std::endl;

    // simple operation to show that everything is working; set every output valu
      e to be the corresponding input value scaled by the given scale
    for( unsigned int row = 0; row < input_mat.rows_; ++row )
    {
        for( unsigned int col = 0; col < input_mat.cols_; ++col )
        {
            // an operator[] is also available but the indexing is less intuitive
       for >1d matrices
            // it is useful for 1xN or Nx1 matrices, however
            output_mat.at( row, col ) = input_mat.at( row, col ) * scale;
        }
    }

    // uses the implicit cast operator for mxArray* to store output_mat value in
      the list of output values
    // this is identical to: plhs[0] = output_mat.getMat();
    plhs[0] = output_mat;
}
```

4.2 yprime.cpp

This example is copied verbatim from MATLAB/extern/examples/mex/yprime.c (note: it has been renamed to yprime.cpp to allow for correct compilation)

```
/*=================================================================
 *
 * YPRIME.C Sample .MEX file corresponding to YPRIME.M
 *          Solves simple 3 body orbit problem
 *
 * The calling syntax is:
 *
 *      [yp] = yprime(t, y)
 *
 *  You may also want to look at the corresponding M-code, yprime.m.
 *
 * This is a MEX-file for MATLAB.
 * Copyright 1984-2006 The MathWorks, Inc.
 *
 *=================================================================*/
/* $Revision: 1.10.6.4 $ */
#include <math.h>
#include "mex.h"

/* Input Arguments */

#define T_IN    prhs[0]
#define Y_IN    prhs[1]


/* Output Arguments */

#define YP_OUT  plhs[0]

#if !defined(MAX)
#define MAX(A, B)   ((A) > (B) ? (A) : (B))
#endif

#if !defined(MIN)
#define MIN(A, B)   ((A) < (B) ? (A) : (B))
#endif

static  double  mu = 1/82.45;
static  double  mus = 1 - 1/82.45;


static void yprime(
           double   yp[],
           double   *t,
           double   y[]
           )
{
    double  r1,r2;

    (void) t;     /* unused parameter */

    r1 = sqrt((y[0]+mu)*(y[0]+mu) + y[2]*y[2]);
    r2 = sqrt((y[0]-mus)*(y[0]-mus) + y[2]*y[2]);

    /* Print warning if dividing by zero. */
    if (r1 == 0.0 || r2 == 0.0 ){
    mexWarnMsgTxt("Division by zero!\n");
    }

    yp[0] = y[1];
    yp[1] = 2*y[3]+y[0]-mus*(y[0]+mu)/(r1*r1*r1)-mu*(y[0]-mus)/(r2*r2*r2);
```

```
    yp[2] = y[3];
    yp[3] = -2*y[1] + y[2] - mus*y[2]/(r1*r1*r1) - mu*y[2]/(r2*r2*r2);
    return;
}

void mexFunction( int nlhs, mxArray *plhs[],
          int nrhs, const mxArray*prhs[] )

{
    double *yp;
    double *t,*y;
    mwSize m,n;

    /* Check for proper number of arguments */

    if (nrhs != 2) {
    mexErrMsgTxt("Two input arguments required.");
    } else if (nlhs > 1) {
    mexErrMsgTxt("Too many output arguments.");
    }

    /* Check the dimensions of Y.  Y can be 4 X 1 or 1 X 4. */

    m = mxGetM(Y_IN);
    n = mxGetN(Y_IN);
    if (!mxIsDouble(Y_IN) || mxIsComplex(Y_IN) ||
    (MAX(m,n) != 4) || (MIN(m,n) != 1)) {
    mexErrMsgTxt("YPRIME requires that Y be a 4 x 1 vector.");
    }

    /* Create a matrix for the return argument */
    YP_OUT = mxCreateDoubleMatrix(m, n, mxREAL);

    /* Assign pointers to the various parameters */
    yp = mxGetPr(YP_OUT);

    t = mxGetPr(T_IN);
    y = mxGetPr(Y_IN);

    /* Do the actual computations in a subroutine */
    yprime(yp,t,y);
    return;

}
```

4.3 yprime_with_mat.cpp

This example shows how to use the matlab::Mat wrapper to perform the same task as yprime

```
/*=========================================================================
 *
 * YPRIME.C Sample .MEX file corresponding to YPRIME.M
 *          Solves simple 3 body orbit problem
 *
 * The calling syntax is:
 *
 *      [yp] = yprime(t, y)
 *
 *  You may also want to look at the corresponding M-code, yprime.m.
 *
 * This is a MEX-file for MATLAB.
 * Copyright 1984-2006 The MathWorks, Inc.
```

```
 *
 *=================================================================*/
/* $Revision: 1.10.6.4 $ */

/*************************************************************************
 *  src/yprime_with_mat.cpp
 *  --------------------
 *
 *  Copyright (c) 2011, Edward T. Kaszubski ( ekaszubski@gmail.com )
 *  All rights reserved.
 *
 *  Redistribution and use in source and binary forms, with or without
 *  modification, are permitted provided that the following conditions are
 *  met:
 *
 *  * Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  * Redistributions in binary form must reproduce the above
 *    copyright notice, this list of conditions and the following disclaimer
 *    in the documentation and/or other materials provided with the
 *    distribution.
 *  * Neither the name of usc-ros-pkg nor the names of its
 *    contributors may be used to endorse or promote products derived from
 *    this software without specific prior written permission.
 *
 *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 *  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 *  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 *  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 *  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 *  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 *  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 *  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 *  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 *  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 *************************************************************************/

#include <math.h>
#include "mex.h"
#include <matlab/mat.h>

/* Input Arguments */

#define T_IN    prhs[0]
#define Y_IN    prhs[1]


/* Output Arguments */

#define YP_OUT  plhs[0]

#if !defined(MAX)
#define MAX(A, B)   ((A) > (B) ? (A) : (B))
#endif

#if !defined(MIN)
#define MIN(A, B)   ((A) < (B) ? (A) : (B))
#endif

static  double  mu = 1/82.45;
static  double  mus = 1 - 1/82.45;

using namespace matlab;

static void yprime(
```

```
            Mat<double> & yp,
            const double & t,
            const Mat<double> & y
            )
{
    double  r1,r2;

