AUBO

Aubo SDK for C++

Documentation

AUBO(USA) Robotics Technology Co., Ltd.

Contents

I. Introduction

The AUBO SDK for C++ provides a C++ API for the AUBO robot. The API is based on the network, it provides plenty of interfaces that can be used to control the robot and related I/O.

The AUBO API uses physical quantities with the following units:

Length	meters
Angle	radians
Joint velocity	rad/s
Joint acceleration	rad/s^2

II. Getting started

2.1 Including the SDK in your project

2.1.1 Compilers and libraries

The AUBO C++ API works on the Linux, which is provided in libprotobuf.a, libour_alg_i5p.a, libauborobotcontroller.a, llog4cplus.so. Separate libraries are provided for 32-bit, x86 architectures and 64bit, x64 architectures.

2.1.2 Constructer and namespace

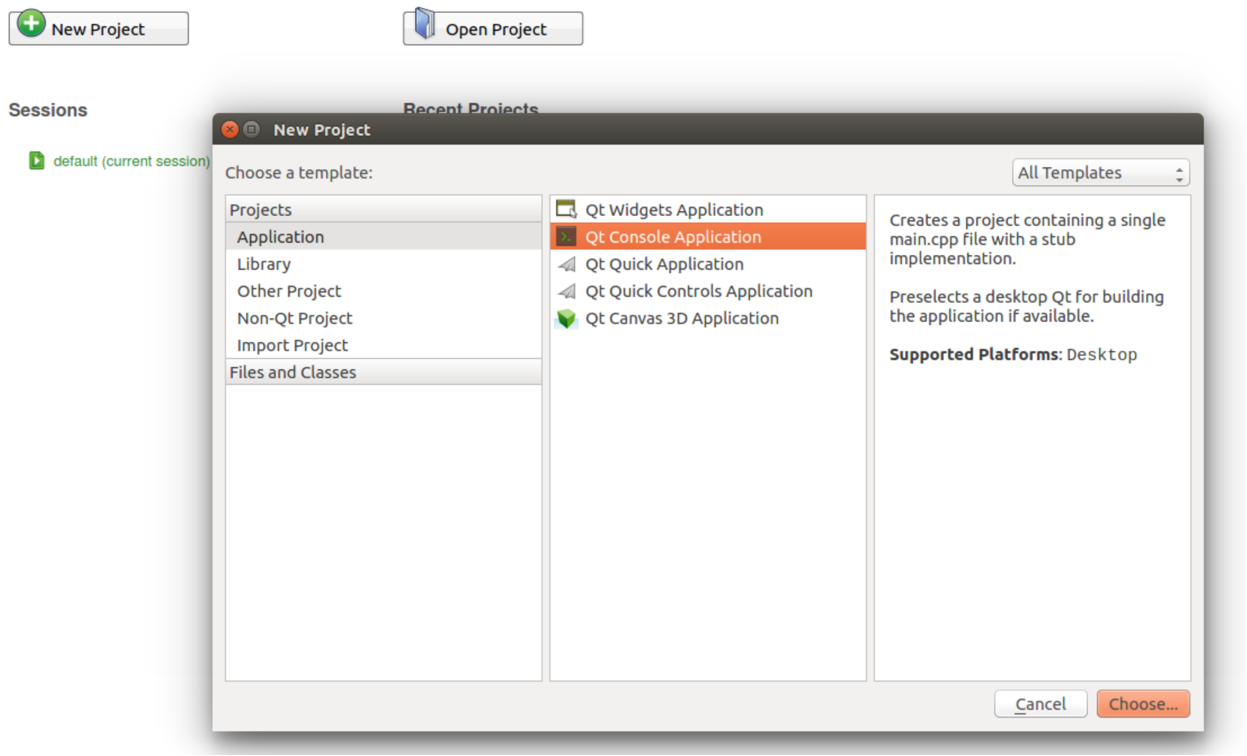
Constructer	ServiceInterface()
Namespace	aubo_robot_namespace

In the AUBO C++ API, the external interface class “ServiceInterface” provides all development interfaces for users, all the interfaces are declared in the <serviceinterface.h>. The API also gives a specific namespace “aubo_robot_namespace” to provides a scope to the identifiers and data structures.

2.1.3 An example

This example shows how to setup the programming environment in 32bit Linux by using Qt

1. create a new project

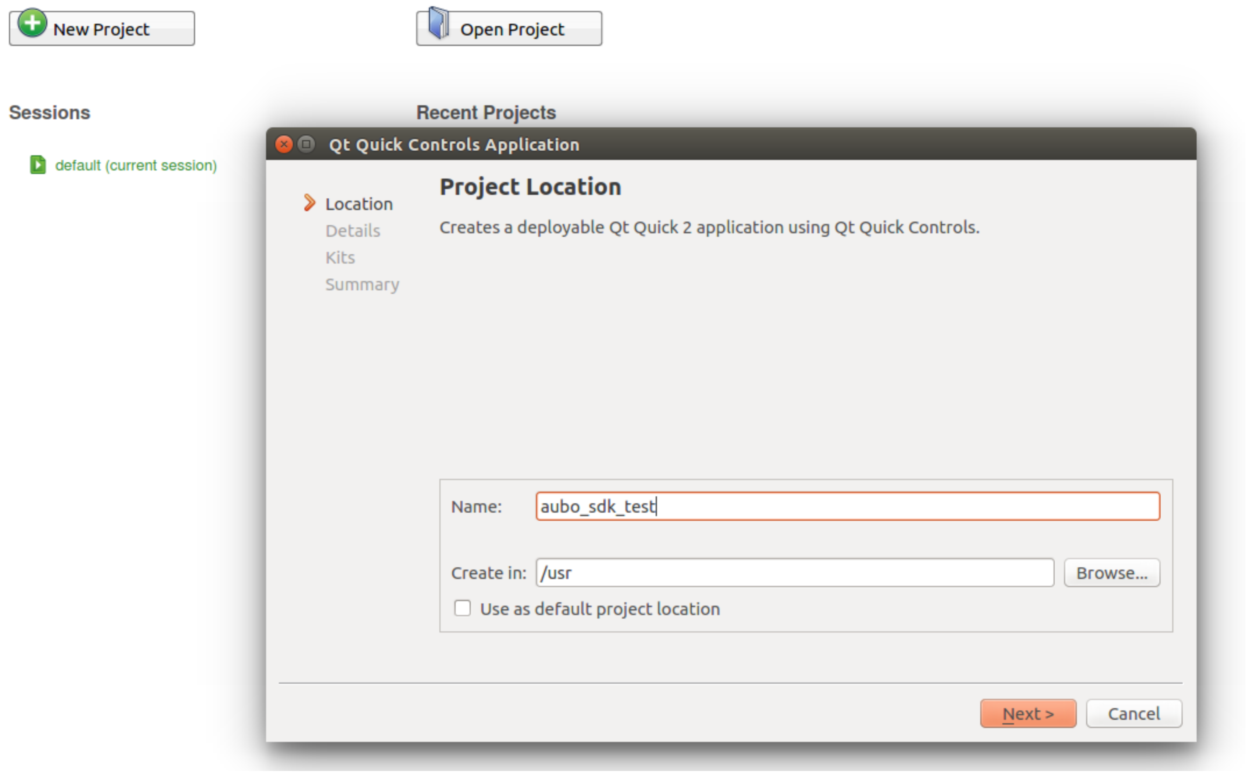


Creates a project containing a single main.cpp file with a stub implementation.

Preselects a desktop Qt for building the application if available.

Supported Platforms: Desktop

2. give the project a name



Project Location

Creates a deployable Qt Quick 2 application using Qt Quick Controls.

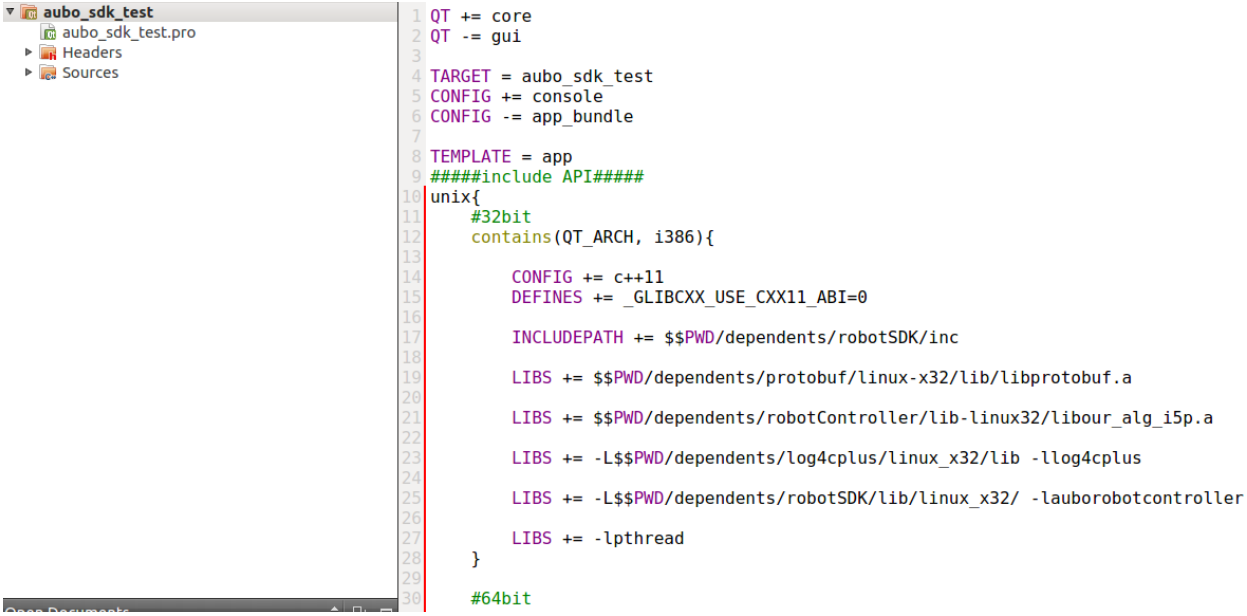
Name: aubo_sdk_test

Create in: /usr

☐ Use as default project location

3. Include Aubo API library, in QT, after putting the API file into the workspace, just paste the code below into

aubo_sdk_test.pro file.

