Tele-Robótica
Departamento de Ingeniería de Sistemas y Automática
Universidad Politécnica de Valencia

EUROPEAN COMMISSION. DIRECTORATE GENERAL - JRC
Joint Research Centre – Ispra
Institute for the Protection and the Security of the Citizen (IPSC)

virtualrobot

VIRTUAL ROBOT AUXILIARY LIBRARIES

October 2012, Version 6.7b (VRAULS 6.7b)

Developed by

robótica
Department of System Engineering and Control (DISA)
Polytechnic University of Valencia (UPV)

Contact Address

vrs@isa.upv.es
http://robotica.isa.upv.es/virtualrobot
DISA - UPV
Camino de Vera s/n E-46022, Valencia (Spain)
Tf: +34-963.879.575 Fax: +34-963.879.579

Supported by

European Commission. Directorate General - JRC
Joint Research Centre - Ispra
Institute for the Protection and the Security of the Citizen (IPSC)

Index

1. Objective

The objective of this document is to explain three auxiliary libraries developed to increase the possibilities to program *Virtual Robot Simulator* (*VRS*) from a user process using VRPROL or VREAL. The libraries are:

- VRMaths for mathematics usually required in robotics applications
- VRTransf for operations with transformations
- VRStdIO for Standard Data Input/Output operations

2. Implementation

Every library has a .h definition file, a .lib library file and a .dll execution file. The first two files must be used for programming, compilation and linkation while the third must be used for execution.

3. VRMaths

This library has the following definitions, mathematics functions and trigonometric functions:

- Definitions:

```
#define RET_OK                      0
#define RET_ERROR                   1

#define FALSE                       0
#define TRUE                        1

// General Errors Definitions
#define VR_MATHS_ERROR              1000
#define INVALID_SIN_VALUE           VR_MATHS_ERROR  + 1
#define INVALID_COS_VALUE           VR_MATHS_ERROR  + 2

#define EPSILON                     1E-12
#define PI     3.14159265358979323846264338327950288419716939937511
#define HALF_PI                     (PI/2.0)
#define PI2                         (PI*2.0)
#define DEG_TO_RAD                  (PI/180.0)
#define RAD_TO_DEG                  (180.0/PI)
#define ROOT_OF_2        1.41421356237309504880168872420969
#define ROOT_OF_3        1.73205080756887729352744634150587
```

- Mathematic Functions:

```
double vrSign(double x);
```
This function computes sign(x) returning +1 if x is positive or -1 if x is negative.

```
double vrAbs(double x);
```
This function computes absolute value of x.

- Trigonometric Functions:

```
double vrAtan2(double y, double x);
```
This function computes arctan(y/x) according to quadrant.
The value is returned in radians.

```
int vrInvSin(double sinx, double *angle);
```
This function computes positive solution of inverse Sin. The angle is considered in radians. `RET_ERROR` is returned if input data is not valid.

```
int vrInvCos(double cosx, double *angle);
```
This function computes positive solution of inverse Cos. The angle is considered in radians. `RET_ERROR` is returned if input data is not valid.

4. VRTransf

This library has the following definitions and transformation functions:

- Definitions:

```
#define RET_OK                      0
#define RET_ERROR                   1
#define FALSE                       0
#define TRUE                        1

// General Errors Definitions
#define VR_TRANSF_ERROR             2000
#define INVALID_NO_POINTS           VR_TRANSF_ERROR + 1
#define INVALID_TRANSFORMATION      VR_TRANSF_ERROR + 2
#define INVALID_AXIS                VR_TRANSF_ERROR + 3
#define INVALID_EULER_TYPE          VR_TRANSF_ERROR + 4
#define INVALID_OPERATION           VR_TRANSF_ERROR + 5
#define INVALID_QUATERNION          VR_TRANSF_ERROR + 6

#define MAX_NO_POINTS               100

// Data Types
#define EULER_ANGLES_TYPE_1         1
#define EULER_ANGLES_TYPE_2         2
#define EULER_ANGLES_TYPE_3         3
#define TRANSFORMATION_TYPE         4
#define QUATERNIONS                 5

// Axis Types
#define X_AXIS                      0
#define Y_AXIS                      1
#define Z_AXIS                      2
#define U_AXIS                      3
#define V_AXIS                      4
#define W_AXIS                      5

// Operations Type
#define TRANSLATION                 0
#define ROTATION                    1

// Access To Euler Locations [6]
#define X_TRANSLATION               0
#define Y_TRANSLATION               1
#define Z_TRANSLATION               2
#define ALPHA_ANGLE                 3
#define BETA_ANGLE                  4
#define GAMMA_ANGLE                 5
```

