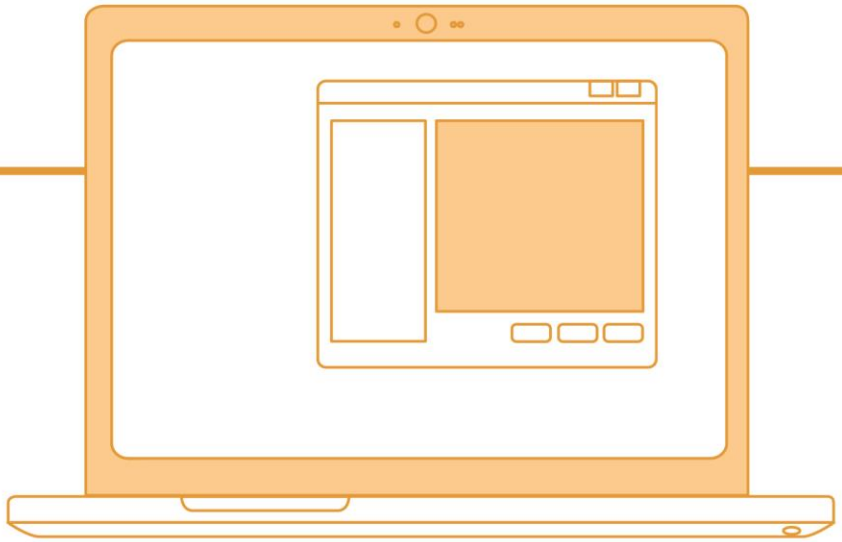
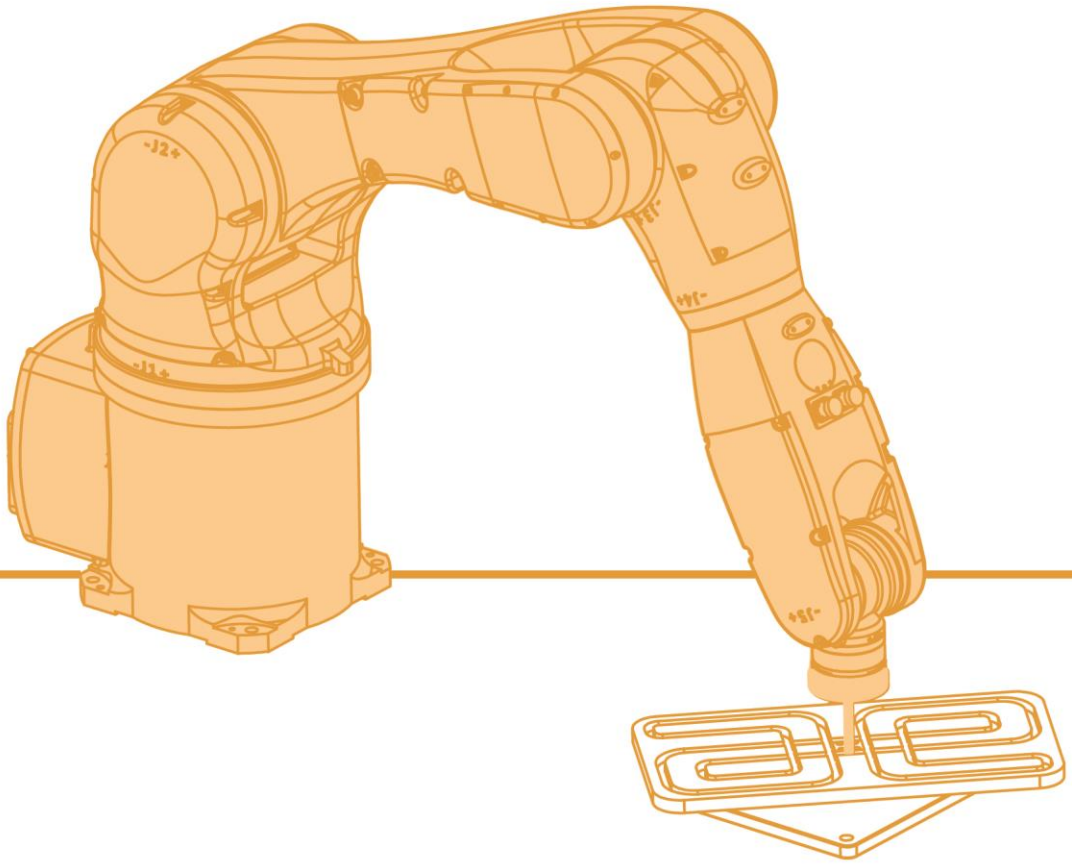
PEITIAN ROBOTICS

Programming Interface Library Instruction Manual

V2.0.6





Foreword

About this manual

The programming interface library is a software product that Petian Robot provides to customers for secondary development. Through the programming interface library, customers can perform a series of control and operations on the robot.

This manual mainly introduces how to use the programming interface library, as well as the functions of each interface function, and provides sample codes for each interface function.

Operating prerequisites

Before using the industrial robot, please read the relevant manual carefully, and use the industrial robot correctly under the premise of understanding its content.

Target groups

- Operators
- Product technicians
- Technical service personnel
- Robot teachers

Meaning of common signs

See Table1 for the signs and their meanings in this manual.

Table 1 Signs used in this manual

Sign	Meaning
 Danger	Failure to follow the instructions will cause accidents, resulting in serious or fatal personal injury, or serious damage to items
 Warning	Failure to follow the instructions may cause accidents, resulting in serious or fatal personal injury, or serious damage to items
 Notice	You are prompted to keep in mind environmental conditions and important matters, or quick operation methods
 Tip	You are prompted to refer to other literature and instructions for additional information or more details about operation instructions

Description of this manual

The document-related information is shown in Table 2.

Table 2 Document-related information

Document No.	UM-S01500000003-006
Document Ver.	V2.0.6
Software ver.	2.6.5

The contents of this manual are subject to supplementation and modification. Please visit "Download Center" on the website regularly to obtain the latest version of this manual in a timely manner.

Website URL: http://robot.peitian.com/

Revision record

The revision record accumulates instructions for each manual update. The latest version of the manual contains updated content from all previous manual versions.

Table 3 Version update list

Version	Release time	Modification instructions
V2.0.3	2021/12/01	Added 'New interface for obtaining variable values through variable names'
V2.0.4	2022/01/10	Fix known bugs
V2.0.5	2022/06/15	Add new interface description
V2.0.6	2022/10/14	Add class pointer parameters to the interface to support simultaneous connection of multiple robots

Contents

1 Instructions

1.1 Development environment setup

The programming interface library is written in C++, and the Windows version uses Visual Studio 2012 to compile C++ version, and includes both 32-bit and 64-bit versions.

The entire interface library includes 4 files, namely:

- robot_api_csharp.h: Programming interface function definition header file
- RobotAPI.dll : Programming interface dynamic library
- RobotAPI.lib : Programming interface static library
- RobotComm.dll: The robot communication dynamic library that the programming interface library depends on

Please install Visual Studio 2012 and choose to install all C++ class libraries at the same time.

If the installed Visual Studio is 2015/2017 or other high version, please use the 2012 version of the programming interface library file.

1.2 Development configuration

C # clients generally choose the Windows platform, and the following steps are only for the Windows platform.

Step1. Create a C # project.

Step2. Taking the debug version as an example, place the RobotAPI.dll, RobotAPI.lib, and RobotComm.dll files in the version library into the bin\debug folder in the project root directory (x64 bit corresponds to bin\x64\debug).

Step3. Refer to the. cs file demo in the SDK to change the Program. cs file.

Step4. You can call the robot_api_csharp.h normally in the project, the interface functions provided in the robot_api_csharp.h can be compiled normally.

1.3 Debugging and running

Before debugging or running, in Windows environment, please copy RobotAPI.dll and RobotComm.dll to the running directory, you can execute the directory where the file is located. In Linux environment, please copy the two files libRobotAPI.so and libRobotComm.so to the program can be run under the searchable directory .

At the same time, the following configuration is required:

- Configure the IP address of the robot user (for the specific configuration method, please refer to the operation manual of the robot, and the IP address and the computer running the program are in the same network segment).
- Through the robot teach pendant interface, turn on "External API Control Enable" under the "System" -> "Configuration" -> "System Settings" page.

2 Interface class description

The programming interface library declares its own namespace RobotAPI, which can be used during development through the using namespace RobotAPI. The programming interface class CRobotAPI describes all the methods and variables involved in managing a robot connection instance.

The interface class structure is as follows:

```
class ROBOTAPI_API CRobotAPI
{
public:
        /*
        *@brief Constructor, creating robot connection instance
        *@para index(in): The serial number of the robot connection created, starting from 0, with a maximum of
max_ Connection_ Count -1
        */
        CRobotAPI(unsigned int index);
    /*
        *@brief Destructor, delete robot connection instance, serial number cannot be used anymore
        */
        ~CRobotAPI();
    /*
    *@brief Query the current control right, which only refers to the main control right. Even if there is no main
    control right, the status can still be queried, but it cannot control the robot's movement
        */
        bool HasControlPermission();
        ……// Other member functions
private:
        unsigned int m_robot_index;
        char* m_server_addr;
        bool m_connected;
        unsigned int m_wobj_num;
};
```

3 Interface function description

The programming interface library declares its own namespace RobotAPI, which can be used by using namespace RobotAPI during development. The return error codes of all programming interface library functions are based on the interface library, and it may not be possible to locate the specific failure reason from these error codes.

E.g:

Control the robot movement interface Move2Joint. If the set axis joint angle exceeds its motion range, the error code 18 (control robot movement failure) will be returned. This error code alone cannot locate the specific cause. In this case, you need to get the robot through GetAlarmList. In the alarm list, the alarm list will contain the alarms of axis overrun.

C # cannot directly call c++class member functions, so programming interface library functions are all encapsulated by the corresponding member functions of the interface class CRobotAPI. The first parameter is the interface class pointer mentioned earlier. The following text provides usage methods for various interface functions for C # users.

3.1 Robot management

3.1.1 CreateCRobotAPI (Create robot connection)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern IntPtr CreateCRobotAPI(uint index);
```

Function

Create robot connections, instances, and return robot connection instance pointers.

Parameter

index(in)	The robot connection index created starts from 0 and reaches a maximum of max_ Connection_ Count -1.

Return

pRobot	Pointer to type CRobotAPI *

Usage

```
uint index = 1;
IntPtr pRobot = CreateCRobotAPI(index);
```

3.1.2 DestoryCRobotAPI (Destroy robot connection instance)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern void DestoryCRobotAPI(IntPtr pRobot);
```

Function

Destroy robot connection instance.

