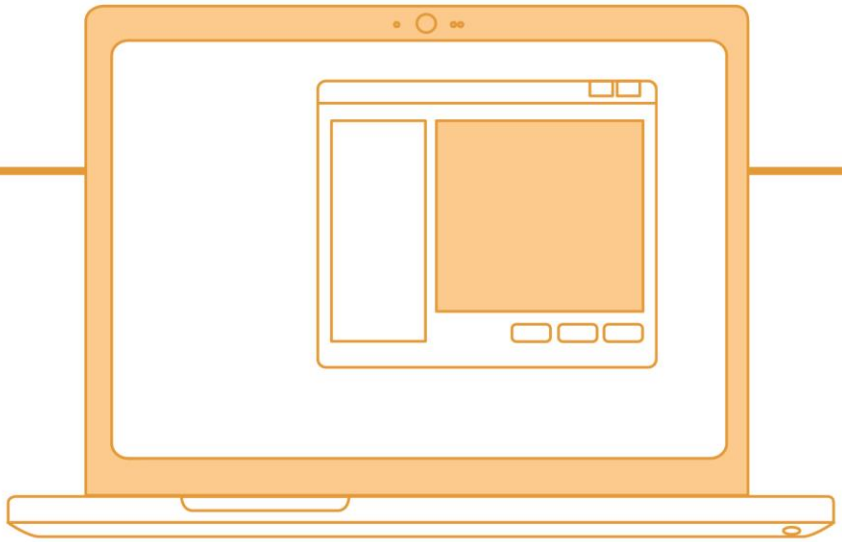
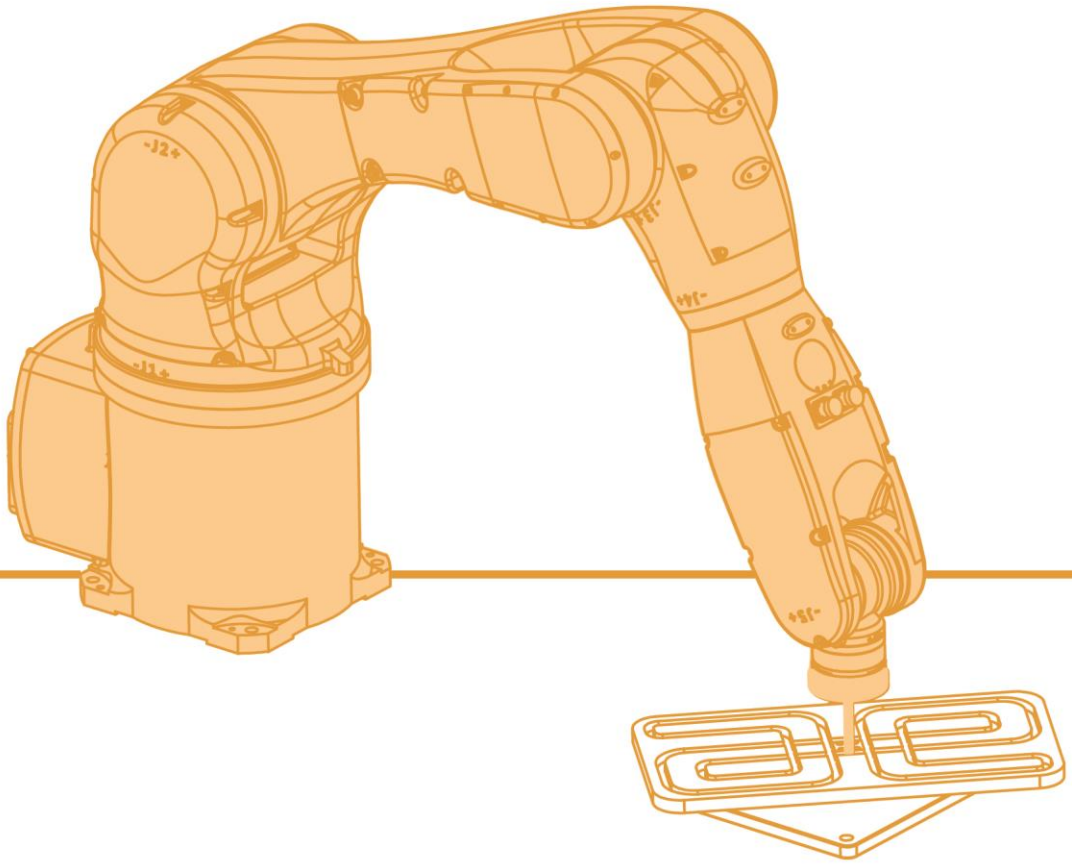
PEITIAN ROBOTICS

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 document update. The latest version of the document contains updated content from all previous document versions.

Table 3 Identification used in this manual

Version	Release time	Modification instructions
V2.0.3	2021/12/01	Added 'Related 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	2022/10/17	Encapsulating interfaces as class member functions, supporting simultaneous connection of multiple robots

Contents

1 Instructions

1.1 Development environment setup

1.1.1 Windows

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

The entire interface library includes 4 files, namely:

- robot_api_c++.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.

1.1.2 Linux

The programming interface library is written in C++, and the Linux version is compiled with gcc/g++-4.8.4.

The entire interface library includes 3 files, namely:

- robot_api_c++.h: Programming interface function definition header file
- libRobotAPI.so: Programming interface dynamic library
- libRobotComm.so: The robot communication dynamic library that the programming interface library depends on

Please use gcc/g++-4.8.4 or higher version compiler to compile.

1.2 Development configuration

1.2.1 Windows

Step1. After creating a C++ project, copy robot_api_c++.h and RobotAPI.lib to an arbitrary directory. It is recommended to copy to the project directory, in the project properties-configuration properties-C/C++-general "additional inclusion directory" Add the directory where robot_api_c++.h is located, as shown in Figure 1-1 below.

