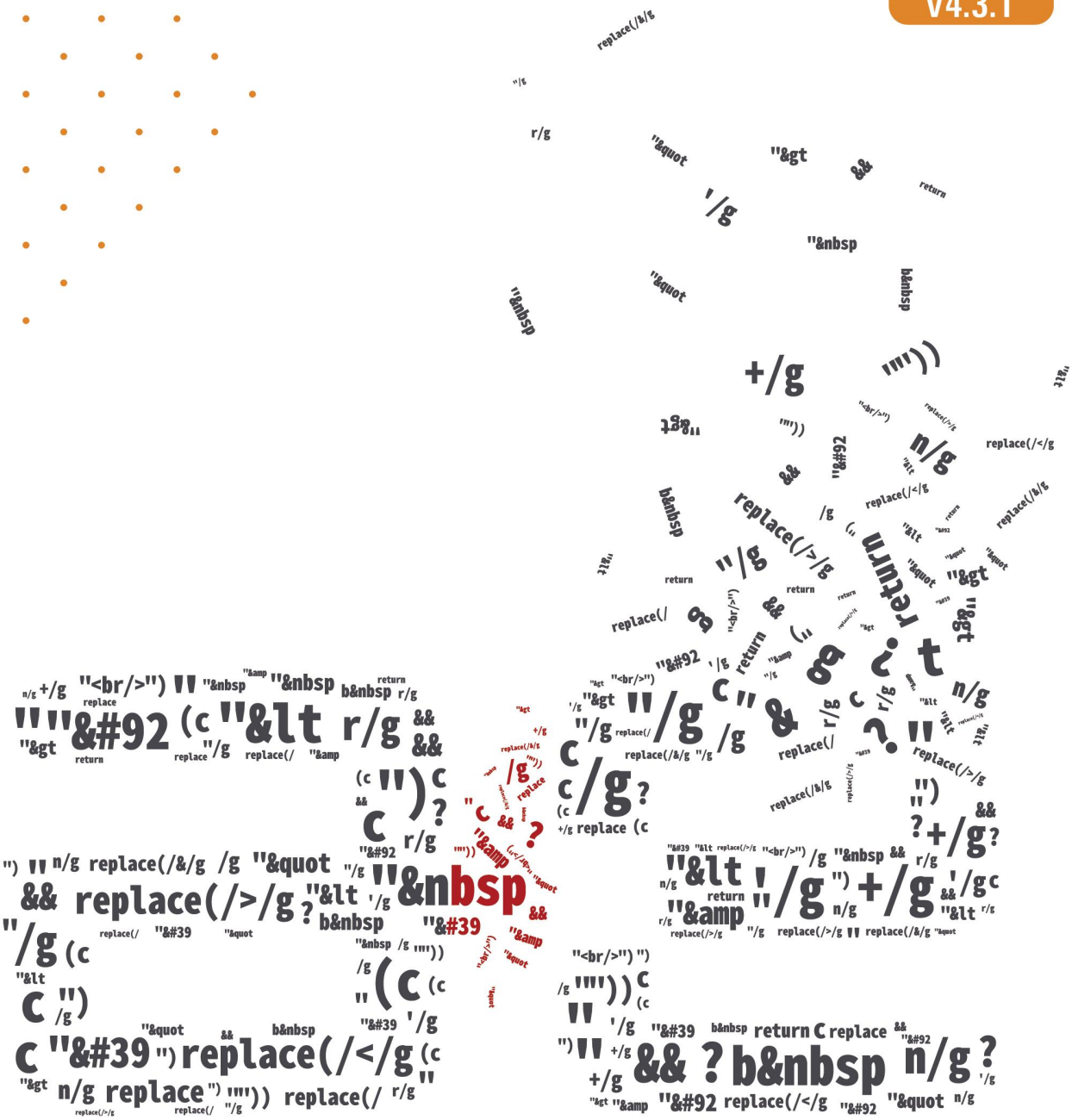
ARL Programming Manual

V4.3.1



Foreword

About this manual

ARL Programming Manual describes the ARL programming format specifications and the programming instructions supported by ARL.

Target groups

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

Meaning of common signs

See Table 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.	BJM/SS-UG-02-003
Document Ver.	V4.3.1
Software ver.	2.6.4

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://ligentrobot.com/

General safety instructions

Thank you for your purchase of our products. The contents of this manual must be observed in order to use our products safely. Please read the relevant manual carefully and understand the contents before operation.


