libdrdc

0.1-0 Mon Sep 8 13:18:46 MDT 2008

Generated by Doxygen 1.5.6

Mon Sep 8 13:18:53 2008

Contents

Chapter 1

libdrdc reference

The libdrc package contains the libdrdc software standards library including internal nomenclature, definitions, units of measure, coordinate reference frames, and representations for use in autonomous systems research. This library is a configurable, portable C-function wrapped C++ / Object Oriented C library developed to be independent of software middleware, system architecture, processor, or operating system. It is designed to use the automatically-tuned linear algebra suite (ATLAS) and Basic Linear Algebra Suite (BLAS) and port to firmware and software. The library goal is to unify data collection and representation for various microcontrollers and Central Processing Unit (CPU) cores and to provide a common Application Binary Interface (ABI) for research projects at all scales. The library supports multi-platform development and currently works on Windows, Unix, GNU/Linux, and Real-Time Executive for Multiprocessor Systems (RTEMS). This library is made available under LGPL version 2.1 license.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Module Documentation

5.1 Matrix Operations

Functions

- int vectorCompare (const int n, const double *vector1, const double *vector2, const double *fuzzVector)

  Vector Comparison.

- double vectorDot (const int n, const double *vector1, const double *vector2)

  Vector Dot Product.

- double vectorMagnitude (const int n, const double *vector)

  Vector Magnitude.

- void vectorNormalize (const int n, double *vector)

  Unitize Vector.

- void vectorCross (const double *vector1, const double *vector2, double *resultVector)

  Vector Cross Product.

- void vectorScale (const int n, const double alpha, double *vector)

  Vector Scaling.

- void vectorCopy (const int n, const double *srcVector, double *destVector)

  Copy Vector.

- void vectorAxpy (const int n, const double alpha, const double *x, double *y)

  Scalar-Vector Product.

- void vectorZeros (const int n, double *vector)

  Zero Vector.

- void vectorSplit (const int n, const int m, const double *orgVector, double *vector1, double *vector2)

  Split Vector.

- void vectorJoin (const int n, const int m, const double *vector1, const double *vector2, double *resultVector)

  Join Vector.

- void matrixMvp (const char transA, const double alpha, const double *a, const double *x, const double beta, double *y)

  4x4 Matrix-Vector Product

- void matrixMmp (const char transA, const char transB, const double alpha, const double *a, const double *b, const double beta, double *c)

  4x4 Matrix-Matrix Product

- double matrixDeterminant (const double *matrix)

  4x4 Matrix Determint

- void matrixTranspose (double *matrix)

  4x4 Matrix Transpose

- void matrixIdentity (double *matrix)

  4x4 Identity Matrix.

5.1.1 Function Documentation

5.1.1.1 double matrixDeterminant (const double * *matrix*)

4x4 Matrix Determint

This function computes the determint of a 4x4 matrix.

Parameters:

matrix Input. 4x4 matrix.

Returns:

The determint of the matrix.

5.1.1.2 void matrixIdentity (double * *matrix*)

4x4 Identity Matrix.

This function sets an 4x4 identity matrix.

Parameters:

matrix Input/Output. 4x4 matrix.

5.1.1.3 void matrixMmp (const char *transA*, const char *transB*, const double *alpha*, const double * *a*, const double * *b*, const double *beta*, double * *c*)

4x4 Matrix-Matrix Product

This function performs a 4x4 matrix-matrix operation defined as c = alpha∗op(a)∗op(b) + beta∗c, where op(x) is one of op(x) = x or op(x) = x'. x' is the transpose of x.

Parameters:

transA Input. Specifies the operation to be performed on A. N or n - No transpose. T or t - Transpose.

transB Input. Specifies the operation to be performed on B. N or n - No transpose. T or t - Transpose.

alpha Input. Scalar.

a Input. 4x4 matrix.

b Input. 4x4 matrix.

beta Input. Scalar.

c Output. 4x4 matrix.

Returns:

5.1.1.4 void matrixMvp (const char *transA*, const double *alpha*, const double * *a*, const double * *x*, const double *beta*, double * *y*)

4x4 Matrix-Vector Product

This function performs a 4x4 matrix-vector operation defined as y = alpha∗a∗x + beta∗y or y = alpha∗a'∗x + beta∗y. a' is the transpose of a.

Parameters:

transA Input. Specifies the operation to be performed. N or n - No transpose. T or t - Transpose.

alpha Input. Scalar.

a Input. 4x4 matrix.

x Input. 4-vector.

beta Input. Scalar.

y Output. 4-vector.

5.1.1.5 void matrixTranspose (double * *matrix*)

4x4 Matrix Transpose

This function transposes a matrix.

Parameters:

matrix Input/Output. 4x4 matrix.

5.1.1.6 void vectorAxpy (const int *n*, const double *alpha*, const double * *x*, double * *y*)

Scalar-Vector Product.

This function performs a vector-vector operation defined as y = alpha*x + y.

Parameters:

n Input. Number of vector elements.

alpha Input. Scalar

x Input. n-vector.

y Output. n-vector.

5.1.1.7 int vectorCompare (const int *n*, const double * *vector1*, const double * *vector2*, const double * *fuzzVector*)

Vector Comparison.

Parameters:

n Input. Number of vector elements.

vector1 Input. n-vector 1.

vector2 Input. n-vector 2.

fuzzVector Input. n-vector. How close the two vectors are considered to be equal.

Returns:

It returns zero if not equal. Otherwise, it returns one if two vectors are equal.

5.1.1.8 void vectorCopy (const int *n*, const double * *srcVector*, double * *destVector*)

Copy Vector.

This function performs a vector-vector operation defined as Y = X.

Parameters:

n Input. Number of vector elements.

srcVector Input. n-vector.

destVector Output. n-vector.

5.1.1.9 void vectorCross (const double * *vector1*, const double * *vector2*, double * *resultVector*)

Vector Cross Product.

Parameters:

vector1 Input. 3-vector 1.

vector2 Input. 3-vector 2.

resultVector Output. 3-vector. The cross product result.

5.1.1.10 double vectorDot (const int *n*, const double * *vector1*, const double * *vector2*)

Vector Dot Product.

Parameters:

n Input. Number of vector elements.

vector1 Input. n-vector 1.

vector2 Input. n-vector 2.

Returns:

It returns the dot prorduct result of two vectors.

5.1.1.11 void vectorJoin (const int *n*, const int *m*, const double * *vector1*, const double * *vector2*, double * *resultVector*)

Join Vector.

This function joins a n-vector vector1 and a m-vector vector2 into a n+m vector resultVector.

Parameters:

n Input. Number of vector1 vector elements.

m Input. Number of vector2 vector elements.

vector1 Input. n-vector.

vector2 Input. m-vector.

resultVector Output. n+m vector.

5.1.1.12 double vectorMagnitude (const int *n*, const double * *vector*)

Vector Magnitude.

Parameters:

n Input. Number of vector elements.

vector Input. n-vector.

Returns:

It returns the magnitude of the vector.

5.1.1.13 void vectorNormalize (const int *n*, double * *vector*)

Unitize Vector.

Parameters:

n Input. Number of vector elements.

vector Input. n-vector.

Returns:

It returns zero if succesful. Otherwise, it returns one if a divied-by-zero error occurs.

5.1.1.14 void vectorScale (const int *n*, const double *alpha*, double * *vector*)

Vector Scaling.

This function performs a vector-vector operation defined as X = alpha*X.

Parameters:

n Input. Number of vector elements.

alpha Input. Scalar.

vector Input. n-vector.

Returns:

5.1.1.15 void vectorSplit (const int *n*, const int *m*, const double * *orgVector*, double * *vector1*, double * *vector2*)

Split Vector.

This function splits a n+m vector orgVector into a n-vector vector1 and a m-vector vector2.

Parameters:

n Input. Number of vector1 vector elements.

m Input. Number of vector2 vector elements.

orgVector Input. n+m vector.

vector1 Output. n-vector.

vector2 Output. m-vector.

5.1.1.16 void vectorZeros (const int *n*, double * *vector*)

Zero Vector.

This function sets all the elements of a vector to zeros.

Parameters:

n Input. Number of vector elements.

vector Output. n-vector.

5.2 Coordinate

Functions

- int LatLon_to_UTM (const double a, const double f, const double Lat, const double Long, double *UTMNorthing, double *UTMEasting, unsigned char *UTMZoneNum, char *UTMZoneCh)

  Lat/Lon to UTM Northing/Easting Conversion.

- int UTM_to_LatLon (const double a, const double f, const double UTMNorthing, const double UTMEasting, const unsigned char UTMZoneNum, const char UTMZoneCh, double *Lat, double *Long)

  UTM to Lat/Lon Conversion

- int WUTMPosn_to_WPosn (const double a, const double f, const unsigned char zoneNumber, const char zoneLetter, const double *worldUTMPosition, double *worldPosition)

  UTM Position to World Position

- int WPosn_to_WUTMPosn (const double a, const double f, const double *worldPosition, unsigned char *zoneNumber, char *zoneLetter, double *worldUTMPosition)

  World Position to UTM Position.

- int cartesian_to_spherical (const double *cartesian, double *spherical)

  Cartesian to Spherical Conversion

- int spherical_to_cartesian (const double *spherical, double *cartesian)

  Spherical to Cartesian Conversion

- int WPosn_to_lPosn (const double a, const double f, const double falseElevation, const double *worldPosition, double *cartesian)

  World Position to Cartesian Conversion.

- int lPosn_to_WPosn (const double a, const double f, const double falseElevation, const double *cartesian, double *worldPosition)

  Cartesian to World Position Conversion

- double DMS_to_DecimalDegree (const double degrees, const double minutes, const double seconds)

  Degree-Minute-Second to Decimal Degree Conversion.

5.2.1 Function Documentation

5.2.1.1 int cartesian_to_spherical (const double * *cartesian*, double * *spherical*)

Cartesian to Spherical Conversion

This function converts ECEF XYZ coordinate spherical coordinate.

Parameters:

cartesian Cartesian vector - [x, y, z].

spherical Spherical vector - [Azimuth, Zenith, Radius].

Returns:

It returns zero if succesful. Otherwise, it returns one if errors occur.

5.2.1.2 double DMS_to_DecimalDegree (const double *degrees*, const double *minutes*, const double *seconds*)

Degree-Minute-Second to Decimal Degree Conversion.

Parameters:

degrees Degree value.

minutes Minute value.

seconds Second value.

Returns:

Decimal degree value.

5.2.1.3 int LatLon_to_UTM (const double *a*, const double *f*, const double *Lat*, const double *Long*, double * *UTMNorthing*, double * *UTMEasting*, unsigned char * *UTMZoneNum*, char * *UTMZoneCh*)

Lat/Lon to UTM Northing/Easting Conversion.

This function converts the specified lat/lon coordinate to UTM coordinate.

Parameters:

a Input. Ellipsoid semi-major axis, in meters. (For WGS84 datum, use 6378137.0).

f Input. Ellipsoid flattening. (For WGS84 datum, use 1 / 298.257223563).

Lat Input. Latitude in radians.

Long Input. Longtitude in radians.

UTMNorthing Output. UTM Northing in meters.

UTMEasting Output. UTM Easting in meters.

UTMZoneNum Output. UTM zone number.

UTMZoneCh Output. UTM zone character.

Returns:

It returns zero if succesful. Otherwise, it returns one if errors occur.

5.2.1.4 int lPosn_to_WPosn (const double *a*, const double *f*, const double *falseElevation*, const double * *cartesian*, double * *worldPosition*)

Cartesian to World Position Conversion

This function converts ECEF XYZ coordinate to global geographical coordinate.

Parameters:

a Input. Ellipsoid semi-major axis, in meters. (For WGS84 datum, use 6378137.0).

f Input. Ellipsoid flattening. (For WGS84 datum, use 1 / 298.257223563).

falseElevation False elevation.

cartesian Cartesian vector - [x, y, z].

worldPosition World position vector - [longtitude, latitude, elevation].

Returns:

It returns zero if succesful. Otherwise, it returns one if errors occur.

5.2.1.5 int spherical_to_cartesian (const double * *spherical*, double * *cartesian*)

Spherical to Cartesian Conversion

This function converts spherical coordinate to ECEF XYZ coordinate.

Parameters:

cartesian Cartesian vector -[x, y, z].

spherical Spherical vector - [Azimuth, Zenith, Radius].

Returns:

It returns zero if succesful. Otherwise, it returns one if errors occur.

5.2.1.6 int UTM_to_LatLon (const double *a*, const double *f*, const double *UTMNorthing*, const double *UTMEasting*, const unsigned char *UTMZoneNum*, const char *UTMZoneCh*, double * *Lat*, double * *Long*)

UTM to Lat/Lon Conversion

This function converts the specified UTM coordinate lat/lon coordinate.

Parameters:

a Input. Ellipsoid semi-major axis, in meters. (For WGS84 datum, use 6378137.0).

f Input. Ellipsoid flattening. (For WGS84 datum, use 1 / 298.257223563).

UTMNorthing Input. UTM Northing in meters.

UTMEasting Input. UTM Easting in meters.

UTMZoneNum Input. UTM zone number.

UTMZoneCh Input. UTM zone character.

Lat Output. Latitude in radians.

Long Output. Longtitude in radians.

Returns:

5.2.1.7 int WPosn_to_lPosn (const double *a*, const double *f*, const double *falseElevation*, const double * *worldPosition*, double * *cartesian*)

World Position to Cartesian Conversion.

This function converts global geographical coordinate to ECEF XYZ coordinate.

Parameters:

a Input. Ellipsoid semi-major axis, in meters. (For WGS84 datum, use 6378137.0).

f Input. Ellipsoid flattening. (For WGS84 datum, use 1 / 298.257223563).

falseElevation False elevation.

worldPosition World position vector - [longtitude, latitude, elevation].

cartesian Cartesian vector - [x, y, z].

Returns:

It returns zero if succesful. Otherwise, it returns one if errors occur.

5.2.1.8 int WPosn_to_WUTMPosn (const double *a*, const double *f*, const double * *worldPosition*, unsigned char * *zoneNumber*, char * *zoneLetter*, double * *worldUTMPosition*)

World Position to UTM Position.

This function is a wrapper of LatLon_to_UTM.

Parameters:

a Input. Ellipsoid semi-major axis, in meters. (For WGS84 datum, use 6378137.0).

f Input. Ellipsoid flattening. (For WGS84 datum, use 1 / 298.257223563).

worldPosition Input. World position vector - [longtiude, latitude, elevation].

zoneNumber Output. UTM zone number.

zoneLetter Output. UTM zone character.

worldUTMPosition Output. UTM vector - [easting, northing, elevation].

Returns:

It returns zero if succesful. Otherwise, it returns one if errors occur.

5.2.1.9 int WUTMPosn_to_WPosn (const double *a*, const double *f*, const unsigned char *zoneNumber*, const char *zoneLetter*, const double * *worldUTMPosition*, double * *worldPosition*)

UTM Position to World Position

This function is a wrapper of UTM_to_LatLon.

Parameters:

a Input. Ellipsoid semi-major axis, in meters. (For WGS84 datum, use 6378137.0).

f Input. Ellipsoid flattening. (For WGS84 datum, use 1 / 298.257223563).

zoneNumber Input. UTM zone number.

zoneLetter Input. UTM zone character.

worldUTMPosition Input. UTM position vector - [easting, northing, elevation].

worldPosition Output. World position vector- [longtiude, latitude, elevation].

Returns:

It returns zero if succesful. Otherwise, it returns one if errors occur.

5.3 C Exception Handling

Defines

- #define init_exception_context(ec) ((void)((ec) → penv = 0))
- #define Try
- #define exception__catch(action)
- #define Catch(e) exception__catch(((e) = the_exception_context → v.etmp, 0))
- #define Throw
- #define Catch_anonymous exception__catch(0)
- #define define_exception_type(etype)

5.3.1 Detailed Description

=== cexcept.h 2.0.0 (2001-Jul-12-Thu) Adam M. Costello <amc@cs.berkeley.edu> An interface for exception-handling in ANSI C (C89 and subsequent ISO standards), developed jointly with Cosmin Truta <cosmin@cs.toronto.edu>.

Only user-defined exceptions are supported, not "real" exceptions like division by zero or memory segmentation violations.

If this interface is used by multiple .c files, they shouldn't include this header file directly. Instead, create a wrapper header file that includes this header file and then invokes the define_exception_type macro (see below), and let your .c files include that header file.

The interface consists of one type, one well-known name, and six macros /**

5.3.2 Define Documentation

5.3.2.1 #define Catch(e) exception__catch(((e) = the_exception_context → v.etmp, 0))

5.3.2.2 #define Catch_anonymous exception__catch(0)

Try statement Catch_anonymous statement

When the value of the exception is not needed, a Try/Catch statement can use Catch_anonymous instead of Catch (expression).

5.3.2.3 #define define_exception_type(etype)

Value:

```
struct exception_context { \
  jmp_buf *penv; \
  int caught; \
  volatile struct { etype etmp; } v; \
}
```

define_exception_type(type_name);

This macro is used like an external declaration. It specifies the type of object that gets copied from the exception thrower to the exception catcher. The type_name can be any type that can be assigned to, that is, a non-constant arithmetic type, struct, union, or pointer. Examples:

define_exception_type(int);

enum exception { out_of_memory, bad_arguments, disk_full }; define_exception_type(enum exception);

struct exception { int code; const char *msg; }; define_exception_type(struct exception);

Because throwing an exception causes the object to be copied (not just once, but twice), programmers may wish to consider size when choosing the exception type.

etmp must be volatile because the application might use automatic storage for the_exception_context, and etmp is modified between the calls to setjmp() and longjmp(). A wrapper struct is used to avoid warnings about a duplicate volatile qualifier in case etype already includes it.

5.3.2.4 #define exception__catch(action)

Value:

```
else { } \
      the_exception_context->caught = 0; \
    } \
    else { \
      the_exception_context->caught = 1; \
    } \
    the_exception_context->penv = exception__prev; \
  } \
  if (!the_exception_context->caught || action) { } \
  else
```

5.3.2.5 #define init_exception_context(ec) ((void)((ec) → penv = 0))

struct exception_context;

This type may be used after the define_exception_type() macro has been invoked. A struct exception_context must be known to both the thrower and the catcher. It is expected that there be one context for each thread that uses exceptions. It would certainly be dangerous for multiple threads to access the same context. One thread can use multiple contexts, but that is likely to be confusing and not typically useful. The application can allocate this structure in any way it pleases–automatic, static, or dynamic. The application programmer should pretend not to know the structure members, which are subject to change.

struct exception_context ∗the_exception_context;

The Try/Catch and Throw statements (described below) implicitly refer to a context, using the name the_exception_context. It is the application's responsibility to make sure that this name yields the address of a mutable (non-constant) struct exception_context wherever those statements are used. Subject to that constraint, the application may declare a variable of this name anywhere it likes (inside a function, in a parameter list, or externally), and may use whatever storage class specifiers (static, extern, etc) or type qualifiers (const, volatile, etc) it likes. Examples:

static struct exception_context const the_exception_context =

{ struct exception_context ∗the_exception_context = bar; ... }

int blah(struct exception_context ∗the_exception_context, ...);

extern struct exception_context the_exception_context[1];

The last example illustrates a trick that avoids creating a pointer object separate from the structure object.

The name could even be a macro, for example:

struct exception_context ec_array[numthreads]; define the_exception_context (ec_array + thread_id)

Be aware that the_exception_context is used several times by the Try/Catch/Throw macros, so it shouldn't be expensive or have side effects. The expansion must be a drop-in replacement for an identifier, so it's safest to put parentheses around it.

void init_exception_context(struct exception_context ∗ec);

For context structures allocated statically (by an external definition or using the "static" keyword), the implicit initialization to all zeros is sufficient, but contexts allocated by other means must be initialized using this macro before they are used by a Try/Catch statement. It does no harm to initialize a context more than once (by using this macro on a statically allocated context, or using this macro twice on the same context), but a context must not be re-initialized after it has been used by a Try/Catch statement.

5.3.2.6 #define Throw

Value:

```
for (;; longjmp(*the_exception_context->penv, 1)) \
    the_exception_context->v.etmp =
```

Throw expression;

A Throw statement is very much like a return statement, except that the expression is required. Whereas return jumps back to the place where the current function was called, Throw jumps back to the Catch clause of the innermost enclosing Try clause. The expression must be compatible with the type passed to define_exception_type(). The exception must be caught, otherwise the program may crash.

Slight limitation: If the expression is a comma-expression it must be enclosed in parentheses.

5.3.2.7 #define Try

Value:

```
{ \
    jmp_buf *exception__prev, exception__env; \
    exception__prev = the_exception_context->penv; \
    the_exception_context->penv = &exception__env; \
    if (setjmp(exception__env) == 0) { \
      if (&exception__prev)
```

Try statement Catch (expression) statement

The Try/Catch/Throw macros are capitalized in order to avoid confusion with the C++ keywords, which have subtly different semantics.

A Try/Catch statement has a syntax similar to an if/else statement, except that the parenthesized expression goes after the second keyword rather than the first. As with if/else, there are two clauses, each of which may be a simple statement ending with a semicolon or a brace-enclosed compound statement. But whereas the else clause is optional, the Catch clause is required. The expression must be a modifiable lvalue (something capable of being assigned to) of the same type (disregarding type qualifiers) that was passed to define_exception_type().

If a Throw that uses the same exception context as the Try/Catch is executed within the Try clause (typically within a function called by the Try clause), and the exception is not caught by a nested Try/Catch statement, then a copy of the exception will be assigned to the expression, and control will jump to the Catch clause. If no such Throw is executed, then the assignment is not performed, and the Catch clause is not executed.

The expression is not evaluated unless and until the exception is caught, which is significant if it has side effects, for example:

Try foo(); Catch (p[++i].e) { ... }

IMPORTANT: Jumping into or out of a Try clause (for example via return, break, continue, goto, longjmp) is forbidden–the compiler will not complain, but bad things will happen at run-time. Jumping into or out of a Catch clause is okay, and so is jumping around inside a Try clause. In many cases where one is tempted to return from a Try clause, it will suffice to use Throw, and then return from the Catch clause. Another option is to set a flag variable and use goto to jump to the end of the Try clause, then check the flag after the Try/Catch statement.

IMPORTANT: The values of any non-volatile automatic variables changed within the Try clause are undefined after an exception is caught. Therefore, variables modified inside the Try block whose values are needed later outside the Try block must either use static storage or be declared with the "volatile" type qualifier.

Try ends with if(), and Catch begins and ends with else. This ensures that the Try/Catch syntax is really the same as the if/else syntax.