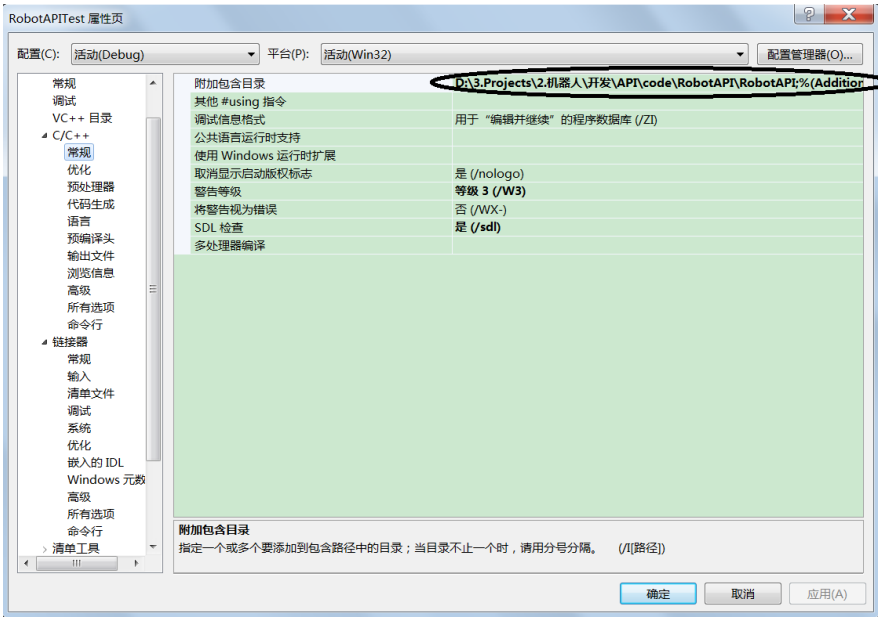


Figure 1-1Add the directory where robot_api_c++.h is located in "Additional Include Directory"

Step2. Add the directory where RobotAPI.lib is located in the project properties-configuration properties-linker-general "additional library directory", as shown in Figure 1-2 below.

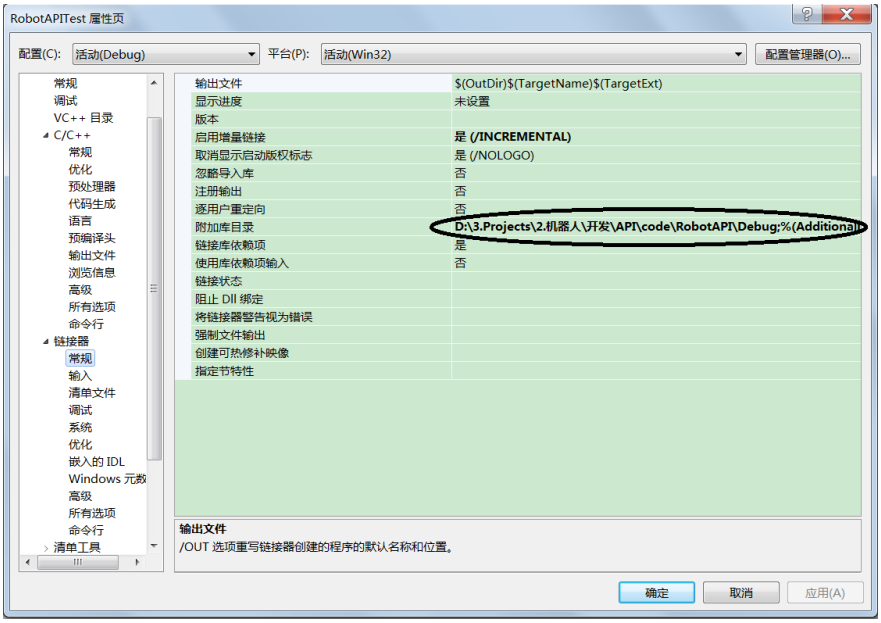


Figure 1-2 Add the directory where RobotAPI.lib is located in "Additional Library Directory"

Step3. Add the RobotAPI.lib file name in the "Additional Dependencies" of Project Properties-Configuration Properties-Linker-Input, as shown in Figure 1-3 below.

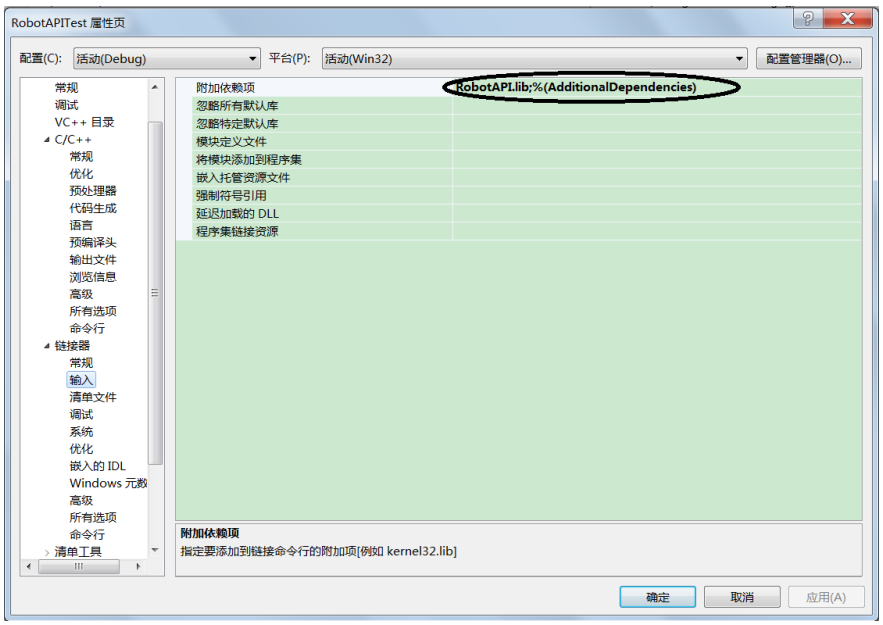


Figure 1-3 Add the directory where RobotAPI.lib is located in "Additional Dependencies"

Step4. In the project, the interface functions provided in robot_api_c++.h can be called normally, and they can be compiled normally.