    //(void) t;     /* unused parameter */

    // Notice that we can index yp and y as if it was a double[]
    // Furthermore, to get proper row/col access, we can do:
    // yp.at( row, col )
    // to get the 4th row and the 1st column ( equivalent to yp[3] since yp must
      be a 4x1 or 1x4 ):
    // yp.at( 3, 0 ) or yp.at( 0, 3 )
    // the Mat::at() function can be used to read or assign values:
    // yp.at( 3, 0 ) = 5;

    r1 = sqrt((y[0]+mu)*(y[0]+mu) + y[2]*y[2]);
    r2 = sqrt((y[0]-mus)*(y[0]-mus) + y[2]*y[2]);

    /* Print warning if dividing by zero. */
    if (r1 == 0.0 || r2 == 0.0 ){
    mexWarnMsgTxt("Division by zero!\n");
    }

    yp[0] = y[1];
    yp[1] = 2*y[3]+y[0]-mus*(y[0]+mu)/(r1*r1*r1)-mu*(y[0]-mus)/(r2*r2*r2);
    yp[2] = y[3];
    yp[3] = -2*y[1] + y[2] - mus*y[2]/(r1*r1*r1) - mu*y[2]/(r2*r2*r2);
    return;
}

void mexFunction( int nlhs, mxArray *plhs[], int nrhs, const mxArray*prhs[] )
{

    /* Check for proper number of arguments */

    if (nrhs != 2) {
    mexErrMsgTxt("Two input arguments required.");
    } else if (nlhs > 1) {
    mexErrMsgTxt("Too many output arguments.");
    }

    /* Check the dimensions of Y.  Y can be 4 X 1 or 1 X 4. */

    // wrap a matlab::Mat around Y_IN
    const Mat<double> y( Y_IN );

    // we know that T_IN is a 1x1 mat (ie just a double)
    // to get this double, we can do a few equivalent things
    // double t = Mat<double>( T_IN )[0]; // get item at 0th index
    // double t = Mat<double>( T_IN ).first(); // get the first item
    // double t = Mat<double>( T_IN ).at( 0, 0 ); // get the item at the 0th row
      and 0th column
    // or, we can use the implicit cast operator which will call matlab::Mat::fir
      st() for us:
    const double t = Mat<double>( T_IN );

    const int & m = y.rows_;
    const int & n = y.cols_;

    if( ( MAX( m, n ) != 4 ) || ( MIN( m, n ) != 1 ) )
    {
        mexErrMsgTxt( "YPRIME requires that Y be a 4 x 1 vector." );
    }
```

```
    /* Create a matrix for the return argument */
    // equivalent to:
    // Mat<double> yp( m, n );
    // Mat<double> yp( y.rows_, y.cols_ );
    Mat<double> yp( y.getDim() );

    /* Do the actual computations in a subroutine */
    yprime( yp, t, y );

    /* pass yp back to MATLAB */
    // equivalent to:
    // YP_OUT = yp.getMat();
    YP_OUT = yp;

