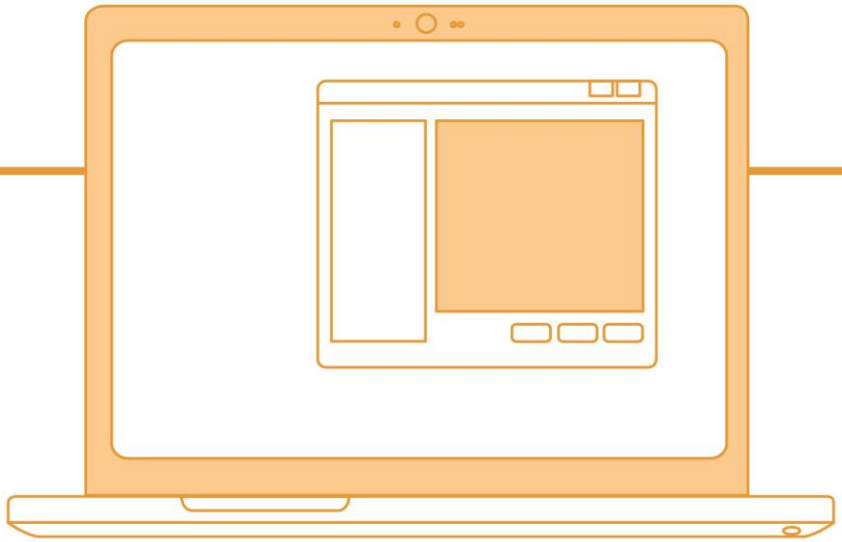
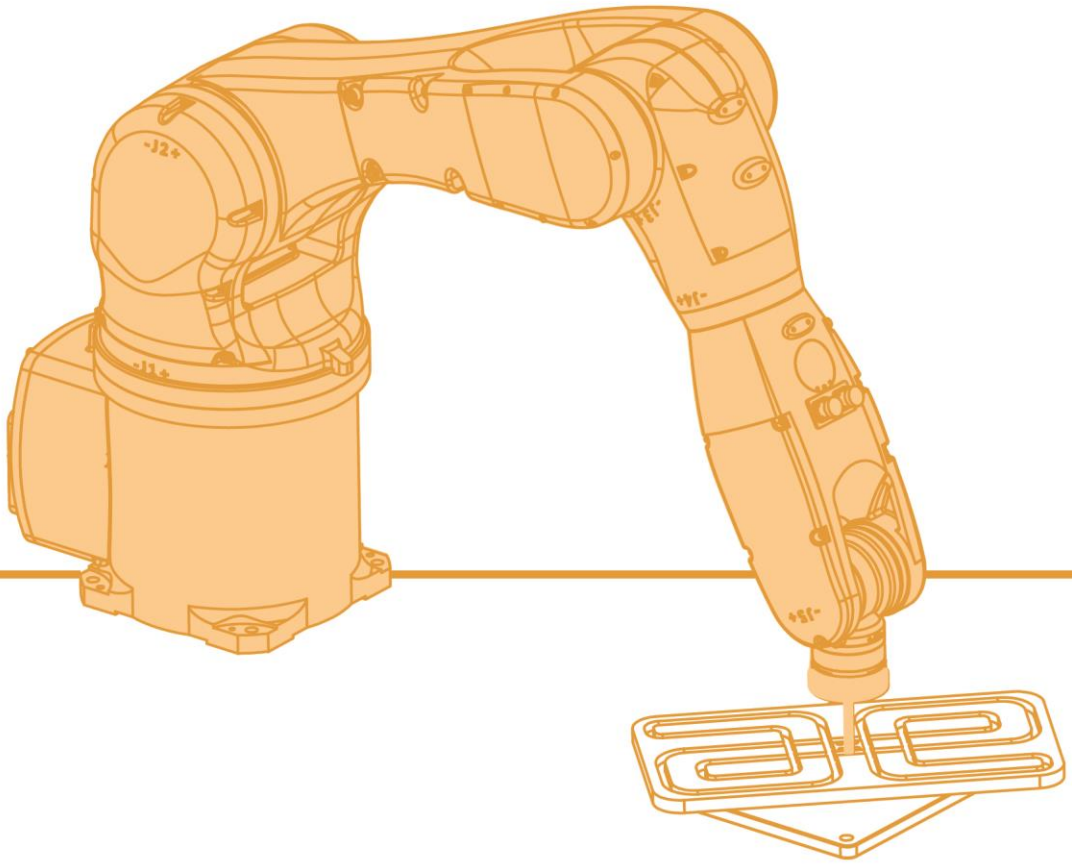
PEITIAN ROBOTICS