Parameter

pRobot(in)	Pointer to the robot connection being operated on

Return

0	Destroyed successfully.
Non-0	Error code.

Usage

```
int res = DestoryCRobotAPI(pRobot);
```

3.1.3 ConnectRobot (Initialize and connect the robot)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int ConnectRobot(IntPtr pRobot, string robot_addr);
```

Function

Initialize and connect the robot.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
robot_addr(in)	The IP address of the robot that needs to be connected.

Return

0	Connection succeeded.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
string server_addr = "10.20.1.1";
res = ConnectRobot(pRobot, server_addr);
if(ERROR_OK == res) { // Connection succeeded
} else { // Connection failed, please check whether the network is normal and whether external API control has been enabled on the robot side}
```

This interface function is the first step in all operations after creating a connected instance. If this interface is not called, other interfaces will not work properly.

3.1.4 DisconnectRobot (Disconnect the robot)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern void DisconnectRobot(IntPtr pRobot);
```

Function

Disconnect the robot.

Parameter

pRobot(in)	Pointer to the robot connection being operated on

Return

0	Successfully disconnected.
Non-0	Error code.

Usage

```
DisconnectRobot(pRobot);
```

3.1.5 EnableApiControl (Enable or disable external API control)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int EnableApiControl(IntPtr pRobot, byte enable);
```

Function

Enable or disable external API control.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
enable (in)	Whether to enable.

Return

0	The setting is successful.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
```

```
// Enable external API control
byte enable = 1;
res = EnableApiControl(pRobot, enable);
if(ERROR_OK == res) { // Set successfully
} else { // Set failed
}
```

3.1.6 SetControlMode(Set robot control mode)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int SetControlMode(IntPtr pRobot, RobotMode mode);
```

Function

Set the robot control mode.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
mode (in)	Set mode.

Return

0	The setting is successful.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
// Set to manual low speed mode
res = SetControlMode(pRobot, RobotMode.ROBOT_MODE_MANUAL);
if(ERROR_OK == res) { // Set successfully
} else { // Setup failed
}
```

The control mode is defined as follows:

```
ROBOT_MODE_MANUAL, // Manual low speed mode
ROBOT_MODE_AUTO, // Automatic mode
ROBOT_MODE_MANUFAST // Manual high speed mode
```

3.1.7 SetProgStepMode (Set the robot's single step continuous operation mode)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int SetProgStepMode(IntPtr pRobot, ProgramStepMode mode);
```

Function

Set the robot's single step continuous operation mode.

Parameter

pRobot(in)	Pointer to the robot connection being operated on.
mode (in)	Set the mode.

Return

0	The setting is successful.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
// Set to single step mode
res = SetProgStepMode(pRobot, ProgramStepMode.STEP);
if(ERROR_OK == res) {    // Set successfully
} else {    // Set failed
}
```

The control mode is defined as follows:

```
STEP,    // Single-step mode
CONTINUE,    // Continuous mode
MSTEP    // Segment debugging mode
```

3.1.8 SetProgDirMode (Set the forward and reverse operation mode of the robot)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int SetProgDirMode(IntPtr pRobot, ProgramDirMode mode);
```

Function

Set the forward and reverse operation mode of the robot.

Parameter

pRobot(in)	Pointer to the robot connection being operated on.
mode (in)	Set the mode.

Return

0	The setting is successful.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
// Set to forward running
res = SetProgDirMode(pRobot, ProgramDirMode.FORWARD);
if(ERROR_OK == res) {    // Set successfully
} else {    // Set failed
}
```

The control mode is defined as follows:

```
FORWARD,    // Forward
BACKWARD,    // Reverse
```

3.1.9 SwitchChannel (Switching channels)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int SwitchChannel(IntPtr pRobot, uint channel_num)
```

Function

Switch channels.

Parameter

pRobot(in)	Pointer to the robot connection being operated on.
channel_num(in)	The set channel number starts from 1, and is the foreground channel and the background channel in turn.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
// Switch to channel 2
res = SwitchChannel(pRobot, 2);
if(ERROR_OK == res) { // Switch successfully
} else { // Switch failed
}
```

3.1.10 PowerOn (Robot powered on)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
```

public static extern int PowerOn(IntPtr pRobot)

Function

The robot is powered on.

Parameter

pRobot(in)	Pointer to the robot connection being operated on.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
res = PowerOn(pRobot);
if(ERROR_OK == res) { // Power on successfully
} else { // Power-on failed, please check whether there is an alarm currently, and the power cannot be powered on in the alarm state
}
```


Caution

It is recommended to check whether it is currently in an alarm state through GetAlarmState before powering on. If it is in an alarm state, use ClearAlarm to clear the alarm.

3.1.11 PowerOff (Robot powered off)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int PowerOff(IntPtr pRobot);
```

Function

The robot is powered off.

Parameter

pRobot(in)	Pointer to the robot connection being operated on.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
```

```
res = PowerOff(pRobot);

if(ERROR_OK == res) { // Power off successfully

} else { // Power-off failed, please check whether it is currently running, and the robot is not allowed to power-off while it is moving

}
```

3.1.12 ClearAlarm (Clear robot alarm)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int ClearAlarm(IntPtr pRobot);
```

Function

Clear the robot alarm.

Parameter

pRobot(in)	Pointer to the robot connection being operated on.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
res = ClearAlarm(pRobot);
if(ERROR_OK == res) { // The clear alarm command was issued successfully
} else {
}
```


Caution

This interface only sends a clear alarm command to the robot, but it cannot guarantee that all alarms can be cleared. Therefore, it is recommended to wait about 100ms after clearing the alarm, and then use GetAlarmState to check whether the clearing is successful.

3.2 Movement

3.2.1 Move2Home (The robot returns to zero)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int Move2Home(IntPtr pRobot, uint home_index);
```

Functionz

The robot returns to zero.

Parameter

pRobot(in)	Pointer to the robot connection being operated on.
home_index(in)	The home point number that the user wants the robot to return to, with a range of 1-5

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
uint home_index = 3;
res = Move2Home(pRobot, home_index);
if(ERROR_OK == res) {    // Successfully sent the zero return instruction
} else {    // Zero return failure
}
```


Caution

The interface only starts the zero point return action, and the zero point has not reached when the interface returns. Therefore, if you want to wait for the zero point to arrive, you need to use GetJoint to determine whether the angle of each axis has reached the zero point.

3.2.2 Move2Joint (Movej controls the movement of each axis to a certain angle)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int Move2Joint(IntPtr pRobot, RobotJoint robot_joint, double speed_percent, double speed_exj =
0);
```

Function

The movej controls the movement of each axis to a certain angle.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
robot_joint(in)	The target angle of each axis, unit: degree.
speed_percent(in)	The movement speed percentage, relative to the maximum speed of each axis, the range is 0.001-100.
speed_exj(in)	Movement speed of external axis, unit: degree/second, if there is no external axis, this parameter can be defaulted.

Return

0	Success.

Non-0	Error code.

Usage

```
int res = ERROR_OK;
// Set the mode first before exercise
SetControlMode(pRobot, RobotMode.CONTROL_MODE_MANUAL);
// Set speed override to 50%
SetSpeedRatio(pRobot, 50);
RobotJoint joint;
joint.j[0] = 10; // Control the first axis to move to 10 degrees, the other axes remain unchanged
res = Move2Joint(pRobot, joint, 100);
if(ERROR_OK == res) { // Movement command sent successfully
} else { // Failure
}
```

Before moving, it is recommended to set the control mode and speed override first. The overrides used by all motion instructions in the programming interface library are SPEED_RATIO_JOG type overrides.

3.2.3 CheckPtpPosValid (Check if it is possible to move to certain positions through PTP)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int CheckPtpPosValid(IntPtr pRobot, RobotPos[] target_pos, uint target_pos_size, uint tool_index,
uint wobj_index, byte[] valid, uint valid_size);
```

Function

Check if it is possible to move to certain positions through PTP.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
target_pos(in)	Target pose
target_pos_size(in)	Target pose array length
tool_index(in)	Tool coordinate system serial number, 0: flange coordinate system, 1-32: custom tool coordinate system
wobj_index(in)	Workpiece coordinate system serial number, 0: World coordinate system, 1-3: Channel base coordinate system, 4-3+WOBJ_NUM (Number of workpiece coordinate systems): Customize the workpiece coordinate system.
valid(out)	Wether it is reachable.
valid_size(in)	Wether the target point reachable to array length

Return

0	Success.

Non-0	Error code.

Usage

```
int res = ERROR_OK;
// Prepare the target point pose array RobotPos * target_ Pos, wether it is the reachable flag array bool * valid
RobotPos pos1 = new RobotPos()
        {
            x = 1.0,
            y = 1.0,
            z = 1.0,
            a = 1.0,
            b = 1.0,
            c = 1.0,
            cfg = 1,
            turn = 1,
            ej1 = 2.0,
            ej2 = 1.0,
            ej3 = 1.0,
            ej4 = 1.0,
            ej5 = 1.0,
            ej6 = 3.0
        };
        RobotPos pos2 = new RobotPos()
        {
            x = 2.0,
            y = 2.0,
            z = 2.0,
            a = 2.0,
            b = 2.0,
            c = 2.0,
            cfg = 2,
            turn = 2,
            ej1 = 2.0,
            ej2 = 2.0,
            ej3 = 2.0,
            ej4 = 2.0,
            ej5 = 2.0,
            ej6 = 2.0
        };
RobotPos[] target_pos = new RobotPos[] { pos1, pos2 };
byte[] valid = {0, 0};
res = CheckPtpPosValid(pRobot, target_pos, 3, 0, 0, valid, 2);
if(ERROR_OK == res) {    // The check was successful, and the values in valid indicate whether these points are reachable
} else {    // Check failed
```

```
}
```

3.2.4 Move2Pos (Single Position) (PTP controls the robot to move to a certain position)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int Move2Pos(IntPtr pRobot, RobotPos robot_pos, uint tool_index, uint wobj_index, double speed_percent, double speed_exj = 0);
```