1.2.2 Linux

Please include the robot_api_c++.h header file in the project, and set a dependency on the libRobotAPI.so dynamic library. Before running, please set the directory where libRobotAPI.so and libRobotComm.so are located as the searchable directory when the program is running.

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 description

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

The programming interface library declares its own namespace RobotAPI, which can be used during development through the using namespace RobotAPI. The programming interface library functions are all member functions of the interface class CRobotAPI, and are called through interface class pointers when used. 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.

3.1 Robot management

3.1.1 CRobotAPI (Interface class constructor)

Function declaration

CRobotAPI(unsigned int _index)

Function

Constructor to create a robot connection instance.

Parameter

index(in)	The robot connection index created starts from 0 and reaches a maximum of max_ connection_ count – 1.

Return

None

Usage

```
unsigned int index = 1;
CRobotAPI* pRobot = new CRobotAPI(index);
```

3.1.2 ~CRobotAPI (Interface class destructor)

Function declaration

~CRobotAPI()

Function

Destructor to delete robot connection instances.

Parameter

None

Return

None

Usage

```
unsigned int index = 1;
CRobotAPI* p = new CRobotAPI(index);
delete p;
p = NULL;
```

3.1.3 ConnectRobot (Initialize and connect the robot)

Function declaration

```
int ConnectRobot(char* robot_addr)
```

Function

Initialize and connect the robot.

Parameter

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;
char* server_addr = "10.20.1.1";
res = pRobot->ConnectRobot(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}
```


Caution

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

```
void DisconnectRobot()
```

Function

Disconnect the robot.

Return

No

Usage

```
pRobot->DisconnectRobot();
```

3.1.5 EnableApiControl (Enable or disable external API control)

Function declaration

```
int EnableApiControl(const bool enable)
```

Function

Enable or disable external API control.

Parameter

enable (in)	Whether to enable.

Return

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

Usage

```
int res = ERROR_OK;
// Enable external API control
res = pRobot->EnableApiControl(true);
if(ERROR_OK == res) {// Set successfully
} else {// Setup failed
}
```

3.1.6 SetControlMode (Set robot control mode)

Function declaration

```
int SetControlMode(const RobotMode mode)
```

Function

Set the robot control mode.

Parameter

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 = pRobot->SetControlMode(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

```
int SetProgStepMode(ProgramStepMode mode)
```

Function

Set the robot's single step continuous operation mode.

Parameter

mode (in)	Set the mode.

Return

0	Successfully set.
Non 0	Error code.

Usage

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

The control mode is defined as follows:

```
STEP,    // 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

int SetProgDirMode(ProgramDirMode mode)

Function

Set the forward and reverse operation mode of the robot.

Parameter

mode (in)	Set the mode.

Return

0	Successfully set.
Non 0	Error code.

Usage

```
int res = ERROR_OK;
// Set to forward running
res = pRobot->SetProgDirMode(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

int SwitchChannel(unsigned int channel_num)

Function

Switch channels.

Parameter

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 = pRobot->SwitchChannel(2);
if(ERROR_OK == res) { // Switch successfully
} else { // Switch failed
}
```

3.1.10 PowerOn (Robot powered on)

Function declaration

```
int PowerOn()
```

Function

The robot is powered on.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
res = pRobot->PowerOn();
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
}
```

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

```
int PowerOff()
```

Function

The robot is powered off.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
res = pRobot->PowerOff();
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

```
int ClearAlarm()
```

Function

Clear the robot alarm.

Return

0	Success.
Non-0	Error code.

Usage

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

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 (Robot returns to zero point)

Function declaration

```
int Move2Home(const unsigned int home_index)
```

Functionz

The robot returns to zero.

Parameter

home_index(in)	The home point number that the user wants the robot to move to

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
const unsigned int home_index = 3;
res = pRobot->Move2Home(home_index);
if(ERROR_OK == res) { // Return to zero command sent successfully
} else { // Failed to return to zero
}
```


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

```
int Move2Joint(const RobotJoint robot_joint, double speed_percent, double speed_exj = 0)
```

Function

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

Parameter

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
pRobot->SetControlMode(CONTROL_MODE_MANUAL);
// Set speed override to 50%
pRobot->SetSpeedRatio(50);
RobotJoint joint;
joint.j[0] = 10; // Control the first axis to move to 10 degrees, the other axes remain unchanged
res = pRobot->Move2Joint(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

int CheckPtpPosValid(const RobotPos* target_pos, const unsigned int target_pos_size, unsigned int tool_index, unsigned int wobj_index, bool* valid, const unsigned int valid_size)