We use &exception__prev instead of 1 to appease compilers that warn about constant expressions inside if(). Most compilers should still recognize that &exception__prev is never zero and avoid generating test code.

5.4 Homogeneous

Functions

- int HMatrixSetTranslation (const double *vector, double *hMatrix)
  Set the Translational Component of a Homogeneous Matrix.
- int HMatrixGetTranslation (const double *hMatrix, double *vector)
  Get the Translational Component of a Homogeneous Matrix.
- int RotVectorToHMatrix (const double theta, const double *vector, double *hMatrix)
  Rotation Vector to Homogeneous Matrix.
- int HMatrixToRotVector (const double *hMatrix, double *theta, double *vector)
  Homogeneous Matrix to Rotation Vector.
- int EulerRPYToHMatrix (const double *rpyVector, double *hMatrix)
  Euler Angle RPY to Homogeneous Matrix.
- int HMatrixToEulerRPY (const double *hMatrix, double *rpyVector)
  Homogeneous Matrix to Euler Angle RPY.
- int HMatrixNormalize (double *hMatrix)
  Normalize a Homogeneous Matrix.
- int RotAxisToHMatrix (Axis_t axis, const double theta, double *hMatrix)
  Rotation about an Axis to Homogeneous Matrix.
- int HMatrixInverse (double *hMatrix)
  Inverse a Homogeneous Matrix.
- int HMatrixLink (const double linkLength, const double twistAngle, const double jointOffset, const double jointAngle, double *hMatrix)
  Homogeneous Matrix from Manipulator Linkage Specification.

5.4.1 Function Documentation

5.4.1.1 int EulerRPYToHMatrix (const double * *rpyVector*, double * *hMatrix*)

Euler Angle RPY to Homogeneous Matrix.

Parameters:

rpyVector Input. RPY vector - [roll, pitch, yaw].

hMatrix Output. Homogeneous matrix represented by a 16-vector (column major).

5.4.1.2 int HMatrixGetTranslation (const double * *hMatrix*, double * *vector*)

Get the Translational Component of a Homogeneous Matrix.

Parameters:

hMatrix Input. Homogeneous matrix represented by a 16-vector (column major).

vector Output. Cartesian vector - [x, y, z].

5.4.1.3 int HMatrixInverse (double * *hMatrix*)

Inverse a Homogeneous Matrix.

Parameters:

hMatrix Input/Output. Homogeneous matrix represented by a 16-vector (column major).

5.4.1.4 int HMatrixLink (const double *linkLength*, const double *twistAngle*, const double *jointOffset*, const double *jointAngle*, double * *hMatrix*)

Homogeneous Matrix from Manipulator Linkage Specification.

Parameters:

linkLength Link length (a).

twistAngle Twist angle (alpha).

jointOffset Joint offset (d).

jointAngle Joint angle (theta).

hMatrix Result homogeneous matrix.

5.4.1.5 int HMatrixNormalize (double * *hMatrix*)

Normalize a Homogeneous Matrix.

Parameters:

hMatrix Input/Output. Homogeneous matrix represented by a 16-vector (column major).

5.4.1.6 int HMatrixSetTranslation (const double * *vector*, double * *hMatrix*)

Set the Translational Component of a Homogeneous Matrix.

Parameters:

vector Input. Cartesian vector - [x, y, z].

hMatrix Input/Output. Homogeneous matrix represented by a 16-vector (column major).

5.4.1.7 int HMatrixToEulerRPY (const double * *hMatrix*, double * *rpyVector*)

Homogeneous Matrix to Euler Angle RPY.

Parameters:

hMatrix Input. Homogeneous matrix represented by a 16-vector (column major).

rpyVector Output. RPY vector - [roll, pitch, yaw].

5.4.1.8 int HMatrixToRotVector (const double * *hMatrix*, double * *theta*, double * *vector*)

Homogeneous Matrix to Rotation Vector.

Parameters:

hMatrix Input. Homogeneous matrix represented by a 16-vector (column major).

theta Output. Rotation angle in radians

vector Output. Unit vector represented by a cartesian vector - [x, y, z].

Returns:

It returns zero if succesful.

5.4.1.9 int RotAxisToHMatrix (Axis_t *axis*, const double *theta*, double * *hMatrix*)

Rotation about an Axis to Homogeneous Matrix.

Computes a homogeneous transformation representing a rotation of Theta radians about the axis.

Parameters:

axis Input. Enumeration type, AXIS_X, AXIS_Y, or AXIS_Z.

theta Input. Rotation angle in radians.

hMatrix Output. Homogeneous matrix represented by a 16-vector (column major).

Returns:

It returns zero if succesful. Otherwise, it returns one if a undefined enumeration type error occurs.

5.4.1.10 int RotVectorToHMatrix (const double *theta*, const double * *vector*, double * *hMatrix*)

Rotation Vector to Homogeneous Matrix.

Parameters:

theta Input. Rotation angle in radians.

vector Input. Unit vector represented by a cartesian vector - [x, y, z].

hMatrix Output. Homogeneous matrix represented by a 16-vector (column major).

5.5 Quaternion

Functions

- int EulerRPYToQuat (const double *rpyVector, double *quaternion)

  RPY to Quaternion

- int QuatToEulerRPY (const double *quaternion, double *rpyVector)

  Quaternion to RPY

- int QuatQuatMult (double *quaternion1, double *quaternion2, double *resultQuaternion)

  Quaternion Multiplication

- int QuatQuatDiv (double *quaternion1, double *quaternion2, double *resultQuaternion)

  Quaternion Division

- int QuatInverse (double *quaternion)

  Quaternion Inversion

- int QuatConjugate (double *quaternion)

  Conjugate Quaternion.

- double QuatMagnitude (const double *quaternion)

  Quaterion Magnitude.

- int QuatVectorMult (double *quaternion, double *vector, double *resultVector)

  Quaternion and Vector Product

- int HMatrixToQuat (const double *hMatrix, double *quaternion)

  Quaternion to Homogeneous Transformation

- int QuatToHMatrix (const double *quaternion, double *hMatrix)

  Homogeneous Transformation to Quaternion.

- int QuatNormalize (double *quaternion)

  Unitize a Quaternion.

- int RotAxisToQuat (const Axis_t axis, const double theta, double *quaternion)

  Rotation about an Axis to Quaternion.

- int QuatToRotVector (const double *quaternion, double *theta, double *vector)

  Quaternion to Rotation Vector

- int RotVectorToQuat (const double theta, const double *vector, double *quaternion)

  Rotation Vector to Quaternion.

5.5.1 Function Documentation

5.5.1.1 int EulerRPYToQuat (const double * *rpyVector*, double * *quaternion*)

RPY to Quaternion

These functions handle quaternion representations and operations, exclusive of the type of orientation used. The type of operations contained herein are:

- Conversion to / from Euler Angle RPY (rotation sequence ZYX)
- Quaternion conjugates
- Quaternion multiplication
- Quaternion inverse
- Quaternion division
- Conversion to / from angle axis / rotation vector representation
- Cinversion to / from rotation matrix

The quaternion representation used here follows this convention:

$\vec{q} \equiv (q_s, \langle q_x, q_y, q_z \rangle)$

$\vec{q} \equiv (q_s, \vec{\mathbf{q}}) \equiv [q_s, < q_x, q_y, q_z >]^T$

If you need help with doxygen TeX formulas, refer to http://en.wikipedia.org/wiki/Help:Formula for a helpful page. Converts the rotation described by the three Euler angles (yaw, pitch, row) into the four-element quaternion vector q

Parameters:

rpyVector Input. RPY vector - [roll, pitch, yaw].

quaternion Output. Quaternion vector - $[q_s, < q_x, q_y, q_z >]^T$.

5.5.1.2 int HMatrixToQuat (const double * *hMatrix*, double * *quaternion*)

Quaternion to Homogeneous Transformation

Converts a quaternion to a rotation matrix (inside homogeneous matrix).

Parameters:

hMatrix Input. Homogeneous matrix described by a 16-vector.

quaternion Output. Quaternion vector - $[q_s, < q_x, q_y, q_z >]^T$.

5.5.1.3 int QuatConjugate (double * *quaternion*)

Conjugate Quaternion.

Computes the conjugate of a Quaternion.

Parameters:

quaternion Input/Output. Input is overwritten by the conjugate Quaternion - $[q_s, < q_x, q_y, q_z >]^T$.

5.5.1.4 int QuatInverse (double * *quaternion*)

Quaternion Inversion

Computes the inversion of a Quaternion.

Parameters:

quaternion Input/Output. Input is overwritten by the inversed Quaternion - $[q_s, < q_x, q_y, q_z >]^T$.

Returns:

It returns zero if succesful. Otherwise, it returns one if a divied-by-zero error occurs.

5.5.1.5 double QuatMagnitude (const double * *quaternion*)

Quaterion Magnitude.

Computes the magnitude of a quaternion.

Parameters:

quaternion Intput. Quaternion vector - $[q_s, < q_x, q_y, q_z >]^T$.

Returns:

It returns quaterion magnitude.

5.5.1.6 int QuatNormalize (double * *quaternion*)

Unitize a Quaternion.

Computes normalized quaternion.

Parameters:

quaternion Input/Output. Input is overwritten by the unitized Quaternion - $[q_s, < q_x, q_y, q_z >]^T$.

Returns:

It returns zero if succesful. Otherwise, it returns one if a divied-by-zero error occurs.

5.5.1.7 int QuatQuatDiv (double * *quaternion1*, double * *quaternion2*, double * *resultQuaternion*)

Quaternion Division

Computes Quaternion-Quaternion division.

Parameters:

quaternion1 Input. Quaternion vector1 - $[q_{s_1}, < q_{x_1}, q_{y_1}, q_{z_1} >]^T$.

quaternion2 Input. Quaternion vector2 - $[q_{s_2}, < q_{x_2}, q_{y_2}, q_{z_2} >]^T$.

resultQuaternion Output. Quaternion Product result - $[q_{s_r}, < q_{x_r}, q_{y_r}, q_{z_r} >]^T$.

Returns:

It returns zero if succesful. Otherwise, it returns one if a divied-by-zero error occurs.

5.5.1.8 int QuatQuatMult (double * *quaternion1*, double * *quaternion2*, double * *resultQuaternion*)

Quaternion Multiplication

Computes Quaternion-Quaternion product.

Parameters:

quaternion1 Input. Quaternion vector1 - $[q_{s_1}, < q_{x_1}, q_{y_1}, q_{z_1} >]^T$.

quaternion2 Input. Quaternion vector2 - $[q_{s_2}, < q_{x_2}, q_{y_2}, q_{z_2} >]^T$.

resultQuaternion Output. Quaternion Product result - $[q_{s_r}, < q_{x_r}, q_{y_r}, q_{z_r} >]^T$.

5.5.1.9 int QuatToEulerRPY (const double * *quaternion*, double * *rpyVector*)

Quaternion to RPY

Converts the four-element unit quaternion $[q_s, < q_x, q_y, q_z >]^T$ into the equivalent thre e Euler angle rotations (yaw, pitch, row).

Parameters:

quaternion Intput. Quaternion vector - $[q_s, < q_x, q_y, q_z >]^T$.

rpyVector Output. RPY vector - [roll, pitch, yaw].

5.5.1.10 int QuatToHMatrix (const double * *quaternion*, double * *hMatrix*)

Homogeneous Transformation to Quaternion.

Converts a rotation matrix (inside homogeneous matrix) to a quaternion.

Parameters:

quaternion Intput. Quaternion vector - $[q_s, < q_x, q_y, q_z >]^T$.

hMatrix Output. Homogeneous matrix described by a 16-vector.

Returns:

5.5.1.11 int QuatToRotVector (const double * *quaternion*, double * *theta*, double * *vector*)

Quaternion to Rotation Vector

Computes a rotation vector from a quaternion.

Parameters:

quaternion Input. Quaternion vector - $[q_s, < q_x, q_y, q_z >]^T$.

theta Output. Rotation angle in radians.

vector Output. Cartesian vector - [x, y, z].

Returns:

5.5.1.12 int QuatVectorMult (double * *quaternion*, double * *vector*, double * *resultVector*)

Quaternion and Vector Product

Parameters:

quaternion Intput. Quaternion vector - $[q_s, < q_x, q_y, q_z >]^T$.

vector Input. Cartesian vector - [x, y, z].

resultVector Output. Cartesian vector - [x, y, z].

Returns:

It returns zero if succesful. Otherwise, it returns one if divied-by-zero error occurs.

5.5.1.13 int RotAxisToQuat (const Axis_t *axis*, const double *theta*, double * *quaternion*)

Rotation about an Axis to Quaternion.

Computes a quaternion from a rotation about an cartesian axis.

Parameters:

axis Input. Enumeration type, AXIS_X, AXIS_Y, or AXIS_Z.

theta Input. Rotation angle in radians.

quaternion Output. Quaternion vector - $[q_s, < q_x, q_y, q_z >]^T$.

Returns:

It returns zero if succesful. Otherwise, it returns one if a undefined AXIS type error occurs.

5.5.1.14 int RotVectorToQuat (const double *theta*, const double * *vector*, double * *quaternion*)

Rotation Vector to Quaternion.

Computes a quaternion from a rotation about a vector.

Parameters:

theta Input. Rotation angle in radians.

vector Input. Cartesian vector - [x, y, z].

quaternion Output. Quaternion vector - $[q_s, < q_x, q_y, q_z >]^T$.

5.6 Time Conversions

Functions

- struct timeval timeUtcToGps (const short leapSec, const struct timeval *utcTime)
  UTC time to GPS time.
- struct timeval timeGpsToUtc (const short leapSec, const struct timeval *gpsTime)
  GPS time to UTC time.
- int jausTimeConvert (const struct timeval *timeVal, JausTime_t *time)
  Convert Time to Jaus Timestamp and Datestamp.
- int jausTimeStampPack (JausTime_t *time)
  Pack Jaus Timestamp.
- int jausDateStampPack (JausTime_t *time)
  Pack Jaus Datestamp.
- int jausTimeToString (JausTime_t *time, char *buffer)
  Convert Jaus Time Structure to String.
- int jausTimeStampUnpack (JausTime_t *time)
  Unpack Jaus Timestamp.
- int jausDateStampUnpack (JausTime_t *time)
  Unpack Jaus Datestamp.

5.6.1 Function Documentation

5.6.1.1 int jausDateStampPack (JausTime_t * *time*)

Pack Jaus Datestamp.

Parameters:

time Input/Output. Jaus time structure.

Returns:

It returns zero if succesful. Otherwise, it returns one.

5.6.1.2 int jausDateStampUnpack (JausTime_t * *time*)

Unpack Jaus Datestamp.

Parameters:

time Input/Output. Jaus time structure.

Returns:

It returns zero if succesful. Otherwise, it returns one.

5.6.1.3 int jausTimeConvert (const struct timeval * *timeVal*, JausTime_t * *time*)

Convert Time to Jaus Timestamp and Datestamp.

Parameters:

timeVal Input. Unix time value.
time Output. Jaus time structure.

Returns:

It returns zero if succesful. Otherwise, it returns one.

5.6.1.4 int jausTimeStampPack (JausTime_t * *time*)

Pack Jaus Timestamp.

Parameters:

time Input/Output. Jaus time structure.

Returns:

It returns zero if succesful. Otherwise, it returns one.

5.6.1.5 int jausTimeStampUnpack (JausTime_t * *time*)

Unpack Jaus Timestamp.

Parameters:

time Input/Output. Jaus time structure.

Returns:

It returns zero if succesful. Otherwise, it returns one.

5.6.1.6 int jausTimeToString (JausTime_t * *time*, char * *buffer*)

Convert Jaus Time Structure to String.

Parameters:

time time Input. Jaus time structure.

buffer Output. String of Timestamp and datestamp.

Returns:

It returns zero if succesful. Otherwise, it returns one.

5.6.1.7 struct timeval timeGpsToUtc (const short *leapSec*, const struct timeval * *gpsTime*) [read]

GPS time to UTC time.

Parameters:

leapSec Input. Leap second adjustment.

gpsTime Input. GPS time.

Returns:

It returns UTC time.

5.6.1.8 struct timeval timeUtcToGps (const short *leapSec*, const struct timeval * *utcTime*) [read]

UTC time to GPS time.

Parameters:

leapSec Input. Leap second adjustment.

utcTime Input. UTC time.

Returns:

It returns GPS time.

5.7 Unit Conversions

Functions

- double unitToSiVolume (const char *unit)

  Converts the volume units to the SI unit - m^3.

- double unitToSiDistance (const char *unit)

  Converts the distance units to the SI unit - m.

- double unitToSiMass (const char *unit)

  Converts the mass units to the SI unit - kg .

- double unitToSiVelocity (const char *unit)

  Converts the velocity units to the SI unit - m/s.

- double unitToSiArea (const char *unit)

  Converts the area units to the SI unit - m^2.

- double unitConvertVolume (const char *srcUnit, const char *destUnit)

  Converts between volume units.

- double unitConvertDistance (const char *srcUnit, const char *destUnit)

  Converts between distance units.

- double unitConvertMass (const char *srcUnit, const char *destUnit)

  Converts between mass units.

- double unitConvertVelocity (const char *srcUnit, const char *destUnit)

  Converts between velocity units.

- double unitConvertArea (const char *srcUnit, const char *destUnit)

  Converts between area units.

5.7.1 Function Documentation

5.7.1.1 double unitConvertArea (const char * *srcUnit*, const char * *destUnit*)

Converts between area units.

Parameters:

srcUnit Input. Source unit string.

destUnit Input. Destination unit string

Returns:

It returns the conversion multiplication factor needed for converting the source units to the destination units.

5.7.1.2 double unitConvertDistance (const char * *srcUnit*, const char * *destUnit*)

Converts between distance units.

Parameters:

srcUnit Input. Source unit string

destUnit Input. Destination unit string

Returns:

It returns the conversion multiplication factor needed for converting the source units to the destination units.

5.7.1.3 double unitConvertMass (const char * *srcUnit*, const char * *destUnit*)

Converts between mass units.

Parameters:

srcUnit

destUnit Input. Destination unit string

Returns:

It returns the conversion multiplication factor needed for converting the source units to the destination units.

5.7.1.4 double unitConvertVelocity (const char * *srcUnit*, const char * *destUnit*)

Converts between velocity units.

Parameters:

srcUnit Input. Source unit string.

destUnit Input. Destination unit string

Returns:

It returns the conversion multiplication factor needed for converting the source units to the destination units.

5.7.1.5 double unitConvertVolume (const char * *srcUnit*, const char * *destUnit*)

Converts between volume units.

Parameters:

srcUnit Input. Source unit string.

destUnit Input. Destination unit string.

Returns:

It returns the conversion multiplication factor needed for converting the source units to the destination units.

5.7.1.6 double unitToSiArea (const char * *unit*)

Converts the area units to the SI unit - m^2.

Units:

- cm^2
- mm^2
- km^2
- ft^2
- yard^2
- mile^2

Parameters:

unit Input. Unit string.

Returns:

It returns the conversion multiplication factor needed for converting the input units to the SI unit.

5.7.1.7 double unitToSiDistance (const char * *unit*)

Converts the distance units to the SI unit - m.

Units:

- mm
- cm
- km
- inch
- foot
- yard
- mile

Parameters:

unit Input. Unit string.

Returns:

It returns the conversion multiplication factor needed for converting the input units to the SI unit.

5.7.1.8 double unitToSiMass (const char * *unit*)

Converts the mass units to the SI unit - kg .

Units:

- g
- lbs
- uston
- brton

Parameters:

unit Input. Unit string.

Returns:

It returns the conversion multiplication factor needed for converting the input units to the SI unit.

5.7.1.9 double unitToSiVelocity (const char * *unit*)

Converts the velocity units to the SI unit - m/s.

Units:

- mm/s
- cm/s
- km/s
- kph
- mph

Parameters:

unit Input. Unit string.

Returns:

It returns the conversion multiplication factor needed for converting the input units to the SI unit.

5.7.1.10 double unitToSiVolume (const char * *unit*)

Converts the volume units to the SI unit - m^3.

Units:

- usgallon
- brgallon
- ml
- cm^3
- mm^3
- liter

Parameters:

unit Input. Unit string.

Returns:

It returns the conversion multiplication factor needed for converting the input units to the SI unit.

5.8 Data Format Conversions

Functions

- ieeeFpd32 ieeeFpd32FromFloat (const float value)

  Converts single to float point data represented by 32-bit unsigned integer format.

- ieeeFpd64 ieeeFpd64FromDouble (const double value)

  Converts double to float point data represented by 64-bit unsigned integer format.

- float ieeeFpd32ToFloat (const ieeeFpd32 input)

  Converts float point data represented by 32-bit unsigned integer format to single.

- double ieeeFpd64ToDouble (const ieeeFpd64 input)

  Converts float point data represented by 64-bit unsigned integer format to double.

- uint8_t jausByteFromDouble (double value, const double min, const double max)

  Double to Jaus Byte Conversion.

- double jausByteToDouble (const uint8_t value, const double min, const double max)

  Jaus Byte to Double Conversion.

- double jausIntegerToDouble (const sint32_t value, const double min, const double max)

  Jaus Integer to Double Type Conversion.

- sint32_t jausIntegerFromDouble (double value, const double min, const double max)

  Double to Jaus Integer Type Conversion.

- double jausLongToDouble (const sint64_t value, const double min, const double max)

  Jaus Long Type to Double Conversion.

- sint64_t jausLongFromDouble (double value, const double min, const double max)

  Double to Jaus Long Type Conversion.

- double jausShortToDouble (const sint16_t value, const double min, const double max)

  Jaus Short Type to Double Conversion.

- sint16_t jausShortFromDouble (double value, const double min, const double max)

  Double to Jaus Short Type Conversion.

- double jausUnsignedIntegerToDouble (const uint32_t value, const double min, const double max)

  Jaus Unsigned Integer Type to Double Conversion.

- uint32_t jausUnsignedIntegerFromDouble (double value, const double min, const double max)

  Double to Jaus Unsigned Integer Type Conversion.

- double jausUnsignedLongToDouble (const uint64_t value, const double min, const double max)

  Jaus Unsigned Long Type to Double Conversion.

- uint64_t jausUnsignedLongFromDouble (double value, const double min, const double max)

  Double to Jaus Unsigned Long Type Conversion.

- double jausUnsignedShortToDouble (const uint16_t value, const double min, const double max)

  Jaus Unsigned Short Type to Double Conversion.

- uint16_t jausUnsignedShortFromDouble (double value, const double min, const double max)

  Double to Jaus Unsigned Short Type Conversion.

5.8.1 Function Documentation

5.8.1.1 ieeeFpd32 ieeeFpd32FromFloat (const float *value*)

Converts single to float point data represented by 32-bit unsigned integer format.