Function

The ptp controls the robot to move to a certain pose.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
robot_pos(in)	Target pose.
tool_index(in)	Tool coordinate system number, 0: flange coordinate system, 1-32: custom tool coordinate system
wobj_index(in)	Workpiece coordinate system serial number, 0: World coordinate system, 1-3: Channel base coordinate system, 4-353+WOBJ_NUM (Number of workpiece coordinate systems): Customize the workpiece coordinate system.
speed_percent(in)	The movement speed percentage, relative to the maximum speed of each axis, the range is 0.001-100.
speed_exj(in)	Movement speed of external axis, unit: degree/second, if there is no external axis, this parameter can be defaulted.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
// Set mode first before exercising
SetControlMode(pRobot, RobotMode.CONTROL_MODE_MANUAL);
// Set the speed multiplier to 50%
SetSpeedRatio(pRobot, 50);
RobotPos pos = pos1;          //pos1 is defined in 3.2.3
res = Move2Pos(pRobot, pos, 0,0,100);
if(ERROR_OK == res) {    // Successfully sent motion instruction
} else {    // Failed
}
```

- Before moving, it is recommended to set the control mode and speed override first. The overrides used by all motion instructions in the programming interface library are SPEED_RATIO_JOG type overrides.
- Unlike the Move2Joint instruction, the x, y, z, a, b, and c in the RobotPos type parameters of this interface need to be set and cannot be defaulted.

3.2.5 Move2MultiPos (Multi Position) (PTP controls the robot to move to several positions in sequence)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int Move2MultiPos(IntPtr pRobot, RobotPos[] robot_pos, uint robot_pos_size, uint tool_index, uint wobj_index, double speed_percent, double speed_exj = 0);
```

Function

The ptp controls the robot to move to several positions in sequence.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
robot_pos(in)	The target pose array.
Robot_pos_size(in)	Target pose array length
tool_index(in)	Tool coordinate system number, 0: flange coordinate system, 1-32: custom tool coordinate system.
wobj_index(in)	Workpiece coordinate system serial number, 0: World coordinate system, 1-3: Channel base coordinate system, 4-353+WOBJ_NUM (Number of workpiece coordinate systems): Customize the workpiece coordinate system.
speed_percent(in)	The movement speed percentage, relative to the maximum speed of each axis, the range is 0.001-100.
speed_exj(in)	Movement speed of external axis, unit: degree/second, if there is no external axis, this parameter can be defaulted.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
// Set mode first before exercising
SetControlMode(pRobot, RobotMode.ROBOT_MODE_MANUAL);
// Set the speed multiplier to 50%
SetSpeedRatio(50);
RobotPos multi_pos = target_pos;        //target_ pos is defined in 3.2.3
res = Move2MultiPos(pRobot, multi_pos, 2, 0,0,100);
if(ERROR_OK == res) {    // Successfully sent motion instruction
} else {    // Failed
}
```

- Before moving, it is recommended to set the control mode and speed override first. The overrides used by all motion instructions in the programming interface library are SPEED_RATIO_JOG type overrides.
- Unlike the Move2Joint instruction, the x, y, z, a, b, and c in the RobotPos type parameters of this interface need to be set and cannot be defaulted. This interface is different from the unit pose Move2Pos. This interface is a blocking function, that is, the function will not return until the last pose arrives. If you need to perform other

operations while this interface is running, please consider multi-threaded processing.

3.2.6 CheckLinePosValid (Check if it is possible to move to certain positions through a straight line)

Function declaration

[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int CheckLinePosValid(IntPtr pRobot, RobotPos start_pos, RobotPos[] target_pos, uint target_pos_size, uint tool_index, uint wobj_index, byte[] valid, uint valid_size);

Function

Check if it is possible to move to certain positions through a straight line.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
start_pos(in)	Starting pose
target_pos(in)	Target pose
Target_pos_size(in)	Target pose array length
tool_index(in)	Tool coordinate system serial number, 0: flange coordinate system, 1-32: custom tool coordinate system
wobj_index(in)	Workpiece coordinate system serial number, 0: World coordinate system, 1-3: Channel base coordinate system, 4-3+WOBJ_NUM (Number of workpiece coordinate systems): Customize the workpiece coordinate system.
valid(out)	Wether it is reachable.
valid_size	Wether the target point reachable to array length

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
// Prepare the starting point RobotPos start_ Pos, target point pose array RobotPos * target_ Pos, reachable flag array bool * valid
RobotPos start_pos = new RobotPos()
                {
                    x = 1.0,
                    y = 1.0,
                    z = 1.0,
                    a = 1.0,
                    b = 1.0,
                    c = 1.0,
                    cfg = 1,
```

```
                turn = 1,
                ej1 = 2.0,
                ej2 = 1.0,
                ej3 = 1.0,
                ej4 = 1.0,
                ej5 = 1.0,
                ej6 = 3.0
};
byte[] valid = {0, 0};
res = CheckLinePosValid(pRobot, start_pos, target_pos, 2, 0, 0, valid, 2);    //target_pos is defined in 3.2.3
if(ERROR_OK == res) {    // The check was successful, and the values in valid indicate whether these points are reachable
} else {    // Check failed
}
```

3.2.7 Line2Pos(Single Position) (Lin controls the robot to move in a straight line to a certain position)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int Line2Pos(IntPtr pRobot, RobotPos robot_pos, uint tool_index, uint wobj_index, double speed_tcp, double speed_ori, double speed_exj = 0);
```

Function

Lin controls the robot to move in a straight line to a certain position.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
robot_pos(in)	Target pose.
tool_index(in)	Tool coordinate system serial number, 0: flange coordinate system, 1-32: custom tool coordinate system.
wobj_index(in)	Workpiece coordinate system serial number, 0: World coordinate system, 1-3: Channel base coordinate system, 4-3+WOBJ_NUM (Number of workpiece coordinate systems): Customize the workpiece coordinate system.
speed_tcp(in)	TCP point movement speed, in mm/s.
speed_exj(in)	Tool coordinate system attitude rotation speed, in degrees per second.
speed_exj(in)	The movement speed of the external axis, in degrees per second. If there is no external axis, this parameter can be defaulted.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
// Set mode first before exercising
SetControlMode(pRobot, RobotMode.CONTROL_MODE_MANUAL);
// Set the speed multiplier to 50%
SetSpeedRatio(50);
RobotPos pos = new RobotPos()
                {
                    x = 1.0,
                    y = 1.0,
                    z = 1.0,
                    a = 1.0,
                    b = 1.0,
                    c = 1.0,
                    cfg = 1,
                    turn = 1,
                    ej1 = 2.0,
                    ej2 = 1.0,
                    ej3 = 1.0,
                    ej4 = 1.0,
                    ej5 = 1.0,
                    ej6 = 3.0
};
res = Line2Pos(pRobot, pos, 0,0,50,50,100);// TCP and ORI speeds 50
if(ERROR_OK == res) {    // Successfully sent motion instruction
} else {    // Failed
}
```

- Before exercising, it is recommended to first set the control mode and speed multiplier. The multiplier used for all motion instructions in the programming interface library is SPEED_ RATIO_ JOG type magnification.
- Unlike the Move2Joint instruction, the x, y, z, a, b, and c in the RobotPos type parameters in this interface need to be set and cannot be defaulted.

3.2.8 Line2MultiPos(Multi Position) (Lin controls the robot to move in a straight line to several positions in sequence)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int Line2MultiPos(IntPtr pRobot, RobotPos[] robot_pos, uint robot_pos_size, uint tool_index, uint wobj_index, double speed_tcp, double speed_ori, double speed_exj = 0);
```

Function

Lin controls the robot to move in a straight line to several positions in sequence.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
robot_pos(in)	The target pose array.
robot_pos_size	Target pose array length
tool_index(in)	Tool coordinate system number, 0: flange coordinate system, 1-32: custom tool coordinate system.
wobj_index(in)	Workpiece coordinate system serial number, 0: World coordinate system, 1-3: Channel base coordinate system, 4-353+WOBJ_NUM (Number of workpiece coordinate systems): Customize the workpiece coordinate system.
speed_tcp(in)	TCP point moving speed, unit: mm/s.
speed_ori(in)	Rotation speed of tool coordinate system posture, unit: degree/second.
speed_exj(in)	Movement speed of external axis, unit: degree/second, if there is no external axis, this parameter can be defaulted.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
// Set mode first before exercising
SetControlMode(pRobot, RobotMode.CONTROL_MODE_MANUAL);
// Set the speed multiplier to 50%
SetSpeedRatio(pRobot, 50);
RobotPos[] pos = target_pos;        //target_pos is defined in 3.2.3
res = Line2MultiPos(pRobot, pos, 3, 0,0,50,50,100);// TCP and ORI speeds 50
if(ERROR_OK == res) {   // Successfully sent motion instruction
} else {    // Failed
}
```

Caution

- Before moving, it is recommended to set the control mode and speed override first. The overrides used by all motion instructions in the programming interface library are SPEED_RATIO_JOG type overrides.
- Unlike the Move2Joint instruction, the x, y, z, a, b, and c in the RobotPos type parameters of this interface need to be set and cannot be defaulted. This interface is different from the unit pose Line2Pos. This interface is a blocking function, that is, the function will not return until the last pose arrives. If you need to perform other operations while this interface is running, please consider multi-threaded processing.

3.2.9 Circle2Pos(Cir controls the robot to move to a certain position)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int Circle2Pos(IntPtr pRobot, RobotPos robot_pos, RobotPos m, double CA, uint tool_index, uint wobj_index, double speed_tcp, double speed_ori, double speed_exj = 0);
```

Function

The cir controls the robot to move to a certain pose.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
robot_pos(in)	Target pose.
m(in)	Auxiliary point pose.
CA(in)	Center angle, if the user specifies the CA parameter, the robot_pos parameter is only used to determine the geometric shape of the arc together with the auxiliary point, not the real target point. The real target point is automatically calculated by the user-specified central angle.
tool_index(in)	Tool coordinate system number, 0: flange coordinate system, 1-32: custom tool coordinate system.
wobj_index(in)	Workpiece coordinate system serial number, 0: World coordinate system, 1-3: Channel base coordinate system, 4-353+WOBJ_NUM (Number of workpiece coordinate systems): Customize the workpiece coordinate system.
speed_tcp(in)	TCP point moving speed, unit: mm/s.
speed_ori(in)	Rotation speed of tool coordinate system posture, unit: degree/second.
speed_exj(in)	Movement speed of external axis, unit: degree/second, if there is no external axis, this parameter can be defaulted.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
// Set mode first before exercising
SetControlMode(pRobot, RobotMode.CONTROL_MODE_MANUAL);
// Set the speed multiplier to 50%
SetSpeedRatio(pRobot, 50);
RobotPos pos = pos1;// Move to positions x: 100, y: 100, z: 100, a: 0, b: 0, c: 0, pos1 defined in 3.2.3
RobotPos m = pos2;// The definition of auxiliary point pose pos2 is shown in 3.2.3
res = Circle2Pos(pRobot, pos, m, 180, 0, 0, 50, 50, 100);// Central angle 180 degrees, TCP and ORI speed 50
if(ERROR_OK == res) {    // Successfully sent motion instruction
} else {    // Failed
}
```

- Before moving, it is recommended to set the control mode and speed override first. The overrides used by all motion instructions in the programming interface library are SPEED_RATIO_JOG type overrides.
- Unlike the Move2Joint instruction, the x, y, z, a, b, and c in the RobotPos type parameters of this interface need to be set and cannot be defaulted.