Function

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

Parameter

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)	Whether it is reachable
valid_size(in)	Is 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, Whether it is reach the flag array bool * valid
RobotPos target_pos[3] = {
RobotPos(100,100,100,0,0,0),
RobotPos(200,300,400,0,0,0),
RobotPos(300,400,500,0,0,0)
};
bool valid[3];
res = pRobot->CheckPtpPosValid( target_pos, 3, 0, 0, valid, 3);
if(ERROR_OK == res) {    // The check was successful, and the values in valid indicate whether these points are reachable
} else {    // Check failed
```

}

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

Function declaration

int Move2Pos(const RobotPos robot_pos, unsigned int tool_index, unsigned int wobj_index, double speed_percent, double speed_exj = 0)

Function

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

Parameter

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, 4-3+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 the mode first before exercise
pRobot->SetControlMode(CONTROL_MODE_MANUAL);
// Set speed override to 50%
pRobot->SetSpeedRatio(50);
RobotPos pos(100,100,100,0,0,0);// Move to x: 100, y: 100, z: 100, a:0, b:0, c:0 pose
res = pRobot->Move2Pos(pos, 0,0,100);
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.
- 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

int Move2MultiPos(const RobotPos* robot_pos, unsigned int robot_pos_size, unsigned int tool_index, unsigned int wobj_index, double speed_percent, double speed_exj = 0)

Function

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

Parameter

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
pRobot->SetControlMode(CONTROL_MODE_MANUAL);
// Set the speed multiplier to 50%
pRobot->SetSpeedRatio(50);
RobotPos pos[3] = {
RobotPos(100,100,100,0,0,0),
RobotPos(200,300,400,0,0,0),
RobotPos(300,400,500,0,0,0)
};
res = pRobot->Move2MultiPos(pos, 3, 0,0,100);
if(ERROR_OK == res) {    // Successfully sent motion instruction
} else {    // Fail
}
```

- 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

int CheckLinePosValid(const RobotPos start_pos, const RobotPos* target_pos, unsigned int target_pos_size, unsigned int tool_index, unsigned int wobj_index, bool* valid, unsigned int valid_size)

Function

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

Parameter

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)	Whether it is reachable
valid_size	Is 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, whether it is reach the flag array bool * valid
RobotPos start_pos = RobotPos(0, 0, 0, 0, 0);
RobotPos target_pos[3] = {
RobotPos(100,100,100,0,0,0),
RobotPos(200,300,400,0,0,0),
RobotPos(300,400,500,0,0,0)
};
bool valid[3];
res = pRobot->CheckLinePosValid(start_pos, target_pos, 3, 0, 0, valid, 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

int Line2Pos(const RobotPos robot_pos, unsigned int tool_index, unsigned int wobj_index, double speed_tcp, double speed_ori, double speed_exj = 0)

Function

The lin controls the robot to move to a certain pose in a straight line.

Parameter

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, 4-3+WOBJ_NUM (Number of workpiece coordinate systems): Customize the workpiece coordinate system.
speed_tcp(in)	TCP point moving speed, unit: mm/s.
speed_exj(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 the mode first before exercise
pRobot->SetControlMode(CONTROL_MODE_MANUAL);
// Set speed override to 50%
pRobot->SetSpeedRatio(50);
RobotPos pos(100,100,100,0,0,0);// Move to x: 100, y: 100, z: 100, a:0, b:0, c:0 pose
res = pRobot->Line2Pos(pos, 0,0,50,50,100);// TCP and ORI speed 50
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.
- 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.

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

Function declaration

int Line2MultiPos(const RobotPos* robot_pos, unsigned int robot_pos_size, unsigned int tool_index, unsigned int 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

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
pRobot->SetControlMode(CONTROL_MODE_MANUAL);
// Set the speed multiplier to 50%
pRobot->SetSpeedRatio(50);
RobotPos pos[3] = {
RobotPos(100,100,100,0,0,0),
RobotPos(200,300,400,0,0,0),
RobotPos(300,400,500,0,0,0),
};
res = pRobot->Line2MultiPos(pos, 3, 0,0,50,50,100);// TCP and ORI speeds 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. 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

int Circle2Pos(const RobotPos robot_pos, const RobotPos m, double CA, unsigned int tool_index, unsigned int 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

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 the mode first before exercise
pRobot->SetControlMode(CONTROL_MODE_MANUAL);
// Set speed override to 50%
pRobot->SetSpeedRatio(50);
RobotPos pos(100,100,100,0,0,0);// Move to x: 100, y: 100, z: 100, a:0, b:0, c:0 pose
RobotPos m(200,100,100,0,0,0);// Auxiliary point pose
res = pRobot->Circle2Pos(pos, m, 180, 0, 0, 50, 50, 100);// Center angle 180 degrees, TCP and ORI speed 50
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.
- 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

int StopMove()

Function

Control the robot to stop moving.

Return

0	Success.
Non-0	Error code.

Usage

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

3.2.11 StartJog (Control the robot to start JOG motion)

Function declaration

int StartJog(JogMode jog_mode, JogDir jog_dir, unsigned int tool_index, unsigned int wobj_index, unsigned int axis_index, bool link = false)

Function

Control the robot to start JOG movement.

Parameter

jog_mode(in)	JOG mode, single-axis mode controls the movement of a single axis, and Cartesian mode controls the movement in the Cartesian coordinate system.
jog_dir(in)	JOG direction
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, 4-3+WOBJ_NUM (Number of workpiece coordinate systems): Customize the workpiece coordinate system.
axis_index(in)	In single-axis mode, it indicates the axis number. For example, for a 6-axis robot, 1-6 indicate the main body axis, and 7-12 indicate the external axis. In Cartesian mode, 1-6 respectively correspond to the X/Y/Z/A/B/C directions .

link(in)	Whether to link, the default is false.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
// Set the mode first before action
pRobot->SetControlMode(CONTROL_MODE_MANUAL);
// Set speed override to 50%
pRobot->SetSpeedRatio(50);
res = pRobot->StartJog(JOG_CARTESIAN, DIR_POSITIVE, 0, 0, 1); // Control the robot to move in the positive direction of the X axis
if(ERROR_OK == res) { // JOG motion command sent successfully
} else { // Failure
}
res = StopJog();
if(ERROR_OK == res) { // Stop JOG motion instruction sent successfully
} else { // Failure
}
res = pRobot->StartJog(JOG_SINGLE_AXIS, DIR_POSITIVE, 0, 0, 1); // Control the robot's 1-axis forward movement
if(ERROR_OK == res) { // JOG motion command sent successfully
} else { // Failure
}
res = pRobot->StopJog();
if(ERROR_OK == res) { // Stop JOG motion instruction 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.