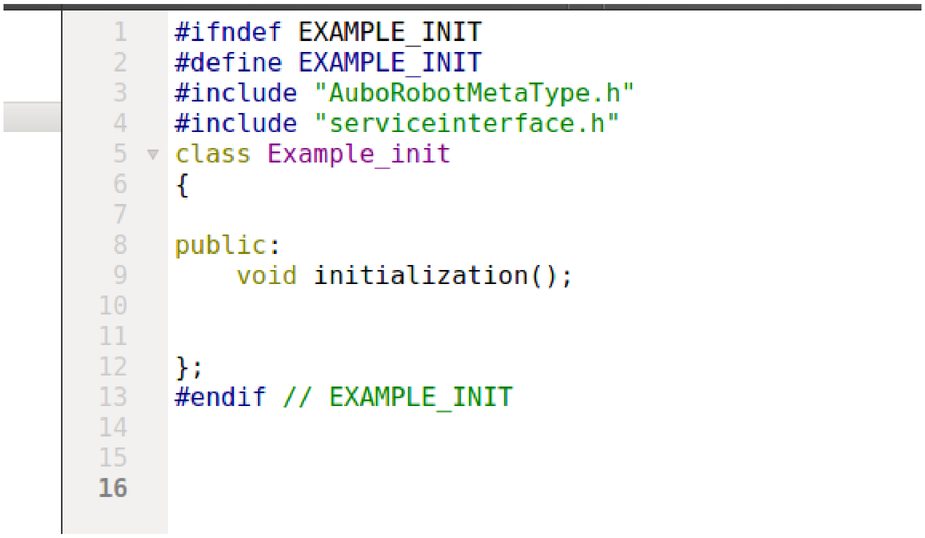


```
QT += core
QT -= gui

TARGET = aubo_sdk_test
CONFIG += console
CONFIG -= app_bundle

TEMPLATE = app
#####include API#####
unix{
    #32bit
    contains(QT_ARCH, i386){

        CONFIG += c++11
        DEFINES += _GLIBCXX_USE_CXX11_ABI=0

        INCLUDEPATH += $$PWD/dependents/robotSDK/inc

        LIBS += $$PWD/dependents/protobuf/linux-x32/lib/libprotobuf.a

        LIBS += $$PWD/dependents/robotController/lib-linux32/libour_alg_i5p.a

        LIBS += -L$$PWD/dependents/log4cplus/linux_x32/lib -llog4cplus

        LIBS += -L$$PWD/dependents/robotSDK/lib/linux_x32/ -lauborobotcontroller

        LIBS += -lpthread
    }

    #64bit
```

```
##********including API**********************************
unix{
    #32bit
    contains(QT_ARCH, i386){

        CONFIG += c++11
        DEFINES += _GLIBCXX_USE_CXX11_ABI=0

        INCLUDEPATH += $$PWD/dependents/robotSDK/inc

        LIBS += $$PWD/dependents/protobuf/linux-x32/lib/libprotobuf.a

        LIBS += $$PWD/dependents/robotController/lib-linux32/libour_alg_i5p.a

        LIBS += -L$$PWD/dependents/log4cplus/linux_x32/lib -llog4cplus

        LIBS += -L$$PWD/dependents/robotSDK/lib/linux_x32/ -lauborobotcontroller

        LIBS += -lpthread
    }

    #64bit
    contains(QT_ARCH, x86_64){

        CONFIG += c++11

        INCLUDEPATH += $$PWD/dependents/robotSDK/inc
        LIBS += -L$$PWD/dependents/protobuf/linux-x64/lib/ -lprotobuf

        LIBS += $$PWD/dependents/robotController/lib-linux64/libour_alg_i5p.a

        LIBS += -L$$PWD/dependents/log4cplus/linux_x64/lib -llog4cplus

        LIBS += -L$$PWD/dependents/robotSDK/lib/linux_x64/ -lauborobotcontroller

         LIBS += -L$$PWD/dependents/libconfig/linux_x64/lib/ -lconfig

        LIBS += -lpthread
    }
}
```

4. When write the program, remember to include the constructer and namespace

```
#include "AuboRobotMetaType.h"
#include "serviceinterface.h"
```



```
#ifndef EXAMPLE_INIT
#define EXAMPLE_INIT
#include "AuboRobotMetaType.h"
#include "serviceinterface.h"
class Example_init
{

public:
    void initialization();


};
#endif // EXAMPLE_INIT
```

For other IDE, the idea is the same. After including the specific library, the constructer and namespace, the API should be ready to use.

III. API reference

3.1 Connection and disconnection module

This interface includes establishing network connection with the manipulator server, disconnecting, and checking the current connection status. Successful building network connection with the server is a prerequisite for using other interfaces. The interface implementation uses the TCP/IP protocol

3.1.1 Interface login

```
int  robotServiceLogin(const char* host, int port, const char *userName, const char* possword);
```

Description: Establishing network connection with the manipulator sever

Parameters:

1. **host:** The IP address of the manipulator server

2. **port:** Port number of the robot server, default port is 8899

3. **username:** Default username: AUBO

4. **password:** Default password: 123456

Return: Successful call returns 0; errors return error code

3.1.2 Interface logout

```
int  robotServiceLogout();
```

Description: Disconnecting from the manipulator server

Return: Successful call returns 0; errors return error code

3.1.3 Connection status

```
void robotServiceGetConnectStatus(bool &connectStatus);
```

Description: Checking the current connection status with the manipulator server.

Parameters:

1. connectStatus: Output parameter, true means connect successful, false means disconnected

Return: Successful call returns 0; errors return error code

3.2 Manipulator initialization module

The initialization and close of the manipulator, initialization will power the manipulator and release the brake etc. Initialization of the manipulator is a prerequisite when work on the real robot.

3.2.1 Manipulator startup

```
int rootServiceRobotStartup(const aubo_robot_namespace::ToolDynamicsParam &toolDynamicsParam,
                            uint8   collisionClass,
                            bool readPose,
                            bool staticCollisionDetect,
                            int boardBaxAcc, aubo_robot_namespace::ROBOT_SERVICE_STATE  &result,
                            bool IsBolck = true);
```

Description: Initializing the manipulator, including power on, release the brake, set up the collision class, set up the kinematics parameters. This function needs a longtime to complete, so user can set its block mode to adjust the return time of the function. When it is set to unblocked mode, it will return immediately after calling function, the result will be notified by event. When it is set to block mode, return value represents whether the interface has been called successfully.

Parameters:

toolDynamicsParam: The kinematics parameters of the tool, if tool is loaded on the end-effector, the parameter should be set according to the actual needs; if tool is not loaded, each section of this parameter should be set to 0.

The Data type of ToolDynamicsParam is showed below

```
typedef struct
{
    double positionX;    //The x-coordinate of center of gravity of the tool

    double positionY;    //The y-coordinate of center of gravity of the tool

    double positionZ;    //The z-coordinate of center of gravity of the tool

    double payload;      //Tool weight

    ToolInertia toolInertia;  //Tool inertia

}ToolDynamicsParam;
```

Inside it, the data type of ToolInertia is

```
typedef struct
{
    double xx;
    double xy;
    double xz;
    double yy;
    double yz;
    double zz;
}ToolInertia;
```

2. **collisionClass:** The collision class of the manipulator

3. **readPose:** Whether interface can get position and pose, default is true

4. **staticCollisionDetect**: Whether the interface can detect collision, default is true

5. **boardBaxAcc:** default is 1000

6. **result:** The startup results of the manipulator showing result==ROBOT_SERVICE_WORKING represents that the manipulator start successful, otherwise, it means startup failed.

The struct of ROBOT_SERVICE_STATE is showed below

```
enum ROBOT_SERVICE_STATE{
    ROBOT_SERVICE_READY=0,
    ROBOT_SERVICE_STARTING,
    ROBOT_SERVICE_WORKING,
    ROBOT_SERVICE_CLOSING,
    ROBOT_SERVICE_CLOSED,
    ROBOT_SETVICE_FAULT_POWER,
    ROBOT_SETVICE_FAULT_BRAKE,
    ROBOT_SETVICE_FAULT_NO_ROBOT
};
```

7. **IsBolck:** To set the block mode when calling interface.

Return: Successful call returns 0; errors return error code

3.2.2 Manipulator shutdown

```
int  robotServiceRobotShutdown(bool IsBolck = true);
```

Description: Shutdown the manipulator, including power off, hold the brake

Parameters:

1. IsBolck: To set whether it is blocked when calling interface: setting blocked, the function will return until manipulator shutdown; setting unblocked, it will return immediately, result will be returned by event push.

Return: Successful call returns 0; errors return error code

3.2.3 Example

The example for connection and initialization module, please refer to the “Example_init” part of the example program

3.3 Motion module

The interfaces in this part are related to the robot movement, including the settings of the movement property and control robot movement. The relative movement property needs to be set before using.