    return;
}
```

Chapter 5

Deprecated List

Class ros::Message This base-class is deprecated in favor of a template-based serialization and traits system

Chapter 6

Directory Hierarchy

6.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

Chapter 7

Class Index

7.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 8

Class Index

8.1 Class List

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

Chapter 9

Directory Documentation

9.1 roscpp_simple/include/ Directory Reference

Directories

- directory ros

9.2 matlab/include/ Directory Reference

Directories

- directory matlab

9.3 matlab/include/matlab/ Directory Reference

Files

- file **mat.h**

9.4 matlab/ Directory Reference

Directories

- directory include

9.5 test_matlab_full_ros/nodes/ Directory Reference

Files

- file **test_ros.cpp**

9.6 roscpp_simple/include/ros/ Directory Reference

Files

- file **message.h**
- file **shared_memory_publisher.h**
- file **shared_memory_storage.h**
- file **shared_memory_subscriber.h**
- file **shared_memory_user.h**

9.7 roscpp_simple/ Directory Reference

Directories

- directory include
- directory src

9.8 test_matlab_no_ros/src/ Directory Reference

Files

- file **test.cpp**
- file **yprime.cpp**
- file **yprime_with_mat.cpp**

9.9 test_matlab_basic_ros/src/ Directory Reference

Files

- file **test_ros.cpp**

9.10 roscpp_simple/src/ Directory Reference

Files

- file **shared_memory_publisher.cpp**
- file **shared_memory_storage.cpp**
- file **shared_memory_subscriber.cpp**
- file **shared_memory_user.cpp**

9.11 test_matlab_basic_ros/ Directory Reference

Directories

- directory src

9.12 test_matlab_full_ros/ Directory Reference

Directories

- directory nodes

9.13 test_matlab_no_ros/ Directory Reference

Directories

- directory src

Chapter 10

Class Documentation

10.1 converter< __Data > Struct Template Reference

Static Public Member Functions

- static void **convert** ()

10.1.1 Detailed Description

template<class __Data> struct converter< __Data >

Definition at line 11 of file shared_memory_subscriber.h.

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

- roscpp_simple/include/ros/shared_memory_subscriber.h

10.2 ros_adapters::converter< __ToMessage > Struct Template Reference

10.2.1 Detailed Description

template<class __ToMessage> struct ros_adapters::converter< __ToMessage >

Definition at line 49 of file test_ros.cpp.

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

- test_matlab_full_ros/nodes/test_ros.cpp

10.3 ros_adapters::converter< geometry_msgs::Vector3 > Struct Template Reference

Public Types

- typedef geometry_msgs::Vector3 **_ToMsg**

Static Public Member Functions

- static _ToMsg **convert** (const test_matlab::Vector3 &msg)

10.3.1 Detailed Description

template<> struct ros_adapters::converter< geometry_msgs::Vector3 >

Definition at line 67 of file test_ros.cpp.

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

- test_matlab_full_ros/nodes/test_ros.cpp

10.4 converter< std::string > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (void *ptr)

10.4.1 Detailed Description

template<> struct converter< std::string >

Definition at line 17 of file shared_memory_subscriber.h.

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

- roscpp_simple/include/ros/shared_memory_subscriber.h

10.5 ros_adapters::converter< test_matlab::Vector3 > Struct Template Reference

Public Types

- typedef test_matlab::Vector3 **_ToMsg**

Static Public Member Functions

- static _ToMsg **convert** (const geometry_msgs::Vector3 &msg)

10.5.1 Detailed Description

template<> struct ros_adapters::converter< test_matlab::Vector3 >

Definition at line 52 of file test_ros.cpp.

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

- test_matlab_full_ros/nodes/test_ros.cpp

10.6 matlab::MatBase::Dim< __Data > Struct Template Reference

Public Types

- typedef __Data **_Data**

Public Member Functions

- **Dim** (const __Data &rows=0, const __Data &cols=0)
- Dim & operator= (const Dim &other)

  Assign data from an existing Dim<__Data>

Public Attributes

- __Data **rows_**
- __Data **cols_**
- __Data & **x_**
- __Data & **y_**

Friends

- std::ostream & **operator**<< (std::ostream &out, const Dim< __Data > &dim)

10.6.1 Detailed Description

template<class __Data = int> struct matlab::MatBase::Dim< __Data >

Definition at line 169 of file mat.h.

10.6.2 Member Function Documentation

10.6.2.1 template<class __Data = int> Dim& matlab::MatBase::Dim< __Data >::operator= (const Dim< __Data > & *other*) [inline]

Assign data from an existing Dim<__Data>

This is necessary because we have non-const reference values (x_, y_) that are not handled automatically by the compiler

Definition at line 188 of file mat.h.

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

- matlab/include/matlab/mat.h

10.7 matlab::is_matlab_compatible< __Data > Struct Template Reference

Defines whether or not a data type can be stored in a matlab array.

`#include <mat.h>`

Static Public Attributes

- static const bool **value** = is_matlab_compatible_helper<matlab_get_class<__Data>::value>::value

10.7.1 Detailed Description

template<class __Data> struct matlab::is_matlab_compatible< __Data >

Defines whether or not a data type can be stored in a matlab array. Specifically, value is true if the corresponding class to __Data is not mxUKNOWN_CLASS and false otherwise

Definition at line 147 of file mat.h.

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

- matlab/include/matlab/mat.h

10.8 matlab::is_matlab_compatible_helper< __ClassId__ > Struct Template Reference

Generic helper for is_matlab_compatible.

`#include <mat.h>`

Static Public Attributes

- static const bool **value** = true

10.8.1 Detailed Description

template<unsigned int __ClassId__> struct matlab::is_matlab_compatible_helper< __ClassId__ >

Generic helper for is_matlab_compatible.

Definition at line 131 of file mat.h.

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

- matlab/include/matlab/mat.h

10.9 matlab::is_matlab_compatible_helper< mxUNKNOWN_CLASS > Struct Template Reference

Specialized helper for is_matlab_compatible.

`#include <mat.h>`

Static Public Attributes

- static const bool **value** = false

10.9.1 Detailed Description

template<> struct matlab::is_matlab_compatible_helper< mxUNKNOWN_CLASS >

Specialized helper for is_matlab_compatible. mxUKNOWN_CLASS is the only incompatible mxClassID

Definition at line 139 of file mat.h.

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

- matlab/include/matlab/mat.h

10.10 matlab::util::is_same_value< __Data, __Id1__, __Id2__ > Struct Template Reference