3.2.12 StopJog (Control the robot to stop JOG motion)

Function declaration

```
int StopJog()
```

Function

Control the robot to stop JOG movement.

Return

0	Success.

Non-0	Error code.

Usage

```
unsigned int index = 1;
res = pRobot->StopJog();
if(ERROR_OK == res) { // Command sent successfully
} else { // Failure
}
```

3.3 IO

3.3.1 GetDigitalIn (Obtain the digital input value of a certain path)

Function declaration

```
int GetDigitalIn(unsigned int index, unsigned int& value)
```

Function

Get the digital input value of a certain channel.

Parameter

index(in)	DI serial number, starting from 1.
value(out)	DI value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
unsigned int value;
res = pRobot->GetDigitalIn(1, 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

```
int GetMultiDigitalIn(unsigned int start_index, unsigned int end_index, unsigned int& value)
```

Function

Obtain a set of digital input values

Parameter

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;
unsigned int value;
res = pRobot->GetMultiDigitalIn(1, 10,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

```
int GetDigitalOut(unsigned int index, unsigned int& value)
```

Function

Get the digital output value of a certain way.

Parameter

index(in)	DO serial number, starting from 1.
value(out)	DO value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
unsigned int value;
res = pRobot->GetDigitalOut(1, 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

int GetMultiDigitalOut(unsigned int start_index, unsigned int end_index, unsigned int& value)

Function

Obtain a set of digital output values.

Parameter

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;
unsigned int value;
res = pRobot->GetMultiDigitalOut(1, 10,value);
if(ERROR_OK == res) {    // Successfully obtained DO value
} else {    // DO value obtained failed
}
```

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

Function declaration

int SetDigitalOut(unsigned int index, unsigned int value)

Function

Set the digital output value of a certain channel.

Parameter

index(in)	DO serial number, starting from 1.
value(out)	DO value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
unsigned int value = 1;
```

```
res = SetDigitalOut(1, value);// Set the first DO to high level
if(ERROR_OK == res) { // The DO value is set successfully
} else { // Failed to set DO value
}
```

3.3.6 SetMultiDigitalOut (Set a set of digital output values)

Function declaration

```
int SetMultiDigitalOut(unsigned int start_index, unsigned int end_inde, unsigned int value)
```

Function

Set a set of digital output values.

Parameter

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;
unsigned int value = 0xff;
res = pRobot->SetMultiDigitalOut(1, 10,value);// Set 1-10 DO channels to high level
if(ERROR_OK == res) {    // DO value set successfully
} else {    // DO value setting failed
}
```

3.4 Configuration

3.4.1 SetSpeedRatio (Set speed multiplier)

Function declaration

```
int SetSpeedRatio(unsigned int ratio)
```

Function

Set speed multiplier.

Parameter

ratio(in)	The magnification value, the range is 1-100.

Return

0	Success.
Non-0	Error code.

Usage

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

3.4.2 SetToolCoordinateByIndex (Set tool coordinate system values through index)

Function declaration

```
int SetToolCoordinateByIndex (unsigned int tool_index, RobotTool tool)
```

Function

Set tool coordinate system values through index

Parameter

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(10,20,30,0,0,0,false, "WORLD");
res = pRobot->SetToolCoordinate ByIndex (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

```
int SetToolCoordinateByName (char* tool_name, RobotTool tool)
```

Function

Set tool coordinate system values by name.

Parameter

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(10,20,30,0,0,0,false,"WORLD");
res = pRobot->SetToolCoordinateByName( "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

```
int SetWorkpieceCoordinateByIndex (unsigned int workpiece_index, RobotWorkpiece workpiece)
```

Function

Set the workpiece coordinate system value through index.

Parameter

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,false,"WORLD");
res = pRobot->SetWorkpieceCoordinateByIndex (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

int SetWorkpieceCoordinateByName (char* workpiece_name, RobotWorkpiece workpiece)

Function

Set the workpiece coordinate system value by name.

Parameter

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(10,20,30,0,0,0,false,"WORLD");
res = pRobot->SetWorkpieceCoordinateByName ( "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

int SetIntVariableByIndex (unsigned int index, int value)

Function

Set integer variable values through index.

Parameter

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 = pRobot->SetIntVariableByIndex (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

```
int SetIntVariableByName (char* name, int value)
```

Function

Set integer variable value by name.

Parameter

name(in)	Variable name.
value(in)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
int value = 100;
res = pRobot->SetIntVariableByName ( "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

```
int SetDoubleVariableByIndex (unsigned int index, double value)
```

Function

Set floating point variable values through index.

Parameter

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 = pRobot->SetDoubleVariableByIndex (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

```
int SetDoubleVariableByName (char* name, double value)
```

Function

Set the value of a floating-point variable by name.

Parameter

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 = pRobot->SetDoubleVariableByName ( "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

```
int SetBoolVariableByIndex (unsigned int index, bool value)
```

Function

Set Boolean variable value through index.

Parameter

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 = pRobot->SetBoolVariableByIndex (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

```
int SetBoolVariableByName (char* name, bool value)
```

Function

Set Boolean variable value by name.

Parameter

name(in)	Variable name.
value(in)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
int value;
res = pRobot->SetBoolVariableByName ( "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

```
int SetStringVariableByIndex (unsigned int index, char* value)
```

Function

Set string variable values through index.

Parameter

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

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
char value[200];
res = pRobot->SetStringVariableByIndex (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

```
int SetStringVariableByName (char* name, char* value)
```

Function

Set string variable values by name

Parameter

name(in)	Variable name.
value(in)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
char value[200];
res = pRobot->SetStringVariableByName ( "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

int SetPoseVariableByIndex (unsigned int index, RobotPos value , int tool_index = -1, int wobj_index = -1)

Function

Set pose variable values through index

Parameter

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;
int tool_index = 0;
int wobj_index = 0;
res = pRobot->SetStringVariableByIndex (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

int SetPoseVariableByName (char* name, RobotPos value , int tool_index = -1, int wobj_index = -1)

Function

Set pose variable values by name.

Parameter

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;
int tool_index = 0;
int wobj_index = 0;
res = pRobot->SetStringVariableByName ( "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