Programming Interface Library Instruction Manual

V2.0.0





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.	V2.0.0
Software ver.	2.6.3

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/

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++ and C# versions respectively, and includes both 32-bit and 64-bit versions.

The entire interface library includes 4 files, namely:

- robot_api.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.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

1.2.3 Windows

Step1. After creating a C++ project, copy robot_api.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.h is located, as shown in Figure 1-1 below.

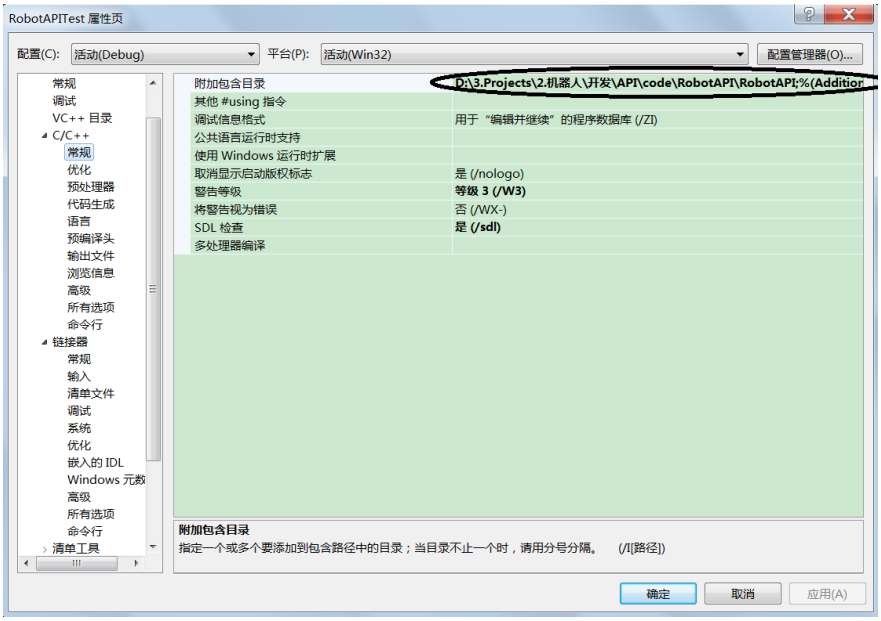


Figure 1-1Add the directory where robot_api.h is located in "Additional Include Directory"

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.