Simple utility struct to determine if two statically-known data types contain the same value.

`#include <mat.h>`

Static Public Attributes

- static const bool **value** = false

10.10.1 Detailed Description

template<class __Data, __Data __Id1__, __Data __Id2__> struct matlab::util::is_same_value< __Data, __Id1__, __Id2__ >

Simple utility struct to determine if two statically-known data types contain the same value. Specialization for two different values

Definition at line 53 of file mat.h.

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

- matlab/include/matlab/mat.h

10.11 matlab::util::is_same_value< __Data, __Id__, __Id__ > Struct Template Reference

Simple utility struct to determine if two statically-known data types contain the same value.

`#include <mat.h>`

Static Public Attributes

- static const bool **value** = true

10.11.1 Detailed Description

template<class __Data, __Data __Id__> struct matlab::util::is_same_value< __Data, __Id__, __Id__ >

Simple utility struct to determine if two statically-known data types contain the same value. Specialization for two identical values

Definition at line 61 of file mat.h.

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

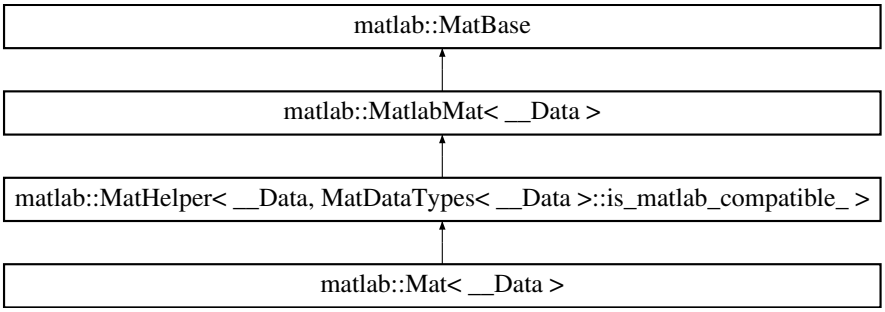
- matlab/include/matlab/mat.h

10.12 matlab::Mat< __Data > Class Template Reference

A generic Mat class; can be matlab-specific.

`#include <mat.h>`

Inheritance diagram for matlab::Mat< __Data >:



Public Types

- typedef MatHelper< __Data, MatDataTypes< __Data >::is_matlab_compatible_ > **_Parent**

Public Member Functions

- template<class... __ParentArgs>
  Mat (__ParentArgs...parent_args)

  Generic constructor to pass all arguments to parent class.

10.12.1 Detailed Description

template<class __Data> class matlab::Mat< __Data >

A generic Mat class; can be matlab-specific. This class will automatically wrap either a MatlabMat or an StdMat; Specifically, if MatDataTypes<__Data>::is_matlab_compatible_ is true, then this class will wrap a MatlabMat<__Data> Otherwise, this class will wrap an StdMat<__Data>

Definition at line 630 of file mat.h.

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

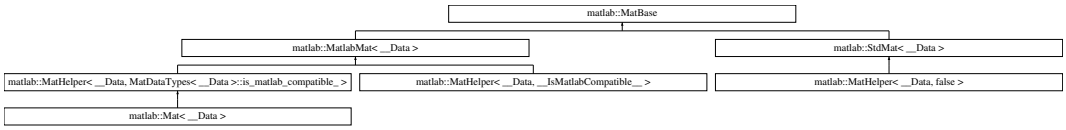
- matlab/include/matlab/mat.h

10.13 matlab::MatBase Class Reference

Base Mat class with dimensions but no storage.

`#include <mat.h>`

Inheritance diagram for matlab::MatBase:



Classes

- struct Dim

Public Types

- typedef Dim< unsigned int > **_Dim**

Public Member Functions

- **MatBase** (const _Dim &dim)
- **MatBase** (const _Dim::_Data &rows, const _Dim::_Data &cols)
- MatBase & operator= (const MatBase &other)

  Assign data from an existing MatBase.

- const _Dim & **getDim** () const

Public Attributes

- _Dim::_Data & **rows_**
- _Dim::_Data & **cols_**

Protected Attributes

- _Dim **dim_**

Friends

- std::ostream & **operator**<< (std::ostream &out, const MatBase &mat)

10.13.1 Detailed Description

Base Mat class with dimensions but no storage. Stores a Dim and provides assignment operators

Definition at line 165 of file mat.h.

10.13.2 Member Function Documentation

10.13.2.1 MatBase& matlab::MatBase::operator= (const MatBase & *other*) [inline]

Assign data from an existing MatBase.

This is necessary because we have non-const reference values (rows_, cols_) that are not handled automatically by the compiler

Definition at line 227 of file mat.h.

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

- matlab/include/matlab/mat.h

10.14 matlab::MatDataTypes< __Data > Struct Template Reference

Type traits for Mats of type __Data.

`#include <mat.h>`

Public Types

- typedef matlab_get_type< matlab_class_ >::type **_MatlabDataType**

Static Public Attributes

- static const bool **is_floating_** = std::is_floating_point<__Data>::value
- static const mxClassID **matlab_class_** = matlab_get_class<__Data>::value
- static const bool **is_matlab_compatible_** = is_matlab_compatible<__Data>::value

10.14.1 Detailed Description

template<class __Data> struct matlab::MatDataTypes< __Data >

Type traits for Mats of type __Data.

Definition at line 154 of file mat.h.

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

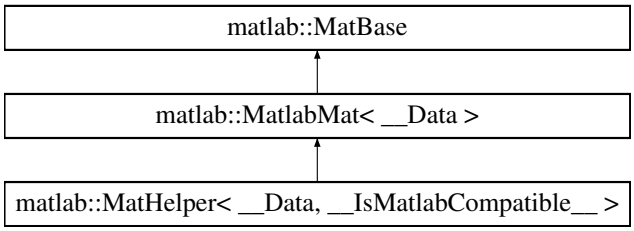
- matlab/include/matlab/mat.h

10.15 matlab::MatHelper< __Data, __IsMatlabCompatible__ > Class Template Reference

Adapter for types that are compatible with matlab.

`#include <mat.h>`

Inheritance diagram for matlab::MatHelper< __Data, __IsMatlabCompatible__ >:



Public Types

- typedef MatlabMat< __Data > **_Parent**

Public Member Functions

- template<class... __ParentArgs>
  MatHelper (__ParentArgs...parent_args)

  Generic constructor to pass all arguments to parent class.

10.15.1 Detailed Description

template<class __Data, bool __IsMatlabCompatible__> class matlab::MatHelper< __Data, __IsMatlabCompatible__ >

Adapter for types that are compatible with matlab. To form this adapter, we simply inherit from MatlabMat<__Data>

See also

MatHelper<__Data, false>, MatlabMat

Definition at line 591 of file mat.h.

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

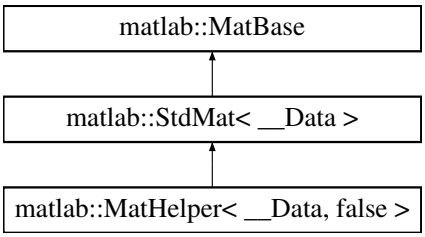
- matlab/include/matlab/mat.h

10.16 matlab::MatHelper< __Data, false > Class Template Reference

Adapter for types that are not compatible with matlab.

`#include <mat.h>`

Inheritance diagram for matlab::MatHelper< __Data, false >:



Public Types

- typedef StdMat< __Data > **_Parent**

Public Member Functions

- template<class... __ParentArgs>
  MatHelper (__ParentArgs...parent_args)

  Generic constructor to pass all arguments to parent class.

10.16.1 Detailed Description

template<class __Data> class matlab::MatHelper< __Data, false >

Adapter for types that are not compatible with matlab. To form this adapter, we simply inherit from StdMat<__Data>

See also

MatHelper<>, StdMat

Definition at line 610 of file mat.h.

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

- matlab/include/matlab/mat.h

10.17 matlab::matlab_get_class< __Data > Struct Template Reference

Defines the mxClassID that corresponds to the __Data.

`#include <mat.h>`

Static Public Attributes

- static const mxClassID **value** = matlab_get_class_helper<__Data>::value

10.17.1 Detailed Description

template<class __Data> struct matlab::matlab_get_class< __Data >

Defines the mxClassID that corresponds to the __Data.

Definition at line 117 of file mat.h.

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

- matlab/include/matlab/mat.h

10.18 matlab::matlab_get_class_helper< __Data > Struct Template Reference

Generic helper for matlab_get_class.