Parameters:

value Input. Single float point data.

Returns:

Converted float point data represented by uint32 data type.

5.8.1.2 float ieeeFpd32ToFloat (const ieeeFpd32 *input*)

Converts float point data represented by 32-bit unsigned integer format to single.

Parameters:

input Input. Float point data represented by uint32 data type.

Returns:

Converted single float point data.

5.8.1.3 ieeeFpd64 ieeeFpd64FromDouble (const double *value*)

Converts double to float point data represented by 64-bit unsigned integer format.

Parameters:

value Input. Double float point data array.

Returns:

Converted float point data represented by uint64 data type.

5.8.1.4 double ieeeFpd64ToDouble (const ieeeFpd64 *input*)

Converts float point data represented by 64-bit unsigned integer format to double.

Parameters:

input Input. Float point data array represented by uint64 data type.

Returns:

Converted double float point data.

5.8.1.5 uint8_t jausByteFromDouble (double *value*, const double *min*, const double *max*)

Double to Jaus Byte Conversion.

Parameters:

value Data to be converted.

min Lower limit.

max Upper limit.

Returns:

It returns unsigned 8-bit Jaus byte.

5.8.1.6 double jausByteToDouble (const uint8_t *value*, const double *min*, const double *max*)

Jaus Byte to Double Conversion.

Parameters:

value Data to be converted.

min Lower limit.

max Upper limit.

Returns:

It returns double float point data.

5.8.1.7 sint32_t jausIntegerFromDouble (double *value*, const double *min*, const double *max*)

Double to Jaus Integer Type Conversion.

Parameters:

value Data to be converted.

min Lower limit.

max Upper limit.

Returns:

It returns signed 32-bit Jaus integer data.

5.8.1.8 double jausIntegerToDouble (const sint32_t *value*, const double *min*, const double *max*)

Jaus Integer to Double Type Conversion.

Parameters:

value Data to be converted.

min Lower limit.

max Upper limit.

Returns:

It returns double float point data.

5.8.1.9 sint64_t jausLongFromDouble (double *value*, const double *min*, const double *max*)

Double to Jaus Long Type Conversion.

Parameters:

value Data to be converted.

min Lower limit.

max Upper limit.

Returns:

It returns signed 64-bit Jaus long integer data.

5.8.1.10 double jausLongToDouble (const sint64_t *value*, const double *min*, const double *max*)

Jaus Long Type to Double Conversion.

Parameters:

value Data to be converted.

min Lower limit.

max Upper limit.

Returns:

It returns double float point data.

5.8.1.11 sint16_t jausShortFromDouble (double *value*, const double *min*, const double *max*)

Double to Jaus Short Type Conversion.

Parameters:

value Data to be converted.

min Lower limit.

max Upper limit.

Returns:

It returns signed 16-bit Jaus short integer data.

5.8.1.12 double jausShortToDouble (const sint16_t *value*, const double *min*, const double *max*)

Jaus Short Type to Double Conversion.

Parameters:

value Data to be converted.

min Lower limit.

max Upper limit.

Returns:

It returns double float point data.

5.8.1.13 uint32_t jausUnsignedIntegerFromDouble (double *value*, const double *min*, const double *max*)

Double to Jaus Unsigned Integer Type Conversion.

Parameters:

value Data to be converted.

min Lower limit.

max Upper limit.

Returns:

It returns unsigned 32-bit Jaus integer data.

5.8.1.14 double jausUnsignedIntegerToDouble (const uint32_t *value*, const double *min*, const double *max*)

Jaus Unsigned Integer Type to Double Conversion.

Parameters:

value Data to be converted.

min Lower limit.

max Upper limit.

Returns:

It returns double float point data.

5.8.1.15 uint64_t jausUnsignedLongFromDouble (double *value*, const double *min*, const double *max*)

Double to Jaus Unsigned Long Type Conversion.

Parameters:

value Data to be converted.

min Lower limit.

max Upper limit.

Returns:

It returns unsigned 64-bit Jaus long integer data.

5.8.1.16 double jausUnsignedLongToDouble (const uint64_t *value*, const double *min*, const double *max*)

Jaus Unsigned Long Type to Double Conversion.

Parameters:

value Data to be converted.

min Lower limit.

max Upper limit.

Returns:

It returns double float point data.

5.8.1.17 uint16_t jausUnsignedShortFromDouble (double *value*, const double *min*, const double *max*)

Double to Jaus Unsigned Short Type Conversion.

Parameters:

value Data to be converted.

min Lower limit.

max Upper limit.

Returns:

It returns unsigned 16-bit Jaus short integer data.

5.8.1.18 double jausUnsignedShortToDouble (const uint16_t *value*, const double *min*, const double *max*)

Jaus Unsigned Short Type to Double Conversion.

Parameters:

value Data to be converted.

min Lower limit.

max Upper limit.

Returns:

It returns double float point data.

Chapter 6

Class Documentation

6.1 DoubleUnion_t Union Reference

Union for Double precision Floating Point Data.

`#include <libdrdc.h>`

Public Attributes

- uint64_t dWord
- uint32_t word [2]
- uint8_t byte [8]
- double value

6.1.1 Detailed Description

Union for Double precision Floating Point Data.

6.1.2 Member Data Documentation

6.1.2.1 uint64_t DoubleUnion_t::dWord

6.1.2.2 uint32_t DoubleUnion_t::word[2]

6.1.2.3 uint8_t DoubleUnion_t::byte[8]

6.1.2.4 double DoubleUnion_t::value

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

- src/libdrdc.h

6.2 drdc_Datum_t Struct Reference

Global Datum

`#include <wrapper.h>`

Public Attributes

- double a
- double f

6.2.1 Detailed Description

Global Datum

6.2.2 Member Data Documentation

6.2.2.1 double drdc_Datum_t::a

6.2.2.2 double drdc_Datum_t::f

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

- src/wrapper/wrapper.h

6.3 drdc_DMS_t Struct Reference

Degree-Minute-Second Structure.

`#include <wrapper.h>`

Public Attributes

- double degree
- double minute
- double second

6.3.1 Detailed Description

Degree-Minute-Second Structure.

6.3.2 Member Data Documentation

6.3.2.1 double drdc_DMS_t::degree

6.3.2.2 double drdc_DMS_t::minute

6.3.2.3 double drdc_DMS_t::second

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

- src/wrapper/wrapper.h

6.4 drdc_eOrient Class Reference

C++ Euler RPY Representation.

`#include <wrapper.h>`

Collaboration diagram for drdc_eOrient:

Public Member Functions

- drdc_eOrient ()
- void set (double roll, double pitch, double yaw)

  Set the object by specifying roll, pitch and yaw values.

- void set (const double *array)

  Set the object by using array data type.

- void toArray (double *array)

  Convert data in the object to data array type.

- int compare (drdc_eOrient eulerRPY, drdc_eOrient fuzz)

  Compare the two objects.

- double getRoll ()

  Return roll value.

- double getPitch ()

  Return pitch value.

- double getYaw ()

  Return yaw value.

- drdc_eOrient & operator= (drdc_eOrient eulerRPY)

  Copy object.

- double operator[ ] (int n)

  Return the nth element from the object.

- drdc_eOrient operator+ (drdc_eOrient eulerRPY)

  Plus operator.

- drdc_eOrient operator- ()

  Negate operator.

- drdc_eOrient operator- (drdc_eOrient eulerRPY)

  Minus operator.

- drdc_eOrient operator∗ (double a)

  Scale operator.

- drdc_HMatrix to_HMatrix ()

  Convert Euler RPY to Homogeneous Matrix.

- drdc_Orient to_Orient ()

  Convert Euler RPY to Quaternion.

Protected Attributes

- drdc_Vector_t ∗ vector

6.4.1 Detailed Description

C++ Euler RPY Representation.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 drdc_eOrient::drdc_eOrient ()

6.4.3 Member Function Documentation

6.4.3.1 void drdc_eOrient::set (double *roll*, double *pitch*, double *yaw*)

Set the object by specifying roll, pitch and yaw values.

6.4.3.2 void drdc_eOrient::set (const double ∗ *array*)

Set the object by using array data type.

6.4.3.3 void drdc_eOrient::toArray (double ∗ *array*)

Convert data in the object to data array type.

6.4.3.4 int drdc_eOrient::compare (drdc_eOrient *eulerRPY*, drdc_eOrient *fuzz*)

Compare the two objects.

The fuzz determines how close they are considered to be equal.

6.4.3.5 double drdc_eOrient::getRoll ()

Return roll value.

6.4.3.6 double drdc_eOrient::getPitch ()

Return pitch value.

6.4.3.7 double drdc_eOrient::getYaw ()

Return yaw value.

6.4.3.8 drdc_eOrient& drdc_eOrient::operator= (drdc_eOrient *eulerRPY*)

Copy object.

6.4.3.9]

double drdc_eOrient::operator[ ] (int *n*)

Return the nth element from the object.

6.4.3.10 drdc_eOrient drdc_eOrient::operator+ (drdc_eOrient *eulerRPY*)

Plus operator.

6.4.3.11 drdc_eOrient drdc_eOrient::operator- ()

Negate operator.

6.4.3.12 drdc_eOrient drdc_eOrient::operator- (drdc_eOrient *eulerRPY*)

Minus operator.

6.4.3.13 drdc_eOrient drdc_eOrient::operator∗ (double *a*)

Scale operator.

6.4.3.14 drdc_HMatrix drdc_eOrient::to_HMatrix ()

Convert Euler RPY to Homogeneous Matrix.

Returns:

Homogeneous Matrix

6.4.3.15 drdc_Orient drdc_eOrient::to_Orient ()

Convert Euler RPY to Quaternion.

Returns:

Quaternion

6.4.4 Member Data Documentation

6.4.4.1 drdc_Vector_t∗ drdc_eOrient::vector [protected]

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

- src/wrapper/wrapper.h

6.5 drdc_eOrient_t Struct Reference

OOC Euler RPY Representation.

`#include <wrapper.h>`

Public Attributes

- const void * vector

  Protected data vector.

- double(* index)(void *self, unsigned char n)

  Return the nth element from the object data vector.

- int(* compare)(void *self, void *object, void *fuzzObject)

  Compare two objects, self and object. The fuzzObject determines how close they are considered to be equal.

- void(* copy)(void *self, void *dest)

  Copy data in the self object to the dest object.

- void(* setFromArray)(void *self, const double *array)

  Set the self object by using array data type.

- void(* toArray)(const void *self, double *array)

  Convert data in the self object to data array type.

- void(* destroy)(void *self)

  Delete the self object.

- void(* set)(void *self, double roll, double pitch, double yaw)

  Set the Euler RPY object by specifying roll, pitch and yaw values.

- double(* getRoll)(void *self)

  Return roll component of Euler RPY.

- double(* getPitch)(void *self)

  Return pitch component of Euler RPY.

- double(* getYaw)(void *self)

  Return yaw component of Euler RPY.

- void(* plus)(void *self, const void *object, void *resultObject)

Plus operator, resultObject = self + object.

- void(∗ minus)(void ∗self, void ∗object, void ∗resultObject)
  Minus operator, resultObject = self - object.
- void(∗ negate)(void ∗self, void ∗object, void ∗resultObject)
  Negate operator, resultObject = - self.
- void(∗ scale)(void ∗self, double scalar, void ∗resultObject)
  Scale operator, resultObject = self ∗ scalar.
- void(∗ to_HMatrix)(void ∗self, void ∗homogeneous)
  Convert Euler RPY to Homogeneous Matrix.
- void(∗ to_Orient)(void ∗self, void ∗quaternion)
  Convert Euler RPY to Quaternion.

6.5.1 Detailed Description

OOC Euler RPY Representation.

6.5.2 Member Data Documentation

6.5.2.1 const void∗ drdc_eOrient_t::vector

Protected data vector.

6.5.2.2 double(∗ drdc_eOrient_t::index)(void ∗self, unsigned char n)

Return the nth element from the object data vector.

6.5.2.3 int(∗ drdc_eOrient_t::compare)(void ∗self, void ∗object, void ∗fuzzObject)

Compare two objects, self and object. The fuzzObject determines how close they are considered to be equal.

6.5.2.4 void(∗ drdc_eOrient_t::copy)(void ∗self, void ∗dest)

Copy data in the self object to the dest object.

6.5.2.5 void(* drdc_eOrient_t::setFromArray)(void *self, const double *array)

Set the self object by using array data type.

6.5.2.6 void(* drdc_eOrient_t::toArray)(const void *self, double *array)

Convert data in the self object to data array type.

6.5.2.7 void(* drdc_eOrient_t::destroy)(void *self)

Delete the self object.

6.5.2.8 void(* drdc_eOrient_t::set)(void *self, double roll, double pitch, double yaw)

Set the Euler RPY object by specifying roll, pitch and yaw values.

6.5.2.9 double(* drdc_eOrient_t::getRoll)(void *self)

Return roll component of Euler RPY.

6.5.2.10 double(* drdc_eOrient_t::getPitch)(void *self)

Return pitch component of Euler RPY.

6.5.2.11 double(* drdc_eOrient_t::getYaw)(void *self)

Return yaw component of Euler RPY.

6.5.2.12 void(* drdc_eOrient_t::plus)(void *self, const void *object, void *resultObject)

Plus operator, resultObject = self + object.

6.5.2.13 void(* drdc_eOrient_t::minus)(void *self, void *object, void *resultObject)

Minus operator, resultObject = self - object.

6.5.2.14 void(* drdc_eOrient_t::negate)(void *self, void *object, void *resultObject)

Negate operator, resultObject = - self.

6.5.2.15 void(* drdc_eOrient_t::scale)(void *self, double scalar, void *resultObject)

Scale operator, resultObject = self * scalar.

6.5.2.16 void(* drdc_eOrient_t::to_HMatrix)(void *self, void *homogeneous)

Convert Euler RPY to Homogeneous Matrix.

6.5.2.17 void(* drdc_eOrient_t::to_Orient)(void *self, void *quaternion)

Convert Euler RPY to Quaternion.

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

- src/wrapper/wrapper.h

6.6 drdc_HMatrix Class Reference

C++ Homogeneous Matrix Representation.

`#include <wrapper.h>`

Collaboration diagram for drdc_HMatrix:

Public Member Functions

- drdc_HMatrix ()

  Constructor.

- void set (const double *array)

  Set the object by using array data type.

- void toArray (double *array)

  Convert data in the object to data array type.

- int compare (drdc_HMatrix homogeneous, drdc_HMatrix fuzz)

  Compare the two objects.

- drdc_lPosn getPosition ()

  Return the translation part of the Homogeneous Matrix.

- void setPosition (drdc_lPosn localPosition)

  Set the translation part of the Homogeneous Matrix.

- drdc_HMatrix & operator= (drdc_HMatrix homogeneous)

  Copy object.

- double operator[ ] (int n)

  Return the nth element from the object.

- drdc_HMatrix operator+ (drdc_HMatrix homogeneous)

  Plus operator.

- drdc_HMatrix operator- ()

  Negate operator.

- drdc_HMatrix operator- (drdc_HMatrix homogeneous)

  Minus operator.

- drdc_HMatrix operator∗ (double a)

  Scale operator.

- drdc_HMatrix operator∗ (drdc_HMatrix homogeneous)

  Homogeneous Matrix multiplication operator.

- drdc_lPosn operator∗ (drdc_lPosn localPosition)

  Homogeneous Matrix - Local Position multiplication operator.

- drdc_WUTMPosn operator∗ (drdc_WUTMPosn worldUTMPosition)

  Homogeneous Matrix - World UTM Position multiplication operator.

- void normalize ()

  Normalize a Homogeneous Matrix.

- drdc_HMatrix transpose ()

  Transpose a Homogeneous Matrix.

- void fromRotAxis (const Axis_t axis, const double theta)

  Construct Homogeneous Matrix from rotation of an axis.

- void fromRotVec (double theta, drdc_lPosn unitVector)

  Construct Homogeneous Matrix from rotation vector.

- void fromDH (double linkLength, double twistAngle, double jointOffset, double jointAngle)

  Construct Homogeneous Matrix from DH representation.

- void fromLink (drdc_LinkSpecs_t linkSpec, double jointOffsetOrAngle)

  Construct Homogeneous Matrix from link specification struct.

- void toRotVec (double ∗theta, drdc_lPosn ∗unitVector)

  Convert Homogeneous Matrix to rotation vector.

- void identity ()

  Create an identity Homogeneous Matrix.

- drdc_HMatrix inverse ()

  Inverse a Homogeneous Matrix.

- drdc_lPose to_lPose ()

  Convert Homogeneous Matrix to Local Pose.

- drdc_lePose to_lePose ()

  Convert Homogeneous Matrix to Local Euler Pose.

- drdc_Orient to_Orient ()

  Convert Homogeneous Matrix to Quaternion.

- drdc_eOrient to_eOrient ()

  Convert Homogeneous Matrix to Euler RPY.

Protected Attributes

- drdc_Vector_t * vector

  Data vector.

6.6.1 Detailed Description

C++ Homogeneous Matrix Representation.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 drdc_HMatrix::drdc_HMatrix ()

Constructor.

6.6.3 Member Function Documentation

6.6.3.1 void drdc_HMatrix::set (const double * *array*)

Set the object by using array data type.

6.6.3.2 void drdc_HMatrix::toArray (double * *array*)

Convert data in the object to data array type.

6.6.3.3 int drdc_HMatrix::compare (drdc_HMatrix *homogeneous*, drdc_HMatrix *fuzz*)

Compare the two objects.

The fuzz determines how close they are considered to be equal.

6.6.3.4 drdc_lPosn drdc_HMatrix::getPosition ()

Return the translation part of the Homogeneous Matrix.

6.6.3.5 void drdc_HMatrix::setPosition (drdc_lPosn *localPosition*)

Set the translation part of the Homogeneous Matrix.

6.6.3.6 drdc_HMatrix& drdc_HMatrix::operator= (drdc_HMatrix *homogeneous*)

Copy object.

6.6.3.7]

double drdc_HMatrix::operator[ ] (int *n*)

Return the nth element from the object.

6.6.3.8 drdc_HMatrix drdc_HMatrix::operator+ (drdc_HMatrix *homogeneous*)

Plus operator.

6.6.3.9 drdc_HMatrix drdc_HMatrix::operator- ()

Negate operator.

6.6.3.10 drdc_HMatrix drdc_HMatrix::operator- (drdc_HMatrix *homogeneous*)

Minus operator.

6.6.3.11 drdc_HMatrix drdc_HMatrix::operator∗ (double *a*)

Scale operator.

6.6.3.12 drdc_HMatrix drdc_HMatrix::operator∗ (drdc_HMatrix *homogeneous*)

Homogeneous Matrix multiplication operator.

6.6.3.13 drdc_lPosn drdc_HMatrix::operator∗ (drdc_lPosn *localPosition*)

Homogeneous Matrix - Local Position multiplication operator.

6.6.3.14 drdc_WUTMPosn drdc_HMatrix::operator∗ (drdc_WUTMPosn *worldUTMPosition*)

Homogeneous Matrix - World UTM Position multiplication operator.

6.6.3.15 void drdc_HMatrix::normalize ()

Normalize a Homogeneous Matrix.

6.6.3.16 drdc_HMatrix drdc_HMatrix::transpose ()

Transpose a Homogeneous Matrix.

6.6.3.17 void drdc_HMatrix::fromRotAxis (const Axis_t *axis*, const double *theta*)

Construct Homogeneous Matrix from rotation of an axis.

6.6.3.18 void drdc_HMatrix::fromRotVec (double *theta*, drdc_lPosn *unitVector*)

Construct Homogeneous Matrix from rotation vector.

6.6.3.19 void drdc_HMatrix::fromDH (double *linkLength*, double *twistAngle*, double *jointOffset*, double *jointAngle*)

Construct Homogeneous Matrix from DH representation.

6.6.3.20 void drdc_HMatrix::fromLink (drdc_LinkSpecs_t *linkSpec*, double *jointOffsetOrAngle*)

Construct Homogeneous Matrix from link specification struct.

6.6.3.21 void drdc_HMatrix::toRotVec (double * *theta*, drdc_lPosn * *unitVector*)

Convert Homogeneous Matrix to rotation vector.

6.6.3.22 void drdc_HMatrix::identity ()

Create an identity Homogeneous Matrix.

6.6.3.23 drdc_HMatrix drdc_HMatrix::inverse ()

Inverse a Homogeneous Matrix.

Returns:

6.6.3.24 drdc_lPose drdc_HMatrix::to_lPose ()

Convert Homogeneous Matrix to Local Pose.

Returns:

6.6.3.25 drdc_lePose drdc_HMatrix::to_lePose ()

Convert Homogeneous Matrix to Local Euler Pose.

Returns:

6.6.3.26 drdc_Orient drdc_HMatrix::to_Orient ()

Convert Homogeneous Matrix to Quaternion.

Returns:

6.6.3.27 drdc_eOrient drdc_HMatrix::to_eOrient ()

Convert Homogeneous Matrix to Euler RPY.

Returns:

6.6.4 Member Data Documentation

6.6.4.1 drdc_Vector_t* drdc_HMatrix::vector [protected]

Data vector.

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

- src/wrapper/wrapper.h

6.7 drdc_HMatrix_t Struct Reference

OOC Homogeneous Matrix Representation.

`#include <wrapper.h>`

Public Attributes

- const void * vector

  Protected data vector.

- double(* index)(void *self, unsigned char n)

  Return the nth element from the object data vector.

- int(* compare)(void *self, void *object, void *fuzzObject)

  Compare two objects, self and object. The fuzzObject determines how close they are considered to be equal.

- void(* copy)(void *self, void *dest)

  Copy data in the self object to the dest object.

- void(* setFromArray)(void *self, const double *array)

  Set the self object by using array data type.

- void(* toArray)(const void *self, double *array)

  Convert data in the self object to data array type.

- void(* destroy)(void *self)

  Delete the self object.

- void(* mult)(void *self, void *object, void *resultObject)

  Homogeneous Matrix multiplication operator.

- void(* multVec)(void *self, void *object, void *resultObject)

  Homogeneous-Vector multiplication operator.

- void(* setPosition)(void *self, drdc_lPosn_t *position)

  Set position component of Homogeneous Matrix.

- void(* getPosition)(void *self, drdc_lPosn_t *position)

  Return position component of Homogeneous Matrix.

- void(* normalize)(void *self)

Normalize a Homogeneous Matrix.