- Transformation Functions:

```
void SetFilterConfig(
     double FilterValue,
     int DoFilter)
```
This function sets the internal value of the filter used in the functions of the librart internally. When a transformation is returned by a funtion, a filter is passed to avoid small values. The `DoFilter` parameter indicates if this filtering is made in these functiones or not. The `FilterValue` parameter specifies the tolerance value of the filter. The default value is 10E-6 with filter active.

```
void GetFilterConfig(
     double *FilterValue,
     int *DoFilter)
```
This function gets the library configuration about internal filtering.

```
void FilterTransformation(
     double transformation[4][4],
     double epsilon,
     double resulttransformation[4][4])
```
Sets all values between –epsilon to epsilon to zero, and copy the rest of the values. The `resulttransformation` is the output parameter.

```
void InitTransformation(
     double transformation[4][4]);
```
Initialize a transformation
Transformation is a homogeneous transformation matrix 4x4
transformation is the output parameter

```
void AssignTransformations(
     double transformation[4][4],
     double resulttransformation[4][4]);
```
Assigns a transformation to other transformation:
resulttransformation=transformation

```
int  TransformationsEqual(
     double Atransformation[4][4],
     double Btransformation[4][4],
     double tolerance=1e-6);
```
Compares two transformations and returns (Atransformation=Btransformation)
A tolerance is used for comparation.
When the call to this function omits the tolerance, a default value of 10E-6 is used

```
void InvertTransformation(
     double transformation[4][4],
     double invtransformation[4][4]);
```
Invert a transformation: invtransformation=transformation^{-1}
Transformation is a homogeneous transformation matrix 4x4
transformation is the input parameter, invtransformation is the output parameter

```
void ComposeTransformations(
     double Atransformation[4][4],
     double Btransformation[4][4],
     double ABtransformation[4][4]);
```
Compute transformation composition:
ABtransformation=Atransformation*Btransformation
Transformation is a homogeneous transformation matrix 4x4
Atransformation and Btransformation are the input parameters,
ABtransformation is the output parameter

```
void ComposeThreeTransformations(
     double Atransformation[4][4],
     double Btransformation[4][4],
     double Ctransformation[4][4],
     double ABCtransformation[4][4]);
```
Compute transformation composition:
ABCtransformation=Atransformation*Btransformation*Ctransformation
Transformation is a homogeneous transformation matrix 4x4
Atransformation, Btransformation and Ctransformation are the input parameters, ABCtransformation is the output parameter

```
void ComposeFourTransformations(
     double Atransformation[4][4],
     double Btransformation[4][4],
     double Ctransformation[4][4],
     double Dtransformation[4][4],
     double ABCDtransformation[4][4]);
```
Compute transformation composition:
ABCDtransformation=
=Atransformation*Btransformation*Ctransformation*Dtransformation
Transformation is a homogeneous transformation matrix 4x4
Atransformation, Btransformation, Ctransformation and Dtransformation are the input parameters, ABCDtransformation is the output parameter

```
void ApplyXYZTranslation(
     double transformation[4][4],
     double x, double y, double z,
     double offsettransformation[4][4]);
```

Apply a translation related to fixed frame OXYZ:
offsettransformation=DX,x*DY,y*DZ,z*transformation
Transformation is a homogeneous transformation matrix 4x4
transformation and x,y,z are the input parameters, offsettransformation is the output parameter

```
void ApplyUVWTranslation(
     double transformation[4][4],
     double x, double y, double z,
     double offsettransformation[4][4]);
```