3.3.1 Frequently-used data type in motion module

```
/** Movement mode enumeration  **/
enum move_mode
{
    NO_MOVEMODE = 0,
    MODEJ,
    MODEL,
    MODEP
};
```

Description: This data type enumerates three movement modes: Move Joint(MODEJ), Move Line(MODEL), and Move Track(MODEP). The Move Line and Move Track belong to the end type movement. The detail explanation about movement mode, please refer to the user manual of robot.

```
/** Coordinate system enumeration  **/ typedef struct
enum coordinate_refer
{
    BaseCoordinate = 0,
    EndCoordinate,
    WorldCoordinate
};
```

Description: This data type enumerates three coordinate systems: base coordinate system(BaseCoordinate), end effector coordinate system(EndCoordinate) and user defined coordinate system(WorldCoordinate）. The detail explanation about coordinate system, please refer to the user manual of robot.

```
typedef struct
{
    coordinate_refer    coordType;       //Coordinate type:  When coordType==BaseCoordinate or
coordType==EndCoordinate, the system does not process the following 3 parameters:

    CoordCalibrateMathod methods;        //Coordinate calibration method

    JointParam       wayPointArray[3];   //Used to calibrate the 3 points (joint angle) of the coordinate,
corresponding to center of the flange, based on the base coordinate.

    ToolInEndDesc    toolDesc;           //The tool description used in calibration

}CoordCalibrateByJointAngleAndTool;
```

Description: This data type shows the parameter that need to define a custom coordinate system. The "CoordCalibrateMathod" and "ToolInEndDesc" data types are introduced below

```
typedef struct
{
    Pos         toolInEndPosition;     //The tool position corresponding to the end coordinate

    Ori         toolInEndOrientation;  //the tool posture corresponding to the end coordinate
}ToolInEndDesc;
```

Description: This data type defines the tool position and posture corresponding to the end coordinate

```
/** Coordinate system calibration method enumeration **/
enum CoordCalibrateMathod
{
    Origin_AnyPointOnPositiveXAxis_AnyPointOnPositiveYAxis,            // The origin, the positive x-axis,
the y-axis
    Origin_AnyPointOnPositiveYAxis_AnyPointOnPositiveZAxis,            // The origin, the positive y-axis,
the z-axis
    Origin_AnyPointOnPositiveZAxis_AnyPointOnPositiveXAxis,            // The origin, the positive z-axis,
the x-axis
    Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrantOfXOYPlane,  // The origin, the positive x-axis
or any point of the first quadrant of the plane of the x and y axis
    Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrantOfXOZPlane,  // The origin, the positive x-axis
or any point of the first quadrant of the plane of the x and z axis
    Origin_AnyPointOnPositiveYAxis_AnyPointOnFirstQuadrantOfYOZPlane,  // The origin, the positive y-axis
or any point of the first quadrant of the  plane of the y and z axis
    Origin_AnyPointOnPositiveYAxis_AnyPointOnFirstQuadrantOfYOXPlane,  // The origin, the positive y-axis
or any point of the first quadrant of the plane of the y and x axis
    Origin_AnyPointOnPositiveZAxis_AnyPointOnFirstQuadrantOfZOXPlane,  // The origin, the positive z-axis
or any point of the first quadrant of the plane of the z and x axis
     Origin_AnyPointOnPositiveZAxis_AnyPointOnFirstQuadrantOfZOYPlane,  // The origin, the positive z-axis
or any point of the first quadrant of the plane of the z and y axis


    CoordTypeCount
};
```

Description: This data type enumerates the coordinate system calibration method, these data type is only used when define the user defined coordinate system

```
typedef struct
{
    bool   ena;                        //Whether it enable offset

    float relativePosition[3];         //offset x,y,z

    Ori    relativeOri;

}MoveRelative;
```

Description: This data type defines whether enable offset and the offset value for each axis

```
enum {ARM_DOF = 6};           /** Robot joint amount **/
```

Description: This data type defines the robot joint amount, default is 6.

```
typedef struct
{
    double jointPara[ARM_DOF];
}JointVelcAccParam;
```

```
typedef struct
{
    double jointPos[ARM_DOF];
}JointParam;
```

Description: These two struct define two arrays to describe the joint parameter. “JointVelcAccParam” is used to store the velocity and acceleration property of each joint. “JointParam” is used to store the joint angle of each joint.

```
/** The representation of the waypoint position information  **/
struct Pos
{
    double x;
    double y;
    double z;
};

/** The representation of the waypoint position information  **/
union cartesianPos_U
{
    Pos position;
    double positionVector[3];
};


/** The representation of the quaternion of the posture  **/
struct Ori
{
    double w;
    double x;
    double y;
    double z;
};

/** The representation of the Euler of the posture  **/
struct Rpy
{
    double rx;
    double ry;
    double rz;
};


/** This describes the waypoint of the robot  **/
typedef struct
{
    cartesianPos_U cartPos;     //Robot position information  X,Y,Z

    Ori orientation;            //Robot posture information, represented as quaternion, can be transformed
from Euler angle through tool function.

    double jointpos[ARM_DOF];   //Robot joint angle information
}wayPoint_S;
```

Description: These structs are used to define the waypoint. Waypoint is an important part of AUBO robot, which represents the position that the end effector needs to arrive. Usually the trajectory of the end consists of two or more waypoints. The waypoint is defined in “waypoint_S” struct, which includes position and posture of end effector, and the angle of each joint.

3.3.2 Initialize the movement property

```
int  robotServiceInitGlobalMoveProfile();
```

Description: Initialize the movement property, set properties to default.

Movement property:

1.The maximum velocity and acceleration of joint movement. The default maximum acceleration for each joint is 25 degrees/s2, the maximum velocity for each joint is 25 degrees/s. This property takes effect in the joint movement.

2.The maximum linear velocity and linear acceleration of end movement. The default maximum linear acceleration for each joint is 3m/s2, the maximum linear velocity for each joint is 3m/s. This property takes effect in the end movement type.

3. Waypoint information cache, it is used in trajectory movement, default is null.

4. Blend radius, it is used for the subtype of trajectory movement MOVEP, default is 0.02m.

5. Cycles of circle (valid when the track type is ARC_CIR, when its property (CircularLoopTimes)is 0, it is arc track; when its property (CircularLoopTimes)is greater than 0， it is circular track, default is 0)

6. The offset property of the movement property, default is no offset. This property takes effect is the non-teaching movement.

7. Tool parameter property

8. Set the coordinate system of the teaching movement, default is base coordinate system, This property takes effect in the teaching movement only.

Return: Successful call returns 0; errors return error code

3.3.3 Setting and obtaining the maximum velocity and acceleration of the joint movement

```
int  robotServiceSetGlobalMoveJointMaxAcc  (const aubo_robot_namespace::JointVelcAccParam &moveMaxAcc);
int  robotServiceSetGlobalMoveJointMaxVelc(const aubo_robot_namespace::JointVelcAccParam  &moveMaxVelc);
void robotServiceGetGlobalMoveJointMaxAcc  (aubo_robot_namespace::JointVelcAccParam        &moveMaxAcc);
void robotServiceGetGlobalMoveJointMaxVelc(aubo_robot_namespace::JointVelcAccParam        &moveMaxVelc);
```

Description: 1. Set the maximum acceleration of each joint.

2. Set the maximum velocity of each joint.

3. Obtain the maximum acceleration of each joint.

4. Obtain the maximum velocity of each joint.

The maximum velocity is no more than 180 degrees/s, and the maximum acceleration is not more than 180 degrees/s^2.

Parameters:

1. moveMaxAcc: The maximum acceleration of joint, the value is no more than π rad/s^2.

2. moveMaxVelc: The maximum velocity of joint, the value is no more than π rad/s

Return: Successful call returns 0; errors return error code

3.3.4 Setting and obtaining the maximum velocity and acceleration of the end type movement

```
int   robotServiceSetGlobalMoveEndMaxLineAcc  (double  moveMaxAcc);
int   robotServiceSetGlobalMoveEndMaxLineVelc(double  moveMaxVelc);
void  robotServiceGetGlobalMoveEndMaxLineAcc  (double  &moveMaxAcc);
void  robotServiceGetGlobalMoveEndMaxLineVelc(double  &moveMaxVelc);
```

Description: 1. Set the maximum acceleration of end effector movement.

2. Set the maximum velocity of end effector movement.

3. Obtain the maximum acceleration of end effector movement.

4. Obtain the maximum velocity of end effector movement.

Parameters:

1. moveMaxAcc: The maximum acceleration of end type movement, the value is no more than 5 m/s^2.

2. moveMaxVelc: The maximum velocity of end type movement, the value is no more than 5 m/s.

Return: Successful call returns 0; errors return error code

3.3.5 Setting and obtaining the maximum angular velocity and angular acceleration of the end type movement

```
int   robotServiceSetGlobalMoveEndMaxAngleAcc  (double  moveMaxAcc);
int   robotServiceSetGlobalMoveEndMaxAngleVelc(double  moveMaxVelc);
void  robotServiceGetGlobalMoveEndMaxAngleAcc  (double  &moveMaxAcc);
void  robotServiceGetGlobalMoveEndMaxAngleVelc(double  &moveMaxVelc);
```

Description: 1. Set the maximum angular acceleration of end effector movement.