`#include <mat.h>`

Static Public Attributes

- static const mxClassID **value** = mxUNKNOWN_CLASS

10.18.1 Detailed Description

template<class __Data> struct matlab::matlab_get_class_helper< __Data >

Generic helper for matlab_get_class. If there are no specialized matches for __Data then we say that the ClassID that corresponds to this data type is unknown

Definition at line 71 of file mat.h.

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

- matlab/include/matlab/mat.h

10.19 matlab::matlab_get_class_helper< bool > Struct Template Reference

Static Public Attributes

- static const mxClassID **value** = mxLOGICAL_CLASS

10.19.1 Detailed Description

template<> struct matlab::matlab_get_class_helper< bool >

Definition at line 101 of file mat.h.

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

- matlab/include/matlab/mat.h

10.20 matlab::matlab_get_class_helper< char > Struct Template Reference

Static Public Attributes

- static const mxClassID **value** = mxCHAR_CLASS

10.20.1 Detailed Description

template<> struct matlab::matlab_get_class_helper< char >

Definition at line 102 of file mat.h.

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

- matlab/include/matlab/mat.h

10.21 matlab::matlab_get_class_helper< double > Struct Template Reference

Static Public Attributes

- static const mxClassID **value** = mxDOUBLE_CLASS

10.21.1 Detailed Description

template<> struct matlab::matlab_get_class_helper< double >

Definition at line 104 of file mat.h.

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

- matlab/include/matlab/mat.h

10.22 matlab::matlab_get_class_helper< float > Struct Template Reference

Static Public Attributes

- static const mxClassID **value** = mxSINGLE_CLASS

10.22.1 Detailed Description

template<> struct matlab::matlab_get_class_helper< float >

Definition at line 105 of file mat.h.

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

- matlab/include/matlab/mat.h

10.23 matlab::matlab_get_class_helper< int16_t > Struct Template Reference

Static Public Attributes

- static const mxClassID **value** = mxINT16_CLASS

10.23.1 Detailed Description

template<> struct matlab::matlab_get_class_helper< int16_t >

Definition at line 108 of file mat.h.

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

- matlab/include/matlab/mat.h

10.24 matlab::matlab_get_class_helper< int32_t > Struct Template Reference

Static Public Attributes

- static const mxClassID **value** = mxINT32_CLASS

10.24.1 Detailed Description

template<> struct matlab::matlab_get_class_helper< int32_t >

Definition at line 110 of file mat.h.

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

- matlab/include/matlab/mat.h

10.25 matlab::matlab_get_class_helper< int64_t > Struct Template Reference

Static Public Attributes

- static const mxClassID **value** = mxINT64_CLASS

10.25.1 Detailed Description

template<> struct matlab::matlab_get_class_helper< int64_t >

Definition at line 112 of file mat.h.

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

- matlab/include/matlab/mat.h

10.26 matlab::matlab_get_class_helper< int8_t > Struct Template Reference

Static Public Attributes

- static const mxClassID **value** = mxINT8_CLASS

10.26.1 Detailed Description

template<> struct matlab::matlab_get_class_helper< int8_t >

Definition at line 106 of file mat.h.

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

- matlab/include/matlab/mat.h

10.27 matlab::matlab_get_class_helper< uint16_t > Struct Template Reference

Static Public Attributes

- static const mxClassID **value** = mxUINT16_CLASS

10.27.1 Detailed Description

template<> struct matlab::matlab_get_class_helper< uint16_t >

Definition at line 109 of file mat.h.

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

- matlab/include/matlab/mat.h

10.28 matlab::matlab_get_class_helper< uint32_t > Struct Template Reference

Static Public Attributes

- static const mxClassID **value** = mxUINT32_CLASS

10.28.1 Detailed Description

template<> struct matlab::matlab_get_class_helper< uint32_t >

Definition at line 111 of file mat.h.

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

- matlab/include/matlab/mat.h

10.29 matlab::matlab_get_class_helper< uint64_t > Struct Template Reference

Static Public Attributes

- static const mxClassID **value** = mxUINT64_CLASS

10.29.1 Detailed Description

template<> struct matlab::matlab_get_class_helper< uint64_t >

Definition at line 113 of file mat.h.

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

- matlab/include/matlab/mat.h

10.30 matlab::matlab_get_class_helper< uint8_t > Struct Template Reference

Static Public Attributes

- static const mxClassID **value** = mxUINT8_CLASS

10.30.1 Detailed Description

template<> struct matlab::matlab_get_class_helper< uint8_t >

Definition at line 107 of file mat.h.

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

- matlab/include/matlab/mat.h

10.31 matlab::matlab_get_class_helper< void > Struct Template Reference

Static Public Attributes

- static const mxClassID **value** = mxVOID_CLASS

10.31.1 Detailed Description

template<> struct matlab::matlab_get_class_helper< void >

Definition at line 103 of file mat.h.

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

- matlab/include/matlab/mat.h

10.32 matlab::matlab_get_type< __ClassId__ > Struct Template Reference

Defines the data type that corresponds to __ClassId__.

`#include <mat.h>`

Public Types

- typedef matlab_get_type_helper< __ClassId__ >::type **type**

10.32.1 Detailed Description

template<unsigned int __ClassId__> struct matlab::matlab_get_type< __ClassId__ >

Defines the data type that corresponds to __ClassId__.

Definition at line 124 of file mat.h.

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

- matlab/include/matlab/mat.h

10.33 matlab::matlab_get_type_helper< __ClassId__ > Struct Template Reference

Generic helper for matlab_get_type.

`#include <mat.h>`

Public Types

- typedef struct UnknownType **type**

10.33.1 Detailed Description

template<unsigned int __ClassId__> struct matlab::matlab_get_type_helper< __ClassId__ >

Generic helper for matlab_get_type. If there are no specialized matches for __ClassId__ then we say that the type that corresponds to this ClassID is unknown

Definition at line 79 of file mat.h.

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

- matlab/include/matlab/mat.h

10.34 matlab::matlab_get_type_helper< mxCHAR_CLASS > Struct Template Reference

Public Types

- typedef char **type**

10.34.1 Detailed Description

template<> struct matlab::matlab_get_type_helper< mxCHAR_CLASS >