Warning

Programming should be performed outside the safety fence as much as possible. When it is necessary to perform programming inside the safety fence, you should pay attention to the followings:

1. Check the conditions inside the safety fence carefully and ensure that there is no danger before entering the inside of the fence.
2. Ensure that the emergency stop button can be pressed at any time.
3. Operate the manipulator at low speed.
4. Start work after confirming the status of the entire system to prevent the operating personnel from danger due to remote control instructions or actions on peripheral equipment.


Notice

Always perform test running according to the prescribed steps after programming. At this time, the operating personnel must work outside the safety fence.


Tip

The programming personnel must receive appropriate training by the company.

Notation convention

The following is an example of simplified syntax with instruction TPWrite.

movej j:,[v: | vp:],[s: | sp: | sl:],[t:],[dura:]

- There are no mandatory parameters in brackets.
- Use square brackets "[ ]" to enclose optional parameters, these parameters can be ignored.
- The mutually exclusive parameters cannot exist in the same instruction at the same time, and they must be separated by a vertical bar "|" in the same instruction.
- Use curly brackets "{ }" to enclose the parameters that can be repeated any number of times.

The above example uses the following parameters:

- j is a mandatory parameter.
- v, vp, s, sp, sl, t and dura are optional parameters.
- v and vp are mutually exclusive.
- s, sp, and sl are mutually exclusive.

Revision record

The revision record accumulates the description of each document update. The latest version of the document contains the updated content of all previous document versions.

Table 2 Identifiers used in this article

version	release time	Explanation of modification
V4.2.0	2020/10/30	**The fifth official release** Software version upgraded to V2.6.3 ■ New structure type: Centroid_Pos (tool load centroid)/Inertia_Tensor (tool load inertia principal axis direction)/ToolInertiaPara (tool load type) ■ Added motion instruction: ccir (continuous arc motion) ■ Add IO related functions: getnostopdi (non-stop forward-looking asynchronous acquisition of single-channel DI) / syncao (synchronous output analog signal) / getao (obtain analog output signal) ■ Add system variables: $WOBJ_OFFSET (workpiece coordinate system offset)/$TOOL_OFFSET (tool coordinate system offset)
V4.3.0	2021/09/02	**The sixth official release** Software version upgraded to V2.6.4 ■ Added smooth motion command spl ■ Added the command startdetect to enable collision detection and enddetect to disable collision detection ■ See section "5.13.8~5.13.11" for the new function to record the foreground running program name and line number ■ Added "5.6.4 ModBusTCP related functions" ■ Added "5.7.5 ftobytes (float to bytes)" and "5.7.6 tofloat (bytes to float)"
V4.3.1	2022/05/05	**The seventh official release** ■ Fixed known bugs ■ Added "File Size Query Function, File Rename Function, and File Delete Function"

Precautions for Safety

Before operating the manipulator and peripheral equipment and the manipulator system thereof, the precautions for safety of the operating personnel and the system must be fully considered. Fig. 1 shows the schematic diagram of safe operation of industrial robot.

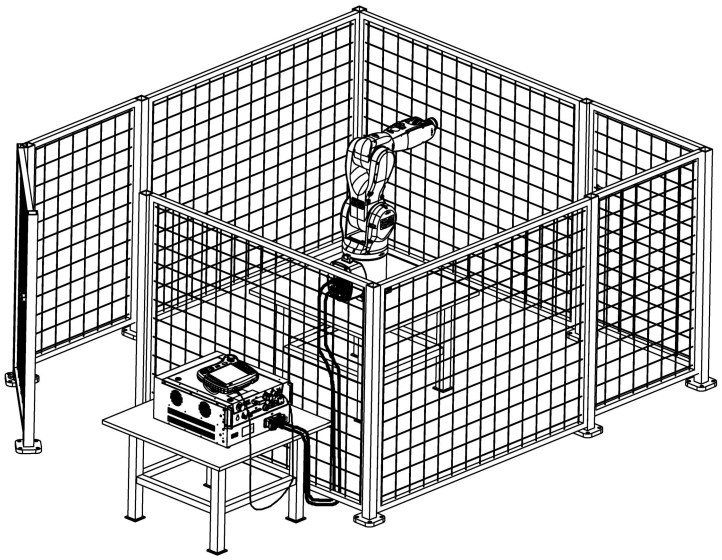


Fig. 1 Safe operation of industrial robot

Definition of operating personnel

The manipulator's operating personnel mainly includes operators, teachers and maintenance engineers, who all should meet the following conditions:

Operator

- He switches on/off the power supply of the manipulator.
- He starts the operator program via the operator panel.
- He has no right to work inside the safety fence.