- void(∗ identity)(void ∗self)
  Create identity Homogeneous Matrix.
- void(∗ inverse)(void ∗self, void ∗resultObject)
  Inverse a Homogeneous Matrix.
- void(∗ transpose)(void ∗self, void ∗resultObject)
  Transpose a Homogeneous Matrix.
- void(∗ fromDH)(void ∗self, double linkLength, double twistAngle, double jointOffset, double jointAngle)
  Construct Homogeneous Matrix from DH representation.
- void(∗ fromLink)(void ∗linkSpec, double jointOffsetOrAngle)
  Construct Homogeneous Matrix from link specification struct.
- void(∗ fromRotAxis)(void ∗self, Axis_t axis, double angle)
  Construct Homogeneous Matrix from rotation of an axis.
- void(∗ fromRotVec)(void ∗self, double angle, void ∗unitVector)
  Construct Homogeneous Matrix from rotation vector.
- void(∗ toRotVec)(void ∗self, double ∗angle, void ∗unitVector)
  Convert Homogeneous Matrix to rotation vector.
- void(∗ to_Orient)(void ∗self, void ∗homogeneous)
  Convert Homogeneous Matrix to Quaternion.
- void(∗ to_eOrient)(void ∗self, drdc_eOrient_t ∗eulerRPY)
  Convert Homogeneous Matrix to Euler RPY.
- void(∗ to_lPose)(void ∗self, void ∗localPose)
  Convert Homogeneous Matrix to Local Pose.
- void(∗ to_lePose)(void ∗self, void ∗localEulerPose)
  Convert Homogeneous Matrix to Local Euler Pose.

6.7.1 Detailed Description

OOC Homogeneous Matrix Representation.

6.7.2 Member Data Documentation

6.7.2.1 const void* drdc_HMatrix_t::vector

Protected data vector.

6.7.2.2 double(* drdc_HMatrix_t::index)(void *self, unsigned char n)

Return the nth element from the object data vector.

6.7.2.3 int(* drdc_HMatrix_t::compare)(void *self, void *object, void *fuzzObject)

Compare two objects, self and object. The fuzzObject determines how close they are considered to be equal.

6.7.2.4 void(* drdc_HMatrix_t::copy)(void *self, void *dest)

Copy data in the self object to the dest object.

6.7.2.5 void(* drdc_HMatrix_t::setFromArray)(void *self, const double *array)

Set the self object by using array data type.

6.7.2.6 void(* drdc_HMatrix_t::toArray)(const void *self, double *array)

Convert data in the self object to data array type.

6.7.2.7 void(* drdc_HMatrix_t::destroy)(void *self)

Delete the self object.

6.7.2.8 void(* drdc_HMatrix_t::mult)(void *self, void *object, void *resultObject)

Homogeneous Matrix multiplication operator.

resultObject = self * object

Object and resultObject are Homogeneous Matrix type..

6.7.2.9 void(* drdc_HMatrix_t::multVec)(void *self, void *object, void *resultObject)

Homogeneous-Vector multiplication operator.

resultObject = self * object

Object and resultObject are drdc_lPosn or drdc_WUTMPosn type.

6.7.2.10 void(* drdc_HMatrix_t::setPosition)(void *self, drdc_lPosn_t *position)

Set position component of Homogeneous Matrix.

6.7.2.11 void(* drdc_HMatrix_t::getPosition)(void *self, drdc_lPosn_t *position)

Return position component of Homogeneous Matrix.

6.7.2.12 void(* drdc_HMatrix_t::normalize)(void *self)

Normalize a Homogeneous Matrix.

6.7.2.13 void(* drdc_HMatrix_t::identity)(void *self)

Create identity Homogeneous Matrix.

6.7.2.14 void(* drdc_HMatrix_t::inverse)(void *self, void *resultObject)

Inverse a Homogeneous Matrix.

6.7.2.15 void(* drdc_HMatrix_t::transpose)(void *self, void *resultObject)

Transpose a Homogeneous Matrix.

6.7.2.16 void(* drdc_HMatrix_t::fromDH)(void *self, double linkLength, double twistAngle, double jointOffset, double jointAngle)

Construct Homogeneous Matrix from DH representation.

6.7.2.17 void(* drdc_HMatrix_t::fromLink)(void *linkSpec, double jointOffsetOrAngle)

Construct Homogeneous Matrix from link specification struct.

6.7.2.18 void(* drdc_HMatrix_t::fromRotAxis)(void *self, Axis_t axis, double angle)

Construct Homogeneous Matrix from rotation of an axis.

6.7.2.19 void(* drdc_HMatrix_t::fromRotVec)(void *self, double angle, void *unitVector)

Construct Homogeneous Matrix from rotation vector.

6.7.2.20 void(* drdc_HMatrix_t::toRotVec)(void *self, double *angle, void *unitVector)

Convert Homogeneous Matrix to rotation vector.

6.7.2.21 void(* drdc_HMatrix_t::to_Orient)(void *self, void *homogeneous)

Convert Homogeneous Matrix to Quaternion.

6.7.2.22 void(* drdc_HMatrix_t::to_eOrient)(void *self, drdc_eOrient_t *eulerRPY)

Convert Homogeneous Matrix to Euler RPY.

6.7.2.23 void(* drdc_HMatrix_t::to_lPose)(void *self, void *localPose)

Convert Homogeneous Matrix to Local Pose.

6.7.2.24 void(* drdc_HMatrix_t::to_lePose)(void *self, void *localEulerPose)

Convert Homogeneous Matrix to Local Euler Pose.

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

- src/wrapper/wrapper.h

6.8 drdc_lePose Class Reference

C++ Local Euler Pose Representation.

`#include <wrapper.h>`

Collaboration diagram for drdc_lePose:

Public Member Functions

- drdc_lePose ()
  Constructor.
- void set (drdc_lPosn position, drdc_eOrient orientation)
  Set the object by specifying Local Position and Euler RPY orientation.
- void set (const double *array)
  Set the object by using array data type.
- void toArray (double *array)
  Convert data in the object to data array type.
- int compare (drdc_lePose localEulerPose, drdc_lePose fuzz)
  Compare the two objects.
- drdc_lPosn getPosition ()
  Return Local Position.
- drdc_eOrient getOrientation ()
  Return Euler RPY orientation.
- drdc_lePose & operator= (drdc_lePose localEulerPose)
  Copy object.
- double operator[ ] (int n)
  Return the nth element from the object.
- drdc_lPose to_lPose ()
  Convert Local Euler Pose to Local Pose.
- drdc_HMatrix to_HMatrix ()
  Convert Local Euler Pose to Homogeneous Matrix.

Protected Attributes

- drdc_Vector_t * vector

  Data vector.

6.8.1 Detailed Description

C++ Local Euler Pose Representation.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 drdc_lePose::drdc_lePose ()

Constructor.

6.8.3 Member Function Documentation

6.8.3.1 void drdc_lePose::set (drdc_lPosn *position*, drdc_eOrient *orientation*)

Set the object by specifying Local Position and Euler RPY orientation.

6.8.3.2 void drdc_lePose::set (const double * *array*)

Set the object by using array data type.

6.8.3.3 void drdc_lePose::toArray (double * *array*)

Convert data in the object to data array type.

6.8.3.4 int drdc_lePose::compare (drdc_lePose *localEulerPose*, drdc_lePose *fuzz*)

Compare the two objects.

The fuzz determines how close they are considered to be equal.

6.8.3.5 drdc_lPosn drdc_lePose::getPosition ()

Return Local Position.

6.8.3.6 drdc_eOrient drdc_lePose::getOrientation ()

Return Euler RPY orientation.

6.8.3.7 drdc_lePose& drdc_lePose::operator= (drdc_lePose *localEulerPose*)

Copy object.

6.8.3.8]

double drdc_lePose::operator[ ] (int *n*)

Return the nth element from the object.

6.8.3.9 drdc_lPose drdc_lePose::to_lPose ()

Convert Local Euler Pose to Local Pose.

Returns:

6.8.3.10 drdc_HMatrix drdc_lePose::to_HMatrix ()

Convert Local Euler Pose to Homogeneous Matrix.

Returns:

6.8.4 Member Data Documentation

6.8.4.1 drdc_Vector_t* drdc_lePose::vector [protected]

Data vector.

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

- src/wrapper/wrapper.h

6.9 drdc_lePose_t Struct Reference

OOC Local Euler Pose Representation.

`#include <wrapper.h>`

Public Attributes

- const void * vector

  Protected data vector.

- double(* index)(void *self, unsigned char n)

  Return the nth element from the object data vector.

- int(* compare)(void *self, void *object, void *fuzzObject)

  Compare two objects, self and object. The fuzzObject determines how close they are considered to be equal.

- void(* copy)(void *self, void *dest)

  Copy data in the self object to the dest object.

- void(* setFromArray)(void *self, const double *array)

  Set the self object by using array data type.

- void(* toArray)(const void *self, double *array)

  Convert data in the self object to data array type.

- void(* destroy)(void *self)

  Delete the self object.

- void(* set)(void *self, void *position, void *orientation)

  Set the Local Euler Pose object by specifying Local Position and Euler RPY..

- void(* getPosition)(void *self, void *position)

  Return Local Position of Local Euler Pose.

- void(* getOrientation)(void *self, void *orientation)

  Return Euler RPY orientation of Local Euler Pose.

- void(* to_lPose)(void *self, void *localPose)

  Convert Local Euler Pose to Local Pose.

- void(* to_HMatrix)(void *self, void *homogeneous)

Convert Local Euler Pose to Homogeneous Matrix.

6.9.1 Detailed Description

OOC Local Euler Pose Representation.

6.9.2 Member Data Documentation

6.9.2.1 const void* drdc_lePose_t::vector

Protected data vector.

6.9.2.2 double(* drdc_lePose_t::index)(void *self, unsigned char n)

Return the nth element from the object data vector.

6.9.2.3 int(* drdc_lePose_t::compare)(void *self, void *object, void *fuzzObject)

Compare two objects, self and object. The fuzzObject determines how close they are considered to be equal.

6.9.2.4 void(* drdc_lePose_t::copy)(void *self, void *dest)

Copy data in the self object to the dest object.

6.9.2.5 void(* drdc_lePose_t::setFromArray)(void *self, const double *array)

Set the self object by using array data type.

6.9.2.6 void(* drdc_lePose_t::toArray)(const void *self, double *array)

Convert data in the self object to data array type.

6.9.2.7 void(* drdc_lePose_t::destroy)(void *self)

Delete the self object.

6.9.2.8 void(* drdc_lePose_t::set)(void *self, void *position, void *orientation)

Set the Local Euler Pose object by specifying Local Position and Euler RPY..

6.9.2.9 void(* drdc_lePose_t::getPosition)(void *self, void *position)

Return Local Position of Local Euler Pose.

6.9.2.10 void(* drdc_lePose_t::getOrientation)(void *self, void *orientation)

Return Euler RPY orientation of Local Euler Pose.

6.9.2.11 void(* drdc_lePose_t::to_lPose)(void *self, void *localPose)

Convert Local Euler Pose to Local Pose.

6.9.2.12 void(* drdc_lePose_t::to_HMatrix)(void *self, void *homogeneous)

Convert Local Euler Pose to Homogeneous Matrix.

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

- src/wrapper/wrapper.h

6.10 drdc_LinkSpecs_t Struct Reference

Manipulator Linkage Specification.

`#include <wrapper.h>`

Public Attributes

- uint8_t jointType
- double linkLength
- double twistAngle
- double jointOffsetOrAngle

6.10.1 Detailed Description

Manipulator Linkage Specification.

6.10.2 Member Data Documentation

6.10.2.1 uint8_t drdc_LinkSpecs_t::jointType

6.10.2.2 double drdc_LinkSpecs_t::linkLength

6.10.2.3 double drdc_LinkSpecs_t::twistAngle

6.10.2.4 double drdc_LinkSpecs_t::jointOffsetOrAngle

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

- src/wrapper/wrapper.h

6.11 drdc_lPose Class Reference

C++ Local Pose Representation.

`#include <wrapper.h>`

Collaboration diagram for drdc_lPose:

Public Member Functions

- drdc_lPose ()

  Constructor.

- void set (drdc_lPosn position, drdc_Orient orientation)

  Set the object by specifying Local Position and Quaternion orientation.

- void set (const double *array)

  Set the object by using array data type.

- void toArray (double *array)

  Convert data in the object to data array type.

- int compare (drdc_lPose localPose, drdc_lPose fuzz)

  Compare the two objects.

- drdc_lPosn getPosition ()

  Return Local Position.

- drdc_Orient getOrientation ()

  Return Quaternion orientation.

- drdc_lPose & operator= (drdc_lPose localPose)

  Copy object.

- double operator[ ] (int n)

  Return the nth element from the object.

- drdc_lPose operator* (drdc_lPose localPose)

  Local Pose multiplication operator.

- drdc_lPosn operator* (drdc_lPosn localPosition)

  Local Pose - Local Position multiplication operator.

- drdc_WPose to_WPose (double falseElevation)

  Convert ECEF Pose to World Pose.

- drdc_lePose to_lePose ()

  Convert Local Pose to Local Euler Pose.

- drdc_HMatrix to_HMatrix ()

  Convert Local Pose to Homogeneous Matrix.

- drdc_WUTMPose to_WUTMPose (drdc_UTMZone_t zone)

  Convert Local Pose to World UTM Pose.

Protected Attributes

- drdc_Vector_t * vector

6.11.1 Detailed Description

C++ Local Pose Representation.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 drdc_lPose::drdc_lPose ()

Constructor.

6.11.3 Member Function Documentation

6.11.3.1 void drdc_lPose::set (drdc_lPosn *position*, drdc_Orient *orientation*)

Set the object by specifying Local Position and Quaternion orientation.

6.11.3.2 void drdc_lPose::set (const double * *array*)

Set the object by using array data type.

6.11.3.3 void drdc_lPose::toArray (double * *array*)

Convert data in the object to data array type.

6.11.3.4 int drdc_lPose::compare (drdc_lPose *localPose*, drdc_lPose *fuzz*)

Compare the two objects.

The fuzz determines how close they are considered to be equal.

6.11.3.5 drdc_lPosn drdc_lPose::getPosition ()

Return Local Position.

6.11.3.6 drdc_Orient drdc_lPose::getOrientation ()

Return Quaternion orientation.

6.11.3.7 drdc_lPose& drdc_lPose::operator= (drdc_lPose *localPose*)

Copy object.

6.11.3.8]

double drdc_lPose::operator[ ] (int *n*)

Return the nth element from the object.

6.11.3.9 drdc_lPose drdc_lPose::operator∗ (drdc_lPose *localPose*)

Local Pose multiplication operator.

6.11.3.10 drdc_lPosn drdc_lPose::operator∗ (drdc_lPosn *localPosition*)

Local Pose - Local Position multiplication operator.

6.11.3.11 drdc_WPose drdc_lPose::to_WPose (double *falseElevation*)

Convert ECEF Pose to World Pose.

Parameters:

falseElevation

Returns:

6.11.3.12 drdc_lePose drdc_lPose::to_lePose ()

Convert Local Pose to Local Euler Pose.

Returns:

6.11.3.13 drdc_HMatrix drdc_lPose::to_HMatrix ()

Convert Local Pose to Homogeneous Matrix.

Returns:

6.11.3.14 drdc_WUTMPose drdc_lPose::to_WUTMPose (drdc_UTMZone_t *zone*)

Convert Local Pose to World UTM Pose.

Parameters:

zone

Returns:

6.11.4 Member Data Documentation

6.11.4.1 drdc_Vector_t* drdc_lPose::vector [protected]

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

- src/wrapper/wrapper.h

6.12 drdc_lPose_t Struct Reference

OOC Local Pose Representation.

`#include <wrapper.h>`

Public Attributes

- const void * vector

  Protected data vector.

- double(* index)(void *self, unsigned char n)

  Return the nth element from the object data vector.

- int(* compare)(void *self, void *object, void *fuzzObject)

  Compare two objects, self and object. The fuzzObject determines how close they are considered to be equal.

- void(* copy)(void *self, void *dest)

  Copy data in the self object to the dest object.

- void(* setFromArray)(void *self, const double *array)

  Set the self object by using array data type.

- void(* toArray)(const void *self, double *array)

  Convert data in the self object to data array type.

- void(* destroy)(void *self)

  Delete the self object.

- void(* set)(void *self, void *position, void *orientation)

  Set the Local Pose object by specifying Local Position and Quaternion.

- void(* getPosition)(void *self, void *position)

  Return Local Position of Local Pose.

- void(* getOrientation)(void *self, void *orientation)

  Return Quaternion orientation of Local Pose.

- void(* mult)(void *self, void *object, void *resultObject)

  Local Pose multiplication operator.

- void(* multVec)(void *self, void *object, void *resultObject)

Local Pose - Local Position multiplication operator.

- void(∗ to_WPose)(void ∗self, double falseElevation, void ∗worldPose)

  Convert Local Pose to World Pose.

- void(∗ to_WUTMPose)(void ∗self, drdc_UTMZone_t zone, void ∗worldUTMPose)

  Map World UTM Pose to Local Pose It is a simple mapping function.

- void(∗ to_lePose)(void ∗self, void ∗localEulerPose)

  Convert Local Pose to Local Euler Pose.

- void(∗ to_HMatrix)(void ∗self, void ∗homogeneous)

  Convert Local Pose to Homogeneous Matrix.

6.12.1 Detailed Description

OOC Local Pose Representation.

6.12.2 Member Data Documentation

6.12.2.1 const void∗ drdc_lPose_t::vector

Protected data vector.

6.12.2.2 double(∗ drdc_lPose_t::index)(void ∗self, unsigned char n)

Return the nth element from the object data vector.

6.12.2.3 int(∗ drdc_lPose_t::compare)(void ∗self, void ∗object, void ∗fuzzObject)

Compare two objects, self and object. The fuzzObject determines how close they are considered to be equal.

6.12.2.4 void(∗ drdc_lPose_t::copy)(void ∗self, void ∗dest)

Copy data in the self object to the dest object.

6.12.2.5 void(* drdc_lPose_t::setFromArray)(void *self, const double *array)

Set the self object by using array data type.

6.12.2.6 void(* drdc_lPose_t::toArray)(const void *self, double *array)

Convert data in the self object to data array type.

6.12.2.7 void(* drdc_lPose_t::destroy)(void *self)

Delete the self object.

6.12.2.8 void(* drdc_lPose_t::set)(void *self, void *position, void *orientation)

Set the Local Pose object by specifying Local Position and Quaternion.

6.12.2.9 void(* drdc_lPose_t::getPosition)(void *self, void *position)

Return Local Position of Local Pose.

6.12.2.10 void(* drdc_lPose_t::getOrientation)(void *self, void *orientation)

Return Quaternion orientation of Local Pose.

6.12.2.11 void(* drdc_lPose_t::mult)(void *self, void *object, void *resultObject)

Local Pose multiplication operator.

resultObject = self * object

Object and resultObject are drdc_lPose type.

6.12.2.12 void(* drdc_lPose_t::multVec)(void *self, void *object, void *resultObject)

Local Pose - Local Position multiplication operator.

resultObject = self * object.

Object and resultObject are drdc_lPosn type.

6.12.2.13 void(* drdc_lPose_t::to_WPose)(void *self, double falseElevation, void *worldPose)

Convert Local Pose to World Pose.

6.12.2.14 void(* drdc_lPose_t::to_WUTMPose)(void *self, drdc_UTMZone_t zone, void *worldUTMPose)

Map World UTM Pose to Local Pose It is a simple mapping function.

The mapping scheme is shown as follows:

- x - Easting
- y - Northing
- z - Elevation
- Orientation - Orientation

Other mapping schemes can be achieved by using coordinate transformations.

6.12.2.15 void(* drdc_lPose_t::to_lePose)(void *self, void *localEulerPose)

Convert Local Pose to Local Euler Pose.

6.12.2.16 void(* drdc_lPose_t::to_HMatrix)(void *self, void *homogeneous)

Convert Local Pose to Homogeneous Matrix.

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

- src/wrapper/wrapper.h

6.13 drdc_lPosn Class Reference

C++ Local Position Representation.

`#include <wrapper.h>`

Collaboration diagram for drdc_lPosn:

Public Member Functions

- drdc_lPosn ()

  Constructor.

- void set (double x, double y, double z)

  Set the object by specifying cartesian x, y and z values.

- void set (const double *array)

  Set the object by using array data type.

- void toArray (double *array)

  Convert data in the object to data array type.

- int compare (drdc_lPosn localPosition, drdc_lPosn fuzz)

  Compare two objects, self and object.

- double getX ()

  Return the cartesian x value from the object.

- double getY ()

  Return the cartesian y value from the object.

- double getZ ()

  Return the cartesian z value from the object.

- drdc_lPosn & operator= (drdc_lPosn localPosition)

  Copy object.

- double operator[ ] (int n)

  Return the nth element from the object.

- drdc_lPosn operator+ (drdc_lPosn localPosition)

  Plus operator.

- drdc_lPosn operator- ()

  Negate operator.

- drdc_lPosn operator- (drdc_lPosn localPosition)

  Minus operator.

- drdc_lPosn operator∗ (double a)

  Scale operator.

- double dot (drdc_lPosn localPosition)

  Vector dot operator.

- drdc_lPosn cross (drdc_lPosn localPosition)

  Vector cross operator.

- void normalize ()

  Normalize the object.

- double magnitude ()

  Return the magnitude of the object.

- drdc_WUTMPosn to_WUTMPosn (drdc_UTMZone_t zone)

  Map Local Position to World UTM Position.

- drdc_WPosn to_WPosn (double falseElevation)

  Convert ECEF Position to World Position.

Protected Attributes

- drdc_Vector_t ∗ vector

  Protected data vector.

6.13.1 Detailed Description

C++ Local Position Representation.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 drdc_lPosn::drdc_lPosn ()

Constructor.

6.13.3 Member Function Documentation

6.13.3.1 void drdc_lPosn::set (double *x*, double *y*, double *z*)

Set the object by specifying cartesian x, y and z values.

6.13.3.2 void drdc_lPosn::set (const double * *array*)

Set the object by using array data type.

6.13.3.3 void drdc_lPosn::toArray (double * *array*)

Convert data in the object to data array type.

6.13.3.4 int drdc_lPosn::compare (drdc_lPosn *localPosition*, drdc_lPosn *fuzz*)

Compare two objects, self and object.
The fuzz determines how close they are considered to be equal.

6.13.3.5 double drdc_lPosn::getX ()

Return the cartesian x value from the object.

6.13.3.6 double drdc_lPosn::getY ()

Return the cartesian y value from the object.

6.13.3.7 double drdc_lPosn::getZ ()

Return the cartesian z value from the object.

6.13.3.8 drdc_lPosn& drdc_lPosn::operator= (drdc_lPosn *localPosition*)

Copy object.

6.13.3.9]

double drdc_lPosn::operator[ ] (int *n*)

Return the nth element from the object.