Apply a translation related to mobile frame OUVW:
offsettransformation=transformation*DU,x*DV,y*DW,z
Transformation is a homogeneous transformation matrix 4x4
transformation and x,y,z are the input parameters, offsettransformation is the output parameter

```
int  ApplyTranslation(
     double transformation[4][4],
     int axis,
     double value,
     double offsettransformation[4][4]);
```

Apply a translation related to the axis indicated (`X_AXIS, Y_AXIS, Z_AXIS, U_AXIS, V_AXIS, W_AXIS`).
`transformation` is an homogeneous transformation matrix 4x4.
An error can be returned if axis type is invalid.
`Transformation`, `axis` and `value` are the input parameters, `offsettransformation` is the output parameter.

```
void ApplyXRotation(
     double transformation[4][4],
     double alpha,
     double rotatedtransformation[4][4]);
```

Apply a rotation related to fixed axis X:
rotatedtransformation=RX,alpha*transformation
Transformation is a homogeneous transformation matrix 4x4
The angle alpha is IN DEGREES
transformation and alpha are the input parameters, rotatedtransformation is the output parameter

```
void ApplyYRotation(
     double transformation[4][4],
     double beta,
     double rotatedtransformation[4][4]);
```

Apply a rotation related to fixed axis Y:
rotatedtransformation=RY,beta*transformation
Transformation is a homogeneous transformation matrix 4x4
The angle beta is IN DEGREES
transformation and beta are the input parameters, rotatedtransformation is the output parameter

```
void ApplyZRotation(
     double transformation[4][4],
     double gamma,
     double rotatedtransformation[4][4]);
```

Apply a rotation related to fixed axis Z:
rotatedtransformation=RZ,gamma*transformation
Transformation is a homogeneous transformation matrix 4x4
The angle gamma is IN DEGREES
transformation and gamma are the input parameters, rotatedtransformation is the output parameter

```
void ApplyURotation(
     double transformation[4][4],
     double alpha,
     double rotatedtransformation[4][4]);
```

Apply a rotation related to fixed axis U:
rotatedtransformation=transformation*RU,alpha
Transformation is a homogeneous transformation matrix 4x4
The angle alpha is IN DEGREES
transformation and alpha are the input parameters, rotatedtransformation is the output parameter

```
void ApplyVRotation(
     double transformation[4][4],
     double beta,
     double rotatedtransformation[4][4]);
```

Apply a rotation related to fixed axis V:
rotatedtransformation=transformation*RV,beta
Transformation is a homogeneous transformation matrix 4x4
The angle beta is IN DEGREES
transformation and beta are the input parameters, rotatedtransformation is the output parameter

```
void ApplyWRotation(
     double transformation[4][4],
     double gamma,
     double rotatedtransformation[4][4]);
```

Apply a rotation related to fixed axis W:
rotatedtransformation=transformation*RW,gamma
Transformation is a homogeneous transformation matrix 4x4
The angle gamma is IN DEGREES
transformation and gamma are the input parameters, rotatedtransformation is the output parameter

```
int  ApplyRotation(
     double transformation[4][4],
     int axis,
     double angle,
     double rotatedtransformation[4][4]);
```

Apply a rotation related to axis indicated (`X_AXIS, Y_AXIS, Z_AXIS, U_AXIS, V_AXIS, W_AXIS`).
`transformation` is an homogeneous transformation matrix 4x4.
The angle is IN DEGREES.
An error can be returned if axis type is invalid.
`transformation`, `axis` and `angle` are the input parameters, `rotatedtransformation` is the output parameter.

```
int  ApplyOperation(
     double transformation[4][4],
     int operation_type,
     int axis,
     double value,
     double outputtransformation[4][4]);
```