3.2.10 StopMove (Control the robot to stop moving)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int StopMove(IntPtr pRobot);
```

Function

Control the robot to stop moving.

Parameter

pRobot(in)	Pointer to the robot connection being operated on

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
res = StopMove(pRobot);
if(ERROR_OK == res) { // Instruction sent successfully
} else { // Failure
}
```

3.2.11 StartJog (Control the robot to start JOG movement)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int StartJog(IntPtr pRobot, JogMode jog_mode, JogDir jog_dir, uint tool_index, uint wobj_index,
uint axis_index, byte link = 0);
```

Function

Control the robot to start JOG movement.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
jog_mode(in	JOG mode controls single axis motion in single axis mode, and in Cartesian mode controls motion in Cartesian coordinate system.
jog_dir(in)	JOG direction
tool_index(in)	Tool coordinate system serial number, 0: flange coordinate system, 1-32: custom tool coordinate system.
wobj_index(in)	Workpiece coordinate system serial number, 0: World coordinate system, 1-3: Channel base coordinate system, 4-3+WOBJ_NUM (Number of workpiece coordinate systems): Customize the workpiece coordinate system.
axis_index(in)	When in single axis mode, it represents the axis number. For example, for 6-axis robots, 1-6 represents the main axis, 7-12 represents the external axis. In Cartesian mode, 1-6 corresponds to the X/Y/Z/A/B/C direction, respectively.
link(in)	Linkage, default to false.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
// Set mode first before exercising
SetControlMode(pRobot, RobotMode.CONTROL_MODE_MANUAL);
// Set the speed multiplier to 50%
SetSpeedRatio(pRobot, 50);
res = StartJog(pRobot, JogMode.JOG_CARTESIAN, JogDir.DIR_POSITIVE, 0, 0, 1); // Control the robot to move forward on the X-axis
if(ERROR_OK == res) { // JOG motion instruction sent successfully
} else { // Failed
}
res = StopJog(pRobot);
if(ERROR_OK == res) { // Successfully sent JOG motion instruction to stop
} else { // Failed
}
res = StartJog(pRobot, JogMode.JOG_SINGLE_AXIS, JogDir.DIR_POSITIVE, 0, 0, 1); // Control robot 1 axis forward movement
if(ERROR_OK == res) { // JOG motion instruction sent successfully
} else { // Failed
}
res = StopJog(pRobot);
if(ERROR_OK == res) { // Successfully sent JOG motion instruction to stop
} else { // Failed
}
```

Before moving, it is recommended to set the control mode and speed override first. The overrides used by all motion instructions in the programming interface library are SPEED_RATIO_JOG type overrides.

3.2.12 StopJog (Control the robot to stop JOG movement)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int StopJog(IntPtr pRobot);
```

Function

Control the robot to stop JOG movement.

Parameter

pRobot(in)	Pointer to the robot connection being operated on

Return

0	Success.
Non-0	Error code.

Usage

```
res = StopJog(pRobot);
if(ERROR_OK == res) { // Command sent successfully
} else { // Failed
}
```

3.3 IO

3.3.1 GetDigitalIn (Get the digital input value of a certain channel)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int GetDigitalIn(IntPtr pRobot, int index, ref uint value);
```

Function

Get the digital input value of a certain channel.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
index(in)	DI serial number, starting from 1.
value(out)	DI value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
uint value = 0;
res = GetDigitalIn(pRobot, 1, ref value);
if(ERROR_OK == res) {    // DI value obtained successfully
} else {    // DI value obtained failed
}
```

3.3.2 GetMultiDigitalIn (Obtain a set of digital input values)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int GetMultiDigitalIn(IntPtr pRobot, uint start_index, uint end_index, ref uint value);
```

Function

Obtain a set of digital input values.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
start_index(in)	DI starting sequence number, starting from 1.
end_index(in)	DI end sequence number
value(out)	DI value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
uint value = 0;
res = GetMultiDigitalIn(pRobot, 1, 10,ref value);
if(ERROR_OK == res) {    // DI value obtained successfully
} else {    // DI value obtained failed
}
```

3.3.3 GetDigitalOut (Obtain the digital output value of a certain channel)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int GetDigitalOut(IntPtr pRobot, uint index, ref uint value);
```

Function

Obtain the digital output value of a certain channel.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
index(in)	DO serial number, starting from 1.
value(out)	DO value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
```

```
uint value = 0;
res = GetDigitalOut(pRobot, 1, ref value);
if(ERROR_OK == res) {    // DO value obtained successfully
} else {    // DO value obtained failed
}
```

3.3.4 GetMultiDigitalOut (Obtain a set of digital output values)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int GetMultiDigitalOut(IntPtr pRobot, uint start_index, uint end_index, ref uint value);
```

Function

Obtain a set of digital output values.

Parameter

pRobot(in)	Pointer to the robot connection being operated on.
start_index(in)	DO starting sequence number, starting from 1.
end_index(in)	DO end sequence number.
value(out)	DO value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
uint value = 0;
res = GetMultiDigitalOut(pRobot, 1, 10,ref value);
if(ERROR_OK == res) {    // DO value obtained successfully
} else {    // DO value obtained failed
}
```

3.3.5 SetDigitalOut (Set the digital output value of a certain channel)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int SetDigitalOut(IntPtr pRobot, uint index, uint value);
```

Function

Set the digital output value of a certain channel.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
index(in)	DO serial number, starting from 1.
value(out)	DO value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
uint value = 1;
res = SetDigitalOut(pRobot, 1, value);// Set the 1st DO to high level
if(ERROR_OK == res) { // DO value set successfully
} else { // DO value set failed
}
```

3.3.6 SetMultiDigitalOut (Set a set of digital output values)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int SetMultiDigitalOut(IntPtr pRobot, uint start_index, uint end_inde, uint value);
```

Function

Set a set of digital output values.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
start_index(in)	DO starting sequence number, starting from 1.
end_index(in)	DO end sequence number.
value(out)	DO value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
uint value = 255;
res = SetMultiDigitalOut(pRobot, 1, 10,value);// Set 1-10 DO channels to high level.
if(ERROR_OK == res) {    // DO value set successfully.
} else {    // DO value set failed.
```

```
}
```

3.4 Configuration

3.4.1 SetSpeedRatio(Set speed override)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int SetSpeedRatio(IntPtr pRobot, uint ratio);
```

Function

Set the speed override.

Parameter

pRobot(in)	Pointer to the robot connection being operated on.
ratio(in)	The magnification value, the range is 1-100.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
res = SetSpeedRatio (pRobot, 50);// Set the magnification to 50%
if(ERROR_OK == res) {    // The magnification is set successfully
} else {    // Set failed
}
```

3.4.2 SetToolCoordinateByIndex (Set tool coordinate system values through index)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int SetToolCoordinateByIndex (IntPtr pRobot, uint tool_index, RobotTool tool);
```

Function

Set tool coordinate system values through index.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
tool_index(in)	Tool coordinate system serial number, range 0-31.
tool(in)	The tool coordinate system value set.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotTool tool = new RobotTool() { x = 1, y = 1, z = 1, a = 1, b = 1, c = 1, stationary = 1, mu_name = "WORLD" };
// When initializing a structure, mu_ name cannot be defaulted
res = SetToolCoordinate ByIndex (pRobot, 0, tool);// Set the first tool coordinate system
if(ERROR_OK == res) {    // Coordinate system set successfully
} else {    // Set failed
}
```

3.4.3 SetToolCoordinateByName (Set tool coordinate system values by name)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int SetToolCoordinateByName (IntPtr pRobot, string tool_name, RobotTool tool);
```

Function

Set tool coordinate system values by name.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
tool_name(in)	The name of the tool coordinate system.
tool(in)	The tool coordinate system value set.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotTool tool = new RobotTool() { x = 1, y = 1, z = 1, a = 1, b = 1, c = 1, stationary = 1, mu_name = "WORLD" };
// When initializing a structure, mu_ name cannot be defaulted
res = SetToolCoordinateByName(pRobot, "t1", tool);// Set the t1 tool coordinate system
if(ERROR_OK == res) {    // Coordinate system set successfully
} else {    // Set failed
}
```

3.4.4 SetWorkpieceCoordinateByIndex (Set the workpiece coordinate system value through index)

Function declaration

[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int SetWorkpieceCoordinateByIndex (IntPtr pRobot, uint workpiece_index, RobotWorkpiece workpiece);

Function

Set the workpiece coordinate system value through index.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
workpiece _index(in)	The serial number of the workpiece coordinate system, ranging from 0 to 199.
workpiece (in)	The set value of the workpiece coordinate system.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotWorkpiece workpiece(10,20,30,0,0,0,0,"WORLD");
RobotWorkpiece workpiece = new RobotWorkpiece() { x = 1, y = 1, z = 1, a = 1, b = 1, c = 1, robhold = 1, mu_name = "WORLD" };    // When initializing a structure, mu_ Name cannot be defaulted
res = SetWorkpieceCoordinateByIndex (pRobot, 0, workpiece);// Set the first workpiece coordinate system
if(ERROR_OK == res) {    // Coordinate system set successfully
} else {    // Set failed
}
```

3.4.5 SetWorkpieceCoordinateByName (Set the workpiece coordinate system value by name)

Function declaration

[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int SetWorkpieceCoordinateByName (IntPtrpRobot, string workpiece_name, RobotWorkpiece workpiece);

Function

Set the workpiece coordinate system value by name.