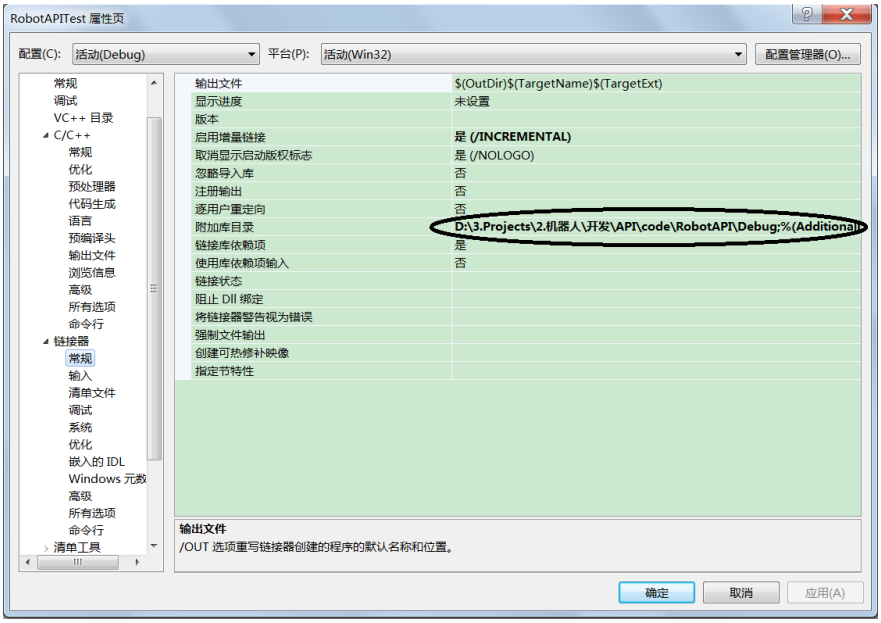


Figure 1-2 Add the directory where RobotAPI.lib is located in "Additional Library Directory"

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.

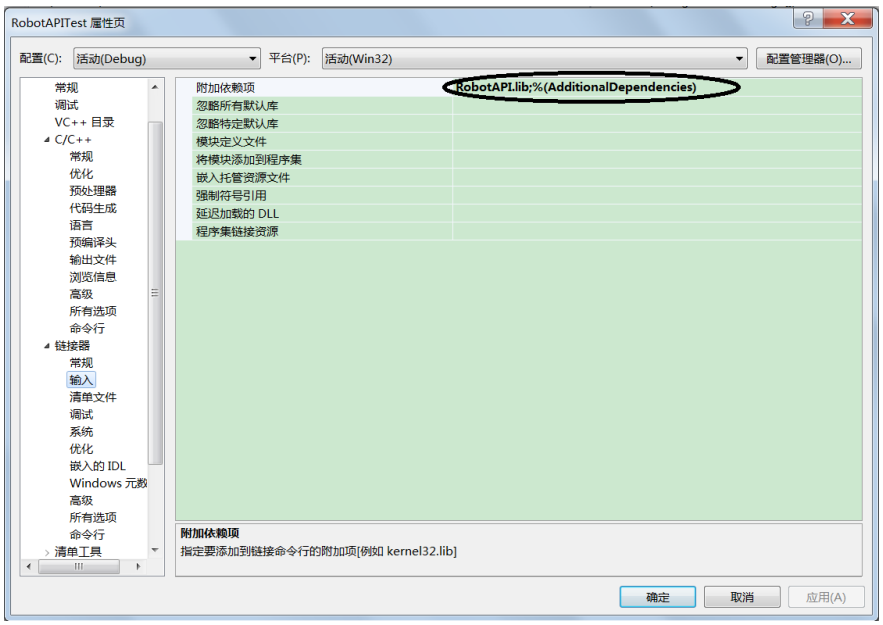


Figure 1-3 Add the directory where RobotAPI.lib is located in "Additional Dependencies"

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

1.2.4 Linux

Please include the robot_api.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 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.

2.1 Robot management

2.1.5 ConnectRobot

Function declaration

ROBOTAPI_API int ConnectRobot(const std::string 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;
std::string server_addr = “10.20.1.1”;
res = 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}
```

This interface function is the first step of all operations. If this interface is not called, other interfaces will not work properly.

2.1.6 DisconnectRobot

Function declaration

ROBOTAPI_API void DisconnectRobot()

Function

Disconnect the robot.

Return

No

Usage

DisconnectRobot();

2.1.7 EnableApiControl

Function declaration

ROBOTAPI_API 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 = EnableApiControl(true);

if(ERROR_OK == res) {   // Set successfully

} else {       // Setup failed

}
```

2.1.8 SetControlMode

Function declaration

ROBOTAPI_API 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 = 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
```

2.1.9 SwitchChannel

Function declaration

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

2.1.10 PowerOn

Function declaration

```
ROBOTAPI_API int PowerOn()
```

Function

The robot is powered on.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
res = 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.

2.1.11 PowerOff

Function declaration

ROBOTAPI_API int PowerOff()

Function

The robot is powered off.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
res = 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
}
```

2.1.12 ClearAlarm

Function declaration

ROBOTAPI_API int ClearAlarm()

Function

Clear the robot alarm.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
res = 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.