6.13.3.10 drdc_lPosn drdc_lPosn::operator+ (drdc_lPosn *localPosition*)

Plus operator.

6.13.3.11 drdc_lPosn drdc_lPosn::operator- ()

Negate operator.

6.13.3.12 drdc_lPosn drdc_lPosn::operator- (drdc_lPosn *localPosition*)

Minus operator.

6.13.3.13 drdc_lPosn drdc_lPosn::operator∗ (double *a*)

Scale operator.

6.13.3.14 double drdc_lPosn::dot (drdc_lPosn *localPosition*)

Vector dot operator.

6.13.3.15 drdc_lPosn drdc_lPosn::cross (drdc_lPosn *localPosition*)

Vector cross operator.

6.13.3.16 void drdc_lPosn::normalize ()

Normalize the object.

6.13.3.17 double drdc_lPosn::magnitude ()

Return the magnitude of the object.

6.13.3.18 drdc_WUTMPosn drdc_lPosn::to_WUTMPosn (drdc_UTMZone_t *zone*)

Map Local Position to World UTM Position.

It is a simple mapping function. The mapping scheme is shown as follows:

- x - Easting
- y - Northing
- z - Elevation

Other mapping schemes can be achieved by using coordinate transformations.

6.13.3.19 drdc_WPosn drdc_lPosn::to_WPosn (double *falseElevation*)

Convert ECEF Position to World Position.

ECEF is a global position representation. Here we use Local Position, which is really a cartesian system, to represent ECEF system.

Parameters:

falseElevation False elevation.

6.13.4 Member Data Documentation

6.13.4.1 drdc_Vector_t* drdc_lPosn::vector [protected]

Protected data vector.

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

- src/wrapper/wrapper.h

6.14 drdc_lPosn_t Struct Reference

OOC Local Position Representation.

`#include <wrapper.h>`

Public Attributes

- const void * vector

  Protected data vector.

- double(* index)(void *self, unsigned char n)

  Return the nth element from the object data vector.

- int(* compare)(void *self, void *object, void *fuzzObject)

  Compare two objects, self and object. The fuzzObject determines how close they are considered to be equal.

- void(* copy)(void *self, void *dest)

  Copy data in the self object to the dest object.

- void(* setFromArray)(void *self, const double *array)

  Set the self object by using array data type.

- void(* toArray)(const void *self, double *array)

  Convert data in the self object to data array type.

- void(* destroy)(void *self)

  Delete the self object.

- void(* set)(void *self, double x, double y, double z)

  Set the Local Position object by specifying cartesian x, y and z values.

- double(* getX)(void *self)

  Return the cartesian x value from the Local Position object.

- double(* getY)(void *self)

  Return the cartesian y value from the Local Position object.

- double(* getZ)(void *self)

  Return the cartesian z value from the Local Position object.

- void(* plus)(void *self, const void *object, void *resultObject)

Plus operator, resultObject = self + object.

- void(∗ minus)(void ∗self, void ∗object, void ∗resultObject)

  Minus operator, resultObject = self - object.

- void(∗ negate)(void ∗self, void ∗resultObject)

  Negate operator, resultObject = - self.

- void(∗ scale)(void ∗self, double scalar, void ∗resultObject)

  Scale operator, resultObject = self ∗ scalar.

- double(∗ dot)(void ∗self, void ∗object)

  Vector dot operator. It returns vector dot result.

- void(∗ cross)(void ∗self, void ∗object, void ∗resultObject)

  Vector cross operator, resultObject = self cross object.

- void(∗ normalize)(void ∗self)

  Normalize the self object (vector norm).

- double(∗ magnitude)(void ∗self)

  Return the magnitude of the self object.

- void(∗ to_WUTMPosn)(void ∗self, drdc_UTMZone_t zone, void ∗worldUTMPosition)

  Map Local Position to World UTM Position.

- void(∗ to_WPosn)(void ∗self, double falseElevation, void ∗worldPosition)

  Convert ECEF Position to World Position.

6.14.1 Detailed Description

OOC Local Position Representation.

6.14.2 Member Data Documentation

6.14.2.1 const void∗ drdc_lPosn_t::vector

Protected data vector.

6.14.2.2 double(* drdc_lPosn_t::index)(void *self, unsigned char n)

Return the nth element from the object data vector.

6.14.2.3 int(* drdc_lPosn_t::compare)(void *self, void *object, void *fuzzObject)

Compare two objects, self and object. The fuzzObject determines how close they are considered to be equal.

6.14.2.4 void(* drdc_lPosn_t::copy)(void *self, void *dest)

Copy data in the self object to the dest object.

6.14.2.5 void(* drdc_lPosn_t::setFromArray)(void *self, const double *array)

Set the self object by using array data type.

6.14.2.6 void(* drdc_lPosn_t::toArray)(const void *self, double *array)

Convert data in the self object to data array type.

6.14.2.7 void(* drdc_lPosn_t::destroy)(void *self)

Delete the self object.

6.14.2.8 void(* drdc_lPosn_t::set)(void *self, double x, double y, double z)

Set the Local Position object by specifying cartesian x, y and z values.

6.14.2.9 double(* drdc_lPosn_t::getX)(void *self)

Return the cartesian x value from the Local Position object.

6.14.2.10 double(* drdc_lPosn_t::getY)(void *self)

Return the cartesian y value from the Local Position object.

6.14.2.11 double(* drdc_lPosn_t::getZ)(void *self)

Return the cartesian z value from the Local Position object.

6.14.2.12 void(* drdc_lPosn_t::plus)(void *self, const void *object, void *resultObject)

Plus operator, resultObject = self + object.

6.14.2.13 void(* drdc_lPosn_t::minus)(void *self, void *object, void *resultObject)

Minus operator, resultObject = self - object.

6.14.2.14 void(* drdc_lPosn_t::negate)(void *self, void *resultObject)

Negate operator, resultObject = - self.

6.14.2.15 void(* drdc_lPosn_t::scale)(void *self, double scalar, void *resultObject)

Scale operator, resultObject = self * scalar.

6.14.2.16 double(* drdc_lPosn_t::dot)(void *self, void *object)

Vector dot operator. It returns vector dot result.

6.14.2.17 void(* drdc_lPosn_t::cross)(void *self, void *object, void *resultObject)

Vector cross operator, resultObject = self cross object.

6.14.2.18 void(* drdc_lPosn_t::normalize)(void *self)

Normalize the self object (vector norm).

6.14.2.19 double(* drdc_lPosn_t::magnitude)(void *self)

Return the magnitude of the self object.

6.14.2.20 void(∗ drdc_lPosn_t::to_WUTMPosn)(void ∗self, drdc_UTMZone_t zone, void ∗worldUTMPosition)

Map Local Position to World UTM Position.

It is a simple mapping function. The mapping scheme is shown as follows:

- x - Easting
- y - Northing
- z - Elevation

Other mapping schemes can be achieved by using coordinate transformations.

6.14.2.21 void(∗ drdc_lPosn_t::to_WPosn)(void ∗self, double falseElevation, void ∗worldPosition)

Convert ECEF Position to World Position.

ECEF is a global position representation. Here we use Local Position, which is really a cartesian system, to represent ECEF system.

Parameters:

falseElevation False elevation.

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

- src/wrapper/wrapper.h

6.15 drdc_Orient Class Reference

C++ Quaternion Representation.

`#include <wrapper.h>`

Collaboration diagram for drdc_Orient:

Public Member Functions

- drdc_Orient ()
  Constructor.
- void set (double w, double x, double y, double z)
  Set the object by specifying Quaternion w, x, y and z components.
- void set (const double *array)
  Set the object by using array data type.
- void toArray (double *array)
  Convert data in the object to data array type.
- int compare (drdc_Orient quaternion, drdc_Orient fuzz)
  Compare the two objects.
- double getW ()
  Return w component of the Quaternion.
- double getX ()
  Return x component of the Quaternion.
- double getY ()
  Return y component of the Quaternion.
- double getZ ()
  Return z component of the Quaternion.
- drdc_Orient & operator= (drdc_Orient quaternion)
  Copy object.
- double operator[ ] (int n)
  Return the nth element from the object.

- drdc_Orient operator+ (drdc_Orient quaternion)

  Plus operator.

- drdc_Orient operator- ()

  Negate operator.

- drdc_Orient operator- (drdc_Orient quaternion)

  Minus operator.

- drdc_Orient operator∗ (double a)

  Scale operator.

- drdc_Orient operator∗ (drdc_Orient quaternion)

  Quaternion multiplication operator.

- drdc_lPosn operator∗ (drdc_lPosn localPosition)

  Quaternion - Local Position multiplication operator.

- drdc_WUTMPosn operator∗ (drdc_WUTMPosn worldUTMPosition)

  Quaternion - World UTM Position multiplication operator.

- void normalize ()

  Normalize Quaternion.

- double magnitude ()

  Return magnitude of Quaternion.

- void fromRotAxis (const Axis_t axis, const double theta)

  Construct Quaternion from Rotation of a axis.

- void fromRotVec (double theta, drdc_lPosn unitVector)

  Construct Quaternion from rotating an angle about a unit vector.

- void toRotVec (double ∗theta, drdc_lPosn ∗unitVector)

  Convert Quaternion to Rotation Vector (angle and unit vector).

- void identity ()

  Create identity Quaternion.

- drdc_Orient inverse ()

  Inverse a Quaternion.

- drdc_HMatrix to_HMatrix ()

  Convert Quaternion to Homogeneous Matrix.

- drdc_eOrient to_eOrient ()

  Convert Quaternion to Euler RPY.

Protected Attributes

- drdc_Vector_t * vector

  Data vector.

6.15.1 Detailed Description

C++ Quaternion Representation.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 drdc_Orient::drdc_Orient ()

Constructor.

6.15.3 Member Function Documentation

6.15.3.1 void drdc_Orient::set (double *w*, double *x*, double *y*, double *z*)

Set the object by specifying Quaternion w, x, y and z components.

6.15.3.2 void drdc_Orient::set (const double * *array*)

Set the object by using array data type.

6.15.3.3 void drdc_Orient::toArray (double * *array*)

Convert data in the object to data array type.

6.15.3.4 int drdc_Orient::compare (drdc_Orient *quaternion*, drdc_Orient *fuzz*)

Compare the two objects.

The fuzz determines how close they are considered to be equal.

6.15.3.5 double drdc_Orient::getW ()

Return w component of the Quaternion.

6.15.3.6 double drdc_Orient::getX ()

Return x component of the Quaternion.

6.15.3.7 double drdc_Orient::getY ()

Return y component of the Quaternion.

6.15.3.8 double drdc_Orient::getZ ()

Return z component of the Quaternion.

6.15.3.9 drdc_Orient& drdc_Orient::operator= (drdc_Orient *quaternion*)

Copy object.

6.15.3.10]

double drdc_Orient::operator[ ] (int *n*)

Return the nth element from the object.

6.15.3.11 drdc_Orient drdc_Orient::operator+ (drdc_Orient *quaternion*)

Plus operator.

6.15.3.12 drdc_Orient drdc_Orient::operator- ()

Negate operator.

6.15.3.13 drdc_Orient drdc_Orient::operator- (drdc_Orient *quaternion*)

Minus operator.

6.15.3.14 drdc_Orient drdc_Orient::operator* (double *a*)

Scale operator.

6.15.3.15 drdc_Orient drdc_Orient::operator* (drdc_Orient *quaternion*)

Quaternion multiplication operator.

6.15.3.16 drdc_lPosn drdc_Orient::operator* (drdc_lPosn *localPosition*)

Quaternion - Local Position multiplication operator.

6.15.3.17 drdc_WUTMPosn drdc_Orient::operator* (drdc_WUTMPosn *worldUTMPosition*)

Quaternion - World UTM Position multiplication operator.

6.15.3.18 void drdc_Orient::normalize ()

Normalize Quaternion.

6.15.3.19 double drdc_Orient::magnitude ()

Return magnitude of Quaternion.

6.15.3.20 void drdc_Orient::fromRotAxis (const Axis_t *axis*, const double *theta*)

Construct Quaternion from Rotation of a axis.

6.15.3.21 void drdc_Orient::fromRotVec (double *theta*, drdc_lPosn *unitVector*)

Construct Quaternion from rotating an angle about a unit vector.

6.15.3.22 void drdc_Orient::toRotVec (double * *theta*, drdc_lPosn * *unitVector*)

Convert Quaternion to Rotation Vector (angle and unit vector).

6.15.3.23 void drdc_Orient::identity ()

Create identity Quaternion.

6.15.3.24 drdc_Orient drdc_Orient::inverse ()

Inverse a Quaternion.

6.15.3.25 drdc_HMatrix drdc_Orient::to_HMatrix ()

Convert Quaternion to Homogeneous Matrix.

Returns:

6.15.3.26 drdc_eOrient drdc_Orient::to_eOrient ()

Convert Quaternion to Euler RPY.

Returns:

6.15.4 Member Data Documentation

6.15.4.1 drdc_Vector_t* drdc_Orient::vector [protected]

Data vector.

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

- src/wrapper/wrapper.h

6.16 drdc_Orient_t Struct Reference

OOC Quaternion Representation.

`#include <wrapper.h>`

Public Attributes

- const void * vector

  Protected data vector.

- double(* index)(void *self, unsigned char n)

  Return the nth element from the object data vector.

- int(* compare)(void *self, void *object, void *fuzzObject)

  Compare two objects, self and object. The fuzzObject determines how close they are considered to be equal.

- void(* copy)(void *self, void *dest)

  Copy data in the self object to the dest object.

- void(* setFromArray)(void *self, const double *array)

  Set the self object by using array data type.

- void(* toArray)(const void *self, double *array)

  Convert data in the self object to data array type.

- void(* destroy)(void *self)

  Delete the self object.

- void(* set)(void *self, double w, double x, double y, double z)

  Set the Auaternion object by specifying w, x, y and z components.

- double(* getW)(void *self)

  Return w component of Quaternion.

- double(* getX)(void *self)

  Return x component of Quaternion.

- double(* getY)(void *self)

  Return y component of Quaternion.

- double(* getZ)(void *self)

Return z component of Quaternion.

- void(* plus)(void *self, const void *object, void *resultObject)
  Plus operator, resultObject = self + object.
- void(* minus)(void *self, void *object, void *resultObject)
  Minus operator, resultObject = self - object.
- void(* negate)(void *self, void *object, void *resultObject)
  Negate operator, resultObject = - self.
- void(* scale)(void *self, double scalar, void *resultObject)
  *Scale operator, resultObject = self * scalar.*
- void(* mult)(void *self, void *object, void *resultObject)
  Quaternion multiplication operator.
- void(* multVec)(void *self, void *object, void *resultObject)
  Quaternion-Vector multiplication operator.
- void(* normalize)(void *self)
  Normalize Quaternion.
- double(* magnitude)(void *self)
  Return magnitude of Quaternion.
- void(* identity)(void *self)
  Create identity Quaternion.
- void(* inverse)(void *self, void *resultObject)
  Inverse a Quaternion.
- void(* fromRotAxis)(void *self, Axis_t axis, double angle)
  Construct Quaternion from Rotation of a axis.
- void(* fromRotVec)(void *self, double angle, void *unitVector)
  Construct Quaternion from rotating an angle about a unit vector.
- void(* toRotVec)(void *self, double *angle, void *unitVector)
  Convert Quaternion to Rotation Vector (angle and unit vector).
- void(* to_HMatrix)(void *self, void *homogeneous)

Convert Quaternion to Homogeneous Matrix.

- void(∗ to_eOrient)(void ∗self, void ∗eulerRPY)

  Convert Quaternion to Euler RPY.

6.16.1 Detailed Description

OOC Quaternion Representation.

6.16.2 Member Data Documentation

6.16.2.1 const void∗ drdc_Orient_t::vector

Protected data vector.

6.16.2.2 double(∗ drdc_Orient_t::index)(void ∗self, unsigned char n)

Return the nth element from the object data vector.

6.16.2.3 int(∗ drdc_Orient_t::compare)(void ∗self, void ∗object, void ∗fuzzObject)

Compare two objects, self and object. The fuzzObject determines how close they are considered to be equal.

6.16.2.4 void(∗ drdc_Orient_t::copy)(void ∗self, void ∗dest)

Copy data in the self object to the dest object.

6.16.2.5 void(∗ drdc_Orient_t::setFromArray)(void ∗self, const double ∗array)

Set the self object by using array data type.

6.16.2.6 void(∗ drdc_Orient_t::toArray)(const void ∗self, double ∗array)

Convert data in the self object to data array type.

6.16.2.7 void(* drdc_Orient_t::destroy)(void *self)

Delete the self object.

6.16.2.8 void(* drdc_Orient_t::set)(void *self, double w, double x, double y, double z)

Set the Auaternion object by specifying w, x, y and z components.

6.16.2.9 double(* drdc_Orient_t::getW)(void *self)

Return w component of Quaternion.

6.16.2.10 double(* drdc_Orient_t::getX)(void *self)

Return x component of Quaternion.

6.16.2.11 double(* drdc_Orient_t::getY)(void *self)

Return y component of Quaternion.

6.16.2.12 double(* drdc_Orient_t::getZ)(void *self)

Return z component of Quaternion.

6.16.2.13 void(* drdc_Orient_t::plus)(void *self, const void *object, void *resultObject)

Plus operator, resultObject = self + object.

6.16.2.14 void(* drdc_Orient_t::minus)(void *self, void *object, void *resultObject)

Minus operator, resultObject = self - object.

6.16.2.15 void(* drdc_Orient_t::negate)(void *self, void *object, void *resultObject)

Negate operator, resultObject = - self.

6.16.2.16 void(* drdc_Orient_t::scale)(void *self, double scalar, void *resultObject)

Scale operator, resultObject = self * scalar.

6.16.2.17 void(* drdc_Orient_t::mult)(void *self, void *object, void *resultObject)

Quaternion multiplication operator.

resultObject = self * object

Object and resultObject are drdc_Orient_t type.

6.16.2.18 void(* drdc_Orient_t::multVec)(void *self, void *object, void *resultObject)

Quaternion-Vector multiplication operator.

resultObject = self * object

Object and resultObject are drdc_lPosn_t or drdc_WUTMPosn_t type.

6.16.2.19 void(* drdc_Orient_t::normalize)(void *self)

Normalize Quaternion.

6.16.2.20 double(* drdc_Orient_t::magnitude)(void *self)

Return magnitude of Quaternion.

6.16.2.21 void(* drdc_Orient_t::identity)(void *self)

Create identity Quaternion.

6.16.2.22 void(* drdc_Orient_t::inverse)(void *self, void *resultObject)

Inverse a Quaternion.

6.16.2.23 void(* drdc_Orient_t::fromRotAxis)(void *self, Axis_t axis, double angle)

Construct Quaternion from Rotation of a axis.

6.16.2.24 void(* drdc_Orient_t::fromRotVec)(void *self, double angle, void *unitVector)

Construct Quaternion from rotating an angle about a unit vector.

6.16.2.25 void(* drdc_Orient_t::toRotVec)(void *self, double *angle, void *unitVector)

Convert Quaternion to Rotation Vector (angle and unit vector).

6.16.2.26 void(* drdc_Orient_t::to_HMatrix)(void *self, void *homogeneous)

Convert Quaternion to Homogeneous Matrix.

6.16.2.27 void(* drdc_Orient_t::to_eOrient)(void *self, void *eulerRPY)

Convert Quaternion to Euler RPY.

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

- src/wrapper/wrapper.h

6.17 drdc_UTMZone_t Struct Reference

UTM Zone Structure

`#include <wrapper.h>`

Public Attributes

- unsigned char zoneNumber
- char zoneLetter

6.17.1 Detailed Description

UTM Zone Structure

6.17.2 Member Data Documentation

6.17.2.1 unsigned char drdc_UTMZone_t::zoneNumber

6.17.2.2 char drdc_UTMZone_t::zoneLetter

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

- src/wrapper/wrapper.h

6.18 drdc_Vector_t Struct Reference

Generic Vector Type.

`#include <wrapper.h>`

Public Attributes

- unsigned char dsize
- double * data

6.18.1 Detailed Description

Generic Vector Type.

6.18.2 Member Data Documentation

6.18.2.1 unsigned char drdc_Vector_t::dsize

6.18.2.2 double* drdc_Vector_t::data

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

- src/wrapper/wrapper.h

6.19 drdc_WPose Class Reference

C++ World Pose Representation.

`#include <wrapper.h>`

Collaboration diagram for drdc_WPose:

Public Member Functions

- drdc_WPose ()
  Constructor.
- void set (drdc_WPosn position, drdc_Orient orientation)
  Set the object by specifying World Position and Quaternion orientation.
- void set (const double *array)
  Set the object by using array data type.
- void toArray (double *array)
  Convert data in the object to data array type.
- int compare (drdc_WPose worldPose, drdc_WPose fuzz)
  Compare the two objects.
- drdc_WPosn getPosition ()
  Return World Position.
- drdc_Orient getOrientation ()
  Return Quaternion orientation.
- drdc_WPose & operator= (drdc_WPose worldPose)
  Copy object.
- double operator[ ] (int n)
  Return the nth element from the object.
- drdc_lPose to_lPose (double falseElevation)
  Convert World Pose to Local Pose.
- drdc_WUTMPose to_WUTMPose ()
  Convert World Pose to World UTM Pose.

Protected Attributes

- drdc_Vector_t * vector

  Data vector.

6.19.1 Detailed Description

C++ World Pose Representation.

6.19.2 Constructor & Destructor Documentation

6.19.2.1 drdc_WPose::drdc_WPose ()

Constructor.

6.19.3 Member Function Documentation

6.19.3.1 void drdc_WPose::set (drdc_WPosn *position*, drdc_Orient *orientation*)

Set the object by specifying World Position and Quaternion orientation.

6.19.3.2 void drdc_WPose::set (const double * *array*)

Set the object by using array data type.

6.19.3.3 void drdc_WPose::toArray (double * *array*)

Convert data in the object to data array type.

6.19.3.4 int drdc_WPose::compare (drdc_WPose *worldPose*, drdc_WPose *fuzz*)

Compare the two objects.

The fuzz determines how close they are considered to be equal.

6.19.3.5 drdc_WPosn drdc_WPose::getPosition ()

Return World Position.

6.19.3.6 drdc_Orient drdc_WPose::getOrientation ()

Return Quaternion orientation.

6.19.3.7 drdc_WPose& drdc_WPose::operator= (drdc_WPose *worldPose*)

Copy object.

6.19.3.8]

double drdc_WPose::operator[ ] (int *n*)

Return the nth element from the object.

6.19.3.9 drdc_lPose drdc_WPose::to_lPose (double *falseElevation*)

Convert World Pose to Local Pose.