Teacher

- He has the duties of the operator.
- He can teach the manipulator inside the safety fence.

Maintenance engineer

- He has the duties of the teacher.
- He can perform the manipulator maintenance (repSOT, adjustment, replacement, etc.)

Safety of operating personnel

Operators, teachers and maintenance engineers must pay attention to their safety during operation, programming and maintenance of the manipulator. At least the following items should be worn before operation:

- Work clothes suitable for the task
- Safety shoes
- Helmet

When an automatic system is used, every effort should be made to guarantee the safety of the operating personnel. It is very dangerous to enter the operating range of the manipulator. Measures should be taken to prevent the operating personnel from entering the operating range of the manipulator.

The general precautions are given below. Please take appropriate measures to guarantee the safety of the operating personnel:

- The operating personnel of the manipulator system should receive training by the company and pass relevant examinations.
- When the manipulator is running, even if it appears to have stopped, it may be waiting for a start signal and is in a state where it is about to operate. In this case, the manipulator should also be regarded as running. In order to guarantee the safety of the operating personnel, it is necessary to confirm that the manipulator is in a running state with an alarm light or sound.
- Always provide a safety fence and a safety door around the system so that the operating personnel cannot enter the safety fence without opening the safety door. The safety door should be equipped with an interlock switch or safety latch so that the manipulator will stop when the operating personnel opens the safety door.
- The peripheral equipment should be electrically grounded.
- The peripheral equipment should be arranged outside the operating range of the manipulator as much as possible.
- The operating range of the manipulator should be clearly marked by drawing lines on the floor so that the operator understands the operating range of the manipulator, including mechanical arm equipped on the manipulator.
- A shim switch or a photoelectric switch should be installed on the floor so that when the operating personnel is about to enter the operating range of the manipulator, an alarm can be sent via a buzzer and light to stop the manipulator, thereby guaranteeing the safety of the operating personnel.
- A lock should be provided as required. Except the operating personnel responsible for operation, none can power on the manipulator.
- The manipulator must be powered off during single commissioning of peripheral equipment.

Safety of teacher

In certain cases, it is necessary to enter the operating range of the manipulator when teaching the manipulator. At this time, particular attention should be paid to safety:

- When it is unnecessary to enter the operating range of the manipulator, be sure to work outside the operating range of the manipulator.
- Before teaching, ensure that the manipulator or peripheral equipment is in a safe state.
- When it is necessary to enter the operating range of the manipulator for teaching, please confirm the position and state of safety devices (such as emergency stop button, teach pendant emergency auto stop switch, etc.) in advance.
- The teacher should keep in mind that irrelevant personnel is not allowed to enter the operating range of the manipulator.
- Before starting the manipulator, confirm that none stays within the operating range of the manipulator and there are no abnormalities.
- After teaching, be sure to perform test running as follows:
    1. Single step at low speed for at least one cycle to confirm that there are no abnormalities.
    2. Run at low speed continuously for at least one cycle to confirm that there are no abnormalities.

3. Run at medium speed continuously for at least one cycle to confirm that there are no abnormalities.
4. Run continuously for one cycle at the running speed to confirm that there are no abnormalities.
5. Execute the program in auto running mode.

- The teacher must evacuate outside the safety fence when the manipulator is in auto running.

Contents

1 Overview of ARL

1.1 Terms and abbreviations

- ARL — AE Robot Language. It specifies the robot programming language.
- CP movement — Cartesian path movement. It mainly includes linear and circular movement and corresponds to Lin and Cir in the movement instructions.
- PTP movement — Point-to-point movement. It corresponds to PTP and Movej in movement instructions.
- DI — Digital input. It refers to digital input signal.
- DO — Digital output. It refers to digital output signal.
- AO — Analog output. It specifies the analog input signal.
- RPP — Return to path point. It means that the robot returns from the current position to the path point.

1.2 Program file

ARL is the abbreviation of AE Robot Language, which is the AE robot programming language. The user can write user programs to control the robot via ARL.