Apply the indicated Operation (`TRANSLATION, ROTATION`) related to axis indicated (`X_AXIS, Y_AXIS, Z_AXIS, U_AXIS, V_AXIS, W_AXIS`).
`transformation` is an homogeneous transformation matrix 4x4.
The value is IN DEGREES (for rotation angles).
An error can be returned if axis type is invalid.
`transformation`, `Operation_Type`, `axis` and `value` are the input parameters, `outputtransformation` is the output parameter

```
void ApplyTransformationToPoint(
     double transformation[4][4],
     double point[3],
     double transformedpoint[3]);
```

Apply a transformation to a point: transformedpoint=transformation*point
`transformation` is an homogeneous transformation matrix 4x4.
point and transformedpoint are 3D column vector points
transformation and point are the input parameters, transformedpoint is the output parameter

```
int  ApplyTransformationToSetofPoints(
     double transformation[4][4],
     int nopoints,
     double setofpoints[3][MAX_NO_POINTS],
     double transformedsetofpoints[3][MAX_NO_POINTS]);
```

Apply a transformation to a set of points:
transformedsetofpoints=transformation*setofpoints
`transformation` is an homogeneous transformation matrix 4x4.
nopoints is the number of points in setofpoints (must be in interval [1..MAX_NO_POINTS], otherwise an error is returned)
setofpoints and transformedsetofpoints are set of 3D column vector points in a matrix
transformation, nopoints and setofpoints are the input parameters, transformedsetofpoints is the output parameter

```
void LocationEuler1ToTransformation(
     double location[6],
     double transformation[4][4]);
```

Convert from Location to Transformation
`transformation` is an homogeneous transformation matrix 4x4.
`location` is an array representing position x,y,z (location[X_TRANSLATION], location[Y_TRANSLATION], location[Z_TRANSLATION]) and orientation (location[alpha_ANGLE], location[BETA_ANGLE], location[GAMMA_ANGLE]).
Orientation is represented with Euler angles alpha, beta and gamma (IN DEGREES)
Euler angles used are type 1, that is, representing the following rotation sequence:
1- Rotation of alpha degrees related to fixed Z axis
2- Rotation of beta degrees related to mobile U axis
3- Rotation of gamma degrees related to mobile W axis
location is the input parameter, transformation is the output parameter

```
int  TransformationToLocationEuler1(
     double transformation[4][4],
     double location[6]);
```

Convert from Transformation to Location
`transformation` is an homogeneous transformation matrix 4x4.
`location` is an array representing position x,y,z (location[X_TRANSLATION], location[Y_TRANSLATION], location[Z_TRANSLATION]) and orientation (location[alpha_ANGLE], location[BETA_ANGLE], location[GAMMA_ANGLE]).
Orientation is represented with Euler angles alpha, beta and gamma (IN DEGREES)
Euler angles used are type 1, that is, representing the following rotation sequence:
1- Rotation of alpha degrees related to fixed Z axis
2- Rotation of beta degrees related to mobile U axis
3- Rotation of gamma degrees related to mobile W axis
An error can be returned if there is no solution
transformation is the input parameter, location is the output parameter

```
void LocationEuler2ToTransformation(
     double location[6],
     double transformation[4][4]);
```

Convert from Location to Transformation

`transformation` is an homogeneous transformation matrix 4x4.

`location` is an array representing position x,y,z (location[X_TRANSLATION], location[Y_TRANSLATION], location[Z_TRANSLATION]) and orientation (location[alpha_ANGLE], location[BETA_ANGLE], location[GAMMA_ANGLE]).

Orientation is represented with Euler angles alpha, beta and gamma (IN DEGREES)

Euler angles used are type 2, that is, representing the following rotation sequence:

1- Rotation of alpha degrees related to fixed Z axis

2- Rotation of beta degrees related to mobile V axis

3- Rotation of gamma degrees related to mobile W axis

location is the input parameter, transformation is the output parameter

```
int  TransformationToLocationEuler2(
     double transformation[4][4],
     double location[6]);
```

Convert from Transformation to Location

`transformation` is an homogeneous transformation matrix 4x4.