2. Set the maximum angular velocity of end effector movement.

3. Obtain the maximum angular acceleration of end effector movement.

4. Obtain the maximum angular velocity of end effector movement.

The maximum velocity is no more than 180 degrees/s, and the maximum acceleration is not more than 180 $degrees/s^2$.

Parameters:

1. moveMaxAcc: The maximum angular acceleration of end type movement, the value is no more than π rad/s^2

2. moveMaxVelc: The maximum angular velocity of end type movement, the value is no more than π rad/s.

Return: Successful call returns 0; errors return error code

3.3.6 Settings and obtaining of waypoint in the movement property

```
int   robotServiceAddGlobalWayPoint(const aubo_robot_namespace::wayPoint_S &wayPoint);
int   robotServiceAddGlobalWayPoint(const double jointAngle[aubo_robot_namespace::ARM_DOF]);
void  robotServiceClearGlobalWayPointVector();
void  robotServiceGetGlobalWayPointVector(std::vector<aubo_robot_namespace::wayPoint_S>&wayPointVector);
```

Description: 1. Add the waypoint by using “waypoint_S” data type.

2. Add the waypoint by using the angle of each joint.

3. Clear the waypoint

4. Obtain the waypoint information

The waypoint is used in Trajectory movement(Move track mode).

Parameters:

1.wayPoint: Waypoint information

2.jointAngle: Angle of each joint

Return: Successful call returns 0; errors return error code

3.3.7 Settings and obtaining of the blend radius in movement property

```
float robotServiceGetGlobalBlendRadius();
int   robotServiceSetGlobalBlendRadius(float value);
```

Description: 1. Get the blend radius.

2. Set the blend radius.

Parameters:

Value: Blend radius, which range is from 0.0 m to 0.05m, the description of blend radius, please refer to the user manual.

Return: Successful call returns 0; errors return error code

3.3.8 Settings and obtaining of the cycles of circle in movement property

```
int   robotServiceGetGlobalCircularLoopTimes();
void  robotServiceSetGlobalCircularLoopTimes(int times);
```

Description: 1. Get the cycles of circle

2. Set the cycles of circle

It will take effect when the type of the track movement equals to ARC_CIR. When the cycles of the circle equal to 0, ARC_CIR represents arc. When the cycles of the circle are greater than 0, ARC_CIR represents circle.

3.3.9 Setting and Obtaining the coordinate system offset

```
int  robotServiceSetMoveRelativeParam(const aubo_robot_namespace::MoveRelative &relativeMoveOnBase);
```

Description: Set the offset based on the base coordinate system

Parameter:

1. relativeMoveonBase: The offset according to each axis of the base coordinate system

Return: Successful call returns 0; errors return error code

```
int  robotServiceSetMoveRelativeParam(const aubo_robot_namespace::MoveRelative   &relativeMoveOnUserCoord,
                                 const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool &userCoord);
```

Description: Set the offset based on the user defined coordinate system

Parameter:

1. relativeMoveonUserCoord: The offset according to each axis of the user define coordinate system, the data type is the same as above.

2. userCoord: The user defined coordinate system.

Return: Successful call returns 0; errors return error code

3.3.10 Joint movement

```
int robotServiceJointMove(aubo_robot_namespace::wayPoint_S   &wayPoint,     bool IsBolck);
```

```
int robotServiceJointMove(double jointAngle[aubo_robot_namespace::ARM_DOF], bool IsBolck);
```

Description: The manipulator moves to the target position through the joint movement (Move Joint), **the target position is described by the angle of each joint**. The maximum velocity and the maximum acceleration of the joint movement should be set before calling it, and the offset attribute should be set if the offset is used. The input has two option, one is the waypoint, or directly gives the angle of each joint.

Parameter:

1. waypoint: Waypoint information, the data type of “waypoint_S” is introduced above. This method only uses the joint angle of the waypoint information.

2. jointAngle: The angle of each joint.

3. IsBolck: To set the block mode when calling interface.

Return: Successful call returns 0; errors return error code.

```
int robotMoveJointToTargetPosition(
           const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool     &userCoord,
           const aubo_robot_namespace::Pos             &toolEndPositionOnUserCoord, //the target position
corresponding to the user coordinate
           const aubo_robot_namespace::ToolInEndDesc   &toolInEndDesc,
           bool  IsBolck = false);                                //blocked or not
```

Description: The manipulator moves to the target position through the joint movement (Move Joint), **the target**

position is described by the end effector's coordinates in the defined coordinate system. The maximum velocity and the maximum acceleration of the joint movement should be set before calling it, and the offset attribute should be set if the offset is used. The input has two option, one is the waypoint, or directly gives the angle of each joint.

Parameter:

1. userCoord: The user defined coordinate system.

2. ToolEndPositionOnUserCoord: The target position, which is the desired end effector's coordinates

3. ToolInEndDesc: The tool position and posture corresponding to the end coordinate

4. IsBolck: To set the block mode when calling interface.

Return: Successful call returns 0; errors return error code.

```
int robotMoveJointToTargetPositionByRelative(
             const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool &userCoord,
             const aubo_robot_namespace::MoveRelative    &relativeMoveOnUserCoord,     // the offset of
current position corresponding to the target position
             bool IsBolck = false);                                          //blocked or not
```

Description: The manipulator moves to the target position through the joint movement (Move Joint), **the target position is described by the offset of current position corresponding to the target position**. The maximum velocity and the maximum acceleration of the joint movement should be set before calling it, and the offset attribute should be set if the offset is used. The input has two option, one is the waypoint, or directly gives the angle of each joint.

Parameter:

1. userCoord: The user defined coordinate system.

2. relativeMoveOnUserCoord: The offset of current position corresponding to the target position

3. IsBolck: To set the block mode when calling interface.

4. Return: Successful call returns 0; errors return error code.

3.3.11 Linear movement

```
int robotServiceLineMove(aubo_robot_namespace::wayPoint_S   &wayPoint, bool IsBolck);
```

```
int robotServiceLineMove(double jointAngle[aubo_robot_namespace::ARM_DOF],  bool IsBolck);
```

Description: The manipulator moves to the target position through the linear movement (Move Line), **the target position is described by the angle of each joint.** The maximum linear velocity and the maximum linear acceleration should be set before calling it, and the offset attribute should be set if the offset is used. The input has two option, one is the waypoint, or directly gives the angle of each joint.

Parameter:

1. waypoint: Waypoint information, the data type of "waypoint_S" is introduced above. This method only uses the joint angle of the waypoint information.

2. jointAngle: The angle of each joint.

3. IsBolck: To set the block mode when calling interface.

Return: Successful call returns 0; errors return error code.

```
int robotMoveLineToTargetPosition(
             const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool  &userCoord,
             const aubo_robot_namespace::Pos              &positionOnUserCoord,
             const aubo_robot_namespace::ToolInEndDesc    &toolInEndDesc,
              bool IsBolck = false);                                        //blocked or not
```

Description: The manipulator moves to the target position through the linear movement (Move Line), **the target position is described by the end effector's coordinates in the defined coordinate system.** The maximum linear velocity and the maximum linear acceleration should be set before calling it, and the offset attribute should be set if the offset is used.

Parameter:

1. userCoord: The user defined coordinate system.

2. PositionOnUserCoord: The target position, which is the desired end effector's coordinates

3. ToolInEndDesc: The tool position and posture corresponding to the end coordinate

4. IsBolck: To set the block mode when calling interface.

Return: Successful call returns 0; errors return error code.

```
int robotMoveLineToTargetPositionByRelative(
             const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool &userCoord,
             const aubo_robot_namespace::MoveRelative     &relativeMoveOnUserCoord,      //the offset of
current position corresponding to the target position
             bool IsBolck);                                                //blocked or not
```

Description: The manipulator moves to the target position through the linear movement (Move Line), **the target position is described by the offset of current position corresponding to the target position.** The maximum linear velocity and the maximum linear acceleration should be set before calling it, and the offset attribute should be set if the offset is used.

Parameter:

1. userCoord: The user defined coordinate system.

2. relativeMoveOnUserCoord: The offset of current position corresponding to the target position

3. IsBolck: To set the block mode when calling interface.

4. Return: Successful call returns 0; errors return error code.

3.3.12 Rotary movement

```
int robotServiceRotateMove(const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool &userCoord, const
double rotateAxisOnUserCoord[3], double rotateAngle,  bool IsBolck);
```

Description: The end effector of the manipulator rotates around the specified axis and keep the current position.

Parameter:

1. userCoord: The user defined coordinate system

2. rotateAxisOnUserCoord: The free vector under the user coordinate system (It can be understood as the vector that corresponds to the origin of the coordinate system to the same value coordinate point). The user defined coordinate system is determined by the first parameter "userCoord".

3. rotateAngle: The rotation angle rotates around specified axis.

4. IsBolck: To set the block mode when calling interface.

Return: Successful call returns 0; errors return error code.