ARL program file postfix is arl, for example: AEMoveObj.arl.

1.3 Code comments

2 formats of code comments in ARL:

- "//" to comment the part of a line following "//"

  Program example:

  ```
  //Waittime 5
  waituntil getdi (2) //Wait until channel 2 DI signal is true
  ```

  
  Tip

  The compiler will, after reading a line, ignore the part following "//", and then execute parsing & compilation.

- "/*" and "*/" to comment an area

  Program example:

  ```
  /* this prog is for
    moving object */
  movej j:{j1 20}
  waittime 5
  ```

1.4 Linefeed connector

When a line in the program is too long and you want to write it in two or more lines, you can use the linefeed connector "\".

Program example:

```
if(~x == -6 && y == 3 && A == 60 && (A&B) == 12 \
&& (A|B) == 61 && (A^B) == 49 && (~A) == -61 \
```

```
&& (A<<2) == 240 && (A>>2) == 15)
return true
else
return false
endif
```

1.5 Subprogram

The subprogram technology can make the robot program more modular, which can help programmers design the program. structure effectively. The purpose is not to write all instructions into one program, but to put specific flows, calculations or processes into separate programs, which achieves the modular reuse.

Tip

- When the subprogram and the main program are in the same directory, the calling method:
  SubProg::func()
- The calling method when the subprogram and the main program are in different directories:
  import "/home/ae/.../SubProg.arl" //Import subroutine file
  SubProg::func()
  When the program pointer is executed to the above block, it will go to the func function of the SubProg program.
- The structure of subprogram will not be significantly different from that of ordinary program, except that the main function can be excluded.
- After the called function of the subprogram ends (that is, after execution to endfunc), the program pointer will return to the calling place. If you want to end the subprogram in advance, you can insert a return instruction in the place where you want to end, which will end the subprogram's operation in advance.

1.6 Functions

In order to reduce the number of characters and the length of program, functions are introduced when processing similar, usually repeated program parts.

- An ARL program can are composed of one or more functions

The function definition format is as follows:

```
func returntype funcname(argtype argname, argtype argname ...)
Internally implemented for the function body
endfunc
```

Where "returntype" is the return value type, "funcname" is the function name, "argtype" is the parameter type, and "argname" is the parameter name.

- Example of the function that implements two integer additions

Program example:

```
func int add(int a, int b)
return a+b
endfunc
```

The function call format is as follows:

```
funcname(arg1, arg2, ...)
```

- The method to call the previously defined add function

Example:

```
int c = add(1,2) //c value is 3
```

- The function return values can continue to be used in expressions

Example:

```
int c = add(1,2)*3 //c value is 9
```

- ARL program has no concept of pointer

ARL program has no concept of pointer, and the function parameters are only passed by values. If you want to modify the value of an incoming parameter within a function, you must declare the parameter as a global variable.

The print result is 1 instead of 2, because the self-add operation of a by the add function is not passed to the variable a.

Program example:

```
func void add(int a)
a++
endfunc
func void main()
init()
int a=1
add(a)
print a
endfunc
```

- main function

ARL program must implement a main function, which is the entry function of the program. After the program is reset, the program pointer will point to the first line of code in the main function. When the program runs, execution will start from the first line of code in the main function.

The main function has no parameters or return values, and is defined as follows:

Definition of main function

```
func void main()
...
endfunc
```

Where void indicates that the function has no return values.

- Write HelloWorld program in ARL language

Program example:

```
//HelloWorld.arl
func void main()
print "Hello world!"
endfunc
```

print is the printout function, which is printed to the HMI message bar by default. After loading and running the program, you will see "Hello world!" that is printed in the message bar on HMI.

It should be noted that if different functions are defined in the same file, the called function must be defined before being called, otherwise the error that the called function cannot be found will be reported when loading.

2 Variables and operations

2.1 Constants

A constant refers to the measure whose value may not be modified during program execution.

ARL supports four types of basic constants as follows:

- Integer (int)
- Double (double)
- Boolean (bool)
- Character string (string)

Where integer constants can be expressed in binary (number followed by b) or hexadecimal (number followed by h); double constants can be expressed in scientific notation.