`location` is an array representing position x,y,z (location[X_TRANSLATION], location[Y_TRANSLATION], location[Z_TRANSLATION]) and orientation (location[alpha_ANGLE], location[BETA_ANGLE], location[GAMMA_ANGLE]).

Orientation is represented with Euler angles alpha, beta and gamma (IN DEGREES)

Euler angles used are type 2, that is, representing the following rotation sequence:

1- Rotation of alpha degrees related to fixed Z axis

2- Rotation of beta degrees related to mobile V axis

3- Rotation of gamma degrees related to mobile W axis

An error can be returned if there is no solution

transformation is the input parameter, location is the output parameter

```
void LocationEuler3ToTransformation(
     double location[6],
     double transformation[4][4]);
```

Convert from Location to Transformation

`transformation` is an homogeneous transformation matrix 4x4.

`location` is an array representing position x,y,z (location[X_TRANSLATION], location[Y_TRANSLATION], location[Z_TRANSLATION]) and orientation (location[alpha_ANGLE], location[BETA_ANGLE], location[GAMMA_ANGLE]).

Orientation is represented with Euler angles alpha, beta and gamma (IN DEGREES)

Euler angles used are type 3 (RPY), that is, representing the following rotation sequence:

1- Rotation of alpha degrees related to fixed X axis

2- Rotation of beta degrees related to fixed Y axis

3- Rotation of gamma degrees related to fixed Z axis

location is the input parameter, transformation is the output parameter

```
int  TransformationToLocationEuler3(
     double transformation[4][4],
     double location[6]);
```

Convert from Transformation to Location

`transformation` is an homogeneous transformation matrix 4x4.

`location` is an array representing position x,y,z (location[X_TRANSLATION], location[Y_TRANSLATION], location[Z_TRANSLATION]) and orientation (location[alpha_ANGLE], location[BETA_ANGLE], location[GAMMA_ANGLE]).

Orientation is represented with Euler angles alpha, beta and gamma (IN DEGREES)

Euler angles used are type 3 (RPY), that is, representing the following rotation sequence:

1- Rotation of alpha degrees related to fixed X axis

2- Rotation of beta degrees related to fixed Y axis

3- Rotation of gamma degrees related to fixed Z axis

An error can be returned if there is no solution

transformation is the input parameter, location is the output parameter

```
int  LocationEulerToTransformation(
     int euler_type,
     double location[6],
     double transformation[4][4]);
```

Convert from Location to Transformation.

`transformation` is an homogeneous transformation matrix 4x4.

`location` is an array representing position x,y,z (location[X_TRANSLATION], location[Y_TRANSLATION], location[Z_TRANSLATION]) and orientation (location[alpha_ANGLE], location[BETA_ANGLE], location[GAMMA_ANGLE]).

Orientation is represented with euler angles alpha, beta and gamma (IN DEGREES). Euler angles used are defined by `euler_type` parameter

An error can be returned if `euler_type` is not valid.

`euler_type` and `location` are the input parameter, `transformation` is the output parameter.

```
int  TransformationToLocationEuler(
     int euler_type,
     double transformation[4][4],
     double location[6]);
```

Convert from Transformation to Location.

`transformation` is an homogeneous transformation matrix 4x4.

`location` is an array representing position x,y,z (location[X_TRANSLATION], location[Y_TRANSLATION], location[Z_TRANSLATION]) and orientation (location[alpha_ANGLE], location[BETA_ANGLE], location[GAMMA_ANGLE]).

Orientation is represented with euler angles alpha, beta and gamma (IN DEGREES). Euler angles used are defined by `euler_type` parameter

An error can be returned if there is no solution.

An error can be returned if `euler_type` is not valid.

`euler_type` and `transformation` are the input parameter, `location` is the output parameter.

```
int  LocationEulerToLocationEuler(
     int input_euler_type,
     int output_euler_type,
     double input_location[6],
     double output_location[6]);
```

Convert from a Euler Type to another.