Parameter

pRobot(in)	Pointer to the robot connection being operated on.
workpiece _name(in)	The name of the workpiece coordinate system.
workpiece (in)	The set value of the workpiece coordinate system.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotWorkpiece workpiece = new RobotWorkpiece() { x = 1, y = 1, z = 1, a = 1, b = 1, c = 1, robhold = 1, mu_name = "WORLD" };    // When initializing a structure, mu_ name cannot be defaulted
res = SetWorkpieceCoordinateByName (pRobot, "w1", workpiece);// Set the w1 workpiece coordinate system
if(ERROR_OK == res) {    // Coordinate system set successfully
} else {    // Set failed
}
```

3.4.6 SetIntVariableByIndex (Set integer variable values through index)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int SetIntVariableByIndex (IntPtr pRobot, uint index, int value);
```

Function

Set integer variable values through index.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
index(in)	Variable sequence number, starting from 0.
value(in)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
int value = 100;
res = SetIntVariableByIndex (pRobot, 0, value);    // Set the first integer value
if(ERROR_OK == res) {    // Set Successfully
} else {    // Set failed
}
```

3.4.7 SetIntVariableByName (Set integer variable value by name)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
```

```
public static extern int SetIntVariableByName (IntPtr pRobot, string name, int value);
```

Function

Set integer variable value by name.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
name(in)	Variable name.
value(in)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
int value = 100;
res = SetIntVariableByName (pRobot, "i1", value); // Set i1 integer value
if(ERROR_OK == res) {    // Set successfully
} else {    // Set failed
}
```

3.4.8 SetDoubleVariableByIndex (Set floating point variable values through index)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int SetDoubleVariableByIndex (IntPtr pRobot, uint index, double value);
```

Function

Set floating point variable values through index.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
index(in)	Variable sequence number, starting from 0.
value(in)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
double value = 100.1;
res = SetDoubleVariableByIndex (pRobot, 0, value);    // Set the first floating-point value
if(ERROR_OK == res) {    // Set successfully
} else {    // Set failed
}
```

3.4.9 SetDoubleVariableByName (Set floating point variable value by name)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int SetDoubleVariableByName (IntPtr pRobot, string name, double value);
```

Function

Set floating point variable value by name.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
name(in)	Variable name.
value(in)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
double value = 100.1;
res = SetDoubleVariableByName (pRobot, "d1", value);    // Set d1 floating-point value
if(ERROR_OK == res) {    // Set successfully
} else {    // Set failed
}
```

3.4.10 SetBoolVariableByIndex (Set Boolean variable value through index)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int SetBoolVariableByIndex (IntPtr pRobot, uint index, byte value);
```

Function

Set Boolean variable value through index.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
index(in)	Variable sequence number, starting from 0.
value(in)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
int value;
res = SetBoolVariableByIndex (pRobot, 0, value);    // Set the first Boolean value
if(ERROR_OK == res) {    // Set successfully
} else {    // Set failed
}
```

3.4.11 SetBoolVariableByName (Set Boolean variable value by name)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int SetBoolVariableByName (IntPtr pRobot, string name, byte value);
```

Function

Set Boolean variable value by name.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
name(in)	Variable name.
value(in)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
int value;
res = SetBoolVariableByName (pRobot, "b1", value);    // Set b1 Boolean value
if(ERROR_OK == res) {    // Set successfully
} else {    // Set failed
```

```
}
```

3.4.12 SetStringVariableByIndex (Set string variable values through index)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int SetStringVariableByIndex (IntPtr pRobot, uint index, string value);
```

Function

Set string variable values through index.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
index(in)	Variable sequence number, starting from 0.
value(in)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
string value = "a3";
res = SetStringVariableByIndex (pRobot, 0, value);    // Set the first string value
if(ERROR_OK == res) {    // Set successfully
} else {    // Set failed
}
```

3.4.13 SetStringVariableByName (Set string variable values by name)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int SetStringVariableByName (IntPtr pRobot, string name, string value);
```

Function

Set string variable values by name.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
name(in)	Variable name.
value(in)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
string value = "s1_name";
res = SetStringVariableByName (pRobot, "s1", value);    // Set s1 string value
if(ERROR_OK == res) {    // Set successfully
} else {    // Set failed
}
```

3.4.14 SetPoseVariableByIndex (Set pose variable values through index)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int SetPoseVariableByIndex (IntPtr pRobot, uint index, RobotPos value , int tool_index = -1, int wobj_index = -1);
```

Function

Set pose variable values through index.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
index(in)	Variable sequence number, starting from 0.
value(in)	Variable value.
tool_index(in)	The tool coordinate system number for teaching this pose, 0: flange coordinate system, 1-TOOL_NUM (Number of Tool Coordinate Systems): Customize the tool coordinate system, with a default value of -1 indicating no setting
wobj_index(in)	The workpiece coordinate system number during this pose teaching, 0: world coordinate system, 1-3: channel base coordinate system, 4-3+WOBJ_NUM (Number of workpiece coordinate systems): Customized workpiece coordinate system, with a default value of -1 indicating no setting

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotPos value = pos1; // Pos1 is defined in 3.2.3
int tool_index = 0;
int wobj_index = 0;
res = SetStringVariableByIndex (pRobot, 0, value, tool_index, wobj_index);    // Set the first pose value, and set the tool coordinate system to the flange coordinate system and the workpiece coordinate system to WORLD
if(ERROR_OK == res) {    // Set successfully
```

```
} else {    // Set failed
}
```

3.4.15 SetPoseVariableByName (Set pose variable values by name)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int SetPoseVariableByName (IntPtr pRobot, string name, RobotPos value , int tool_index = -1, int
wobj_index = -1);
```

Function

Set pose variable values by name.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
index(in)	Variable name.
value(in)	Variable value.
tool_index(in)	The tool coordinate system number for teaching this pose, 0: flange coordinate system, 1-TOOL_NUM (Number of Tool Coordinate Systems): Customize the tool coordinate system, with a default value of -1 indicating no setting
wobj_index(in)	The workpiece coordinate system number during this pose teaching, 0: world coordinate system, 1-3: channel base coordinate system, 4-3+WOBJ_NUM (Number of workpiece coordinate systems): Customized workpiece coordinate system, with a default value of -1 indicating no setting

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotPos value = pos1; // Pos1 is defined in 3.2.3
int tool_index = 0;
int wobj_index = 0;
res = SetStringVariableByName (pRobot, "p1", value);    // Set the pose value of p1, and set the tool coordinate
system to the flange coordinate system and the workpiece coordinate system to WORLD
if(ERROR_OK == res) {    // Set successfully
} else {    // Set failed
}
```

3.4.16 SetJointVariableByIndex (Set joint type variable values through index)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int SetJointVariableByIndex(IntPtr pRobot, uint index, RobotJoint value);
```

Function

Set joint type variable values through index.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
index(in)	Variable sequence number, starting from 0.
value(in)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotJoint value;
value[0] = 10;
res = SetJointVariableByIndex (pRobot, 0, value);    // Set the first joint type value
if(ERROR_OK == res) {    // Set successfully
} else {    // Set failed
}
```

3.4.17 SetJointVariableByName (Set joint type variable values by name)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int SetJointVariableByName (IntPtr pRobot, string name, RobotJoint value);
```

Function

Set joint type variable values by name.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
name(in)	Variable name.
value(in)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
```

```
RobotJoint value;
value[0] = 10;
res = SetJointVariableByName (pRobot, "j1", value);    // Set j1 joint type value
if(ERROR_OK == res) {    // Set successfully
} else {    // Set failed
}
```

3.4.18 SetVariableName (Set system variable name)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int SetVariableName(IntPtr pRobot, VarNameType type, uint index, string name);
```

Function

Set system variable name.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
type(in)	Variable type.
index(in)	Variable sequence number.
name(in)	Variable name

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
res = SetVariableName(pRobot, VarNameType.VAR_INT, 0, "V1");    // Set the first integer variable name to 'V1'
if(ERROR_OK == res) {    // Set successfully
} else {    // Set failed
}
```

3.4.19 SetCurCoordinate (Set JOG tool and workpiece coordinate system)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int SetCurCoordinate(IntPtr pRobot, uint tool_index, uint wobj_index);
```

Function

Set JOG tool and workpiece coordinate system.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
tool_index(in)	Tool coordinate system serial number, 0: flange coordinate system, 1-TOOL_NUM (number of tool coordinate systems): customizing tool coordinate systems
wobj_index(in)	Workpiece coordinate system serial number, 0: World coordinate system, 1-3: Channel base coordinate system, 4-3+WOBJ_NUM (Number of workpiece coordinate systems): Custom workpiece coordinate system

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
res = SetCurCoordinate (pRobot, 0, 0);    // Set the JOG tool coordinate system to the flange coordinate system and the workpiece coordinate system to WORLD
if(ERROR_OK == res) {    // Set successfully
} else {    // Set failed
}
```

3.5 Inquire

3.5.1 GetMaxConnectionCount (Obtain the maximum allowed number of connections)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern unsigned int GetMaxConnectionCount();
```

Function

Obtain the maximum allowed number of connections, the default value is 100.

Return

100

Usage

```
uint res = GetMaxConnectionCount();
```

3.5.2 GetCurrentConnectionCount (Obtain the number of existing connections)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern unt GetCurrentConnectionCount();
```

Function

Obtain the number of existing connections.

Return

0	Connection not created
Non-0	Number of connections created - Number of connections destroyed

Usage

```
uint res = GetMaxConnectionCount();
```

3.5.3 GetControlMode (Query the current robot control mode)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int GetControlMode(IntPtr pRobot, ref RobotMode mode);
```

Function

Query the current robot control mode.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
mode(out)	The current mode.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotMode mode = RobotMode.ROBOT_MODE_MANUAL;
res = GetControlMode (pRobot, ref mode);
if(ERROR_OK == res) {    // Status query successful
} else {    // Query failed
}
```

3.5.4 GetProgramState (Query the current running status of the robot)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int GetProgramState(IntPtr pRobot, ref ProgramState state);
```

Function

Query the current running status of the robot.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
state(out)	Current operating status.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
ProgramState state = ProgramState. PROG_STATE_NOT_LOAD;
res = GetProgramState (pRobot, ref state);
if(ERROR_OK == res) { // Status query successful
} else { // Query failed
}
```

The program running status is defined as follows:

```
PROG_STATE_NOT_LOAD, // The program is not loaded
PROG_STATE_RUNNING, // Running
PROG_STATE_PAUSE, // Time out
PROG_STATE_STOP // Stop
```

3.5.5 GetSpeedRatio(Query the current speed override)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int GetSpeedRatio(IntPtr pRobot, ref uint ratio);
```

Function

Query the current speed override.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
ratio(out)	The current speed override.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
uint ratio = 1;
res = GetSpeedRatio (pRobot, ref ratio);// Query JOG magnification value
if(ERROR_OK == res) {    // Status query successful
} else {    // Query failed
}
```

3.5.6 GetLoadedProg (Query the currently loaded path)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int GetLoadedProg(IntPtr pRobot, StringBuilder file_path);
```

Function

Query the currently loaded path.