Definition at line 102 of file mat.h.

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

- matlab/include/matlab/mat.h

10.35 matlab::matlab_get_type_helper< mxDOUBLE_CLASS > Struct Template Reference

Public Types

- typedef double **type**

10.35.1 Detailed Description

template<> struct matlab::matlab_get_type_helper< mxDOUBLE_CLASS >

Definition at line 104 of file mat.h.

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

- matlab/include/matlab/mat.h

10.36 matlab::matlab_get_type_helper< mxINT16_CLASS > Struct Template Reference

Public Types

- typedef int16_t **type**

10.36.1 Detailed Description

template<> struct matlab::matlab_get_type_helper< mxINT16_CLASS >

Definition at line 108 of file mat.h.

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

- matlab/include/matlab/mat.h

10.37 matlab::matlab_get_type_helper< mxINT32_CLASS > Struct Template Reference

Public Types

- typedef int32_t **type**

10.37.1 Detailed Description

template<> struct matlab::matlab_get_type_helper< mxINT32_CLASS >

Definition at line 110 of file mat.h.

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

- matlab/include/matlab/mat.h

10.38 matlab::matlab_get_type_helper< mxINT64_CLASS > Struct Template Reference

Public Types

- typedef int64_t **type**

10.38.1 Detailed Description

template<> struct matlab::matlab_get_type_helper< mxINT64_CLASS >

Definition at line 112 of file mat.h.

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

- matlab/include/matlab/mat.h

10.39 matlab::matlab_get_type_helper< mxINT8_CLASS > Struct Template Reference

Public Types

- typedef int8_t **type**

10.39.1 Detailed Description

template<> struct matlab::matlab_get_type_helper< mxINT8_CLASS >

Definition at line 106 of file mat.h.

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

- matlab/include/matlab/mat.h

10.40 matlab::matlab_get_type_helper< mxLOGICAL_CLASS > Struct Template Reference

Public Types

- typedef bool **type**

10.40.1 Detailed Description

template<> struct matlab::matlab_get_type_helper< mxLOGICAL_CLASS >

Definition at line 101 of file mat.h.

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

- matlab/include/matlab/mat.h

10.41 matlab::matlab_get_type_helper< mxSINGLE_CLASS > Struct Template Reference

Public Types

- typedef float **type**

10.41.1 Detailed Description

template<> struct matlab::matlab_get_type_helper< mxSINGLE_CLASS >

Definition at line 105 of file mat.h.

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

- matlab/include/matlab/mat.h

10.42 matlab::matlab_get_type_helper< mxUINT16_CLASS > Struct Template Reference

Public Types

- typedef uint16_t **type**

10.42.1 Detailed Description

template<> struct matlab::matlab_get_type_helper< mxUINT16_CLASS >

Definition at line 109 of file mat.h.

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

- matlab/include/matlab/mat.h

10.43 matlab::matlab_get_type_helper< mxUINT32_CLASS > Struct Template Reference

Public Types

- typedef uint32_t **type**

10.43.1 Detailed Description

template<> struct matlab::matlab_get_type_helper< mxUINT32_CLASS >

Definition at line 111 of file mat.h.

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

- matlab/include/matlab/mat.h

10.44 matlab::matlab_get_type_helper< mxUINT64_CLASS > Struct Template Reference

Public Types

- typedef uint64_t **type**

10.44.1 Detailed Description

template<> struct matlab::matlab_get_type_helper< mxUINT64_CLASS >

Definition at line 113 of file mat.h.

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

- matlab/include/matlab/mat.h

10.45 matlab::matlab_get_type_helper< mxUINT8_CLASS > Struct Template Reference

Public Types

- typedef uint8_t **type**

10.45.1 Detailed Description

template<> struct matlab::matlab_get_type_helper< mxUINT8_CLASS >

Definition at line 107 of file mat.h.

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

- matlab/include/matlab/mat.h

10.46 matlab::matlab_get_type_helper< mxVOID_CLASS > Struct Template Reference

Public Types

- typedef void **type**

10.46.1 Detailed Description

template<> struct matlab::matlab_get_type_helper< mxVOID_CLASS >

Definition at line 103 of file mat.h.

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

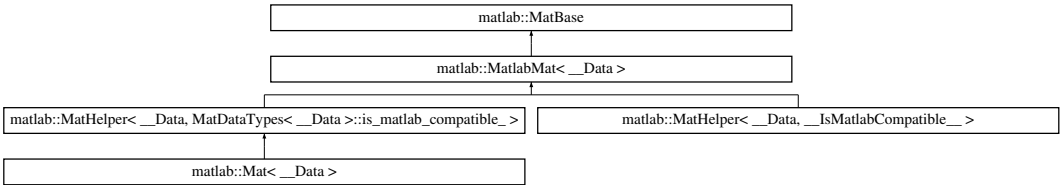
- matlab/include/matlab/mat.h

10.47 matlab::MatlabMat< __Data > Class Template Reference

A generic Mat class with indexing based on MatlabMat.

`#include <mat.h>`

Inheritance diagram for matlab::MatlabMat< __Data >:



Public Types

- typedef MatlabMatTypes::_Mat **_Mat**
- typedef MatlabMatTypes::_MatPtr **_MatPtr**
- typedef MatBase **_Parent**

Public Member Functions

- template<class __Mat , typename std::enable_if<(std::is_same< __Mat, const _Mat >::value), int >::type = 0>
  MatlabMat (__Mat *mat)

  Wrap an existing const mxArray.

- template<class __Mat , typename std::enable_if<(std::is_same< __Mat, _Mat >::value), int >::type = 0>
  MatlabMat (__Mat *mat)

  Wrap an existing non-const mxArray.

- template<class... __ParentArgs>
  MatlabMat (__ParentArgs &&...parent_args)

  Generic constructor to pass all arguments to parent class.

- template<class... __ParentArgs>
  MatlabMat (mxComplexity complexity, __ParentArgs &&...parent_args)

  Generic constructor to read the desired complexity and pass all other arguments to parent class.

- void resize (const MatBase::_Dim &dim)

  Change the dimensions of the matrix.

- void resize (const MatBase::_Dim::_Data &rows, const MatBase::_Dim::_Data &cols)

  Change the dimensions of the matrix.

- template<mxClassID __ClassId__, typename std::enable_if<(!util::is_same_value< decltype(__ClassId__), __ClassId__-, mxCHAR_CLASS >::value), int >::type = 0>
  const std::string dataToString () const

  Convert our data to a formatted string; specialization enabled if __ClassId__ != mxCHAR_CLASS.

- template<mxClassID __ClassId__, typename std::enable_if<(util::is_same_value< decltype(__ClassId__), __ClassId__-, mxCHAR_CLASS >::value), int >::type = 0>
  const std::string dataToString () const

  Convert our data to a formatted string; specialization enabled if __ClassId__ == mxCHAR_CLASS.

- const std::string dataToString () const

  Convert our data to a formatted string.

- const std::string **toString** () const
- _MatPtr **getMat** ()
- __Data * **getData** ()
- __Data *const **getData** () const
- __Data & **operator[ ]** (const unsigned int &index)
- const __Data & **operator[ ]** (const unsigned int &index) const
- __Data & **at** (const unsigned int &row, const unsigned int &col)
- const __Data & **at** (const unsigned int &row, const unsigned int &col) const