```
int SetJointVariableByIndex (unsigned int index, RobotJoint value)
```

Function

Set joint type variable values through index.

Parameter

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;
res = pRobot->SetJointVariableByName (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

int SetJointVariableByName (char* name, RobotJoint value)

Function

Set joint type variable values by name.

Parameter

name(in)	Variable name.
value(in)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotJoint value;
res = pRobot->SetJointVariableByIndex ( "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

int SetVariableName(VarNameType type, unsinged int index, char* name)

Function

Set system variable name.

Parameter

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 = pRobot->SetVariableName(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

```
int SetCurCoordinate(unsigned int tool_index, unsigned int wobj_index)
```

Function

Set JOG tool and workpiece coordinate system.

Parameter

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 = pRobot->SetCurCoordinate (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 GetControlMode (Query the current robot control mode)

Function declaration

```
int GetControlMode(RobotMode& mode)
```

Function

Query the current robot control mode.

Parameter

mode(out)	The current mode.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotMode mode;
res = pRobot->GetControlMode (mode);
if(ERROR_OK == res) { // Status query successful
} else { // Query failed
}
```

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

Function declaration

```
int GetProgramState(ProgramState& state)
```

Function

Query the current running status of the robot.

Parameter

state(out)	Current operating status.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
ProgramState state;
res = pRobot->GetProgramState (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
```

3.5.3 GetSpeedRatio (Query the current speed override)

Function declaration

int GetSpeedRatio(unsigned int& ratio)

Function

Query the current speed override.

Parameter

ratio(out)	The current speed override.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
unsigned int ratio;
res = pRobot->GetSpeedRatio (ratio);// Query JOG magnification value
if(ERROR_OK == res) { // Status query successful
} else { // Query failed
}
```

3.5.4 GetLoadedProg (Query the currently loaded path)

Function declaration

int GetLoadedProg(char* file_path)

Function

Query the currently loaded path.

Parameter

file_path(out)	The current loading file path.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
char file_path[200];
res = pRobot->GetLoadedProg (file_path);// Query loaded file path
if(ERROR_OK == res) {    // Status query successful
```

```
} else {    // Query failed
}
```

3.5.5 IsPowerOn (Query whether it is currently powered on)

Function declaration

```
int IsPowerOn(bool& is_poweron)
```

Function

Query whether it is currently powered on.

Parameter

is_poweron(out)	Whether it has been powered on.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
bool is_poweron = false;
res = pRobot-> IsPowerOn (is_poweron);
if(ERROR_OK == res) { // Status query successful
} else { // Query failed
}
```

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

Function declaration

```
bool IsConnected()
```

Function

Check if it is currently connected to the robot.

Return

true	Connected
false	Not connected

Usage

```
bool is_connected = pRobot->IsConnected();
```

3.5.7 GetPos (Query the current robot pose)

Function declaration

int GetPos(RobotPos& robot_pos)

Function

Query the current robot pose.

Parameter

robot_pos(out)	Current pose.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotPos pos;
res = pRobot->GetPos (pos);
if(ERROR_OK == res) { // Query successful
} else { // Query failed
}
```

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

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

Function declaration

int GetJoint(RobotJoint& robot_joint)

Function

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

Parameter

robot_joint(out)	The current angle of each axis.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotJoint joint;
```

```
res = pRobot->GetJoint (joint);
if(ERROR_OK == res) { // Query successful
} else { // Query failed
}
```

3.5.9 GetAlarmState (Query the current alarm status)

Function declaration

```
int GetAlarmState(bool& state)
```

Function

Query the current alarm status.

Parameter

state(out)	Whether it is currently alarming.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
bool state = false;
res = pRobot->GetAlarmState (state);
if(ERROR_OK == res) { // Query successful
} else { // Query failed
}
```

3.5.10 GetAlarmList (Query the current alarm list)

Function declaration

```
int GetAlarmList(unsigned int* list, const unsigned int list_size)
```

Function

Query the current alarm list.

Parameter

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;
const unsigned int list_size = 500;
unsigned int list[list_size];
res = pRobot->GetAlarmList (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.

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

Function declaration

```
int GetAxisAlarm(int* alarm, const unsigned int alarm_size)
```

Function

The list of driver alarm codes for abnormal axes is currently limited to axes 1-6 of the body.

Parameter

alarm(out)	Data alarm (out): A list of abnormal axis alarm codes, with axis 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	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
const unsigned int alarm_size = 10;
int alarm[alarm_size] = {0};
res = pRobot->GetAxisAlarm (alarm, alarm_size);
if(ERROR_OK == res) {    // Query successful
} else {    // Query failed
}
```

3.5.12 GetAxisData (Obtain current data for each axis)

Function declaration

```
int GetAxisData(DataType data_type, unsigned int axis_index, double& data)
```

Function

Obtain current data for each axis

Parameter

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 = pRobot->GetAxisData (CURRENT_SPEED, 1, data); // Obtain the current speed of the first axis
if(ERROR_OK == res) {    // Query 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
};
```

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

Function declaration

```
int GetRobotRunTime(double& time)
```

Function

Obtain the accumulated running time of the current robot.

Parameter

time(out)	Accumulated running time, in seconds.

Return

0	Success.

Non-0	Error code.

Usage

```
int res = ERROR_OK;
double time;
res = pRobot->GetRobotRunTime (time);
if(ERROR_OK == res) {    // Query successful
} else {    // Query failed
}
```

3.5.14 GetToolCoordinateByIndex (Retrieve tool coordinate system values through index)

Function declaration

```
int GetToolCoordinateByIndex (unsigned int tool_index, RobotTool& tool)
```

Function

Retrieve tool coordinate system values through index.

Parameter

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;
res = pRobot->GetToolCoordinateByIndex (0, tool);// Obtain the first tool coordinate system
if(ERROR_OK == res) {    // Successfully obtained coordinate system
} else {    // Obtained failed
}
```

3.5.15 GetToolCoordinateByName (Obtain tool coordinate system values by name)

Function declaration

```
int GetToolCoordinateByName (char* tool_name, RobotTool& tool)
```

Function

Obtain tool coordinate system values by name.