Parameters:

falseElevation

Returns:

6.19.3.10 drdc_WUTMPose drdc_WPose::to_WUTMPose ()

Convert World Pose to World UTM Pose.

Returns:

6.19.4 Member Data Documentation

6.19.4.1 drdc_Vector_t* drdc_WPose::vector [protected]

Data vector.

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

- src/wrapper/wrapper.h

6.20 drdc_WPose_t Struct Reference

OOC World Pose Representation.

`#include <wrapper.h>`

Public Attributes

- const void * vector

  Protected data vector.

- double(* index)(void *self, unsigned char n)

  Return the nth element from the object data vector.

- int(* compare)(void *self, void *object, void *fuzzObject)

  Compare two objects, self and object. The fuzzObject determines how close they are considered to be equal.

- void(* copy)(void *self, void *dest)

  Copy data in the self object to the dest object.

- void(* setFromArray)(void *self, const double *array)

  Set the self object by using array data type.

- void(* toArray)(const void *self, double *array)

  Convert data in the self object to data array type.

- void(* destroy)(void *self)

  Delete the self object.

- void(* set)(void *self, void *position, void *orientation)

  Set World Pose object by specifying World Position and Quaternion orientation.

- void(* getPosition)(void *self, void *position)

  Return World Position of World Pose.

- void(* getOrientation)(void *self, void *orientation)

  Return Quaternion orientation of World Pose.

- void(* to_WUTMPose)(void *self, void *worldUTMPose)

  Convert World Pose to World UTM Pose.

- void(* to_lPose)(void *self, double falseElevation, void *localPose)

Convert World Pose to Local Pose.

6.20.1 Detailed Description

OOC World Pose Representation.

6.20.2 Member Data Documentation

6.20.2.1 const void* drdc_WPose_t::vector

Protected data vector.

6.20.2.2 double(* drdc_WPose_t::index)(void *self, unsigned char n)

Return the nth element from the object data vector.

6.20.2.3 int(* drdc_WPose_t::compare)(void *self, void *object, void *fuzzObject)

Compare two objects, self and object. The fuzzObject determines how close they are considered to be equal.

6.20.2.4 void(* drdc_WPose_t::copy)(void *self, void *dest)

Copy data in the self object to the dest object.

6.20.2.5 void(* drdc_WPose_t::setFromArray)(void *self, const double *array)

Set the self object by using array data type.

6.20.2.6 void(* drdc_WPose_t::toArray)(const void *self, double *array)

Convert data in the self object to data array type.

6.20.2.7 void(* drdc_WPose_t::destroy)(void *self)

Delete the self object.

6.20.2.8 void(* drdc_WPose_t::set)(void *self, void *position, void *orientation)

Set World Pose object by specifying World Position and Quaternion orientation.

6.20.2.9 void(* drdc_WPose_t::getPosition)(void *self, void *position)

Return World Position of World Pose.

6.20.2.10 void(* drdc_WPose_t::getOrientation)(void *self, void *orientation)

Return Quaternion orientation of World Pose.

6.20.2.11 void(* drdc_WPose_t::to_WUTMPose)(void *self, void *worldUTMPose)

Convert World Pose to World UTM Pose.

6.20.2.12 void(* drdc_WPose_t::to_lPose)(void *self, double falseElevation, void *localPose)

Convert World Pose to Local Pose.

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

- src/wrapper/wrapper.h

6.21 drdc_WPosn Class Reference

C++ World Position Represenation.

`#include <wrapper.h>`

Collaboration diagram for drdc_WPosn:

Public Member Functions

- drdc_WPosn ()
  Constructor.
- void set (double longtitude, double latitude, double elevation)
  Set the object by specifying longtitude, latitude and elevation.
- void set (const double *array)
  Set the object by using array data type.
- void toArray (double *array)
  Convert data in the object to data array type.
- int compare (drdc_WPosn worldPosition, drdc_WPosn fuzz)
  Compare the two objects.
- double getLongtitude ()
  Return the longtitude value from the object.
- double getLatitude ()
  Return the latitude value from the object.
- double getElevation ()
  Return the Elevation value from the object.
- drdc_WPosn & operator= (drdc_WPosn worldPosition)
  Copy object.
- double operator[ ] (int n)
  Return the nth element from the object.
- void normalize ()
  Normalize lontitude to -180 to 180 degree and latitude to -90 to 90 degree.

- drdc_lPosn to_lPosn (double falseElevation)

  Convert World Position to ECEF Position.

- drdc_WUTMPosn to_WUTMPosn ()

  Convert World Position to World UTM Position.

Protected Attributes

- drdc_Vector_t * vector

  Data vector.

6.21.1 Detailed Description

C++ World Position Represenation.

6.21.2 Constructor & Destructor Documentation

6.21.2.1 drdc_WPosn::drdc_WPosn ()

Constructor.

6.21.3 Member Function Documentation

6.21.3.1 void drdc_WPosn::set (double *longtitude*, double *latitude*, double *elevation*)

Set the object by specifying longtitude, latitude and elevation.

6.21.3.2 void drdc_WPosn::set (const double * *array*)

Set the object by using array data type.

6.21.3.3 void drdc_WPosn::toArray (double * *array*)

Convert data in the object to data array type.

6.21.3.4 int drdc_WPosn::compare (drdc_WPosn *worldPosition*, drdc_WPosn *fuzz*)

Compare the two objects.

The fuzz determines how close they are considered to be equal.

6.21.3.5 double drdc_WPosn::getLongtitude ()

Return the longtitude value from the object.

6.21.3.6 double drdc_WPosn::getLatitude ()

Return the latitude value from the object.

6.21.3.7 double drdc_WPosn::getElevation ()

Return the Elevation value from the object.

6.21.3.8 drdc_WPosn& drdc_WPosn::operator= (drdc_WPosn *worldPosition*)

Copy object.

6.21.3.9]

double drdc_WPosn::operator[ ] (int *n*)

Return the nth element from the object.

6.21.3.10 void drdc_WPosn::normalize ()

Normalize lontitude to -180 to 180 degree and latitude to -90 to 90 degree.

6.21.3.11 drdc_lPosn drdc_WPosn::to_lPosn (double *falseElevation*)

Convert World Position to ECEF Position.

Parameters:

falseElevation

Returns:

Cartesian position represented by Local Position

6.21.3.12 drdc_WUTMPosn drdc_WPosn::to_WUTMPosn ()

Convert World Position to World UTM Position.

Returns:

6.21.4 Member Data Documentation

6.21.4.1 drdc_Vector_t* drdc_WPosn::vector [protected]

Data vector.

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

- src/wrapper/wrapper.h

6.22 drdc_WPosn_t Struct Reference

OOC World Position Representation.

`#include <wrapper.h>`

Public Attributes

- const void * vector

  Protected data vector.

- double(* index)(void *self, unsigned char n)

  Return the nth element from the object data vector.

- int(* compare)(void *self, void *object, void *fuzzObject)

  Compare two objects, self and object. The fuzzObject determines how close they are considered to be equal.

- void(* copy)(void *self, void *dest)

  Copy data in the self object to the dest object.

- void(* setFromArray)(void *self, const double *array)

  Set the self object by using array data type.

- void(* toArray)(const void *self, double *array)

  Convert data in the self object to data array type.

- void(* destroy)(void *self)

  Delete the self object.

- void(* set)(void *self, double longtitude, double latitude, double elevation)

  Set the World Position object by specifying longtitude, latitude and elevation values.

- double(* getLongtitude)(void *self)

  Return the longtitude value from the World Position Object.

- double(* getLatitude)(void *self)

  Return the latitude value from the World Position Object.

- double(* getElevation)(void *self)

  Return the elevation value from the World Position Object.

- void(* normalize)(void *self)

Normalize lontitude to -180 to 180 degree and latitude to -90 to 90 degree.

- void(∗ to_lPosn)(void ∗self, double falseElevation, void ∗localPosition)
  Convert World Position to ECEF position.
- void(∗ to_WUTMPosn)(void ∗self, void ∗worldUTMPosition)
  Convert World Position to World UTM Position.

6.22.1 Detailed Description

OOC World Position Representation.

6.22.2 Member Data Documentation

6.22.2.1 const void∗ drdc_WPosn_t::vector

Protected data vector.

6.22.2.2 double(∗ drdc_WPosn_t::index)(void ∗self, unsigned char n)

Return the nth element from the object data vector.

6.22.2.3 int(∗ drdc_WPosn_t::compare)(void ∗self, void ∗object, void ∗fuzzObject)

Compare two objects, self and object. The fuzzObject determines how close they are considered to be equal.

6.22.2.4 void(∗ drdc_WPosn_t::copy)(void ∗self, void ∗dest)

Copy data in the self object to the dest object.

6.22.2.5 void(∗ drdc_WPosn_t::setFromArray)(void ∗self, const double ∗array)

Set the self object by using array data type.

6.22.2.6 void(∗ drdc_WPosn_t::toArray)(const void ∗self, double ∗array)

Convert data in the self object to data array type.

6.22.2.7 void(∗ drdc_WPosn_t::destroy)(void ∗self)

Delete the self object.

6.22.2.8 void(∗ drdc_WPosn_t::set)(void ∗self, double longtitude, double latitude, double elevation)

Set the World Position object by specifying longtitude, latitude and elevation values.

6.22.2.9 double(∗ drdc_WPosn_t::getLongtitude)(void ∗self)

Return the longtitude value from the World Position Object.

6.22.2.10 double(∗ drdc_WPosn_t::getLatitude)(void ∗self)

Return the latitude value from the World Position Object.

6.22.2.11 double(∗ drdc_WPosn_t::getElevation)(void ∗self)

Return the elevation value from the World Position Object.

6.22.2.12 void(∗ drdc_WPosn_t::normalize)(void ∗self)

Normalize lontitude to -180 to 180 degree and latitude to -90 to 90 degree.

6.22.2.13 void(∗ drdc_WPosn_t::to_lPosn)(void ∗self, double falseElevation, void ∗localPosition)

Convert World Position to ECEF position.

6.22.2.14 void(∗ drdc_WPosn_t::to_WUTMPosn)(void ∗self, void ∗worldUTMPosition)

Convert World Position to World UTM Position.

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

- src/wrapper/wrapper.h

6.23 drdc_WUTMPose Class Reference

World UTM Pose Representation.

`#include <wrapper.h>`

Collaboration diagram for drdc_WUTMPose:

Public Member Functions

- drdc_WUTMPose ()
  Constructor.
- void set (drdc_WUTMPosn position, drdc_Orient orientation)
  Set the object by specifying World UTM Position and quaternion.
- void set (const double *array)
  Set the object by using array data type.
- void setZone (drdc_UTMZone_t zone)
  Set UTM zone.
- drdc_UTMZone_t getZone ()
  Return UTM zone.
- void toArray (double *array)
  Convert data in the object to data array type.
- int compare (drdc_WUTMPose worldPose, drdc_WUTMPose fuzz)
  Compare the two objects.
- drdc_WUTMPosn getPosition ()
  Return World UTM Position.
- drdc_Orient getOrientation ()
  Return Quaternion orientation.
- drdc_WUTMPose & operator= (drdc_WUTMPose worldUTMPose)
  Copy object.
- double operator[ ] (int n)
  Return the nth element from the object.

- drdc_WUTMPose operator∗ (drdc_WUTMPose worldUTMPose)

  World UTM Pose multiplication.

- drdc_WUTMPosn operator∗ (drdc_WUTMPosn worldUTMPosition)

  World UTM Pose - Position multiplication.

- drdc_WPose to_WPose ()

  Convert World UTM Pose to World Pose.

- drdc_lPose to_lPose ()

  Map World UTM Pose to Local Pose.

Protected Attributes

- drdc_Vector_t ∗ vector

  Data vector.

- drdc_UTMZone_t zone

  UTM zone.

6.23.1 Detailed Description

World UTM Pose Representation.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 drdc_WUTMPose::drdc_WUTMPose ()

Constructor.

6.23.3 Member Function Documentation

6.23.3.1 void drdc_WUTMPose::set (drdc_WUTMPosn *position*, drdc_Orient *orientation*)

Set the object by specifying World UTM Position and quaternion.

6.23.3.2 void drdc_WUTMPose::set (const double * *array*)

Set the object by using array data type.

6.23.3.3 void drdc_WUTMPose::setZone (drdc_UTMZone_t *zone*)

Set UTM zone.

6.23.3.4 drdc_UTMZone_t drdc_WUTMPose::getZone ()

Return UTM zone.

6.23.3.5 void drdc_WUTMPose::toArray (double * *array*)

Convert data in the object to data array type.

6.23.3.6 int drdc_WUTMPose::compare (drdc_WUTMPose *worldPose*, drdc_WUTMPose *fuzz*)

Compare the two objects.

The fuzz determines how close they are considered to be equal.

6.23.3.7 drdc_WUTMPosn drdc_WUTMPose::getPosition ()

Return World UTM Position.

6.23.3.8 drdc_Orient drdc_WUTMPose::getOrientation ()

Return Quaternion orientation.

6.23.3.9 drdc_WUTMPose& drdc_WUTMPose::operator= (drdc_WUTMPose *worldUTMPose*)

Copy object.

6.23.3.10]

double drdc_WUTMPose::operator[ ] (int *n*)

Return the nth element from the object.

6.23.3.11 drdc_WUTMPose drdc_WUTMPose::operator∗ (drdc_WUTMPose *worldUTMPose*)

World UTM Pose multiplication.

6.23.3.12 drdc_WUTMPosn drdc_WUTMPose::operator∗ (drdc_WUTMPosn *worldUTMPosition*)

World UTM Pose - Position multiplication.

6.23.3.13 drdc_WPose drdc_WUTMPose::to_WPose ()

Convert World UTM Pose to World Pose.

Returns:

6.23.3.14 drdc_lPose drdc_WUTMPose::to_lPose ()

Map World UTM Pose to Local Pose.

Returns:

6.23.4 Member Data Documentation

6.23.4.1 drdc_Vector_t∗ drdc_WUTMPose::vector [protected]

Data vector.

6.23.4.2 drdc_UTMZone_t drdc_WUTMPose::zone [protected]

UTM zone.

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

- src/wrapper/wrapper.h

6.24 drdc_WUTMPose_t Struct Reference

OOC World UTM Pose Representation.

`#include <wrapper.h>`

Collaboration diagram for drdc_WUTMPose_t:

Public Attributes

- const void * vector

  Protected data vector.

- drdc_UTMZone_t zone

  UTM Zone.

- double(* index)(void *self, unsigned char n)

  Return the nth element from the object data vector.

- int(* compare)(void *self, void *object, void *fuzzObject)

  Compare two objects, self and object. The fuzzObject determines how close they are considered to be equal.

- void(* copy)(void *self, void *dest)

  Copy data in the self object to the dest object.

- void(* setFromArray)(void *self, const double *array)

  Set the self object by using array data type.

- void(* toArray)(const void *self, double *array)

  Convert data in the self object to data array type.

- void(* destroy)(void *self)

  Delete the self object.

- void(* set)(void *self, void *position, void *orientation)

  Set the World UTM Pose object by specifying World UTM Position and Quaternion..

- void(* getPosition)(void *self, void *position)

  Return World UTM Position of World UTM Pose.

- void(* getOrientation)(void *self, void *orientation)

  Return Quaternion orientation of World UTM Pose.

- void(∗ setZone)(void ∗self, drdc_UTMZone_t zone)

  Set UTM Zone.

- void(∗ getZone)(void ∗self, drdc_UTMZone_t ∗zone)

  Return UTM Zone.

- void(∗ mult)(void ∗self, void ∗object, void ∗resultObject)

  Local Pose multiplication operator.

- void(∗ multVec)(void ∗self, void ∗object, void ∗resultObject)

  World UTM Pose -World UTM Position multiplication operator.

- void(∗ to_WPose)(void ∗self, void ∗worldPose)

  Convert World UTM Pose to World Pose.

- void(∗ to_lPose)(void ∗self, void ∗localPose)

  Map World UTM Pose to Local Pose.

6.24.1 Detailed Description

OOC World UTM Pose Representation.

6.24.2 Member Data Documentation

6.24.2.1 const void∗ drdc_WUTMPose_t::vector

Protected data vector.

6.24.2.2 drdc_UTMZone_t drdc_WUTMPose_t::zone

UTM Zone.

6.24.2.3 double(∗ drdc_WUTMPose_t::index)(void ∗self, unsigned char n)

Return the nth element from the object data vector.

6.24.2.4 int(∗ drdc_WUTMPose_t::compare)(void ∗self, void ∗object, void ∗fuzzObject)

Compare two objects, self and object. The fuzzObject determines how close they are considered to be equal.

6.24.2.5 void(∗ drdc_WUTMPose_t::copy)(void ∗self, void ∗dest)

Copy data in the self object to the dest object.

6.24.2.6 void(∗ drdc_WUTMPose_t::setFromArray)(void ∗self, const double ∗array)

Set the self object by using array data type.

6.24.2.7 void(∗ drdc_WUTMPose_t::toArray)(const void ∗self, double ∗array)

Convert data in the self object to data array type.

6.24.2.8 void(∗ drdc_WUTMPose_t::destroy)(void ∗self)

Delete the self object.

6.24.2.9 void(∗ drdc_WUTMPose_t::set)(void ∗self, void ∗position, void ∗orientation)

Set the World UTM Pose object by specifying World UTM Position and Quaternion..

6.24.2.10 void(∗ drdc_WUTMPose_t::getPosition)(void ∗self, void ∗position)

Return World UTM Position of World UTM Pose.

6.24.2.11 void(∗ drdc_WUTMPose_t::getOrientation)(void ∗self, void ∗orientation)

Return Quaternion orientation of World UTM Pose.

6.24.2.12 void(* drdc_WUTMPose_t::setZone)(void *self, drdc_UTMZone_t zone)

Set UTM Zone.

6.24.2.13 void(* drdc_WUTMPose_t::getZone)(void *self, drdc_UTMZone_t *zone)

Return UTM Zone.

6.24.2.14 void(* drdc_WUTMPose_t::mult)(void *self, void *object, void *resultObject)

Local Pose multiplication operator.

resultObject = self * object

Object and resultObject are drdc_WUTMPose type.

6.24.2.15 void(* drdc_WUTMPose_t::multVec)(void *self, void *object, void *resultObject)

World UTM Pose -World UTM Position multiplication operator.

resultObject = self * object.

Object and resultObject are drdc_WUTMPosn type.

6.24.2.16 void(* drdc_WUTMPose_t::to_WPose)(void *self, void *worldPose)

Convert World UTM Pose to World Pose.

6.24.2.17 void(* drdc_WUTMPose_t::to_lPose)(void *self, void *localPose)

Map World UTM Pose to Local Pose.

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

- src/wrapper/wrapper.h

6.25 drdc_WUTMPosn Class Reference

C++ World UTM Position Representation.

`#include <wrapper.h>`

Collaboration diagram for drdc_WUTMPosn:

Public Member Functions

- drdc_WUTMPosn ()
  Constructor.
- void set (double easting, double northing, double elevation)
  Set the object by specifying UTM easting, northing and elevation.
- void set (const double *array)
  Set the object by using array data type.
- void setZone (drdc_UTMZone_t zone)
  Set UTM zone.
- drdc_UTMZone_t getZone ()
  Return UTM zone.
- void toArray (double *array)
  Convert data in the object to data array type.
- int compare (drdc_WUTMPosn worldUTMPosition, drdc_WUTMPosn fuzz)
  Compare the two objects.
- double getEasting ()
  Return the Easting value from the object.
- double getNorthing ()
  Return the Northing value from the object.
- double getElevation ()
  Return the Elevation value from the object.
- drdc_WUTMPosn & operator= (drdc_WUTMPosn worldUTMPosition)
  Copy object.

- double operator[ ] (int n)

  Return the nth element from the object.

- drdc_WUTMPosn operator+ (drdc_WUTMPosn worldUTMPosition)

  Plus operator.

- drdc_WUTMPosn operator- ()

  Negate operator.

- drdc_WUTMPosn operator- (drdc_WUTMPosn worldUTMPosition)

  Minus operator.

- drdc_WUTMPosn operator∗ (double a)

  Scale operator.

- double dot (drdc_WUTMPosn worldUTMPosition)

  Vector dot operator.

- drdc_WUTMPosn cross (drdc_WUTMPosn worldUTMPosition)

  Vector cross operator.

- drdc_lPosn to_lPosn ()

  Map World UTM Position to Local Position.

- drdc_WPosn to_WPosn ()

  Convert World UTM Position to World Position.

Protected Attributes

- drdc_Vector_t ∗ vector

  Data vector.

- drdc_UTMZone_t zone

  UTM zone.

6.25.1 Detailed Description

C++ World UTM Position Representation.

6.25.2 Constructor & Destructor Documentation

6.25.2.1 drdc_WUTMPosn::drdc_WUTMPosn ()

Constructor.

6.25.3 Member Function Documentation

6.25.3.1 void drdc_WUTMPosn::set (double *easting*, double *northing*, double *elevation*)

Set the object by specifying UTM easting, northing and elevation.

6.25.3.2 void drdc_WUTMPosn::set (const double * *array*)

Set the object by using array data type.

6.25.3.3 void drdc_WUTMPosn::setZone (drdc_UTMZone_t *zone*)

Set UTM zone.

6.25.3.4 drdc_UTMZone_t drdc_WUTMPosn::getZone ()

Return UTM zone.

6.25.3.5 void drdc_WUTMPosn::toArray (double * *array*)

Convert data in the object to data array type.

6.25.3.6 int drdc_WUTMPosn::compare (drdc_WUTMPosn *worldUTMPosition*, drdc_WUTMPosn *fuzz*)

Compare the two objects.

The fuzz determines how close they are considered to be equal.

6.25.3.7 double drdc_WUTMPosn::getEasting ()

Return the Easting value from the object.

6.25.3.8 double drdc_WUTMPosn::getNorthing ()

Return the Northing value from the object.

6.25.3.9 double drdc_WUTMPosn::getElevation ()

Return the Elevation value from the object.

6.25.3.10 drdc_WUTMPosn& drdc_WUTMPosn::operator= (drdc_WUTMPosn *worldUTMPosition*)

Copy object.

6.25.3.11]

double drdc_WUTMPosn::operator[ ] (int *n*)

Return the nth element from the object.

6.25.3.12 drdc_WUTMPosn drdc_WUTMPosn::operator+ (drdc_WUTMPosn *worldUTMPosition*)

Plus operator.

6.25.3.13 drdc_WUTMPosn drdc_WUTMPosn::operator- ()

Negate operator.

6.25.3.14 drdc_WUTMPosn drdc_WUTMPosn::operator- (drdc_WUTMPosn *worldUTMPosition*)

Minus operator.