`input_location and output_location` are arrays representing position x,y,z (location[X_TRANSLATION], location[Y_TRANSLATION], location[Z_TRANSLATION]) and orientation (location[alpha_ANGLE], location[BETA_ANGLE], location[GAMMA_ANGLE]).

Orientation is represented with euler angles alpha, beta and gamma (IN DEGREES). Euler angles used are defined by `euler_type` parameter

An error can be returned if there is no solution.

An error can be returned if `euler_type` is not valid.

`input_euler_type, output_euler_type` and input_location are the input parameters, `output_location` is the output parameter.

```
void LocQuaternionToTransformation(
     double Qlocation[7],
     double transformation[4][4]);
```

Convert from Quaternion Location to Transformation
transformation is a homogeneous transformation matrix 4x4
Qlocation is an array representing position x,y,z (from Qlocation[0] to Qlocation[2]) and orientation (from Qlocation[3] to Qlocation[6])
Orientation is represented with quaternions q1,q2,q3,q4
Qlocation is the input parameter, transformation is the output parameter

```
int TransformationToLocQuaternion(
     double transformation[4][4],
     double Qlocation[7]);
```

Convert from Transformation to Quaternion Location
transformation is a homogeneous transformation matrix 4x4
Qlocation is an array representing position x,y,z (from Qlocation[0] to Qlocation[2]) and orientation (from Qlocation[3] to Qlocation[6])
Orientation is represented with quaternions q1,q2,q3,q4
An error can be returned if there is no solution
transformation is the input parameter, Qlocation is the output parameter

- Transformation Class:

Based on the previous functions, a transformation class is defined with the following interface:

```
class __declspec (dllimport) CTransformation
{
public:
       // Construction & Destruction
       CTransformation();
       CTransformation(CTransformation &Transformation); //copy constructor
       CTransformation(double InitMatrix[4][4]);
       CTransformation(double InitMatrix[16]);
       ~CTransformation();

       // Own Functions
       void Initialize();
       void Invert();
       int Compare(CTransformation CompareTransformation);

       int SetEulerAngles(int EulerType, double x, double y, double z,
              double alpha, double beta, double gamma);
       int GetEulerAngles(int EulerType, double *x, double *y, double *z,
              double *alpha, double *beta, double *gamma);
       int SetEulerAngles(int EulerType, double EulerVector[6]);
       int GetEulerAngles(int EulerType, double EulerVector[6]);

       int GetMatrix(double Matrix[4][4]);
       int SetMatrix(double Matrix[4][4]);
       int GetMatrix(double Matrix[16]);
       int SetMatrix(double Matrix[16]);

       int SetString(char *TransformationString);
       int GetString(int EulerType, char *TransformationString);
```

```
      int AddOperation(int Kind, int Axis, double Value);

       /* Override operators */
      CTransformation operator = (CTransformation &SourceObject);
        /* (i) Adress of Original Object to be Copied */
       CTransformation operator * (CTransformation &SourceObject);

protected:
       // Internal functions
       void Filter(double FilterEpsilon = 1e-12);
       // Data Members
       double InternalMatrix[4][4];

       //  NOTE:
       //  Use for 16 component Vector
       //  | 0  4  8 12 |
       //  | 1  5  9 13 |
       //  | 2  6 10 14 |
       //  | 3  7 11 15 |
};
```

5. VRStdIO

This library has dialogs for data input/output. When this library is used in a proyect, **MFC must be used in a shared dll** (Project Setting option).

This library has the following definitions and functions:

- Definitions:

```
#DEFINE INPUT_OK                        0
#DEFINE INPUT_CANCEL                    1

#DEFINE INPUT_TYPE                      0
#DEFINE OUTPUT_TYPE                     1
#DEFINE INPUT_OUTPUT_TYPE               2

#DEFINE NUM_DOF                         12

TYPEDEF CHAR STRING[256]

#DEFINE VR_STDIO_ERROR                  3000
#DEFINE INVALID_DIALOG_TYPE             VR_STDIO_ERROR + 1
```

Every function of the library has the DialogType parameter, which controls the behaviour for the dialogs:

- If DialogType is `INPUT_TYPE`, the dialog allows the input of data.
- If DialogType is `OUTPUT_TYPE`, the dialog shows data.
- If DialogType is `INPUT_OUTPUT_TYPE`, the dialog shows data as default values and allows the input of data modifying the default values.