Parameter

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;
res = pRobot->GetToolCoordinateByName( "t1" , tool);// Obtain the t1 tool coordinate system
if(ERROR_OK == res) { // Successfully obtained coordinate system
} else { // Obtained Failed
}
```

3.5.16 GetWorkpieceCoordinateByIndex (Retrieve workpiece coordinate system values through index)

Function declaration

```
int GetWorkpieceCoordinateByIndex (unsigned int workpiece_index, RobotWorkpiece& workpiece)
```

Function

Retrieve workpiece coordinate system values through index.

Parameter

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;
res = pRobot->GetWorkpieceCoordinateByIndex (0, workpiece);// Obtain the first workpiece coordinate system
if(ERROR_OK == res) {    // Successfully obtained coordinate system
} else {    // Obtained Failed
}
```

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

Function declaration

int GetWorkpieceCoordinateByName (char* workpiece_name, RobotWorkpiece& workpiece)

Function

Obtain the workpiece coordinate system value by name.

Parameter

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;
res = pRobot->GetWorkpieceCoordinateByName ( "w1" , workpiece);// Obtain the w1 workpiece coordinate system
if(ERROR_OK == res) {    // Successfully obtained coordinate system
} else {    // Obtained failed
}
```

3.5.18 GetIntVariableByIndex (Query integer variable values through index)

Function declaration

int GetIntVariableByIndex (unsigned int index, int& value)

Function

Query integer variable values through index

Parameter

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;
res = pRobot->GetIntVariableByIndex (0, value);    // Query the first integer value
```

```
if(ERROR_OK == res) {    // Query successfully
} else {    // Query failed
}
```

3.5.19 GetIntVariableByName (Query integer variable values by name)

Function declaration

```
int GetIntVariableByName (char* name, int& value)
```

Function

Query integer variable values by name.

Parameter

name(in)	Variable name.
value(out)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
int value;
res = pRobot->GetIntVariableByName ( "i1" , value);    // Query i1 integer value
if(ERROR_OK == res) {    // Query successfully
} else {    // Query failed
}
```

3.5.20 GetDoubleVariableByIndex (Query floating point variable values through index)

Function declaration

```
int GetDoubleVariableByIndex (unsigned int index, double& value)
```

Function

Query floating point variable values through index.

Parameter

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

3.5.21 GetDoubleVariableByName (Query floating point variable values by name)

Function declaration

```
int GetDoubleVariableByName (char* name, double& value)
```

Function

Query floating point variable values by name.

Parameter

name(in)	Variable name.
value(out)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

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

3.5.22 GetBoolVariableByIndex (Query Boolean variable values through index)

Function declaration

```
int GetBoolVariableByIndex (unsigned int index, bool& value)
```

Function

Query Boolean variable values through index.

Parameter

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;
res = pRobot->GetBoolVariableByIndex (0, value);    // Query the first Boolean value
if(ERROR_OK == res) {    // Query successfully
} else {    // Query failed
}
```

3.5.23 GetBoolVariableByName (Query Boolean variable values by name)

Function declaration

```
int GetBoolVariableByName (char* name, bool& value)
```

Function

Query Boolean variable values by name.

Parameter

name(in)	Variable name.
value(out)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
int value;
res = pRobot->GetBoolVariableByName ( "b1", value);    // Query b1 Boolean Value
if(ERROR_OK == res) {    // Query successfully
} else {    // Query failed
}
```

3.5.24 GetStringVariableByIndex (Retrieve string variable values through index)

Function declaration

```
int GetStringVariableByIndex (unsigned int index, char* value)
```

Function

Retrieve string variable values through index.

Parameter

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

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
char value[200];
res = pRobot->GetStringVariableByIndex (0, value);    // Get the first string value
if(ERROR_OK == res) {    // Query successfully
} else {    // Query failed
}
```

3.5.25 GetStringVariableByName (Obtain string variable values by name)

Function declaration

```
int GetStringVariableByName (char* name, char* value)
```

Function

Obtain string variable values by name.

Parameter

name(in)	Variable name.
value(out)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
char value[200];
res = pRobot->GetStringVariableByName ( "s1", value);    // Obtain the s1 string value
if(ERROR_OK == res) {    // Query successfully
} else {    // Query failed
}
```

3.5.26 GetPoseVariableByIndex (Retrieve pose variable values through index)

Function declaration

int GetPoseVariableByIndex (unsigned int index, RobotPos& value , int& tool_index, int& wobj_index)

Function

Retrieve pose variable values through index.

Parameter

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;
int tool_index, wobj_index;
res = pRobot->GetStringVariableByIndex (0, value, tool_index, wobj_index);    // Obtain the first pose value
if(ERROR_OK == res) {    // Query successfully
} else {    // Query failed
}
```

3.5.27 GetPoseVariableByName (Obtain pose variable values by name)

Function declaration

int GetPoseVariableByName (char* name, RobotPos& value , int& tool_index, int& wobj_index)

Function

Obtain pose variable values by name.