6.25.3.15 drdc_WUTMPosn drdc_WUTMPosn::operator∗ (double *a*)

Scale operator.

6.25.3.16 double drdc_WUTMPosn::dot (drdc_WUTMPosn *worldUTMPosition*)

Vector dot operator.

6.25.3.17 drdc_WUTMPosn drdc_WUTMPosn::cross (drdc_WUTMPosn *worldUTMPosition*)

Vector cross operator.

6.25.3.18 drdc_lPosn drdc_WUTMPosn::to_lPosn ()

Map World UTM Position to Local Position.

It is a simple mapping function. The mapping scheme is shown as follows:

- x - Easting
- y - Northing
- z - Elevation

Other mapping schemes can be achieved by using coordinate transformations.

Returns:

Local Position

6.25.3.19 drdc_WPosn drdc_WUTMPosn::to_WPosn ()

Convert World UTM Position to World Position.

Returns:

World Position

6.25.4 Member Data Documentation

6.25.4.1 drdc_Vector_t* drdc_WUTMPosn::vector [protected]

Data vector.

6.25.4.2 drdc_UTMZone_t drdc_WUTMPosn::zone [protected]

UTM zone.

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

- src/wrapper/wrapper.h

6.26 drdc_WUTMPosn_t Struct Reference

OOC World UTM Position Representation.

`#include <wrapper.h>`

Collaboration diagram for drdc_WUTMPosn_t:

Public Attributes

- const void * vector

  Protected data vector.

- drdc_UTMZone_t zone

  UTM Zone.

- double(* index)(void *self, unsigned char n)

  Return the nth element from the object data vector.

- int(* compare)(void *self, void *object, void *fuzzObject)

  Compare two objects, self and object. The fuzzObject determines how close they are considered to be equal.

- void(* copy)(void *self, void *dest)

  Copy data in the self object to the dest object.

- void(* setFromArray)(void *self, const double *array)
- void(* toArray)(const void *self, double *array)
- void(* destroy)(void *self)
- void(* set)(void *self, double easting, double northing, double elevation)

  Set the World UTM Position object by specifying Easting, Northing and Elevation values.

- double(* getEasting)(void *self)

  Return the Easting value from the World UTM Position object.

- double(* getNorthing)(void *self)

  Return the Northing value from the World UTM Position object.

- double(* getElevation)(void *self)

  Return the Elevation value from the World UTM Position object.

- void(* setZone)(void *self, drdc_UTMZone_t zone)

Set UTM Zone.

- void(∗ getZone)(void ∗self, drdc_UTMZone_t ∗zone)

  Return UTM Zone.

- void(∗ plus)(void ∗self, const void ∗object, void ∗resultObject)

  Plus operator, resultObject = self + object.

- void(∗ minus)(void ∗self, void ∗object, void ∗resultObject)

  Minus operator, resultObject = self - object.

- void(∗ negate)(void ∗self, void ∗object, void ∗resultObject)

  Negate operator, resultObject = - self.

- void(∗ scale)(void ∗self, double scalar, void ∗resultObject)

  Scale operator, resultObject = self ∗ scalar.

- double(∗ dot)(void ∗self, void ∗object)

  Vector dot operator. It returns vector dot result.

- void(∗ cross)(void ∗self, void ∗object, void ∗resultObject)

  Vector cross operator, resultObject = self cross object.

- void(∗ to_lPosn)(void ∗self, void ∗localPosition)

  Map World UTM Position to Local Position It is a simple mapping function.

- void(∗ to_WPosn)(void ∗self, void ∗worldPosition)

  Convert World UTM Position to World Position.

6.26.1 Detailed Description

OOC World UTM Position Representation.

6.26.2 Member Data Documentation

6.26.2.1 const void∗ drdc_WUTMPosn_t::vector

Protected data vector.

6.26.2.2 drdc_UTMZone_t drdc_WUTMPosn_t::zone

UTM Zone.

6.26.2.3 double(* drdc_WUTMPosn_t::index)(void *self, unsigned char n)

Return the nth element from the object data vector.

6.26.2.4 int(* drdc_WUTMPosn_t::compare)(void *self, void *object, void *fuzzObject)

Compare two objects, self and object. The fuzzObject determines how close they are considered to be equal.

6.26.2.5 void(* drdc_WUTMPosn_t::copy)(void *self, void *dest)

Copy data in the self object to the dest object.

6.26.2.6 void(* drdc_WUTMPosn_t::setFromArray)(void *self, const double *array)

6.26.2.7 void(* drdc_WUTMPosn_t::toArray)(const void *self, double *array)

6.26.2.8 void(* drdc_WUTMPosn_t::destroy)(void *self)

6.26.2.9 void(* drdc_WUTMPosn_t::set)(void *self, double easting, double northing, double elevation)

Set the World UTM Position object by specifying Easting, Northing and Elevation values.

6.26.2.10 double(* drdc_WUTMPosn_t::getEasting)(void *self)

Return the Easting value from the World UTM Position object.

6.26.2.11 double(* drdc_WUTMPosn_t::getNorthing)(void *self)

Return the Northing value from the World UTM Position object.

6.26.2.12 double(* drdc_WUTMPosn_t::getElevation)(void *self)

Return the Elevation value from the World UTM Position object.

6.26.2.13 void(* drdc_WUTMPosn_t::setZone)(void *self, drdc_UTMZone_t zone)

Set UTM Zone.

6.26.2.14 void(* drdc_WUTMPosn_t::getZone)(void *self, drdc_UTMZone_t *zone)

Return UTM Zone.

6.26.2.15 void(* drdc_WUTMPosn_t::plus)(void *self, const void *object, void *resultObject)

Plus operator, resultObject = self + object.

6.26.2.16 void(* drdc_WUTMPosn_t::minus)(void *self, void *object, void *resultObject)

Minus operator, resultObject = self - object.

6.26.2.17 void(* drdc_WUTMPosn_t::negate)(void *self, void *object, void *resultObject)

Negate operator, resultObject = - self.

6.26.2.18 void(* drdc_WUTMPosn_t::scale)(void *self, double scalar, void *resultObject)

Scale operator, resultObject = self * scalar.

6.26.2.19 double(* drdc_WUTMPosn_t::dot)(void *self, void *object)

Vector dot operator. It returns vector dot result.

6.26.2.20 void(* drdc_WUTMPosn_t::cross)(void *self, void *object, void *resultObject)

Vector cross operator, resultObject = self cross object.

6.26.2.21 void(* drdc_WUTMPosn_t::to_lPosn)(void *self, void *localPosition)

Map World UTM Position to Local Position It is a simple mapping function.

The mapping scheme is shown as follows:

- x - Easting
- y - Northing
- z - Elevation

Other mapping schemes can be achieved by using coordinate transformations.

6.26.2.22 void(* drdc_WUTMPosn_t::to_WPosn)(void *self, void *worldPosition)

Convert World UTM Position to World Position.

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

- src/wrapper/wrapper.h

6.27 FloatUnion_t Union Reference

Union for Single precision Floating Point Data.

`#include <libdrdc.h>`

Public Attributes

- uint32_t word
- uint8_t byte [4]
- float value

6.27.1 Detailed Description

Union for Single precision Floating Point Data.

6.27.2 Member Data Documentation

6.27.2.1 uint32_t FloatUnion_t::word

6.27.2.2 uint8_t FloatUnion_t::byte[4]

6.27.2.3 float FloatUnion_t::value

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

- src/libdrdc.h

6.28 JausTime_t Struct Reference

JAUS Timestamp and Datestamp Structure.

`#include <time_conv.h>`

Public Attributes

- uint32_t timeStamp
- uint16_t dateStamp
- uint16_t millisec
- uint16_t second
- uint16_t minute
- uint16_t hour
- uint16_t day
- uint16_t month
- uint16_t year

6.28.1 Detailed Description

JAUS Timestamp and Datestamp Structure.

6.28.2 Member Data Documentation

6.28.2.1 uint32_t JausTime_t::timeStamp

6.28.2.2 uint16_t JausTime_t::dateStamp

6.28.2.3 uint16_t JausTime_t::millisec

6.28.2.4 uint16_t JausTime_t::second

6.28.2.5 uint16_t JausTime_t::minute

6.28.2.6 uint16_t JausTime_t::hour

6.28.2.7 uint16_t JausTime_t::day

6.28.2.8 uint16_t JausTime_t::month

6.28.2.9 uint16_t JausTime_t::year

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

- src/time_conv/time_conv.h

6.29 UnitConv_t Struct Reference

Unit Conversion Structure.

`#include <unit_conv.h>`

Public Attributes

- char * unit
- double factor

6.29.1 Detailed Description

Unit Conversion Structure.

6.29.2 Member Data Documentation

6.29.2.1 char* UnitConv_t::unit

6.29.2.2 double UnitConv_t::factor

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

- src/unit_conv/unit_conv.h

Chapter 7

File Documentation

7.1 src/cexcept/cexcept.h File Reference

```
#include <setjmp.h>
```

Include dependency graph for cexcept.h:

This graph shows which files directly or indirectly include this file:

Defines

- #define init_exception_context(ec) ((void)((ec) → penv = 0))
- #define Try
- #define exception__catch(action)
- #define Catch(e) exception__catch(((e) = the_exception_context → v.etmp, 0))
- #define Throw
- #define Catch_anonymous exception__catch(0)

7.2 src/cexcept/except.h File Reference

```
#include "cexcept.h"
```

Include dependency graph for except.h:

This graph shows which files directly or indirectly include this file:

Defines

- #define define_exception_type(etype)

7.3 src/coordinate/coordinate.h File Reference

This graph shows which files directly or indirectly include this file:

Defines

- #define WGS84_A 6378137.0
- #define WGS84_F (1.0 / 298.257223563)

Functions

- int LatLon_to_UTM (const double a, const double f, const double Lat, const double Long, double ∗UTMNorthing, double ∗UTMEasting, unsigned char ∗UTMZoneNum, char ∗UTMZoneCh)

  Lat/Lon to UTM Northing/Easting Conversion.

- int UTM_to_LatLon (const double a, const double f, const double UTMNorthing, const double UTMEasting, const unsigned char UTMZoneNum, const char UTMZoneCh, double ∗Lat, double ∗Long)

  UTM to Lat/Lon Conversion

- int WUTMPosn_to_WPosn (const double a, const double f, const unsigned char zoneNumber, const char zoneLetter, const double ∗worldUTMPosition, double ∗worldPosition)

  UTM Position to World Position

- int WPosn_to_WUTMPosn (const double a, const double f, const double ∗worldPosition, unsigned char ∗zoneNumber, char ∗zoneLetter, double ∗worldUTMPosition)

  World Position to UTM Position.

- int cartesian_to_spherical (const double ∗cartesian, double ∗spherical)

  Cartesian to Spherical Conversion

- int spherical_to_cartesian (const double ∗spherical, double ∗cartesian)

  Spherical to Cartesian Conversion

- int WPosn_to_lPosn (const double a, const double f, const double falseElevation, const double ∗worldPosition, double ∗cartesian)

  World Position to Cartesian Conversion.

- int lPosn_to_WPosn (const double a, const double f, const double falseElevation, const double *cartesian, double *worldPosition)

  Cartesian to World Position Conversion

- double DMS_to_DecimalDegree (const double degrees, const double minutes, const double seconds)

  Degree-Minute-Second to Decimal Degree Conversion.

7.3.1 Define Documentation

7.3.1.1 #define WGS84_A 6378137.0

7.3.1.2 #define WGS84_F (1.0 / 298.257223563)

7.4 src/data_conv/data_conv.h File Reference

```
#include <math.h>
#include "libdrdc.h"
```

Include dependency graph for data_conv.h:

Defines

- #define JAUS_BYTE_RANGE 255.0
- #define JAUS_INTEGER_RANGE 4294967294.0
- #define JAUS_LONG_RANGE (double) 18446744073709551614.0
- #define JAUS_SHORT_RANGE 65534.0
- #define JAUS_UNSIGNED_INTEGER_RANGE 4294967294.0
- #define JAUS_UNSIGNED_LONG_RANGE (double) 1.8446744073709552e19
- #define JAUS_UNSIGNED_SHORT_RANGE 65535.0

Typedefs

- typedef uint32_t ieeeFpd32
- typedef uint64_t ieeeFpd64

Functions

- ieeeFpd32 ieeeFpd32FromFloat (const float value)

  Converts single to float point data represented by 32-bit unsigned integer format.

- ieeeFpd64 ieeeFpd64FromDouble (const double value)

  Converts double to float point data represented by 64-bit unsigned integer format.

- float ieeeFpd32ToFloat (const ieeeFpd32 input)

  Converts float point data represented by 32-bit unsigned integer format to single.

- double ieeeFpd64ToDouble (const ieeeFpd64 input)

  Converts float point data represented by 64-bit unsigned integer format to double.

- uint8_t jausByteFromDouble (double value, const double min, const double max)

  Double to Jaus Byte Conversion.

- double jausByteToDouble (const uint8_t value, const double min, const double max)

  Jaus Byte to Double Conversion.

- double jausIntegerToDouble (const sint32_t value, const double min, const double max)

  Jaus Integer to Double Type Conversion.

- sint32_t jausIntegerFromDouble (double value, const double min, const double max)

  Double to Jaus Integer Type Conversion.

- double jausLongToDouble (const sint64_t value, const double min, const double max)

  Jaus Long Type to Double Conversion.

- sint64_t jausLongFromDouble (double value, const double min, const double max)

  Double to Jaus Long Type Conversion.

- double jausShortToDouble (const sint16_t value, const double min, const double max)

  Jaus Short Type to Double Conversion.

- sint16_t jausShortFromDouble (double value, const double min, const double max)

  Double to Jaus Short Type Conversion.

- double jausUnsignedIntegerToDouble (const uint32_t value, const double min, const double max)

  Jaus Unsigned Integer Type to Double Conversion.

- uint32_t jausUnsignedIntegerFromDouble (double value, const double min, const double max)

  Double to Jaus Unsigned Integer Type Conversion.

- double jausUnsignedLongToDouble (const uint64_t value, const double min, const double max)

  Jaus Unsigned Long Type to Double Conversion.

- uint64_t jausUnsignedLongFromDouble (double value, const double min, const double max)

  Double to Jaus Unsigned Long Type Conversion.

- double jausUnsignedShortToDouble (const uint16_t value, const double min, const double max)

  Jaus Unsigned Short Type to Double Conversion.

- uint16_t jausUnsignedShortFromDouble (double value, const double min, const double max)

  Double to Jaus Unsigned Short Type Conversion.

7.4.1 Define Documentation

7.4.1.1 #define JAUS_BYTE_RANGE 255.0

7.4.1.2 #define JAUS_INTEGER_RANGE 4294967294.0

7.4.1.3 #define JAUS_LONG_RANGE (double) 18446744073709551614.0

7.4.1.4 #define JAUS_SHORT_RANGE 65534.0

7.4.1.5 #define JAUS_UNSIGNED_INTEGER_RANGE 4294967294.0

7.4.1.6 #define JAUS_UNSIGNED_LONG_RANGE (double) 1.8446744073709552e19

7.4.1.7 #define JAUS_UNSIGNED_SHORT_RANGE 65535.0

7.4.2 Typedef Documentation

7.4.2.1 typedef uint32_t ieeeFpd32

7.4.2.2 typedef uint64_t ieeeFpd64

7.5 src/homogeneous/homogeneous.h File Reference

```
#include "libdrdc.h"
#include <float.h>
#include <math.h>
#include "matrix_ops.h"
```

Include dependency graph for homogeneous.h:

This graph shows which files directly or indirectly include this file:

Functions

- int HMatrixSetTranslation (const double *vector, double *hMatrix)

  Set the Translational Component of a Homogeneous Matrix.

- int HMatrixGetTranslation (const double *hMatrix, double *vector)

  Get the Translational Component of a Homogeneous Matrix.

- int RotVectorToHMatrix (const double theta, const double *vector, double *hMatrix)

  Rotation Vector to Homogeneous Matrix.

- int HMatrixToRotVector (const double *hMatrix, double *theta, double *vector)

  Homogeneous Matrix to Rotation Vector.

- int EulerRPYToHMatrix (const double *rpyVector, double *hMatrix)

  Euler Angle RPY to Homogeneous Matrix.

- int HMatrixToEulerRPY (const double *hMatrix, double *rpyVector)

  Homogeneous Matrix to Euler Angle RPY.

- int HMatrixNormalize (double *hMatrix)

  Normalize a Homogeneous Matrix.

- int RotAxisToHMatrix (Axis_t axis, const double theta, double *hMatrix)

  Rotation about an Axis to Homogeneous Matrix.

- int HMatrixInverse (double *hMatrix)

  Inverse a Homogeneous Matrix.

- int HMatrixLink (const double linkLength, const double twistAngle, const double jointOffset, const double jointAngle, double *hMatrix)

  Homogeneous Matrix from Manipulator Linkage Specification.

7.6 src/libdrdc.h File Reference

```
#include <stdio.h>
#include "except.h"
```

Include dependency graph for libdrdc.h:

This graph shows which files directly or indirectly include this file:

Classes

- union DoubleUnion_t

  Union for Double precision Floating Point Data.

- union FloatUnion_t

  Union for Single precision Floating Point Data.

Defines

- #define PI 3.14159265358979323846

  Local constants defines.

- #define HALF_PI 1.57079632679489661923
- #define TWO_PI 6.28318530717958647693
- #define DOUBLE_FUZZ 2.2204460492503131e-16
- #define RADIAN_FUZZ 0.000001
- #define DRDC_BIG_ENDIAN 0
- #define deg2rad PI / 180.0

  Radians to/from Degrees conversions -DRE 2008.

- #define rad2deg 180.0 / PI
- #define DEG2RAD(x) (x ∗ deg2rad)

  Degrees to Radians DEG2RAD -DRE 2008.

- #define RAD2DEG(x) (x ∗ rad2deg)

  Radians to Degrees RAD2DEG -DRE 2008.

- #define SQ(x) ((x)∗(x))

  Square defined as SQ() macro.

- #define CLOSE(a, b, eps) ((fabs((a) - (b)) < (eps)) ? 1 : 0)

- #define DOUBLE_IS_ZERO(d) ((fabs(d) < DOUBLE_FUZZ) ? 1 : 0)
- #define UNRECOGNIZED_UNIT (11)
- #define UNRECOGNIZED_ENUM_ITEM (12)
- #define DIVIDED_BY_ZERO (13)
- #define BLAS_TRANSPOSE_OPTION_ERROR (14)
- #define CALLOC_NULL_ERROR (15)
- #define UNRECOGNIZED_INDEX (16)
- #define UNINITILIZED_UTM_ZONE (17)

Typedefs

- typedef unsigned char uint8_t
- typedef char sint8_t
- typedef unsigned short uint16_t
- typedef short sint16_t
- typedef unsigned long uint32_t
- typedef long sint32_t
- typedef unsigned long long uint64_t
- typedef long long sint64_t

Enumerations

- enum Axis_t { AXIS_ERR, AXIS_X, AXIS_Y, AXIS_Z }

Functions

- define_exception_type (int)

Variables

- struct exception_context the_exception_context [1]

7.6.1 Define Documentation

7.6.1.1 #define BLAS_TRANSPOSE_OPTION_ERROR (14)

7.6.1.2 #define CALLOC_NULL_ERROR (15)

7.6.1.3 #define CLOSE(a, b, eps) ((fabs((a) - (b)) < (eps)) ? 1 : 0)

7.6.1.4 #define DEG2RAD(x) (x * deg2rad)

Degrees to Radians DEG2RAD -DRE 2008.

Parameters:

x,: degree value to convert to radians.

7.6.1.5 #define deg2rad PI / 180.0

Radians to/from Degrees conversions -DRE 2008.

7.6.1.6 #define DIVIDED_BY_ZERO (13)

7.6.1.7 #define DOUBLE_FUZZ 2.2204460492503131e-16

7.6.1.8 #define DOUBLE_IS_ZERO(d) ((fabs(d) < DOUBLE_FUZZ) ? 1 : 0)

7.6.1.9 #define DRDC_BIG_ENDIAN 0

7.6.1.10 #define HALF_PI 1.57079632679489661923

7.6.1.11 #define PI 3.14159265358979323846

Local constants defines.

7.6.1.12 #define RAD2DEG(x) (x * rad2deg)

Radians to Degrees RAD2DEG -DRE 2008.

Parameters:

x,: radian value to convert to degrees.

7.6.1.13 #define rad2deg 180.0 / PI

7.6.1.14 #define RADIAN_FUZZ 0.000001

7.6.1.15 #define SQ(x) ((x)∗(x))

Square defined as SQ() macro.

7.6.1.16 #define TWO_PI 6.28318530717958647693

7.6.1.17 #define UNINITILIZED_UTM_ZONE (17)

7.6.1.18 #define UNRECOGNIZED_ENUM_ITEM (12)

7.6.1.19 #define UNRECOGNIZED_INDEX (16)

7.6.1.20 #define UNRECOGNIZED_UNIT (11)

7.6.2 Typedef Documentation

7.6.2.1 typedef short sint16_t

7.6.2.2 typedef long sint32_t

7.6.2.3 typedef long long sint64_t

7.6.2.4 typedef char sint8_t

7.6.2.5 typedef unsigned short uint16_t

7.6.2.6 typedef unsigned long uint32_t

7.6.2.7 typedef unsigned long long uint64_t

7.6.2.8 typedef unsigned char uint8_t

7.6.3 Enumeration Type Documentation

7.6.3.1 enum Axis_t

Enumerator:

AXIS_ERR

AXIS_X

AXIS_Y
AXIS_Z

7.6.4 Function Documentation

7.6.4.1 define_exception_type (int)

7.6.5 Variable Documentation

7.6.5.1 struct exception_context the_exception_context[1]

7.7 src/matrix_ops/matrix_ops.h File Reference

```
#include "libdrdc.h"
#include <math.h>
#include "except.h"
```

Include dependency graph for matrix_ops.h:

This graph shows which files directly or indirectly include this file:

Functions

- int vectorCompare (const int n, const double *vector1, const double *vector2, const double *fuzzVector)

  Vector Comparison.

- double vectorDot (const int n, const double *vector1, const double *vector2)

  Vector Dot Product.

- double vectorMagnitude (const int n, const double *vector)

  Vector Magnitude.

- void vectorNormalize (const int n, double *vector)

  Unitize Vector.

- void vectorCross (const double *vector1, const double *vector2, double *resultVector)

  Vector Cross Product.

- void vectorScale (const int n, const double alpha, double *vector)

  Vector Scaling.

- void vectorCopy (const int n, const double *srcVector, double *destVector)

  Copy Vector.

- void vectorAxpy (const int n, const double alpha, const double *x, double *y)

  Scalar-Vector Product.