2.2 Movement

2.2.13 Move2Home

Function declaration

ROBOTAPI_API int Move2Home()

Functionz

The robot returns to zero.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;

res = Move2Home();

if(ERROR_OK == res) {   // Return to zero command sent successfully

} else {        // Failed to return to zero

}
```

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.

2.2.14 Move2Joint

Function declaration

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


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.

2.2.15 Move2Pos (Single Position)

Function declaration

ROBOTAPI_API 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 number, 0: world coordinate system, 1-3: channel base coordinate system, 4-35: custom 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

SetControlMode(CONTROL_MODE_MANUAL);

// Set speed override to 50%

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 = 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.

2.2.16 Move2Pos (Multi Position)

Function declaration

```
ROBOTAPI_API int Move2Pos(const vector<RobotPos> robot_pos, 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.
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, 4-35: custom 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

SetControlMode(CONTROL_MODE_MANUAL);

// Set speed override to 50%

SetSpeedRatio(50);

vector <RobotPos> pos;

pos.push_back(RobotPos(100,100,100,0,0,0));

pos.push_back(RobotPos(200,300,400,0,0,0));

pos.push_back(RobotPos(300,400,500,0,0,0));

res = 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. 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.

2.2.17 Line2Pos (Single Position)

Function declaration

ROBOTAPI_API 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, 4-35: custom 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

SetControlMode(CONTROL_MODE_MANUAL);

// Set speed override to 50%

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 = 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.

2.2.18 Line2Pos (Multi Position)

Function declaration

```
ROBOTAPI_API int Line2Pos(const vector<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 several positions in a straight line.

Parameter

robot_pos(in)	The target pose array.
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, 4-35: custom 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

SetControlMode(CONTROL_MODE_MANUAL);

// Set speed override to 50%

SetSpeedRatio(50);

vector <RobotPos> pos;

pos.push_back(RobotPos(100,100,100,0,0,0));

pos.push_back(RobotPos(200,300,400,0,0,0));

pos.push_back(RobotPos(300,400,500,0,0,0));

res = 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. 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.

2.2.19 Circle2Pos

Function declaration

ROBOTAPI_API 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, 4-35: custom 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

SetControlMode(CONTROL_MODE_MANUAL);

// Set speed override to 50%

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 = 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.

2.2.20 StopMove

Function declaration

```
ROBOTAPI_API int StopMove()
```

Function

Control the robot to stop moving.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;

res = StopMove();

if(ERROR_OK == res) {   // Command sent successfully

} else {       // Failure

}
```

2.2.21 StartJog

Function declaration

```
ROBOTAPI_API 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, 4-35: custom 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

SetControlMode(CONTROL_MODE_MANUAL);

// Set speed override to 50%

SetSpeedRatio(50);

res = 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 = 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 = 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.

2.2.22 StopJog

Function declaration

```
ROBOTAPI_API int StopJog()
```

Function

Control the robot to stop JOG movement.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;

res = StopJog();

if(ERROR_OK == res) {   // Command sent successfully

} else {       // Failure

}
```

2.3 IO

2.3.23 GetDigitalIn

Function declaration

```
ROBOTAPI_API 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 = GetDigitalIn(1, value);
if(ERROR_OK == res) {   // DI value obtained successfully
} else {       // DI value obtained failed
}
```

2.3.24 GetDigitalOut

Function declaration

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

2.3.25 SetDigitalOut

Function declaration

ROBOTAPI_API 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
}
```

2.4 Configuration

2.4.26 SetSpeedRatio

Function declaration

ROBOTAPI_API int SetSpeedRatio(unsigned int ratio)

Function

Set the speed override.

Parameter

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

Return

0	Success.
Non-0	Error code.

Usage

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

2.4.27 SetToolCoordinate

Function declaration