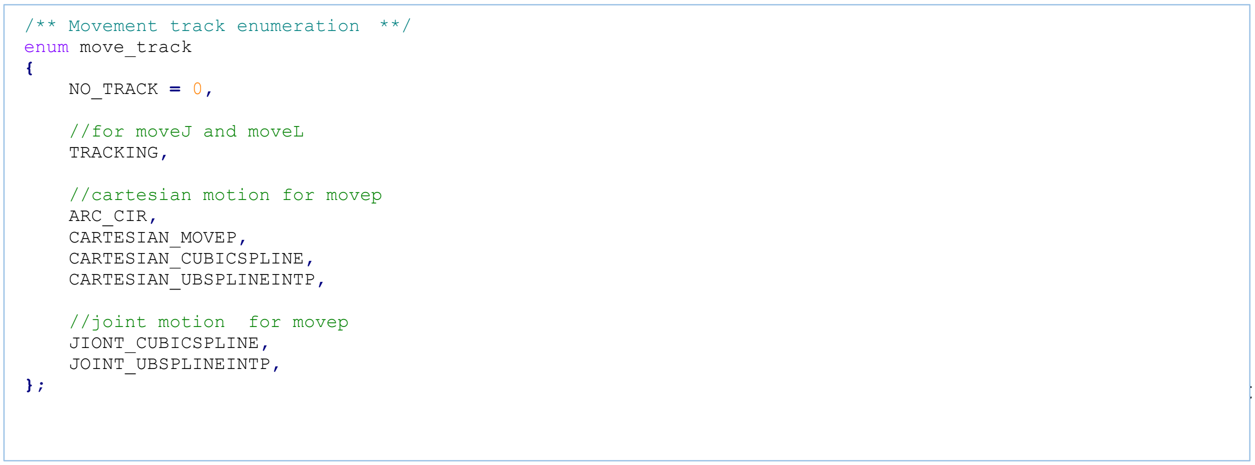
3.3.13 Trajectory movement

```
int robotServiceTrackMove(aubo_robot_namespace::move_track subMoveMode,     bool IsBolck);
```

Description: The track movement of the manipulator. This movement is belonged to different movement type according to the different type of subMoveMode.

Parameter:

1. subMoveMode: It belongs to move_track data type



```
/** Movement track enumeration  **/
enum move_track
{
    NO_TRACK = 0,

    //for moveJ and moveL
    TRACKING,

    //cartesian motion for movep
    ARC_CIR,
    CARTESIAN_MOVEP,
    CARTESIAN_CUBICSPLINE,
    CARTESIAN_UBSPLINEINTP,

    //joint motion  for movep
    JIONT_CUBICSPLINE,
    JOINT_UBSPLINEINTP,
};
```

When **subMoveMode==JIONT_CUBICSPLINE, JOINT_UBSPLINEINTP**, it is joint movement.

When **subMoveMode==ARC_CIR**, it represents circle or arc.

When the cycles property of the circle(CircularLoopTimes)is 0, it is arc track

When the cycles property of the circle(CircularLoopTimes)is greater than 0, it is circle track

When **subMoveMode==CARTESIAN_MOVEP**(MOVEPtrack), user needs to set the property of blend radius.

2. IsBolck: To set the block mode when calling interface.

Return: Successful call returns 0; errors return error code.

3.3.14 The startup and stop of the teaching movement

```
int robotServiceTeachStart(aubo_robot_namespace::teach_mode mode, bool direction);
/** @brief  finish teaching*/
int robotServiceTeachStop();
```

Description: 1. Start the teaching movement

2. Stop the teaching movement

Parameter:

1. mode: It belongs to teach_mode data type. The teach_mode data type is showed below

```
enum teach_mode
{
    NO_TEACH = 0,
    JOINT1,
    JOINT2,
    JOINT3,
    JOINT4,
    JOINT5,
    JOINT6,
    MOV_X,
    MOV_Y,
    MOV_Z,
    ROT_X,
    ROT_Y,
    ROT_Z
};
```

The joint movement teaching: JOINT1, JOINT2, JOINT3, JOINT4, JOINT5, JOINT6,

Position teaching: MOV_X, MOV_Y, MOV_Z

Pose teaching: ROT_X, ROT_Y, ROT_Z;

2. direction: True: Positive direction

False: Negative direction

Return: Successful call returns 0; errors return error code.

3.3.15 Robot movement control: stop, pause, continue

```
int  rootServiceRobotMoveControl(aubo_robot_namespace::RobotMoveControlCommand cmd);
```

Description: Control the movement by using control command

Parameter:

1.cmd: Robot control command, which belongs to the RobotMoveControlCommand datatype

```
typedef enum {
    RobotMoveStop     = 0,
    RobotMovePause    = 1,
    RobotMoveContinue = 2,
}RobotMoveControlCommand;
```

It can stop, pause or continue the robot movement

Return: Successful call returns 0; errors return error code.

3.3.16 Offline trajectory movement

```
int robotServiceOfflineTrackWaypointAppend(
                        const std::vector<aubo_robot_namespace::wayPoint_S> &wayPointVector);
```

Description: Add an offline waypoint to the server through the waypoint container

Parameters:

1. wayPointVector: Waypoint container, the container allows up to 4,000 waypoints.

Return: Successful call returns 0; errors return error code.

```
int robotServiceOfflineTrackWaypointAppend(const char *fileName);
```

Description: Add an offline waypoint to the server through the form of waypoint file

Parameters:

1. filename: Waypoint file directory

Return: Successful call returns 0; errors return error code.

```
int robotServiceOfflineTrackWaypointClear ();
```

Description: Remove waypoint

Return: Successful call returns 0; errors return error code.

```
int robotServiceOfflineTrackMoveStartup   (bool  IsBolck);

int robotServiceOfflineTrackMoveStop      ();
```

Description: 1. Start the offline track movement

2. Stop the offline track movement

Parameters:

1. IsBolck: To set the block mode when calling interface

Return: Successful call returns 0; errors return error code.

3.3.17 Arrival ahead mode

```
int  robotServiceSetArrivalAheadDistanceMode(double distance);
```

Description: Set the arrival ahead mode by using distance to smooth the trajectory

Parameters:

1. distance: The distance (meter).

Return: Successful call returns 0; errors return error code.

```
int  robotServiceSetArrivalAheadTimeMode(double second /*sec*/);
```

Description: Set the arrival ahead mode by using time to smooth the trajectory

Parameters:

1. distance: The time (second).

Return: Successful call returns 0; errors return error code.

```
int  robotServiceSetNoArrivalAhead();
```

Description: Turn off the arrival ahead mode

Return: Successful call returns 0; errors return error code.

3.3.18 Example

The example for motion module, please refer to the “Example_movement” part of the example program

3.4 Real-time manipulator information push module

The interfaces are mainly about the push of the manipulator information. The information push of the manipulator in the interface is implemented by the callback function. The developer needs to define the callback function and register the defined callback function into the system using the following interface to implement the information push. This part of the interface includes the push of real-time joint information, the push of real-time waypoint information, the push of real-time end velocity, and the push of the event information of the manipulator. The following interfaces are used to register the user-defined callback function into our system.

3.4.1 Data type for call back function

```
typedef void (*RealTimeJointStatusCallback)(const aubo_robot_namespace::JointStatus *jointStatus, int size, 
void *arg);
```

Description: Define the function pointer for the Joint status information push

Parameters:

1. jointStatus: It belongs to JointStatus data type, contains serval properties of the manipulator

```
typedef struct PACKED
{
    int     jointCurrentI;       /**< Current of driver   Joint current*/
    int     jointSpeedMoto;      /**< Speed of driver     Joint velocity*/
    float   jointPosJ;           /**< Current position in radian   Joint angle*/
    float   jointCurVol;         /**< Rated voltage of motor. Unit: mV   Joint voltage*/
    float   jointCurTemp;        /**< Current temperature of joint        Current temperature*/
    int     jointTagCurrentI;    /**< Target current of motor            The target current of motor*/
    float   jointTagSpeedMoto;   /**< Target speed of motor              The target velocity of motor*/
    float   jointTagPosJ;        /**< Target position of joint in radian The target joint angle */
    uint16 jointErrorNum;        /**< Joint error of joint num          Joint error code */
}JointStatus;
```

2. size: The size of the parameter(jointStatus)

3. arg: The second parameter that passed by the user

Return: Successful call returns 0; errors return error code.

```
typedef void (*RealTimeRoadPointCallback) (const aubo_robot_namespace::wayPoint_S  *wayPoint, void *arg);
```

Description: Define the function pointer for the waypoint information push

Parameters:

1. waypoint: The waypoint information

2. arg: The second parameter that passed by the user

Return: Successful call returns 0; errors return error code.

```
typedef void (*RealTimeEndSpeedCallback)  (double speed, void *arg);
```

Description: Define the function pointer for the end velocity push

Parameters:

1. speed: The current end velocity

2. arg: The second parameter that passed by the user

Return: Successful call returns 0; errors return error code.

```
typedef void (*RobotEventCallback)    (const aubo_robot_namespace::RobotEventInfo *eventInfo, void *arg);
```

Description: Define the function pointer for the manipulator event information push

Parameters:

1. eventInfo: The event information of the manipulator, it belongs to” RobotEventInfo“ data type

```
typedef struct{
    RobotEventType  eventType;       //Event type number
    int             eventCode;       //
    std::string     eventContent;    //Event content
}RobotEventInfo;
```