- void vectorZeros (const int n, double *vector)

  Zero Vector.

- void vectorSplit (const int n, const int m, const double *orgVector, double *vector1, double *vector2)

Split Vector.

- void vectorJoin (const int n, const int m, const double *vector1, const double *vector2, double *resultVector)

  Join Vector.

- void matrixMvp (const char transA, const double alpha, const double *a, const double *x, const double beta, double *y)

  4x4 Matrix-Vector Product

- void matrixMmp (const char transA, const char transB, const double alpha, const double *a, const double *b, const double beta, double *c)

  4x4 Matrix-Matrix Product

- double matrixDeterminant (const double *matrix)

  4x4 Matrix Determint

- void matrixTranspose (double *matrix)

  4x4 Matrix Transpose

- void matrixIdentity (double *matrix)

  4x4 Identity Matrix.

- double ddot_ (const int *n, const double *dx, const int *incx, const double *dy, const int *incy)
- double dnrm2_ (const int *n, const double *x, const int *incx)
- int dscal_ (const int *n, const double *da, double *dx, const int *incx)
- int dcopy_ (const int *n, const double *dx, const int *incx, double *dy, const int *incy)
- int daxpy_ (const int *n, const double *da, const double *dx, const int *incx, double *dy, const int *incy)
- int dgemv_ (const char *trans, const int *m, const int *n, const double *alpha, const double *a, const int *lda, const double *x, const int *incx, const double *beta, double *y, const int *incy)
- int dgemm_ (const char *transa, const char *transb, const int *m, const int *n, const int *k, const double *alpha, const double *a, const int *lda, const double *b, const int *ldb, const double *beta, double *c, const int *ldc)

7.7.1 Function Documentation

7.7.1.1 int daxpy_ (const int * *n*, const double * *da*, const double * *dx*, const int * *incx*, double * *dy*, const int * *incy*)

7.7.1.2 int dcopy_ (const int * *n*, const double * *dx*, const int * *incx*, double * *dy*, const int * *incy*)

7.7.1.3 double ddot_ (const int * *n*, const double * *dx*, const int * *incx*, const double * *dy*, const int * *incy*)

7.7.1.4 int dgemm_ (const char * *transa*, const char * *transb*, const int * *m*, const int * *n*, const int * *k*, const double * *alpha*, const double * *a*, const int * *lda*, const double * *b*, const int * *ldb*, const double * *beta*, double * *c*, const int * *ldc*)

7.7.1.5 int dgemv_ (const char * *trans*, const int * *m*, const int * *n*, const double * *alpha*, const double * *a*, const int * *lda*, const double * *x*, const int * *incx*, const double * *beta*, double * *y*, const int * *incy*)

7.7.1.6 double dnrm2_ (const int * *n*, const double * *x*, const int * *incx*)

7.7.1.7 int dscal_ (const int * *n*, const double * *da*, double * *dx*, const int * *incx*)

7.8 src/quaternion/quaternion.h File Reference

```
#include <math.h>
#include "libdrdc.h"
#include "matrix_ops.h"
```

Include dependency graph for quaternion.h:

This graph shows which files directly or indirectly include this file:

Functions

- int EulerRPYToQuat (const double *rpyVector, double *quaternion)
  RPY to Quaternion
- int QuatToEulerRPY (const double *quaternion, double *rpyVector)
  Quaternion to RPY
- int QuatQuatMult (double *quaternion1, double *quaternion2, double *resultQuaternion)
  Quaternion Multiplication
- int QuatQuatDiv (double *quaternion1, double *quaternion2, double *resultQuaternion)
  Quaternion Division
- int QuatInverse (double *quaternion)
  Quaternion Inversion
- int QuatConjugate (double *quaternion)
  Conjugate Quaternion.
- double QuatMagnitude (const double *quaternion)
  Quaterion Magnitude.
- int QuatVectorMult (double *quaternion, double *vector, double *resultVector)
  Quaternion and Vector Product
- int HMatrixToQuat (const double *hMatrix, double *quaternion)
  Quaternion to Homogeneous Transformation
- int QuatToHMatrix (const double *quaternion, double *hMatrix)

Homogeneous Transformation to Quaternion.

- int QuatNormalize (double *quaternion)

  Unitize a Quaternion.

- int RotAxisToQuat (const Axis_t axis, const double theta, double *quaternion)

  Rotation about an Axis to Quaternion.

- int QuatToRotVector (const double *quaternion, double *theta, double *vector)

  Quaternion to Rotation Vector

- int RotVectorToQuat (const double theta, const double *vector, double *quaternion)

  Rotation Vector to Quaternion.

7.9 src/time_conv/time_conv.h File Reference

```
#include "libdrdc.h"
#include <time.h>
#include <sys/time.h>
```

Include dependency graph for time_conv.h:

Classes

- struct JausTime_t

  JAUS Timestamp and Datestamp Structure.

Defines

- #define JAUS_TIME_STAMP_SIZE_BYTES JAUS_UNSIGNED_INTEGER_SIZE_BYTES
- #define JAUS_DATE_STAMP_SIZE_BYTES JAUS_UNSIGNED_SHORT_SIZE_BYTES
- #define JAUS_TIME_STAMP_MILLISEC_SHIFT 0
- #define JAUS_TIME_STAMP_SECOND_SHIFT 10
- #define JAUS_TIME_STAMP_MINUTE_SHIFT 16
- #define JAUS_TIME_STAMP_HOUR_SHIFT 22
- #define JAUS_TIME_STAMP_DAY_SHIFT 27
- #define JAUS_DATE_STAMP_DAY_SHIFT 0
- #define JAUS_DATE_STAMP_MONTH_SHIFT 5
- #define JAUS_DATE_STAMP_YEAR_SHIFT 9
- #define JAUS_TIME_STAMP_MILLISEC_MASK 0x1FF
- #define JAUS_TIME_STAMP_SECOND_MASK 0x3F
- #define JAUS_TIME_STAMP_MINUTE_MASK 0x3F
- #define JAUS_TIME_STAMP_HOUR_MASK 0x1F
- #define JAUS_TIME_STAMP_DAY_MASK 0x1F
- #define JAUS_DATE_STAMP_DAY_MASK 0x1F
- #define JAUS_DATE_STAMP_MONTH_MASK 0x0F
- #define JAUS_DATE_STAMP_YEAR_MASK 0x7F

Functions

- struct timeval timeUtcToGps (const short leapSec, const struct timeval *utcTime)

UTC time to GPS time.

- struct timeval timeGpsToUtc (const short leapSec, const struct timeval *gpsTime)

  GPS time to UTC time.

- int jausTimeConvert (const struct timeval *timeVal, JausTime_t *time)

  Convert Time to Jaus Timestamp and Datestamp.

- int jausTimeStampPack (JausTime_t *time)

  Pack Jaus Timestamp.

- int jausDateStampPack (JausTime_t *time)

  Pack Jaus Datestamp.

- int jausTimeToString (JausTime_t *time, char *buffer)

  Convert Jaus Time Structure to String.

- int jausTimeStampUnpack (JausTime_t *time)

  Unpack Jaus Timestamp.

- int jausDateStampUnpack (JausTime_t *time)

  Unpack Jaus Datestamp.

7.9.1 Define Documentation

7.9.1.1 #define JAUS_DATE_STAMP_DAY_MASK 0x1F

7.9.1.2 #define JAUS_DATE_STAMP_DAY_SHIFT 0

7.9.1.3 #define JAUS_DATE_STAMP_MONTH_MASK 0x0F

7.9.1.4 #define JAUS_DATE_STAMP_MONTH_SHIFT 5

7.9.1.5 #define JAUS_DATE_STAMP_SIZE_BYTES JAUS_UNSIGNED_SHORT_SIZE_BYTES

7.9.1.6 #define JAUS_DATE_STAMP_YEAR_MASK 0x7F

7.9.1.7 #define JAUS_DATE_STAMP_YEAR_SHIFT 9

7.9.1.8 #define JAUS_TIME_STAMP_DAY_MASK 0x1F

7.9.1.9 #define JAUS_TIME_STAMP_DAY_SHIFT 27

7.9.1.10 #define JAUS_TIME_STAMP_HOUR_MASK 0x1F

7.9.1.11 #define JAUS_TIME_STAMP_HOUR_SHIFT 22

7.9.1.12 #define JAUS_TIME_STAMP_MILLISEC_MASK 0x1FF

7.9.1.13 #define JAUS_TIME_STAMP_MILLISEC_SHIFT 0

7.9.1.14 #define JAUS_TIME_STAMP_MINUTE_MASK 0x3F

7.9.1.15 #define JAUS_TIME_STAMP_MINUTE_SHIFT 16

7.9.1.16 #define JAUS_TIME_STAMP_SECOND_MASK 0x3F

7.9.1.17 #define JAUS_TIME_STAMP_SECOND_SHIFT 10

7.9.1.18 #define JAUS_TIME_STAMP_SIZE_BYTES JAUS_UNSIGNED_INTEGER_SIZE_BYTES

7.10 src/unit_conv/unit_conv.h File Reference

```
#include "libdrdc.h"
#include <float.h>
#include <string.h>
```

Include dependency graph for unit_conv.h:

Classes

- struct UnitConv_t

  Unit Conversion Structure.

Defines

- #define VOLUME_UNITS_NUM 7
- #define DISTANCE_UNITS_NUM 8
- #define MASS_UNITS_NUM 5
- #define VELOCITY_UNITS_NUM 6
- #define AREA_UNITS_NUM 7

Functions

- double unitToSiVolume (const char *unit)

  Converts the volume units to the SI unit - m^3.

- double unitToSiDistance (const char *unit)

  Converts the distance units to the SI unit - m.

- double unitToSiMass (const char *unit)

  Converts the mass units to the SI unit - kg .

- double unitToSiVelocity (const char *unit)

  Converts the velocity units to the SI unit - m/s.

- double unitToSiArea (const char *unit)

  Converts the area units to the SI unit - m^2.

- double unitConvertVolume (const char *srcUnit, const char *destUnit)

  Converts between volume units.

- double unitConvertDistance (const char *srcUnit, const char *destUnit)
  Converts between distance units.
- double unitConvertMass (const char *srcUnit, const char *destUnit)
  Converts between mass units.
- double unitConvertVelocity (const char *srcUnit, const char *destUnit)
  Converts between velocity units.
- double unitConvertArea (const char *srcUnit, const char *destUnit)
  Converts between area units.

7.10.1 Define Documentation

7.10.1.1 #define AREA_UNITS_NUM 7

7.10.1.2 #define DISTANCE_UNITS_NUM 8

7.10.1.3 #define MASS_UNITS_NUM 5

7.10.1.4 #define VELOCITY_UNITS_NUM 6

7.10.1.5 #define VOLUME_UNITS_NUM 7

7.11 src/wrapper/wrapper.h File Reference

```
#include "libdrdc.h"
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "matrix_ops.h"
#include "homogeneous.h"
#include "quaternion.h"
#include "coordinate.h"
```

Include dependency graph for wrapper.h:

Classes

- struct drdc_UTMZone_t
  UTM Zone Structure
- struct drdc_DMS_t
  Degree-Minute-Second Structure.
- struct drdc_Datum_t
  Global Datum
- struct drdc_LinkSpecs_t
  Manipulator Linkage Specification.
- struct drdc_Vector_t
  Generic Vector Type.
- struct drdc_lPosn_t
  OOC Local Position Representation.
- struct drdc_WUTMPosn_t
  OOC World UTM Position Representation.
- struct drdc_WPosn_t
  OOC World Position Representation.

- struct drdc_eOrient_t

  OOC Euler RPY Representation.

- struct drdc_Orient_t

  OOC Quaternion Representation.

- struct drdc_HMatrix_t

  OOC Homogeneous Matrix Representation.

- struct drdc_WPose_t

  OOC World Pose Representation.

- struct drdc_lPose_t

  OOC Local Pose Representation.

- struct drdc_WUTMPose_t

  OOC World UTM Pose Representation.

- struct drdc_lePose_t

  OOC Local Euler Pose Representation.

- class drdc_lPosn

  C++ Local Position Representation.

- class drdc_WUTMPosn

  C++ World UTM Position Representation.

- class drdc_WPosn

  C++ World Position Represenation.

- class drdc_eOrient

  C++ Euler RPY Representation.

- class drdc_Orient

  C++ Quaternion Representation.

- class drdc_HMatrix

  C++ Homogeneous Matrix Representation.

- class drdc_lPose

  C++ Local Pose Representation.

- class drdc_lePose

  C++ Local Euler Pose Representation.

- class drdc_WPose

  C++ World Pose Representation.

- class drdc_WUTMPose

  World UTM Pose Representation.

Enumerations

- enum drdc_Joint_t { JOINT_TYPE_ERR, JOINT_REVOLUTE, JOINT_PRISMATIC }

Functions

- int wp_compare (void *self, void *object, void *fuzzObject)
- int wp_compareUTM (void *self, void *object, void *fuzzObject)
- double wp_index (void *self, unsigned char n)
- void wp_setFromArray (void *self, const double *array)
- void wp_toArray (void *self, double *array)
- void wp_destroy (void *self)
- void wp_copy (void *self, void *dest)
- void wp_copyUTM (void *self, void *dest)
- void wp_set3 (void *self, double arg0, double arg1, double arg2)
- void wp_print (void *self)
- void wp_set4 (void *self, double arg0, double arg1, double arg2, double arg3)
- double wp_getArg0 (void *self)
- double wp_getArg1 (void *self)
- double wp_getArg2 (void *self)
- double wp_getArg3 (void *self)
- void wp_plus (void *self, void *object, void *resultObject)
- void wp_minus (void *self, void *object, void *resultObject)
- void wp_negate (void *self, void *resultObject)
- void wp_scale (void *self, double scalar, void *resultObject)
- double wp_dot (void *self, void *object)
- void wp_cross (void *self, void *object, void *resultObject)
- void wp_normalize (void *self)
- double wp_magnitude (void *self)
- void wp_lPosn_to_WUTMPosn (void *self, drdc_UTMZone_t zone, void *worldUTMPosition)

- void wp_lPose_to_WUTMPose (void *self, drdc_UTMZone_t zone, void *worldUTMPose)
- void wp_lPosn_to_WPosn (void *self, double falseElevation, void *worldPosition)
- void wp_WUTMPosn_to_WPosn (void *self, void *worldPosition)
- void wp_WUTMPosn_setZone (void *self, drdc_UTMZone_t zone)
- void wp_WUTMPosn_getZone (void *self, drdc_UTMZone_t *zone)
- void wp_WUTMPose_setZone (void *self, drdc_UTMZone_t zone)
- void wp_WUTMPose_getZone (void *self, drdc_UTMZone_t *zone)
- void wp_WPosn_to_lPosn (void *self, double falseElevation, void *localPosition)
- void wp_WPosnNormalize (void *self)
- void wp_WPosn_to_WUTMPosn (void *self, void *worldUTMPosition)
- void wp_eOrient_to_HMatrix (void *self, void *homogeneous)
- void wp_eOrient_to_Orient (void *self, void *quaternion)
- void wp_QuatMult (void *self, void *object, void *resultObject)
- void wp_QuatMultVec (void *self, void *object, void *resultObject)
- void wp_QuatDiv (void *self, void *object, void *resultObject)
- void wp_QuatInverse (void *self, void *resultObject)
- double wp_QuatMagnitude (void *self)
- void wp_QuatNormalize (void *self)
- void wp_Orient_to_eOrient (void *self, void *eulerRPY)
- void wp_Orient_to_HMatrix (void *self, void *homogeneous)
- void wp_QuatToRotVec (void *self, double *theta, void *unitVector)
- void wp_QuatFromRotVec (void *self, double theta, void *unitVector)
- void wp_QuatIdentity (void *self)
- void wp_QuatRotAxis (void *self, const Axis_t axis, const double theta)
- void wp_HMatrixMult (void *self, void *object, void *resultObject)
- void wp_HMatrixMultVec (void *self, void *object, void *resultObject)
- void wp_HMatrixToRotVec (void *self, double *theta, void *unitVector)
- void wp_HMatrixFromRotVec (void *self, double theta, void *unitVector)
- void wp_HMatrixNormalize (void *self)
- void wp_HMatrix_to_eOrient (void *self, void *eulerRPY)
- void wp_HMatrix_to_Orient (void *self, void *quaternion)
- void wp_HMatrixIdentity (void *self)
- void wp_HMatrixRotAxis (void *self, const Axis_t axis, const double theta)
- void wp_HMatrixTranspose (void *self, void *resultObject)
- void wp_HMatrixGetTranslation (void *self, void *resultObject)
- void wp_HMatrixSetTranslation (void *self, void *object)
- void wp_HMatrixFromDH (void *self, double linkLength, double twistAngle, double jointOffset, double jointAngle)
- void wp_HMatrixFromLink (void *self, drdc_LinkSpecs_t *linkSpecs, double jointOffsetOrAngle)

- void wp_HMatrixInverse (void *self, void *resultObject)
- void wp_setPose (void *self, void *position, void *orientation)
- void wp_getPosition (void *self, void *position)
- void wp_getOrientation (void *self, void *orientation)
- void wp_poseMult (void *self, void *object, void *resultObject)
- void wp_poseMultVec (void *self, void *object, void *resultObject)
- void wp_WPose_to_WUTMPose (void *self, void *worldUTMPose)
- void wp_WPose_to_lPose (void *self, double falseElevation, void *localPose)
- void wp_lPose_to_WPose (void *self, double falseElevation, void *worldPose)
- void wp_lPose_to_lePose (void *self, void *localEulerPose)
- void wp_lPose_to_HMatrix (void *self, void *homogeneous)
- void wp_WUTMPose_to_WPose (void *self, void *worldPose)
- void wp_lePose_to_lPose (void *self, void *localPose)
- void wp_lePose_to_HMatrix (void *self, void *homogeneous)
- void wp_HMatrix_to_lPose (void *self, void *localPose)
- void wp_HMatrix_to_lePose (void *self, void *localEulerPose)
- drdc_Vector_t * new_Vector (unsigned char dsize)
- drdc_lPosn_t * new_lPosn ()

  Local Position Constructor.

- drdc_WUTMPosn_t * new_WUTMPosn ()

  World Position (UTM) Constructor.

- drdc_WPosn_t * new_WPosn ()

  World Position Constructor.

- drdc_eOrient_t * new_eOrient ()

  Euler RPY Constructor.

- drdc_Orient_t * new_Orient ()

  Quaternion Constructor.

- drdc_HMatrix_t * new_HMatrix ()

  Homogeneous Constructor.

- drdc_WPose_t * new_WPose ()

  World Pose Constructor.

- drdc_lPose_t * new_lPose ()

  Local Pose Constructor.

- drdc_WUTMPose_t * new_WUTMPose ()

World Pose (UTM) Constructor.

- drdc_lePose_t * new_lePose ()
  Local Pose (Euler RPY) Constructor.

7.11.1 Enumeration Type Documentation

7.11.1.1 enum drdc_Joint_t

Enumerator:

JOINT_TYPE_ERR

JOINT_REVOLUTE

JOINT_PRISMATIC

7.11.2 Function Documentation

7.11.2.1 drdc_eOrient_t* new_eOrient ()

Euler RPY Constructor.

Returns:

7.11.2.2 drdc_HMatrix_t* new_HMatrix ()

Homogeneous Constructor.

Returns:

7.11.2.3 drdc_lePose_t* new_lePose ()

Local Pose (Euler RPY) Constructor.

Returns:

7.11.2.4 drdc_lPose_t* new_lPose ()

Local Pose Constructor.

Returns:

7.11.2.5 drdc_lPosn_t* new_lPosn ()

Local Position Constructor.

Returns:

7.11.2.6 drdc_Orient_t* new_Orient ()

Quaternion Constructor.

Returns:

7.11.2.7 drdc_Vector_t* new_Vector (unsigned char *dsize*)

7.11.2.8 drdc_WPose_t* new_WPose ()

World Pose Constructor.

Returns:

7.11.2.9 drdc_WPosn_t* new_WPosn ()

World Position Constructor.

Returns:

7.11.2.10 drdc_WUTMPose_t* new_WUTMPose ()

World Pose (UTM) Constructor.

Returns:

7.11.2.11 drdc_WUTMPosn_t* new_WUTMPosn ()

World Position (UTM) Constructor.

Returns:

7.11.2.12 int wp_compare (void * *self*, void * *object*, void * *fuzzObject*)

7.11.2.13 int wp_compareUTM (void * *self*, void * *object*, void * *fuzzObject*)

7.11.2.14 void wp_copy (void * *self*, void * *dest*)

7.11.2.15 void wp_copyUTM (void * *self*, void * *dest*)

7.11.2.16 void wp_cross (void * *self*, void * *object*, void * *resultObject*)

7.11.2.17 void wp_destroy (void * *self*)

7.11.2.18 double wp_dot (void * *self*, void * *object*)

7.11.2.19 void wp_eOrient_to_HMatrix (void * *self*, void * *homogeneous*)

7.11.2.20 void wp_eOrient_to_Orient (void * *self*, void * *quaternion*)

7.11.2.21 double wp_getArg0 (void * *self*)

7.11.2.22 double wp_getArg1 (void * *self*)

7.11.2.23 double wp_getArg2 (void * *self*)

7.11.2.24 double wp_getArg3 (void * *self*)

7.11.2.25 void wp_getOrientation (void * *self*, void * *orientation*)

7.11.2.26 void wp_getPosition (void * *self*, void * *position*)

7.11.2.27 void wp_HMatrix_to_eOrient (void * *self*, void * *eulerRPY*)

7.11.2.28 void wp_HMatrix_to_lePose (void * *self*, void * *localEulerPose*)

7.11.2.29 void wp_HMatrix_to_lPose (void * *self*, void * *localPose*)

7.11.2.30 void wp_HMatrix_to_Orient (void * *self*, void * *quaternion*)

7.11.2.31 void wp_HMatrixFromDH (void * *self*, double *linkLength*, double *twistAngle*, double *jointOffset*, double *jointAngle*)

7.11.2.32 void wp_HMatrixFromLink (void * *self*, drdc_LinkSpecs_t * *linkSpecs*, double *jointOffsetOrAngle*)

7.11.2.33 void wp_HMatrixFromRotVec (void * *self*, double *theta*, void * *unitVector*)

7.11.2.34 void wp_HMatrixGetTranslation (void * *self*, void * *resultObject*)

7.11.2.35 void wp_HMatrixIdentity (void * *self*)

7.11.2.36 void wp_HMatrixInverse (void * *self*, void * *resultObject*)

7.11.2.37 void wp_HMatrixMult (void * *self*, void * *object*, void * *resultObject*)

Index