```
ROBOTAPI_API int SetToolCoordinate(unsigned int tool_index, RobotTool tool)
```

Function

Set the tool coordinate system value.

Parameter

tool_index(in)	The serial number of the tool coordinate system, the range is 0-31.
tool(in)	The set tool coordinate system value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotTool tool(10,20,30,0,0,0,false);
res = SetToolCoordinate (0, tool);// Set the first tool coordinate system
if(ERROR_OK == res) {   // The coordinate system is set successfully
} else {       // Setup failed
}
```

2.4.28 SetWorkpieceCoordinate

Function declaration

```
ROBOTAPI_API int SetWorkpieceCoordinate(unsigned int workpiece_index, RobotWorkpiece workpiece)
```

Function

Set the value of the workpiece coordinate system.

Parameter

workpiece _index(in)	The serial number of the workpiece coordinate system, the range is 0-31.
workpiece (in)	The set workpiece coordinate system value.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
RobotWorkpiece workpiece(10,20,30,0,0,0,false);
res = SetWorkpieceCoordinate (0, workpiece);// Set the first workpiece coordinate system
if(ERROR_OK == res) {   // The coordinate system is set successfully
} else {       // Setup failed
}
```

2.4.29 SetIntVariable

Function declaration

```
ROBOTAPI_API int SetIntVariable(unsigned int index, int value)
```

Function

Set the value of an integer variable.

Parameter

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

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
int value = 100;
res = SetIntVariable(0, value);   // Set the first integer value
if(ERROR_OK == res) {   // Set successfully
```

```
} else {       // Setup failed
}
```

2.4.30 SetDoubleVariable

Function declaration

```
ROBOTAPI_API int SetDoubleVariable(unsigned int index, double value)
```

Function

Set the value of a floating-point variable.

Parameter

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

Return

0	Success.
Non-0	Error code.

Usage

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

2.4.31 SetBoolVariable

Function declaration

```
ROBOTAPI_API int SetBoolVariable(unsigned int index, bool value)
```

Function

Set the value of a Boolean variable.

Parameter

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

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
int value;
res = SetBoolVariable(0, value);          // Set the first boolean value
if(ERROR_OK == res) {    // Set successfully
} else {        // Setup failed
}
```

2.5 Inquire

2.5.32 GetControlMode

Function declaration

```
ROBOTAPI_API 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 = GetControlMode (mode);
if(ERROR_OK == res) {    // Status query successful
} else {        // Query failed
}
```

2.5.33 GetProgramState

Function declaration

ROBOTAPI_API 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 = 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
```

2.5.34 GetSpeedRatio

Function declaration

ROBOTAPI_API 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 = GetSpeedRatio (ratio);// Query JOG magnification value
if(ERROR_OK == res) {   // Status query successful
} else {       // Query failed
}
```

2.5.35 IsPowerOn

Function declaration

```
ROBOTAPI_API 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 = IsPowerOn (is_poweron);
if(ERROR_OK == res) {   // Status query successful
} else {       // Query failed
}
```

2.5.36 GetPos

Function declaration

ROBOTAPI_API 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 = GetPos (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.

2.5.37 GetJoint

Function declaration

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

2.5.38 GetAlarmState

Function declaration

```
ROBOTAPI_API 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 = GetAlarmState (state);
if(ERROR_OK == res) {   // Query successful
} else {        // Query failed
}
```

2.5.39 GetAlarmList

Function declaration

```
ROBOTAPI_API int GetAlarmList(std::vector<unsigned int>& list)
```

Function

Query the current alarm list.

Parameter

state(out)	Whether it is currently alarming.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
std::vector<unsigned int> list;
res = GetAlarmList (list);
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.

2.5.40 GetIntVariable

Function declaration

```
ROBOTAPI_API int GetIntVariable(unsigned int index, int& value)
```

Function

Query the value of an integer variable.

Parameter

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

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
int value;
res = GetIntVariable(0, value);  // Query the first integer value
```