Inside it, the data type of “RobotEventType” is showed below, this enumeration contains a lot

```
typedef enum{
    RobotEvent_armCanbusError,           //Robot CAN bus error
    RobotEvent_remoteHalt,               //Remote halt       TODO
    RobotEvent_remoteEmergencyStop,      //Robot remote emergency stop
    RobotEvent_jointError,               //Joint error

    RobotEvent_forceControl,             //Force control
    RobotEvent_exitForceControl,         //Exit from the force control

    RobotEvent_softEmergency,            //Soft emergency stop
    RobotEvent_exitSoftEmergency,        //Exit from the soft emergency stop

    RobotEvent_collision,                //Collision
    RobotEvent_collisionStatusChanged,   //Collision status has changed
    RobotEvent_tcpParametersSucc,        //Tool dynamic parameters set successfully
    RobotEvent_powerChanged,             //Robot power status has changed
    RobotEvent_ArmPowerOff,              //Robot power off
    RobotEvent_mountingPoseChanged,      //Mounting position has changed
    RobotEvent_encoderError,             //Rncoder error

    RobotEvent_encoderLinesError,        //Encoder lines number are not same
    RobotEvent_singularityOverspeed,     //Singularity overspeed
    RobotEvent_currentAlarm,             //Robot current is abnormal
    RobotEvent_toolioError,             //Robot tool error
    RobotEvent_robotStartupPhase,       //The robot startup phase
    RobotEvent_robotStartupDoneResult,  //The result of robot startup
    RobotEvent_robotShutdownDone,       //The result of robot shutdown
    RobotEvent_atTrackTargetPos,        //The signal notification of the robot movement getting in position

    RobotSetPowerOnDone,                //Setting the power status has done
    RobotReleaseBrakeDone,              //Releasing the robot brake has done
    RobotEvent_robotControllerStateChaned,  //Robot control status has changed
    RobotEvent_robotControllerError,        //Robot control error --- generally, it will return when
algorithm planning is wrong
     RobotEvent_socketDisconnected,         //socket disconnected

    RobotEvent_robotControlException,
    RobotEvent_trackPlayInterrupte,

    RobotEvent_staticCollisionStatusChanged,
    RobotEvent_MountingPoseWarning,
    RobotEvent_MacDataInterruptWarning,
    RobotEvent_ToolIoError,
    RobotEvent_InterfacBoardSafeIoEvent,

    RobotEvent_RobotHandShakeSucc,
    RobotEvent_RobotHandShakeFailed,



      RobotEvent_exceptEvent = 100,

    //unknown event
    robot_event_unknown,

    //user event
    RobotEvent_User = 1000,                            // first user event id
    RobotEvent_MaxUser = 65535                         // last user event id

}RobotEventType;
```

Return: Successful call returns 0; errors return error code.

3.4.2 The push of real-time joint status

```
int robotServiceSetRealTimeJointStatusPush(bool enable);
```

Description: To set whether it is allowed real-time joint status to be pushed.

Parameters:

1. enable: True represents allowed, false represents not allowed

Return: Successful call returns 0; errors return error code.

```
int  robotServiceRegisterRealTimeJointStatusCallback(RealTimeJointStatusCallback ptr, void  *arg);
```

Description: Registers the callback function for obtaining the status of the joint. After registering the callback function, the server pushes the current joint status information in real time.

Parameters:

1. ptr: Obtaining the function pointer of real-time joint state information, when the ptr = NULL, it is equivalent to cancel the registration of the callback function, cancelling the information pushing can use this interface as well.

2. arg: This parameter system does not do any processing, but only does the cache. When the callback function is triggered, the parameter is passed back through the parameters of the callback function.

Return: Successful call returns 0; errors return error code.

3.4.3 The push of real-time waypoint information

```
int robotServiceSetRealTimeRoadPointPush(bool enable);
```

Description: To set whether it is allowed real-time waypoint to be pushed.

Parameters:

1. enable: True represents allowed, false represents not allowed

Return: Successful call returns 0; errors return error code.

```
int  robotServiceRegisterRealTimeRoadPointCallback(const RealTimeRoadPointCallback ptr, void  *arg);
```

Description: Registers the callback function for obtaining a real-time waypoint. After registering the callback function, the server pushes the current waypoint information in real time.

Parameters:

1. ptr: Obtaining the function pointer of way point information, when the ptr = NULL, it is equivalent to cancel the registration of the callback function, cancelling the information pushing can use the this interface as well.

2. arg: This parameter system does not do any processing, but only does the cache. When the callback function is triggered, the parameter is passed back through the parameters of the callback function.

Return: Successful call returns 0; errors return error code.

3.4.4 The push of real-time end speed

```
int robotServiceSetRealTimeEndSpeedPush(bool enable);
```

Description: To set whether it is allowed real-time end speed to be pushed.

Parameters:

1. enable: True represents allowed, false represents not allowed

Return: Successful call returns 0; errors return error code.

```
int  robotServiceRegisterRealTimeEndSpeedCallback(const RealTimeEndSpeedCallback ptr, void  *arg);
```

Description: Registers the callback function for obtaining the end speed. After registering the callback function, the server pushes the current end speed in real time.

Parameters:

1. ptr: Obtaining the function pointer of end speed, when the ptr = NULL, it is equivalent to cancel the registration of the callback function, cancelling the information pushing can use the this interface as well.

2. arg: This parameter system does not do any processing, but only does the cache. When the callback function is triggered, the parameter is passed back through the parameters of the callback function.

Return: Successful call returns 0; errors return error code.

3.4.5 The push of real-time information of the manipulator

```
int  robotServiceRegisterRobotEventInfoCallback(RobotEventCallback ptr, void  *arg);
```

Description: Registers the callback function for obtaining the event information of the manipulator. After registering the callback function, the server pushes the event information in real time. Regarding the event information pushing, it does not provide the interface for changing whether it is allowed the information to be pushed because many important notifications of the manipulator are implemented by pushing event information. So, the event information is the system default push, not allowed to cancel.

Parameters:

1. ptr: Obtaining the function pointer of manipulator event information, when the ptr = NULL, it is equivalent to cancel the registration of the callback function, cancelling the information pushing can use this interface as well.

2. arg: This parameter system does not do any processing, but only does the cache. When the callback function is triggered, the parameter is passed back through the parameters of the callback function.

Return: Successful call returns 0; errors return error code.

3.4.6 Example

The example for real-time manipulator information push module, please refer to the “Example_getinfo” part of the example program, the method “eventpush()” shows how to use callback function.

3.5 The setting and obtaining of the manipulator properties

3.5.1 Setting and obtaining the current working mode of the manipulator

```
int robotServiceSetRobotWorkMode(aubo_robot_namespace::RobotWorkMode mode);
```

Description: Set the work mode of the manipulator

Parameters:

1. Mode: It belongs to the RobotWorkMode, which has two choice, simulation mode or real robot mode

```
typedef enum{
    RobotModeSimulator, //Robot simulation mode
    RobotModeReal       //Robot real mode
}RobotWorkMode;
```

Return: Successful call returns 0; errors return error code.

```
int robotServiceGetRobotWorkMode(aubo_robot_namespace::RobotWorkMode &mode);
```

Description: Get the work mode of the manipulator

Parameters:

1. Mode: Output parameter, simulation or real robot mode

Return: Successful call returns 0; errors return error code.

3.5.2 Determine whether the real manipulator is existing

```
int robotServiceGetIsRealRobotExist(bool &value);
```

Description: Determine the real manipulator is existing or not

Parameters:

1. value: Output parameter, true means the real manipulator is existing, false means not existing.

Return: Successful call returns 0; errors return error code.

3.5.3 Setting and obtaining the collision class of the manipulator

```
int robotServiceSetRobotCollisionClass(int grade);
```

Description: Set the collision class of the manipulator

Parameters:

1. grade: The collision class is range from 1 – 10, default is 6

Return: Successful call returns 0; errors return error code.

```
int robotServiceGetRobotCollisionCurrentService(int &collisionGrade);
```

Description: Get the collision class

Parameters:

1. Mode: Output parameter, range 1- 10

Return: Successful call returns 0; errors return error code.

3.5.4 Obtaining the joint status of the manipulator

```
int robotServiceGetRobotJointStatus(aubo_robot_namespace::JointStatus *jointStatus, int size);
```

Description: Get the joint status of the manipulator, the difference between this method and push method is that this method is one-time call, the push method will keep return the information of the manipulator

Parameters:

1. jointStatus: Output parameter, joint Status

2. size: The length of the joint Status buffer

Return: Successful call returns 0; errors return error code.

3.5.5 Obtaining the diagnosis information of the manipulator

```
int robotServiceGetRobotDiagnosisInfo(aubo_robot_namespace::RobotDiagnosis &robotDiagnosisInfo);
```

Description: Get the diagnosis information of the manipulator

Parameters:

1. robotDiagnosisInfo: Output parameter, the diagnosis information, the data type of “RobotDiagnosis” is showed below

```
/****Robot diagnoses****/
typedef struct PACKED
{
    //CAN communication status: 0x01~0x80: Joint CAN communication error(each joint occupies 1bit)
    //0x00: No error 0xff: CAN bus error
    uint8 armCanbusStatus;
    //The current of robot 48V power
    float armPowerCurrent;
    //The voltage of robot 48V power
    float armPowerVoltage;
    //The switch status (on, off)of robot 48V power
    bool  armPowerStatus;
    //The temperature of control cabinet
    char  contorllerTemp;
    //The humidity of control cabinet
    uint8 contorllerHumidity;
    //Remote halt signal
    bool  remoteHalt;
    //Robot soft emergency
    bool  softEmergency;
    //Remote emergency signal
    bool  remoteEmergency;
    //Collision detection bit
    bool  robotCollision;
    //The flag bit of robot starting force control mode
    bool  forceControlMode;
    //Brake status
    bool brakeStuats;
    //End velocity
    float robotEndSpeed;
    //The maximum acceleration
    int robotMaxAcc;
    //The status bit of the software(ORPE)
    bool orpeStatus;
    //The enable bit of getting position and posture
    bool enableReadPose;
    //Mounting position status
    bool robotMountingPoseChanged;
    //Magnetic encoder error status
    bool encoderErrorStatus;
    //Static collision detection switch
    bool staticCollisionDetect;
    //Joint collision detection, each joint occupies 1 bit, 0-collision inexistence 1-collision existence
    uint8 jointCollisionDetect;
    //Optical-electricity encoders are not same, 0-no error, 1-error
    bool encoderLinesError;
    //joint error status
    bool jointErrorStatus;
    //The overspeed alarm of robot singularity
    bool singularityOverSpeedAlarm;
    //The alarm of robot current flow
    bool robotCurrentAlarm;
    //tool error
    uint8 toolIoError;
    //The mounting position of the robot is wrong(Working on the force control only)
    bool robotMountingPoseWarning;
    //The size of the mac buffer
    uint16 macTargetPosBufferSize;
    //The valid data size of the mac buffer
    uint16 macTargetPosDataSize;
    //The mac data interruption
    uint8  macDataInterruptWarning;
}RobotDiagnosis;
```

Return: Successful call returns 0; errors return error code.