Parameter

pRobot(in)	Pointer to the robot connection being operated on.
file_path(out)	The current loading file path.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
StringBuilder file_path = new StringBuilder(100);
res = GetLoadedProg (pRobot, file_path);// Query loaded file path
if(ERROR_OK == res) {    // Status query successful
} else {    // Query failed
}
```

3.5.7 IsPowerOn (Query whether it is currently powered on)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int IsPowerOn(IntPtr pRobot, ref byte is_poweron);
```

Function

Query whether it is currently powered on.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
is_poweron(out)	Whether it has been powered on.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
byte is_poweron = 0;
res = IsPowerOn (pRobot, ref is_poweron);
if(ERROR_OK == res) {    // Status query successful
} else {    // Query failed
}
```

3.5.8 IsConnected (Check if it is currently connected to the robot)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern byte IsConnected(IntPtr pRobot);
```

Function

Check if it is currently connected to the robot.

Parameter

pRobot(in)	Pointer to the robot connection being operated on

Return

true	Connected
false	Not connected

Usage

```
byte is_connected = IsConnected(pRobot);
```

3.5.9 GetPos (Query the current robot pose)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int GetPos(IntPtr pRobot, ref RobotPos robot_pos);
```

Function

Query the current robot pose.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
robot_pos(out)	Current pose.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotPos pos = pos1; // Pos1 is defined in 3.2.3
res = GetPos (pRobot, pos);
if(ERROR_OK == res) {    // Query successful
} else {    // Query failed
}
```


Caution

The pose returned by this interface refers to the workpiece coordinate system set by the previous motion instruction.

3.5.10 GetJoint (Query the current angle of each axis of the robot)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int GetJoint(IntPtr pRobot, ref RobotJoint robot_joint);
```

Function

Query the current angle of each axis of the robot, unit: degree.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
robot_joint(out)	The current angle of each axis.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
```

```
RobotJoint joint;
joint[0] = 10;
res = GetJoint (pRobot, joint);
if(ERROR_OK == res) {    // Query successful
} else {    // Query failed
}
```

3.5.11 GetAlarmState (Query the current alarm status)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int GetAlarmState(IntPtr pRobot, ref byte state);
```

Function

Query the current alarm status.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
state(out)	Whether it is currently alarming.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
byte state = 0;
res = GetAlarmState (pRobot, state);
if(ERROR_OK == res) {    // Query successful
} else {    // Query failed
}
```

3.5.12 GetAlarmList (Query the current alarm list)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int GetAlarmList(IntPtr pRobot, uint[] list, uint list_size);
```

Function

Query the current alarm list.

Parameter

pRobot(in)	Pointer to the robot connection being operated on

list(out)	List of output alarms.
list_size(in)	The length of the alarm list, with a maximum of 500, it is recommended to input a value not greater than 500

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
uint list_size = 10;
uint[] list = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
res = GetAlarmList (pRobot, list, list_size);
if(ERROR_OK == res) {    // Query successful
} else {    // Query failed
}
```

This interface returns a list of alarm codes. For the specific meaning of each alarm code, please refer to the Peking Robot Diagnostic Manual.

3.5.13 GetAxisAlarm (Obtain the list of driver alarm codes for abnormal axes)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int GetAxisAlarm(IntPtr pRobot, int[] alarm, uint alarm_size);
```

Function

Obtain the list of driver alarm codes for abnormal axes. Currently, it is only applicable to the 1-6 axes of the manipulator.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
alarm(out)	Data class alarm (out): A list of abnormal axis driver alarm codes, with axis sequence numbers starting from 1.
alarm_size(in)	Maximum number of axles MAX_ SLAVE_ SERVO_ NUM, current value is 72, it is recommended to pass in a value not greater than 72

Return

0	Query successful.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
uint alarm_size = 6;
int[] alarm = {0, 0, 0, 0, 0, 0};
res = GetAxisAlarm (pRobot, alarm, alarm_size);
if(ERROR_OK == res) {    // Query was successful
} else {    // Query failed
}
```

3.5.14 GetAxisData (Obtain current data for each axis)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int GetAxisData(IntPtr pRobot, DataType data_type, uint axis_index, ref double data);
```

Function

Obtain current data for each axis.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
data_type(in)	Data type
axis_index(in)	Axis number, starting from 1
data(out)	Return data

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
double data;
res = GetAxisData (pRobot, DataType.CURRENT_SPEED, 1, ref data); // Obtain the current speed of the first axis
if(ERROR_OK == res) {    // Query was successful
} else {    // Query failed
}
The types of data available are:
// Axis data type
enum DataType {
    CURRENT_COMMAND = 0, // Current instruction position
    CURRENT_POS,    // Current real-time location
    CURRENT_SPEED,    // Current speed
    CURRENT_TORQUE,    // Current torque
    CURRENT_CURRENT    // Current current
```

```
};
```

3.5.15 GetRobotRunTime (Obtain the accumulated running time of the current robot)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int GetRobotRunTime(IntPtr pRobot, ref double time);
```

Function

Obtain the accumulated running time of the current robot.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
time(out)	Accumulated running time, in seconds.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
double time = 0.0;
res = GetRobotRunTime (pRobot, ref time);
if(ERROR_OK == res) {    // Query was successful
} else {    // Query failed
}
```

3.5.16 GetToolCoordinateByIndex (Retrieve tool coordinate system values through index)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int GetToolCoordinateByIndex (IntPtr pRobot, uint tool_index, ref RobotTool tool);
```

Function

Retrieve tool coordinate system values through index.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
tool_index(in)	Tool coordinate system serial number, range 0-31.
tool(out)	The tool coordinate system value obtained.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotTool tool = new RobotTool() { x = 1, y = 1, z = 1, a = 1, b = 1, c = 1, stationary = 1, mu_name = "WORLD" }; // When initializing a structure, mu_ name cannot be defaulted.
res = GetToolCoordinateByIndex (pRobot, 0, ref tool);// Obtain the first tool coordinate system
if(ERROR_OK == res) {    // Successfully obtained coordinate system
} else {    // Obtain failed
}
```

3.5.17 GetToolCoordinateByName (Obtain tool coordinate system values by name)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int GetToolCoordinateByName (IntPtr pRobot, string tool_name, ref RobotTool tool);
```

Function

Obtain tool coordinate system values by name.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
tool_name(in)	The name of the tool coordinate system.
tool(out)	The tool coordinate system value obtained.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotTool tool = new RobotTool() { x = 1, y = 1, z = 1, a = 1, b = 1, c = 1, stationary = 1, mu_name = "WORLD" }; // When initializing a structure, mu_ name cannot be defaulted
res = GetToolCoordinateByName(pRobot, "t1", ref tool);// Obtain the t1 tool coordinate system
if(ERROR_OK == res) { // Successfully obtained coordinate system
} else { // Obtain failed
}
```

3.5.18 GetWorkpieceCoordinateByIndex (Retrieve workpiece coordinate system values through index)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
```

public static extern int GetWorkpieceCoordinateByIndex (IntPtr pRobot, uint workpiece_index, ref RobotWorkpiece workpiece);

Function

Retrieve workpiece coordinate system values through index.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
workpiece _index(in)	The serial number of the workpiece coordinate system, ranging from 0 to 199.
workpiece (out)	The obtained workpiece coordinate system value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotWorkpiece workpiece = new RobotWorkpiece() { x = 1, y = 1, z = 1, a = 1, b = 1, c = 1, robhold = 1, mu_name = "WORLD" };    // When initializing a structure, mu_ name cannot be defaulted
res = GetWorkpieceCoordinateByIndex (pRobot, 0, ref workpiece);// Obtain the first workpiece coordinate system
if(ERROR_OK == res) {    // Successfully obtained coordinate system
} else {    // Obtained failed
}
```

3.5.19 GetWorkpieceCoordinateByName (Obtain the workpiece coordinate system value by name)

Function declaration

[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

pubic static extern int GetWorkpieceCoordinateByName (IntPtr pRobot, string workpiece_name, ref RobotWorkpiece workpiece);

Function

Obtain the workpiece coordinate system value by name.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
workpiece _name(in)	The name of the workpiece coordinate system.
workpiece (out)	The obtained workpiece coordinate system value.

Return

0	Success.

Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotWorkpiece workpiece = new RobotWorkpiece() { x = 1, y = 1, z = 1, a = 1, b = 1, c = 1, robhold = 1, mu_name = "WORLD" };    // When initializing a structure, mu_ name cannot be defaulted
res = GetWorkpieceCoordinateByName (pRobot, "w1", ref workpiece);// Obtain the w1 workpiece coordinate system
if(ERROR_OK == res) {    // Successfully obtained coordinate system
} else {    // Obtained failed
}
```

3.5.20 GetIntVariableByIndex (Query integer variable values through index)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int GetIntVariableByIndex (IntPtr pRobot, uint index, ref int value);
```

Function

Query integer variable values through index.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
index(in)	Variable sequence number, starting from 0.
value(out)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
int value = 0;
res = GetIntVariableByIndex (pRobot, 0, ref value);    // Query the first integer value
if(ERROR_OK == res) {    // Query was successful
} else {    // Query failed
}
```

3.5.21 GetIntVariableByName (Query integer variable values by name)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int GetIntVariableByName (IntPtr pRobot, string name, ref int value);
```

Function

Query integer variable values by name.

Parameter

pRobot(in)	Pointer to the robot connection being operated on.
name(in)	Variable name
value(out)	Variable value

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
int value = 0;
res = GetIntVariableByName (pRobot, "i1", ref value);    // Query i1 integer value
if(ERROR_OK == res) {    // Query was successful
} else {    // Query failed
}
```

3.5.22 GetDoubleVariableByIndex (Query floating point variable values through index)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int GetDoubleVariableByIndex (IntPtr pRobot, uint index, ref double value);
```

Function

Query floating point variable values through index.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
index(in)	Variable sequence number, starting from 0.
value(out)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
int value = 0;
```

```
res = GetDoubleVariableByIndex (pRobot, 0, ref value);    // Query the first floating-point value
if(ERROR_OK == res) {    // Query was successful
} else {    // Query failed
}
```

3.5.23 GetDoubleVariableByName (Query floating point variable values by name)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int GetDoubleVariableByName (IntPtr pRobot, string name, ref double value);
```

Function

Query floating point variable values by name.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
name(in)	Variable name.
value(out)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
int value = 0;
res = GetDoubleVariableByName (pRobot, "d1", ref value);    // Query d1 floating-point values
if(ERROR_OK == res) {    // Query was successful
} else {    // Query failed
}
```

3.5.24 GetBoolVariableByIndex (Query Boolean variable values through index)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int GetBoolVariableByIndex (IntPtr pRobot, uint index, ref byte value);
```

Function

Query Boolean variable values through index.