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

2.5.41 GetDoubleVariable

Function declaration

```
ROBOTAPI_API int GetDoubleVariable(unsigned int index, double& value)
```

Function

Query the first integer value

Parameter

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

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
int value;
res = GetDoubleVariable(0, value);      // Query the first floating point value
if(ERROR_OK == res) {   // Query successful
} else {        // Query failed
}
```

2.5.42 GetBoolVariable

Function declaration

```
ROBOTAPI_API int GetBoolVariable(unsigned int index, bool& value)
```

Function

Query the value of a Boolean variable.

Parameter

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

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
int value;
res = GetBoolVariable(0, value);        // Query the first boolean value
if(ERROR_OK == res) {   // Query successful
} else {        // Query failed
}
```

2.6 Program running

2.6.43 SendProgram

Function declaration

```
ROBOTAPI_API int SendProgram(std::string 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;
std::string file_path =  “d:\\test_program”;
res = SendProgram (file_path); // Send the entire folder of d:/test_program
if(ERROR_OK == res) {   // Query successful
} else {        // Query failed
}
file_path =  “d:\\prog\\test.arl”;
```

```
res = SendProgram (file_path); // Send d:/prog/test.arl file
if(ERROR_OK == res) {   // Query successful
} else {        // Query failed
}
```

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

2.6.44 LoadProgram

Function declaration

```
ROBOTAPI_API int LoadProgram(std::string 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;
std::string file_path =  “d:\\test_program”;
res = SendProgram (file_path); // Send the entire folder of d:/test_program
if(ERROR_OK == res) {   // Query successful
} else {        // Query failed
}
file_path =  “test_program\\test.arl”;
res = LoadProgram (file_path); // Load the test.arl program in the test_program directory that was sent before
if(ERROR_OK == res) {   // Query successful
} else {        // Query failed
}
```

2.6.45 StartProgram

Function declaration

ROBOTAPI_API int StartProgram()

Function

Start the ARL program.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
std::string file_path = “d:\\test_program”;
res = SendProgram (file_path); // Send the entire folder of d:/test_program
if(ERROR_OK == res) {  // Query successful
} else {       // Query failed
}
file_path = “test_program\\test.arl”;
res = LoadProgram (file_path); // Load the test.arl program in the test_program directory that was sent before
if(ERROR_OK == res) {  // Query successful
} else {       // Query failed
}
res = StartProgram ();     // Starting program
if(ERROR_OK == res) {  // Query successful
} else {       // Query failed
}
```

2.6.46 PauseProgram

Function declaration

ROBOTAPI_API int PauseProgram()

Function

Pause the program.

Return

0	Success.
Non-0	Error code.

Usage

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

2.6.47 ResetProgram

Function declaration

```
ROBOTAPI_API int ResetProgram()
```

Function

Reset the program.

Return

0	Success.
Non-0	Error code.

Usage

```
int res = ERROR_OK;
res = 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 Data structure description

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

3.1 RobotPos

```
// 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 3-1.

Table 3-1 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.

3.2 RobotJoint

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

For the description of RobotJoint related parameters, see Table 3-2.

Table 3-2Description of RobotJoint related parameters

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

3.3 RobotTool

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

```
double z;

double a;

double b;

double c;

bool stationary;

};
```

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

Table 3-3Description 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

3.4 RobotWorkpiece

```
// Workpiece coordinate system

struct RobotWorkpiece {

double x;

double y;

double z;

double a;

double b;

double c;

bool robhold;

};
```

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

Table 3-4Description 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

Name	Description
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

4 Error code description

Table 4-1 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_ENABLE_API_CONTROL_FAILED	Failed to set external API control.
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 / V2.0.0 / 2021.06.10
© 2011-2021 Peitian Robotics Co., Ltd. All right Reserved.

The description about the product characteristics and availability does not constitute a performance guarantee, and is reference only.The scope of services for the products delivered is subject to the specific contract.