3.5.6 Obtaining the joint angle of the manipulator

```
int robotServiceGetJointAngleInfo(aubo_robot_namespace::JointParam &jointParam);
```

Description: Get the joint angle of the manipulator

Parameters:

1. jointParam: Output parameter, joint angle.

Return: Successful call returns 0; errors return error code.

3.5.7 Obtaining the waypoint information of the manipulator

```
int robotServiceGetCurrentWaypointInfo(aubo_robot_namespace::wayPoint_S &wayPoint);
```

Description: Get the waypoint information of the manipulator

Parameters:

1. jointParam: Output parameter, waypoint information

Return: Successful call returns 0; errors return error code.

3.5.8 Setting and obtaining the tool dynamics parameter

```
int  robotServiceSetNoneToolDynamicsParam();
```

Description: Set the tool dynamics parameters to the 0.

Return: Successful call returns 0; errors return error code.

```
int  robotServiceSetToolDynamicsParam(const aubo_robot_namespace::ToolDynamicsParam &toolDynamicsParam);
```

Description: Set the tool dynamics parameters.

Parameters:

1. toolDynamicsParam: Tool dynamics parameter

Return: Successful call returns 0; errors return error code.

```
int  robotServiceGetToolDynamicsParam(aubo_robot_namespace::ToolDynamicsParam &toolDynamicsParam);
```

Description: Get the tool dynamics parameters.

Parameters:

1. toolDynamicsParam: Output parameter, tool dynamics parameter

Return: Successful call returns 0; errors return error code

3.5.9 Setting and obtaining the tool Kinematics parameter

```
int  robotServiceSetNoneToolKinematicsParam();
```

Description: Set the tool kinematics parameters to the 0.

Return: Successful call returns 0; errors return error code.

```
int  robotServiceSetToolKinematicsParam(const aubo_robot_namespace::ToolKinematicsParam
&toolKinematicsParam);
```

Description: Set the tool kinematics parameters.

Parameters:

1. toolKinematicsParam: Tool kinematics parameter

Return: Successful call returns 0; errors return error code.

```
int robotServiceGetToolKinematicsParam(aubo_robot_namespace::ToolKinematicsParam &toolKinematicsParam);
```

Description: Get the tool kinematics parameters.

Parameters:

1. toolKinematicsParam: Output parameter, tool kinematics parameter

Return: Successful call returns 0; errors return error code

The datatype of toolKinematicsParam is same as ToolInEndDesc.

3.5.10 Example

The example for setting and obtaining of the manipulator properties, please refer to the “Example_getinfo” part of the example program, the method “getJointStatus ()” and “ getToolPara() ” show some examples.

3.6 The I/O module

3.6.1 Important data types used in I/O module

RobotIoType: This data type enumerates the I/O types of the interface board

```
typedef enum
{
    RobotBoardControllerDI,    //Interface board controller(digital input)     Read only(generally for
system inner use)
    RobotBoardControllerDO,    //Interface board controller(digital output)     Read only(generally for
system inner use)
    RobotBoardControllerAI,    //Interface board controller(analog input)     Read only(generally for
system inner use)
    RobotBoardControllerAO,    //Interface board controller(analog output)     Read only(generally for
system inner use)

    RobotBoardUserDI,          //Interface board user DI(digital input)    Read-write
    RobotBoardUserDO,          //Interface board user DO(digital output)    Read-write
    RobotBoardUserAI,          //Interface board user AI(analog input)    Read-write
    RobotBoardUserAO,          //Interface board user AO(analog output)    Read-write

    RobotToolDI,               //Tool DI
    RobotToolDO,               //Tool DO
    RobotToolAI,               //Tool AI
    RobotToolAO,               //Tool AO

}RobotIoType;
```

RobotIoDesc: This data type gives a comprehensive description of an I/O

```
typedef struct PACKED
{
    char        ioId[32];      //IO-ID Not used currently
    RobotIoType ioType;        //IO type
    char        ioName[32];    //IO name
    int         ioAddr;        //IO address
    double      ioValue;       //IO status
}RobotIoDesc;
```

ToolPowerType: This data type defines the power output of the tool I/O

```
typedef enum
{
    OUT_0V  = 0,
    OUT_12V = 1,
    OUT_24V = 2
}ToolPowerType;
```

IO_STATUS: This data type defines the I/O status

```
typedef  enum                 //I/o status
{
    IO_STATUS_INVALID = 0, //Valid
    IO_STATUS_VALID        //Invalid
}IO_STATUS;
```

ToolDigitalIOAddr: This data type defines the digital tool I/O address

```
typedef enum
{
    TOOL_DIGITAL_IO_0 = 0,
    TOOL_DIGITAL_IO_1 = 1,
    TOOL_DIGITAL_IO_2 = 2,
    TOOL_DIGITAL_IO_3 = 3

}ToolDigitalIOAddr;
```

ToolIOType: This data type defines the tool I/O is input or output

```
typedef enum            //I/O type
{
    IO_IN = 0,          //input
    IO_OUT              //output
}ToolIOType;
```

3.6.2 The I/O module of the interface board

The I/O interface board is mainly divided into two parts, they are controller I/O and user I/O respectively. Each I/O has corresponding name and address, the state of IO can be set and obtained by name or address.

```
int robotServiceGetBoardIOConfig(const std::vector<aubo_robot_namespace::RobotIoType>&ioType,
std::vector<aubo_robot_namespace::RobotIoDesc>&configVector);
```

Description: Get the configuration information of the one or multiple interface board I/O

Parameters:

1. ioType: The collective input parameters of the I/O type

2. ConfigVector: The collective output parameters of the I/O configuration information

Return: Successful call returns 0; errors return error code.

```
int robotServiceGetBoardIOStatus(const std::vector<aubo_robot_namespace::RobotIoType>  ioType,
std::vector<aubo_robot_namespace::RobotIoDesc>&statusVector);
```

Description: Get the status information of the one or multiple interface board I/O

Parameters:

1. ioType: The collective input parameters of the I/O type

2. statusVector: I/O status

Return: Successful call returns 0; errors return error code.

```
int robotServiceSetBoardIOStatus(aubo_robot_namespace::RobotIoType type, std::string name, double value);
```

Description: Set the status information of the interface board I/O by using type and name

Parameters:

1. type: I/O type

2. name: I/O name

3. value: I/O status

Return: Successful call returns 0; errors return error code.

```
int robotServiceSetBoardIOStatus(aubo_robot_namespace::RobotIoType type, int    addr,      double value);
```

Description: Set the status information of the interface board I/O by using type and address

Parameters:

1. type: I/O type

2. addr: I/O address

3. value: I/O status

Return: Successful call returns 0; errors return error code.

```
int robotServiceGetBoardIOStatus(aubo_robot_namespace::RobotIoType type, std::string name, double &value);
```

Description: Get the status information of the interface board I/O by using type and name

Parameters:

1. type: I/O type

2. name: I/O name

3. value: Output parameter, I/O status

Return: Successful call returns 0; errors return error code.

```
int robotServiceGetBoardIOStatus(aubo_robot_namespace::RobotIoType type, int    addr,      double &value);
```

Description: Get the status information of the interface board I/O by using type and address

Parameters:

1. type: I/O type

2. addr: I/O address

3. value: Output parameter, I/O status

Return: Successful call returns 0; errors return error code.