- __Data & **first** ()
- const __Data & **first** () const
- **operator mxArray** * ()
- **operator __Data &** ()
- **operator const __Data &** () const
- **operator const std::string** () const

Protected Attributes

- _MatPtr **mat_**

Friends

- std::ostream & **operator**<< (std::ostream &out, const MatlabMat< __Data > &mat)

10.47.1 Detailed Description

template<class __Data> class matlab::MatlabMat< __Data >

A generic Mat class with indexing based on MatlabMat.

See also

MatlabMat

Definition at line 264 of file mat.h.

10.47.2 Constructor & Destructor Documentation

10.47.2.1 template<class __Data > template<class __Mat , typename std::enable_if<(std::is_same< __Mat, const _Mat >::value), int >::type = 0> matlab::MatlabMat< __Data >::MatlabMat (__Mat * *mat*) [inline]

Wrap an existing const mxArray.

Note that we const_cast to remove the const property. When using this constructor, one should use the syntax to preserve const-ness: const MatlabMat<Type>(value)

Definition at line 280 of file mat.h.

Referenced by matlab::MatlabMat< __Data >::resize().

10.47.2.2 template<class __Data > template<class... __ParentArgs> matlab::MatlabMat< __Data >::MatlabMat (__ParentArgs &&... *parent_args*) [inline]

Generic constructor to pass all arguments to parent class.

Note that we also allocate a new mxArray since this constructor is only used to create new data rather than wrap existing data

Definition at line 299 of file mat.h.

10.47.2.3 template<class __Data > template<class... __ParentArgs> matlab::MatlabMat< __Data >::MatlabMat (mxComplexity *complexity,* __ParentArgs &&... *parent_args*) [inline]

Generic constructor to read the desired complexity and pass all other arguments to parent class.

Note that we also allocate a new mxArray since this constructor is only used to create new data rather than wrap existing data

Definition at line 309 of file mat.h.

10.47.3 Member Function Documentation

10.47.3.1 template<class __Data > template<mxClassID __ClassId__, typename std::enable_if<(util::is_same_value< decltype(__ClassId__), __ClassId__, mxCHAR_CLASS >::value), int >::type = 0> const std::string matlab::MatlabMat< __Data >::dataToString ( ) const [inline]

Convert our data to a formatted string; specialization enabled if __ClassId__ == mxCHAR_CLASS.

Since we're storing an array of characters, it's possible to compile them directly into an std::string

Definition at line 357 of file mat.h.

10.47.3.2 template<class __Data > const std::string matlab::MatlabMat< __Data >::dataToString ( ) const [inline]

Convert our data to a formatted string.

In order to use std::enable_if in a class, we need to use a dependent type in the template declaration even if we're enabling based on a class template value

Definition at line 364 of file mat.h.

10.47.3.3 template<class __Data > template<mxClassID __ClassId__, typename std::enable_if<(!util::is_same_value< decltype(__ClassId__), __ClassId__, mxCHAR_CLASS >::value), int >::type = 0> const std::string matlab::MatlabMat< __Data >::dataToString ( ) const [inline]

Convert our data to a formatted string; specialization enabled if __ClassId__ != mxCHAR_CLASS.

Print out all stored values, showing row vectors and column vectors

Definition at line 334 of file mat.h.

10.47.3.4 template<class __Data > void matlab::MatlabMat< __Data >::resize (const MatBase::_Dim & *dim*) [inline]

Change the dimensions of the matrix.

Attention

Experimental; reallocates storage for this mat; destroys existing data

Definition at line 318 of file mat.h.

10.47.3.5 template<class __Data > void matlab::MatlabMat< __Data >::resize (const MatBase::_Dim::_Data & *rows,* const MatBase::_Dim::_Data & *cols*) [inline]

Change the dimensions of the matrix.

Attention

Experimental; reallocates storage for this mat; destroys existing data

Definition at line 325 of file mat.h.

References matlab::MatlabMat< __Data >::MatlabMat().

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

- matlab/include/matlab/mat.h

10.48 matlab::MatlabMatTypes Struct Reference

Typedefs for MatlabMat.

`#include <mat.h>`

Public Types

- typedef mxArray **_Mat**
- typedef _Mat * **_MatPtr**

10.48.1 Detailed Description

Typedefs for MatlabMat.

Definition at line 255 of file mat.h.

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

- matlab/include/matlab/mat.h

10.49 ros::Message Class Reference

`#include <message.h>`

Public Types

- typedef boost::shared_ptr< Message > **Ptr**
- typedef boost::shared_ptr< Message const > **ConstPtr**

Public Member Functions

- virtual const std::string **__getDataType** () const =0
- virtual const std::string **__getMD5Sum** () const =0
- virtual const std::string **__getMessageDefinition** () const =0
- virtual uint32_t **serializationLength** () const =0
- virtual uint8_t * **serialize** (uint8_t *write_ptr, uint32_t seq) const =0
- virtual uint8_t * **deserialize** (uint8_t *read_ptr)=0

Static Public Member Functions

- static std::string **__s_getDataType** ()
- static std::string **__s_getMD5Sum** ()
- static std::string **__s_getMessageDefinition** ()

Public Attributes

- uint32_t **__serialized_length**
- boost::shared_ptr< M_string > **__connection_header**

10.49.1 Detailed Description

Deprecated

This base-class is deprecated in favor of a template-based serialization and traits system

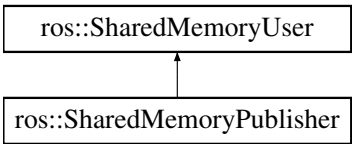
Definition at line 52 of file message.h.

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

- roscpp_simple/include/ros/message.h

10.50 ros::SharedMemoryPublisher Class Reference

Inheritance diagram for ros::SharedMemoryPublisher:



Public Member Functions

- template<class... __Args>
  SharedMemoryPublisher (__Args &&...args)
- template<class __Message >
  std::enable_if<(!boost::is_base_of< ros::Message, __Message >::value), void >::type **publish** (__Message *message, const unsigned int &size)
- template<class __Message >
  std::enable_if<(boost::is_base_of< ros::Message, __Message >::value), void >::type **publish** (const __Message &message)
- template<class __Message >
  std::enable_if<(boost::is_base_of< ros::Message, __Message >::value), void >::type **publish** (const boost::shared_ptr< __Message > &message)

10.50.1 Detailed Description

Definition at line 13 of file shared_memory_publisher.h.

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

- roscpp_simple/include/ros/shared_memory_publisher.h

10.51 ros::SharedMemoryStorage Class Reference

Public Member Functions

- **SharedMemoryStorage** (const std::string &name)
- const std::string & **getName** () const