Parameter

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;
int tool_index, wobj_index;
res = pRobot->GetStringVariableByName ( "p1", value, tool_index, wobj_index);    // Obtain p1 pose value
if(ERROR_OK == res) {    // Query successfully
} else {    // Query failed
}
```

3.5.28 GetJointVariableByIndex(Set joint type variable values through index)

Function declaration

```
int GetJointVariableByIndex (unsigned int index, RobotJoint& value)
```

Function

Set joint type variable values through index.

Parameter

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;
res = pRobot->GetJointVariableByIndex(0, value);    // Obtain the first joint type value
if(ERROR_OK == res) {    // Query successfully
} else {    // Query failed
}
```

3.5.29 GetJointVariableByName (Obtain joint type variable values by name)

Function declaration

```
int GetJointVariableByName(char* name, RobotJoint& value)
```

Function

Obtain joint type variable values by name.

Parameter

name(in)	Variable name.
value(out)	Variable value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotJoint value;
res = pRobot->GetJointVariableByName( "j1", value);    // Obtain j1 joint type value
if(ERROR_OK == res) {    // Query successfully
} else {    // Query failed
}
```

3.5.30 GetCurCoordinate (Obtain JOG tool and workpiece coordinate system)

Function declaration

```
int GetCurCoordinate(unsigned int& tool_index, unsigned int& wobj_index)
```

Function

Obtain JOG tool and workpiece coordinate system.

Parameter

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;
int tool_index, wobj_index;
res = pRobot->GetCurCoordinate (tool_index, wobj_index);    // Obtain the JOG tool coordinate system and workpiece coordinate system
if(ERROR_OK == res) {    // Query successfully
```

```
} else {    // Query failed
}
```

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

Function declaration

```
int FkSolver(RobotJoint joint, unsigned int tool_index, unsigned int wobj_index, RobotPos& pos)
```

Function

Robot pose and joint angle forward solution interface.

Parameter

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;
RobotPos pos;
res = pRobot->FkSolver (joint, tool_index, wobj_index, pos);
if(ERROR_OK == res) {    // Query successfully
} else {    // Query failed
}
```

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

Function declaration

```
int IkSolver(RobotPos pos, unsigned int tool_index, unsigned int wobj_index, RobotJoint& joint)
```

Function

Interface for inverse solution of robot pose and joint angle.

Parameter

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;
RobotPos pos;
res = pRobot->IkSolver (pos, tool_index, wobj_index, joint);
if(ERROR_OK == res) {    // Query successfully
} else {    // Query failed
}
```

3.6 Program running

3.6.1 SendProgram (Send ARL program)

Function declaration

```
int SendProgram(char* file_path)
```

Function

Send the ARL program.

Parameter

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;
char* file_path = "d:\\test_program";
res = pRobot->SendProgram (file_path);    // Send d:/test_ Program entire folder
if(ERROR_OK == res) {    // Sent successfully
} else {    // Send failed
}
```

```
file_path = "d:\\prog\\test.arl";
res = pRobot->SendProgram (file_path);    // Send d:/prog/test.arl file
if(ERROR_OK == res) {    // Sent successfully
} else {    // Send failed
}
```


Caution

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

3.6.2 LoadProgram (Load ARL program)

Function declaration

```
int LoadProgram(char* file_path)
```

Function

Load the ARL program.

Parameter

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;
char* file_path = "d:\\test_program";
res = pRobot->SendProgram (file_path);    // Send d:/test_ Program entire folder
if(ERROR_OK == res) {    // Execution succeeded
} else {    // Execution failed
}
file_path = "test_program\\test.arl";
res = pRobot->LoadProgram (file_path);    // Load the previously sent test.arl program in the tests_ program directory
if(ERROR_OK == res) {    // Execution succeeded
} else {    // Execution failed
}
```

3.6.3 StartProgram (Start ARL program)

Function declaration

```
int StartProgram()
```

Function

Start the ARL program.

Return

0	Success.
Non-0	Error code.

Usage

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

3.6.4 PauseProgram (Pause program)

Function declaration

```
int PauseProgram()
```

Function

Pause the program.

Return

0	Success.
Non-0	Error code.

Usage

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

3.6.5 ResetProgram (Reset program)

Function declaration

int ResetProgram()

Function

Reset the program.

Return

0	Success.
Non-0	Error code.

Usage

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

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

3.7 Control multiple robots

```
#include <iostream>
#include robot_api_c++.h
// Add additional header files as needed

int _tmain(int argc, _TCHAR* argv[])
{
        std::cout << "Program start" << std::endl;
        const unsigned int connect_count = 10;
        CRobotAPI* p[connect_count];
        for (unsigned int i = 0; i < connect_count; i++) {
                p[i] = new CRobotAPI(i);
        }
        int res1 = p[1]->ConnectRobot("10.20.210.254");
        if (res1 != ERROR_OK) {
                std::cout << "ConnectRobot fail!" << std::endl;
        }
        int res2 = p[2]->ConnectRobot("192.168.0.102");
        int res3 = p[3]->ConnectRobot("192.168.0.103");
        const bool enable = true;

        // Enable or disable external API control
```

```
        int res = p[1]->EnableApiControl(enable);
        if(res) {
            std::cout << "EanbleApiControl fail!" << std::endl;
        }
        // Destroy unused robot connections
        delete p[3];
return 0;
}
```

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
struct RobotPos {
double x;
double y;
double z;
double a;
double b;
double c;
int cfg;
int turn;
double ej1;
double ej2;
double ej3;
double ej4;
double ej5;
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
struct RobotJoint {
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
struct RobotTool {
double x;
double y;
```

```
double z;

double a;

double b;

double c;

bool stationary;

char* mu_name;

};
```

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

Table 4-3 Description 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

struct RobotWorkpiece {

double x;

double y;

double z;

double a;

double b;

double c;

bool robhold;

char* mu_name;

};
```

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

Table 4-4 Description 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, // Pause
    PROG_STATE_STOP // Stop
};
```

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 reverse modes)

```
enum ProgramDirMode {
    FORWARD, // Forward
    BACKWARD // Reverse
};
```

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