3.6.3 The I/O module of the tool

```
int robotServiceSetToolPowerVoltageType  (aubo_robot_namespace::ToolPowerType type);
```

Description: Set the output voltage type of tool I/O

Parameters:

1. type: Output voltage type of tool I/O

Return: Successful call returns 0; errors return error code.

```
int robotServiceGetToolPowerVoltageType  (aubo_robot_namespace::ToolPowerType &type);
```

Description: Get the output voltage type of tool I/O

Parameters:

1. type: Output parameter, the voltage type of tool I/O

Return: Successful call returns 0; errors return error code.

```
int robotServiceGetToolPowerVoltageStatus(double &value);
```

Description: Get the output voltage value of tool I/O

Parameters:

1. value: Output parameter, the voltage value of tool I/O

Return: Successful call returns 0; errors return error code.

```
int robotServiceSetToolPowerTypeAndDigitalIOType(aubo_robot_namespace::ToolPowerType type,
                                                  aubo_robot_namespace::ToolIOType io0,
                                                  aubo_robot_namespace::ToolIOType io1,
                                                  aubo_robot_namespace::ToolIOType io2,
                                                  aubo_robot_namespace::ToolIOType io3);
```

Description: Set the output voltage type and all the digital I/O type of tool I/O

Parameters:

1. type: The voltage type of tool I/O

2. io0: I/O type of IO 0

3.io1: I/O type of IO 1

4.io2: I/O type of IO 2

5.io3: I/O type of IO 3

Return: Successful call returns 0; errors return error code.

```
int robotServiceSetToolDigitalIOType(aubo_robot_namespace::ToolDigitalIOAddr addr,
aubo_robot_namespace::ToolIOType type);
```

Description: Set the tool I/O type by using address

Parameters:

1. addr: Address of the digital tool I/O

2. type: I/O type

Return: Successful call returns 0; errors return error code.

```
int robotServiceGetAllToolDigitalIOStatus(std::vector<aubo_robot_namespace::RobotIoDesc>&statusVector);
```

Description: Get the status of all the digital tool I/O

Parameters:

1. statusVector: Output parameter, the I/O status

Return: Successful call returns 0; errors return error code.

```
int robotServiceSetToolDOStatus          (aubo_robot_namespace::ToolDigitalIOAddr addr,
aubo_robot_namespace::IO_STATUS value);
```

Description: Set the status of the tool digital I/O according to the address

Parameters:

1. addr: The I/O address

2. value: Digital I/O status

Return: Successful call returns 0; errors return error code.

```
int robotServiceSetToolDOStatus(std::string name,        aubo_robot_namespace::IO_STATUS value);
```

Description: Set the status of the tool digital I/O according to the name

Parameters:

1. addr: The I/O name

2. value: Digital I/O status

Return: Successful call returns 0; errors return error code.

```
int robotServiceGetToolIoStatus(std::string name, double &value);
```

Description: Get the status of the tool digital I/O according to the name

Parameters:

1. addr: The I/O name

2. value: Digital I/O status

Return: Successful call returns 0; errors return error code.

```
int robotServiceGetAllToolAIStatus       (std::vector<aubo_robot_namespace::RobotIoDesc>&statusVector);
```

Description: Get all tool analog I/O status according to the name

Parameters:

1. statusVector: Output parameter**,** analog I/O status

Return: Successful call returns 0; errors return error code.

3.6.4 Example

The example for the I/O module, please refer to the "Example_IO" part of the example program

3.7 Frequently-used algorithm

3.7.1 Forward kinematics and inverse kinematics

```
int robotServiceRobotFk(const double *jointAngle, int size, aubo_robot_namespace::wayPoint_S &wayPoint);
```

Description: Forward kinematics, get the corresponding position and posture from the joint angle.

Parameters:

1. jointAngle: The joint angle of each joints

2. size: The number of joint, default is 6

3. wayPoint: Output parameter, forward kinematics results

Return: Successful call returns 0; errors return error code.

```
int robotServiceRobotIk(const double *startPointJointAngle,const aubo_robot_namespace::Pos &position, const
aubo_robot_namespace::Ori &ori, aubo_robot_namespace::wayPoint_S &wayPoint);
```

Description: Inverse kinematics, gets the joint angle from the position and posture of the end effector. Because the result is not unique, so it need the start point of the track to calculate the answer.

Parameters:

1. startPointJointAngle: The joint angle of each joint in the track start point

2. position: The end effector position

3. ori: The end effector orientation

4. wayPoint: Output parameter, inverse kinematics results

Return: Successful call returns 0; errors return error code.

3.7.2 Transform between quaternion and Euler angle

```
int quaternionToRPY(const aubo_robot_namespace::Ori &ori, aubo_robot_namespace::Rpy &rpy);
```

Description: Quaternion to Euler angle

Parameters:

1. ori: The orientation described by quaternion

2. rpy: Output parameter, the orientation described by Euler angle

Return: Successful call returns 0; errors return error code.

```
int RPYToQuaternion(const aubo_robot_namespace::Rpy &rpy, aubo_robot_namespace::Ori &ori);
```

Description: Quaternion to Euler angle

Parameters:

1. rpy: The orientation described by Euler angle

2. ori: Output parameter, the orientation described by quaternion

Return: Successful call returns 0; errors return error code.

3.7.3 Coordinates transform

```
static int baseToUserCoordinate(
              const aubo_robot_namespace::Pos &flangeCenterPositionOnBase,
              const aubo_robot_namespace::Ori &flangeCenterOrientationOnBase,
              const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool &userCoord,
              const aubo_robot_namespace::ToolInEndDesc  &toolInEndDesc,
              aubo_robot_namespace::Pos                  &toolEndPositionOnUserCoord,
              aubo_robot_namespace::Ori                  &toolEndOrientationOnUserCoord
                                     );
```

Description: It converts the position and posture of the flange center based on the base coordinate system to the position and posture of the end tool based on the user coordinate system.

Parameters:

1. flangeCenterPositionOnBase: The position information of the flange center based on the base coordinate system

2. flangeCenterOrientationOnBase: The posture information based on the base coordinate

3. userCoord: User coordinate

4. toolInEndDesc: Tool information

5. toolEndPositionOnUserCoord: Output parameter, the position information of the end tool based on the user coordinate

6. toolEndOrientationOnUserCoord: Output parameter, the posture information of the end tool based on the user coordinate

Return: Successful call returns 0; errors return error code.

```
   static int baseToBaseAdditionalTool(
              const aubo_robot_namespace::Pos  &flangeCenterPositionOnBase,
              const aubo_robot_namespace::Ori  &flangeCenterOrientationOnBase,
              const aubo_robot_namespace::ToolInEndDesc  &toolInEndDesc,
              aubo_robot_namespace::Pos                  &toolEndPositionOnBase,
              aubo_robot_namespace::Ori                  &toolEndOrientationOnBase
                                        );
```

Description: It converts the position and posture of the flange center based on the base coordinate system to the position and posture of the end tool based on the base coordinate system.

Parameters:

1. flangeCenterPositionOnBase: The position information of the flange center based on the base coordinate system

2. flangeCenterOrientationOnBase: The posture information based on the base coordinate system

3. toolInEndDesc: Tool information

4. toolEndPositionOnBase: Output parameter, the position information of the end tool based on the base coordinate system

5. toolEndOrientationOnUserBase Output parameter, the posture information of the end tool based on the base coordinate system

Return: Successful call returns 0; errors return error code.

```
static int userToBaseCoordinate(
           const aubo_robot_namespace::Pos &toolEndPositionOnUserCoord,
           const aubo_robot_namespace::Ori &toolEndOrientationOnUserCoord,
           const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool &userCoord,
           const aubo_robot_namespace::ToolInEndDesc  &toolInEndDesc,
           aubo_robot_namespace::Pos                  &flangeCenterPositionOnBase,
           aubo_robot_namespace::Ori                  &flangeCenterOrientationOnBase
                                      );
```

Description: It converts the position and posture of the end tool based on the user coordinate system to the position and posture of the flange center based on the base coordinate system.

Parameters:

1. toolEndPositionOnUserCoord: The position information of the end tool based on the user coordinate

2. toolEndOrientationOnUserCoord: The posture information of the end tool based on the user coordinate

3. userCoord: User coordinate

4. toolInEndDesc: Tool information

5. flangeCenterPositionOnBase: Output parameter, the position information of the flange center based on the base coordinate system

6. flangeCenterOrientationOnBase: Output parameter, the posture information based on the base coordinate system

Return: Successful call returns 0; errors return error code.

```
   static int userCoordPointToBasePoint(
              const aubo_robot_namespace::Pos &userCoordPoint,
              const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool &userCoordSystem,
              aubo_robot_namespace::Pos &basePoint
                                        );
```

Description: converts the position in the user coordinate system to the position in the base coordinate system, this method is only used for converting position

Parameters:

1. userCoordPoint: The position information of the end tool based on the user coordinate

2. userCoordSystem: User coordinate

3. basePoint: The position information of the end tool based on the base coordinate

Return: Successful call returns 0; errors return error code.

```
static int endOrientation2ToolOrientation(
           aubo_robot_namespace::Ori &tcpOriInEnd,
           const aubo_robot_namespace::Ori &endOri,
           aubo_robot_namespace::Ori &toolOri
                                      );
```

Description: Converts the flange orientation to tool orientation

Parameters:

1. tcpOriInEnd: The orientation of the tool center point

2. endOri: Flange orientation

3. toolOri: Output parameter, tool orientation

```
static int toolOrientation2EndOrientation(
           aubo_robot_namespace::Ori &tcpOriInEnd,
           const aubo_robot_namespace::Ori &toolOri,
           aubo_robot_namespace::Ori &endOri
                                     );
```

Description: Converts the tool orientation to flange orientation

Parameters:

1. tcpOriInEnd: The orientation of the tool center point

2. toolOri: Tool orientation

3. endOri: Output parameter, flange orientation

3.7.4 Example

The example for the Frequently-used algorithm, please refer to the “Example_algorithm” part of the example program

IV. Error code

Error code	Error info
0	Success
10001	General failed
10002	Parameter error
10003	Socket connection failed
10004	Socket disconnect
10005	Fail to create request
10006	Request-related internal variable error
10007	Request timeout
10008	Fail to send request information
10009	Response information is null

10010	Fail to resolve response
10011	Forward kinematics failed
10012	Inverse kinematics failed
10013	Tool calibration parameters error
10014	Tool calibration parameters error
10015	Fail to calibrate coordinate system
10016	Fail to convert base coordinate system to user coordinate system
10017	Fail to convert user coordinate system to base coordinate system
10018	Motion-related internal variable error
10019	Motion request fail
10020	Fail to create motion request
10021	Motion is interrupted by event
10022	Motion-related waypoint container size is illegal
10023	Server response return error
10024	The real robot is not existing because some interfaces can be call only if the real robot is existent.