- Functions:

```
int DialogString(
     int DialogType,
     STRING QueryStr,
     STRING String);
```

This function opens a Dialog with `QueryStr` as name for `String` input, output or both depending on the `DialogType` parameter.
If the user cancels the input, the `INPUT_CANCEL` value is returned, otherwise, the `INPUT_OK` value is returned.

```
int DialogInteger(
     int DialogType,
     STRING QueryStr,
     int *IntegerNumber);
```

This function opens a Dialog with `QueryStr` as name for `IntegerNumber` input, output or both depending on the `DialogType` parameter.
If the user cancels the input, the `INPUT_CANCEL` value is returned, otherwise, the `INPUT_OK` value is returned.

```
int DialogDouble(
     int DialogType,
     STRING QueryStr,
     double *DoubleNumber);
```

This function opens a Dialog with `QueryStr` as name for `DoubleNumber` input, output or both depending on the `DialogType` parameter.
If the user cancels the input, the `INPUT_CANCEL` value is returned, otherwise, the `INPUT_OK` value is returned.

```
int DialogLocation(
     int DialogType,
     STRING QueryString,
     double* x, double* y, double* z,
     double* alpha, double* beta, double* gamma)
```

This function opens a Dialog with `QueryStr` as name for location (in `x,y,z,alpha,beta,gamma` format) input, output or both depending on the `DialogType` parameter.
If DialogType is `OUTPUT_TYPE` or `INPUT_OUTPUT_TYPE`, then the dialog allows the user to change the visualization of the Euler angles in the three Euler angles types.
If DialogType is `INPUT_TYPE`, the user can insert the values in all of the three Euler angles types.
By default, the function returns the data in `EULER_ANGLES_TYPE_2`.
If the user cancels the input, the `INPUT_CANCEL` value is returned, otherwise, the `INPUT_OK` value is returned.
Option
The function can be called with a last parameter that indicates the Euler angles type which the programmer wants to receive the data.
The interface is:

```
int DialogLocation(int DialogType, STRING QueryString,
     double* x, double* y, double* z,
     double* alpha, double* beta, double* gamma,
     int Euler_Type)
```

The Euler_Type parameter is defined as in VRTransf library.

```
int DialogLocation(
     int DialogType,
     STRING QueryString,
     double location[6])
```

This function has the same behaviour and options than the previous function, but the data is returned in a vector of 6 components.

```
int DialogTransformation(
     int DialogType,
     STRING QueryStr,
     double Transformation[4][4]);
```

This function opens a Dialog with `QueryStr` as name for `Transformation` input, output or both depending on the `DialogType` parameter.
If the user cancels the input, the `INPUT_CANCEL` value is returned, otherwise, the `INPUT_OK` value is returned.

```
int DialogJoints(
     int DialogType,
     STRING QueryStr,
     double joints[NUM_DOF],
     int NumberOfJoints);
```

This function opens a Dialog with `QueryStr` as name for `joints` (in array format) input, output or both depending on the `DialogType` parameter.
In the `NumberOfJoints` parameter the number of joints to be considered (up to `NUM_DOF`) is specified.
If the user cancels the input, the `INPUT_CANCEL` value is returned, otherwise, the `INPUT_OK` value is returned.

```
int DialogQuaternions(
     int DialogType,
     STRING QueryStr,
     double QLocation[7])
```

This function opens a Dialog with `QueryStr` as name for location (in an array with position and quaternion format) input, output or both depending on the `DialogType` parameter.
If the user cancels the input, the `INPUT_CANCEL` value is returned, otherwise, the `INPUT_OK` value is returned.