Parameter

pRobot(in)	Pointer to the robot connection being operated on

index(in)	Variable sequence number, starting from 0.
value(out)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
byte value = 0;
res = GetBoolVariableByIndex (pRobot, 0, ref value);    // Query the first Boolean value
if(ERROR_OK == res) {    // Query was successful
} else {    // Query failed
}
```

3.5.25 GetBoolVariableByName (Query Boolean variable values by name)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int GetBoolVariableByName (IntPtr pRobot, string name, ref byte value);
```

Function

Query Boolean variable values by name.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
name(in)	Variable name.
value(out)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
byte value = 0;
res = GetBoolVariableByName (pRobot, "b1", ref value);    // Query b1 boolean value
if(ERROR_OK == res) {    // Query was successful
} else {    // Query failed
}
```

3.5.26 GetStringVariableByIndex (Retrieve string variable values through index)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int GetStringVariableByIndex (IntPtr pRobot, uint index, StringBuilder value);
```

Function

Retrieve string variable values through index.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
index(in)	Variable sequence number, starting from 0.
value(out)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
StringBuilder value = new StringBuilder(100);
res = GetStringVariableByIndex (pRobot, 0, value);    // Get the first string value
if(ERROR_OK == res) {    // Query was successful
} else {    // Query failed
}
```

3.5.27 GetStringVariableByName (Obtain string variable values by name)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int GetStringVariableByName (IntPtr pRobot, string name, StringBuilder value);
```

Function

Obtain string variable values by name.

Parameter

pRobot(in)	Pointer to the robot connection being operated on.
name(in)	Variable name
value(out)	Variable value

Return

0	Success.

Non-0	Error code.

Usage

```
int res = ERROR_OK;
StringBuilder value = new StringBuilder(100);
res = GetStringVariableByName (pRobot, "s1", value);    // Obtain the s1 string value
if(ERROR_OK == res) {    // Query was successful
} else {    // Query failed
}
```

3.5.28 GetPoseVariableByIndex (Retrieve pose variable values through index)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int GetPoseVariableByIndex (IntPtr pRobot, uint index, ref RobotPos value , ref int tool_index, ref int wobj_index);
```

Function

Retrieve pose variable values through index.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
index(in)	Variable sequence number, starting from 0.
value(out)	Variable value
tool_index(out)	The tool coordinate system number for teaching this pose, 0: flange coordinate system, 1-TOOL_NUM (number of tool coordinate systems): customizing tool coordinate systems
wobj_index(out)	The workpiece coordinate system number during this pose teaching, 0: world coordinate system, 1-3: channel base coordinate system, 4-3+WOBJ_NUM (Number of workpiece coordinate systems): Custom workpiece coordinate system

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotPos value = pos1;        // Pos1 is defined in 3.2.3
int tool_index = 0, wobj_index = 0;
res = GetStringVariableByIndex (pRobot, 0, ref value, ref tool_index, ref wobj_index);    // Obtain the first pose value
if(ERROR_OK == res) {    // Query was successful
} else {    // Query failed
}
```

3.5.29 GetPoseVariableByName (Obtain pose variable values by name)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int GetPoseVariableByName (IntPtr pRobot, string name, ref RobotPos value , ref int tool_index,
ref int wobj_index);
```

Function

Obtain pose variable values by name.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
index(in)	Variable name
value(out)	Variable value
tool_index(out)	The tool coordinate system number for teaching this pose, 0: flange coordinate system, 1-TOOL_NUM (number of tool coordinate systems): customizing tool coordinate systems
wobj_index(out)	The workpiece coordinate system number during this pose teaching, 0: world coordinate system, 1-3: channel base coordinate system, 4-3+WOBJ_NUM (Number of workpiece coordinate systems): Custom workpiece coordinate system

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotPos value = pos1;        // Pos1 is defined in 3.2.3
int tool_index = 0, wobj_index = 0;
res = GetStringVariableByName (pRobot, "p1", ref value, ref tool_index, ref wobj_index);    // Obtain p1 pose value
if(ERROR_OK == res) {    // Query was successful
} else {    // Query failed
}
```

3.5.30 GetJointVariableByIndex (Set joint type variable values through index)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int GetJointVariableByIndex (IntPtr pRobot, uint index, ref RobotJoint value);
```

Function

Set joint type variable values through index.

Parameter

pRobot(in)	Pointer to the robot connection being operated on.

index(in)	Variable sequence number, starting from 0.
value(out)	Variable value

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotJoint value;
value[0] = 10;
res = GetJointVariableByIndex (pRobot, 0, ref value);    // Obtain the first joint type value
if(ERROR_OK == res) {    // Query was successful
} else {    // Query failed
}
```

3.5.31 GetJointVariableByName (Obtain joint type variable values by name)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int GetJointVariableByName (IntPtr pRobot, string name, ref RobotJoint value);
```

Function

Obtain joint type variable values by name.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
name(in)	Variable name
value(out)	Variable value

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotJoint value;
value[0] = 10;
res = GetJointVariableByName (pRobot, "j1", ref value);    // Obtain j1 joint type value
if(ERROR_OK == res) {    // Query was successful
} else {    // Query failed
}
```

3.5.32 GetCurCoordinate (Obtain JOG tool and workpiece coordinate system)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int GetCurCoordinate(IntPtr pRobot, ref uint tool_index, ref uint wobj_index);
```

Function

Obtain JOG tool and workpiece coordinate system.

Parameter

pRobot(in)	Pointer to the robot connection being operated on.
tool_index(out)	Tool coordinate system serial number, 0: flange coordinate system, 1-TOOL_NUM (number of tool coordinate systems): customizing tool coordinate systems
wobj_index(out)	Workpiece coordinate system serial number, 0: World coordinate system, 1-3: Channel base coordinate system, 4-3+WOBJ_NUM (Number of workpiece coordinate systems): Custom workpiece coordinate system

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
uint tool_index = 0, wobj_index = 0;
res = GetCurCoordinate (pRobot, ref tool_index, ref wobj_index);    // Obtain the JOG tool coordinate system and workpiece coordinate system
if(ERROR_OK == res) {    // Query was successful
} else {    // Query failed
}
```

3.5.33 FkSolver (Robot pose and joint angle forward solution interface)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int FkSolver(IntPtr pRobot, RobotJoint joint, uint tool_index, uint wobj_index, ref RobotPos pos);
```

Function

Robot pose and joint angle forward solution interface.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
joint(in)	Angle of each axis

tool_index(in)	Tool coordinate system serial number, 0: flange coordinate system, 1-TOOL_NUM (number of tool coordinate systems): customizing tool coordinate systems
wobj_index(in)	Workpiece coordinate system serial number, 0: World coordinate system, 1-3: Channel base coordinate system, 4-3+WOBJ_NUM (Number of workpiece coordinate systems): Custom workpiece coordinate system
pos(out)	Pose obtained through forward solution transformation

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotJoint joint;
joint[0] = 10;
uint tool_index = 0, wobj_index = 0;
RobotPos pos = pos1;        // Pos1 is defined in 3.2.3
res = FkSolver (pRobot, joint, tool_index, wobj_index, ref pos);
if(ERROR_OK == res) {    // Query was successful
} else {    // Query failed
}
```

3.5.34 IkSolver (Interface for inverse solution of robot pose and joint angle)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int IkSolver(IntPtr pRobot, RobotPos pos, uint tool_index, uint wobj_index, ref RobotJoint joint);
```

Function

Interface for inverse solution of robot pose and joint angle.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
pos(in)	Robot pose
tool_index(in)	Tool coordinate system serial number, 0: flange coordinate system, 1-TOOL_NUM (number of tool coordinate systems): customizing tool coordinate systems
wobj_index(in)	Workpiece coordinate system serial number, 0: World coordinate system, 1-3: Channel base coordinate system, 4-3+WOBJ_NUM (Number of workpiece coordinate systems): Custom workpiece coordinate system
joint(out)	The angles of each axis obtained through inverse solution transformation

Return

0	Success.

Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotJoint joint;
joint[0] = 10;
uint tool_index = 0, wobj_index = 0;
RobotPos pos = pos1;        // Pos1 is defined in 3.2.3
res = IkSolver (pRobot, pos, tool_index, wobj_index, ref joint);
if(ERROR_OK == res) {    // Query was successful
} else {    // Query failed
}
```

3.6 Program running

3.6.1 SendProgram (Send ARL program)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int SendProgram(IntPtr pRobot, string file_path);
```

Function

Send the ARL program.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
file_path(in)	ARL program file name (including the full path) or folder name (send the entire directory).

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
string file_path = "d:\\test_program";    // File paths are case sensitive
res = SendProgram (pRobot, file_path);    // Send d:/test_ program entire folder
if(ERROR_OK == res) {    // Sent successfully
} else {    // Fail in send
}
```

All files and folders will be sent to the script directory of the robot.

3.6.2 LoadProgram (Load ARL program)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int LoadProgram(IntPtr pRobot, string file_path);
```

Function

Load the ARL program.

Parameter

pRobot(in)	Pointer to the robot connection being operated on
file_path(in)	ARL program file name (the path relative to the script directory).

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
string file_path = "d:\\test_program";    // Path is strictly case sensitive
res = SendProgram (pRobot, file_path);    // Send d:/test_ program entire folder
if(ERROR_OK == res) {    // Execution succeeded
} else {    // Execution failed
}
file_path = "test_program\\test.arl";      // Path is strictly case sensitive
res = LoadProgram (pRobot, file_path);    // Load previously sent the test.arl program in the test_program directory
if(ERROR_OK == res) {    // Execution succeeded
} else {    // Execution failed
}
```

3.6.3 StartProgram (Start ARL program)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int StartProgram(IntPtr pRobot);
```

Function

Start the ARL program.

Parameter

pRobot(in)	Pointer to the robot connection being operated on

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
string file_path = "d:\\test_program";
res = SendProgram (pRobot, file_path);    // Send d:/test_ program entire folder
if(ERROR_OK == res) {    // Execution succeeded
} else {    // Execution failed
}
file_path = "test_program\\test.arl";
res = LoadProgram (pRobot, file_path);    // Load previously sent the test.arl program in the test_program directory
if(ERROR_OK == res) {    // Execution succeeded
} else {    // Execution failed
}
res = StartProgram (pRobot);    // Start program
if(ERROR_OK == res) {    // Execution succeeded
} else {    // Execution failed
}
```

3.6.4 PauseProgram (Pause the program)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int PauseProgram(IntPtr pRobot);
```

Function

Pause the program.

Parameter

pRobot(in)	Pointer to the robot connection being operated on

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
res = PauseProgram (pRobot);
if(ERROR_OK == res) {    // Pause instruction sent successfully
} else {    // Pause failed
}
```

3.6.5 ResetProgram (Reset program)

Function declaration

```
[DllImport("RobotAPI.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int ResetProgram(IntPtr pRobot);
```

Function

Reset the program.