- void **initialize** (unsigned int size)
- template<class __Source >
  void **push** (__Source *source, unsigned int size)
- void * **pull** () const

Private Types

- typedef boost::interprocess::shared_memory_object **_InternalStorage**
- typedef boost::interprocess::mapped_region **_Storage**

Private Member Functions

- int **releaseStorageId** ()

Static Private Member Functions

- static int **setStorageId** (int inc=0)
- static int **makeStorageId** ()
- static std::string **makeName** (unsigned int id)
- static void **removeStorage** (const std::string &name)
- static _InternalStorage **createStorage** (const std::string &name)

Private Attributes

- std::string **name_**
- _InternalStorage **internal_storage_**
- _Storage **storage_**
- bool **initialized_**
- bool **is_ref_**

10.51.1 Detailed Description

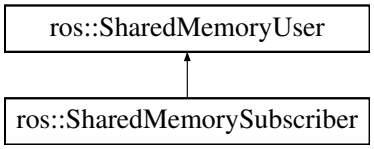
Definition at line 13 of file shared_memory_storage.h.

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

- roscpp_simple/include/ros/shared_memory_storage.h

10.52 ros::SharedMemorySubscriber Class Reference

Inheritance diagram for ros::SharedMemorySubscriber:



Public Member Functions

- template<class... __Args>
  SharedMemorySubscriber (__Args &&...args)
- template<class __Data >
  std::enable_if<(!boost::is_base_of< ros::Message, __Data >::value), __Data >::type **fetch** () const
- template<class __Message >
  std::enable_if<(boost::is_base_of< ros::Message, __Message >::value), __Message >::type **fetch** () const

10.52.1 Detailed Description

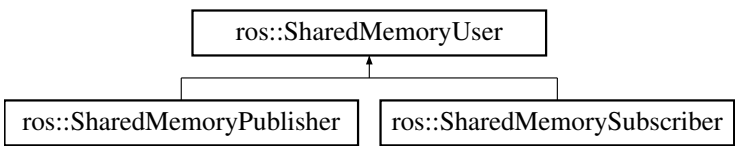
Definition at line 28 of file shared_memory_subscriber.h.

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

- roscpp_simple/include/ros/shared_memory_subscriber.h

10.53 ros::SharedMemoryUser Class Reference

Inheritance diagram for ros::SharedMemoryUser:



Public Member Functions

- **SharedMemoryUser** (const unsigned int &buffer=524288)
- **SharedMemoryUser** (const SharedMemoryStorage &storage)
- **SharedMemoryUser** (const std::string &name)
- const SharedMemoryStorage & **getStorage** () const

Protected Attributes

- SharedMemoryStorage **storage_**

10.53.1 Detailed Description

Definition at line 9 of file shared_memory_user.h.

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

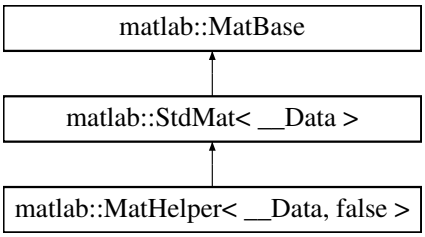
- roscpp_simple/include/ros/shared_memory_user.h

10.54 matlab::StdMat< __Data > Class Template Reference

A generic Mat class with indexing based on MatlabMat.

`#include <mat.h>`

Inheritance diagram for matlab::StdMat< __Data >:



Public Types

- typedef StdMatTypes< __Data > **_StdMatTypes**
- typedef _StdMatTypes::_RowVec **_RowVec**
- typedef _StdMatTypes::_ColVec **_ColVec**
- typedef _StdMatTypes::_Mat **_Mat**
- typedef MatBase **_Parent**

Public Member Functions

- template<class... __ParentArgs>
  StdMat (__ParentArgs...parent_args)

  Generic constructor to pass all arguments to parent class.

- template<class __MData , typename std::enable_if<(!std::is_same< __MData, char >::value), int >::type = 0>
  const std::string dataToString () const

  Convert our data to a formatted string; specialization enabled if __Data != char.

- template<class __MData , typename std::enable_if<(std::is_same< __MData, char >::value), int >::type = 0>
  const std::string dataToString () const

  Convert our data to a formatted string; specialization enabled if __Data == char.

- const std::string dataToString () const

  Convert our data to a formatted string.

- const std::string **toString** () const
- _Mat **getMat** ()
- _Mat::iterator **getData** ()
- _Mat::const_iterator **getData** () const
- __Data & **operator[ ]** (const unsigned int &index)
- const __Data & **operator[ ]** (const unsigned int &index) const
- __Data & **at** (const unsigned int &row, const unsigned int &col)
- const __Data & **at** (const unsigned int &row, const unsigned int &col) const
- __Data & **first** ()
- const __Data & **first** () const
- **operator __Data &** ()
- **operator const __Data &** () const
- **operator const std::string** () const

Protected Attributes

- _Mat **mat_**

Friends

- std::ostream & **operator**<< (std::ostream &out, const StdMat< __Data > &mat)

10.54.1 Detailed Description

template<class __Data> class matlab::StdMat< __Data >

A generic Mat class with indexing based on MatlabMat.

See also

MatlabMat

Definition at line 451 of file mat.h.

10.54.2 Constructor & Destructor Documentation

10.54.2.1 template<class __Data > template<class... __ParentArgs> matlab::StdMat< __Data >::StdMat (__ParentArgs... *parent_args*) [inline]

Generic constructor to pass all arguments to parent class.

Note that since we inherit from MatBase, as soon as we construct _Parent we can use rows_ and cols_ from MatBase

Definition at line 468 of file mat.h.

10.54.3 Member Function Documentation

10.54.3.1 template<class __Data > template<class __MData , typename std::enable_if<(!std::is_same< __MData, char >::value), int >::type = 0> const std::string matlab::StdMat< __Data >::dataToString ( ) const [inline]

Convert our data to a formatted string; specialization enabled if __Data != char.

Print out all stored values, showing row vectors and column vectors

Definition at line 478 of file mat.h.

10.54.3.2 template<class __Data > template<class __MData , typename std::enable_if<(std::is_same< __MData, char >::value), int >::type = 0> const std::string matlab::StdMat< __Data >::dataToString ( ) const [inline]

Convert our data to a formatted string; specialization enabled if __Data == char.

Since we're storing an array of characters, it's possible to compile them directly into an std::string

Definition at line 501 of file mat.h.

10.54.3.3 template<class __Data > const std::string matlab::StdMat< __Data >::dataToString ( ) const [inline]

Convert our data to a formatted string.

In order to use std::enable_if in a class, we need to use a dependent type in the template declaration even if we're enabling based on a class template value

Definition at line 508 of file mat.h.

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

- matlab/include/matlab/mat.h

10.55 matlab::StdMatTypes< __Data > Struct Template Reference

Typedefs for StdMat.

`#include <mat.h>`

Public Types

- typedef std::vector< __Data > **_RowVec**
- typedef _RowVec **_ColVec**
- typedef std::vector< __Data > **_Mat**

10.55.1 Detailed Description

template<class __Data> struct matlab::StdMatTypes< __Data >

Typedefs for StdMat.

Definition at line 247 of file mat.h.

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

- matlab/include/matlab/mat.h

Index