Parameter

pRobot(in)	Pointer to the robot connection being operated on

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
res = ResetProgram (pRobot);
if(ERROR_OK == res) {    // Reset successfully
} else {    // Reset failed
}
```


Caution

The program can be reset only when the system status is paused or stopped.

3.7 Control multiple robots

```
// Function import can be found in the descriptions of each function, and only the main function instance is provided
here
static int Main(string[] args)
{
Console.WriteLine("Program start");
// Obtain maximum allowed connections
        uint max_connection_count = GetMaxConnectionCount();
        Console.WriteLine("max_connection_count: {0}", max_connection_count);

        // Test establishing 5 connections
```

```
        IntPtr[] robot = new IntPtr[5] { IntPtr.Zero, IntPtr.Zero, IntPtr.Zero, IntPtr.Zero, IntPtr.Zero };
        for (uint i = 0; i < 5; i++)
        {
            robot[i] = CreateCRobotAPI(i);
}
        int res = ConnectRobot(robot[1], "192.168.0.101");
        if (res != 0)
        {
            Console.WriteLine("robot[1] connect failed, res = {0}", res);
        }
          res = ConnectRobot(robot[2], "192.168.0.102");
          if (res != 0)
          {
              Console.WriteLine("robot[2] connect failed");
          }
          // Test deleting useless connections
          DestoryCRobotAPI(robot[4]);

          // Obtain the number of existing connections
          uint curr_connection_count = GetCurrentConnectionCount();
          Console.WriteLine("curr_connection_count: {0}", curr_connection_count);
        // Calling functions with robot [1]
          byte enable = 0;
          res = EnableApiControl(robot[1], enable);
          if (res != 0)
          {
              Console.WriteLine("EnableApiControl faileded!");
}
Console.WriteLine("Program over!");
}
```

4 Data structure description

The programming interface library declares its own namespace RobotAPI, which can be used by using namespace RobotAPI during development.

4.1 RobotPos (Robot position and posture data)

```
// Robot position and posture data
public struct RobotPos {
public double x;
public double y;
public double z;
public double a;
public double b;
public double c;
public int cfg;
public int turn;
public double ej1;
public double ej2;
public double ej3;
public double ej4;
public double ej5;
public double ej6;
};
```

For the description of RobotPos related parameters, see Table 4-1.

Table 4-1 Description of RobotPos related parameters

Name	Description
x	The x component of the robot TCP point relative to the coordinates of the current workpiece coordinate system, in millimeters
y	The y component of the robot TCP point relative to the coordinates of the current workpiece coordinate system, in millimeters
z	The z component of the robot TCP point relative to the coordinates of the current workpiece coordinate system, in millimeters

Name	Description
a	The a component expressed by the Euler angle of the robot tool posture in the current workpiece coordinate system, in degree
b	The b component expressed by the Euler angle of the robot tool posture in the current workpiece coordinate system, in degree
c	The c component expressed by the Euler angle of the robot tool posture in the current workpiece coordinate system, in degree
cfg	Robot axis configuration. Since the robot may have several different ways to make the TCP reach the same pose, it is necessary to specify one of the ways through the cfg parameter to uniquely determine a set of robot axis positions
turn	Used with cfg to determine the axis position of the inverse solution Only the lower 6 bits of turn are used here, bit0 means 1 axis, bit1 means 2 axis, and so on. When the turn value is -1, it means that the solution closest to the starting point is automatically selected; when a bit value is 1, it means that the axis selects the solution less than 0; when a bit value is 0, it means that the axis is selected greater than 0 Solution. When the axis limit range is greater than 360 degrees, this parameter may be used to assist solution selection. In most other cases, this value does not need to be set, and the default value is -1.
ej1-ej6	The position of outer 1 axis ~ outer 6 axis, unit degree

Euler angles are different according to the order of the axis around which they rotate. There are 12 different ways of expressing Euler angles. What we use here is the ZYX type Euler angles, that is, the order of converting from the initial coordinate system posture to the transformed coordinate system posture is the first Rotate by angle a around the z axis, then rotate by angle b around the y axis, and finally rotate by angle c around the x axis.

4.2 RobotJoint (Angle data of each axis of the robot)

```
// Angle data of each axis of the robot
public struct RobotJoint {
public double j[12];
};
```

For the description of RobotJoint related parameters, see Table 4-2.

Table 4-2Description of RobotJoint related parameters

Name	Description
j	Angle values of 6 joint axes and 6 external axes, unit: degree

4.3 RobotTool (Tool coordinate system data)

```
// Tool coordinate system
public struct RobotTool {
public double x;
public double y;
```

```
public double z;
public double a;
public double b;
public double c;
public byte stationary;
public string mu_name;
};
```

For the description of RobotPosTool related parameters, see Table 4-3.

Table 4-3Description of RobotTool related parameters

Name	Description
x,y,z	Tool coordinate system x, y, z components, unit: mm
a,b,c	Tool coordinate system a, b, c components, unit: degree
stationary	Whether it is a fixed tool, the non-fixed tool is installed on the robot flange, the fixed tool does not move relative to the world coordinate system, and the default is a non-fixed tool
mu_name	Reference mechanical unit name, defaults to the reference world coordinate system

4.4 RobotWorkpiece (Workpiece coordinate system data)

```
// Workpiece coordinate system
public struct RobotWorkpiece {
public double x;
public double y;
public double z;
public double a;
public double b;
public double c;
public byte robhold;
string mu_name;
};
```

For the description of RobotWorkpiece related parameters, see Table 4-4.

Table 4-4Description of RobotWorkpiece related parameters

Name	Description
x,y,z	Workpiece coordinate system x, y, z components, unit: mm
a,b,c	Workpiece coordinate system a, b, c components, unit: degree
robhold	Whether the robot grabs the workpiece, the robot grabs the workpiece and installs it on the robot flange, the non-robot grabs the workpiece does not move relative to the world coordinate system, the default is the robot grabs the workpiece
mu_name	Reference mechanical unit name, defaults to the reference world coordinate system

4.5 RobotMode (Robot controller mode)

```
enum RobotMode {
    ROBOT_MODE_MANUAL, // Manual low speed mode
    ROBOT_MODE_AUTO, // Automatic mode
    ROBOT_MODE_MANUFAST // Manual high-speed mode
};
```

4.6 ProgramState (Program running status)

```
enum ProgramState{
    PROG_STATE_NOT_LOAD, // Program not loaded
    PROG_STATE_RUNNING,    // In operation
    PROG_STATE_PAUSE,    // Suspend
    PROG_STATE_STOP    // Cease
};
```

4.7 Jog mode

```
enum JogMode{
    JOG_SINGLE_AXIS, // Single axis mode
    JOG_CARTESIAN    // Cartesian mode
};
```

4.8 Jog direction

```
enum JogDir {
```

```
    DIR_POSITIVE, // Forward direction
    DIR_NEGATIVE // Negative direction
};
```

4.9 ProgramStepMode (Program operation mode)

```
enum ProgramStepMode {
    STEP, // Single step
    CONTINUE, // Continuous
    MSTEP // Section debugging
};
```

4.10 ProgramDirMode (Program forward and backward modes)

```
enum ProgramDirMode {
    FORWARD, // Forward mode
    BACKWARD // Backward mode
};
```

4.11 DataType (Axis data type)

```
enum DataType {
    CURRENT_COMMAND = 0, // Current instruction position
    CURRENT_POS, // Current real-time location
    CURRENT_SPEED, // Current speed
    CURRENT_TORQUE, // Current torque
    CURRENT_CURRENT // Current current
};
```

4.12 VarNameType (System Variable Type)

```
enum VarNameType {
    VAR_INT = 0,
    VAR_BOOL,
    VAR_DOUBLE,
```

```
    VAR_STRING,
    VAR_POSE, //$P,$PV1,$PV2,$PV3,$PV4,$PV5,$PV6,$PV7,$PV8,$PV9 use this type uniformly, with serial numbers ranging from 0 to 9999
    VAR_JOINT
};
```

4.13 max_connection_count (Maximum number of connections)

```
// The maximum number of connections allowed by the server, with a default value of 100
ROBOTAPI_API extern const unsigned int max_connection_count;
```

4.14 curr_connection_count (Number of existing connections)

```
// The number of existing connections, equal to the number of times the constructor is called - the number of times the destructor is called
ROBOTAPI_API extern unsigned int curr_connection_count;
```

5 Error code description

Table 5-1 Error code description

Error code	Enumerated value	Description
0	ERROR_OK	Successful operation
1	ERROR_CONNECT_FAILED	Failed to connect with robot
2	ERROR_NO_CONNECTION	Not yet connected to the robot
3	ERROR_CONNECTION_BROKEN	The connection with the robot is interrupted
4	ERROR_ACCESS_REJECTED	Access denied, the robot is set to not allow API access
5	ERROR_CUR_MODE_NOT_SUPPORT	The operation is not supported in the current mode
6	ERROR_SERVO_ON_FAILED	Power-on failure
7	ERROR_POWER_OFF_FAILED	Power-off failed
8	ERROR_SET_MODE_FAILED	Failed to set control mode
9	ERROR_SET_SPEED_RATIO_FAILED	Failed to set speed override, please check if the set override value is valid
10	ERROR_SWITCH_CHANNEL_FAILED	Failed to switch channel, please check whether the channel number set is valid
11	ERROR_START_PROGRAM_FAILED	Failed to start the program, the program can only be started when there is no warning, the program has been successfully loaded, and the system status is paused or stopped.
12	ERROR_RESET_PROGRAM_FAILED	The reset procedure failed, the procedure can be reset only when the system state is paused or stopped.
13	ERROR_LOAD_PROGRAM_FAILED	Failed to load the program, please obtain the alarm list to view the specific failure information.
14	ERROR_PARA_INVALID	Incorrect input parameter range.
15	ERROR_STILL_RUNNING	The system is still running and cannot execute motion instructions
16	ERROR_SET_PARA_FAILED	Failed to set variable
17	ERROR_GET_PARA_FAILED	Query variable failed
18	ERROR_MOVE_FAILED	Control robot movement failed
19	ERROR_ENABLE_API_CONTROL_FAILED	Failed to set external API control.
1000	ERROR_POINTER_INVALID	Invalid pointer parameter
10000	ERROR_THIRD_PARTY	The initial value of the exception code thrown by the third-party library.

PEITIAN ROBOTICS

WeChat Official Account

Official Website

Sevice Hotline : 400-990-0909
Official Website : http://robot.peitian.com

UM-S01500000003-006 / V2.0.6 / 2023.05.26