For example, below are all legal constants:

- 1 //decimal integer
- 10b //binary integer, the value is 2
- 3eh //hexadecimal integer, the value is 62
- 3.245 //double
- 0.15e2 //double expressed in scientific notation, which refers to 0.15 times the square of 10, the value is 15
- true //boolean, the value is true
- false //boolean, the value is false

2.2 Variables and names

A variabl refers to the measure whose value can be modified during program execution. ARL supports five types of basic variables as follows:

- int
- byte
- double
- bool
- string

You must declare the variable before using it. The variable name is user-defined and can be composed of letters, underscores and numbers. Variable names may not begin with a number. Variable names may not be named as system keywords. For ARL system keywords, please refer to *Appendix A "ARL Keywords" .*

- You must declare the variable before using it. The general format of the variable declaration statement is as follows:

  Variable type Variable name

  or

  Variable type Variable name = initial value

  Program example:

```
inta = 1
```

The declaration statement declares a variable of type int and name a, and assigned with an initial value of 1.

- The variables of the same type can be declared on the same line, in the format as follows:

  Variable type Variable name = initial value, variable name = initial value, ...

  Program example:

  ```
  int x = 1, y = 3
  ```

  The declaration statement declares two variables of type int and name x and assigned with an initial value of 1, and a variable of name y and assigned with an initial value of 3.

- If the const keyword is added before the defined variable, it indicates that the variable may not be assigned again.

  Program example:

  ```
  const double pi = 3.1415926
  pi = 3.4
  ```

  An error will be reported when the program is loaded.

2.3 Basic data type

ARL supports 6 types of basic variables:

- int
- uint
- byte
- double
- bool
- string

2.3.1 int(Integer)

The int type data is a 32-bit number, and represents an integer ranging within -2^{31}~2^{31}-1.

Program example:

```
int counter = 4
```

It indicates that an integer variable counter is defined, and its value is equal to 4.

2.3.2 uint(Unsigned integer)

The uint type data is a 32-bit number, and represents an integer ranging within 0~2^{32}-1.

Program example:

```
uint ae = 5
```

It indicates that an unsigned integer variable ae is defined, and its value is equal to 5.

2.3.3 byte(Byte type)

The byte data is an 8-bit number, and represents an integer ranging within 0~255.

Program example:

```
byte b = ffh
```

It indicates that a byte variable b is defined, and its value is equal to ffh, which is 255 in decimal.

2.3.4 double(Floating point)

The double type data represents a decimal ranging within $-1.7*10^{-308}$~$1.7*10^{308}$.

Program example:

```
double scbd = 1
```

It indicates that a double variable scbd is defined, and its value is equal to 1.

The double data has precision problems. Try to avoid expressions that compare equality between two double numbers.

Program example (wrong use):

```
double data = 0
...
if (data == 0)
...
endif
```

Program example (correct use):

```
double data = 0
...
if (abs (data) <0.000001)
...
endif
```

2.3.5 bool(Boolean)

The bool variable is logically true or false. The value of a bool variable can only be true or false.

Example:

```
bool io16 = true
```

It indicates that a bool variable io16 is defined, and its value is true.

2.3.6 string(String type)

The string data represents a string that is composed of one or more characters.

Example:

```
string name = "Vincent"
```

The statement defines a string variable name and initializes it to "Vincent".

However, some characters do not have corresponding separate symbols, such as linefeed character, so you must express them with escape characters that begin with a backslash "\".

The escape characters supported in ARL are as follows:

- \n Linefeed (LF). It moves the current position to the beginning of the next line
- \r Carriage return (CR). It moves the current position to the beginning of the local line

- \t Horizontal tab (HT), go to next tab position
- \" It represents a double quote character
- \\ It represents a backslash character "\"

2.3.7 Implicit type conversion of basic data

In some cases, the system will perform implicit type conversion against basic type variables. The types that are allowed for implicit type conversion are shown in Table 2-1:

Table 2-1 Implicit type conversion

from \ to	byte	int	double	bool
byte		√	√	√
int			√	√

Items marked with " √ " in the table indicate that the two types are allowed to be converted implicitly from the type on the left to the type above. For example, the byte type can be converted implicitly to the int type.

Example:

```
int a = 3
double b = 4
double c = a+b //The system will implicitly convert a to double and add it to b.
int counter = 5
while(counter--)
...
Endwhile //The system will implicitly convert counter to bool data. Non-zero values will be converted to true, and zero values will be converted to false. So the code loop will be executed 5 times.
```

2.4 Structure type (Applicable to six-axis robots)

ARL supports the system-defined structure types. The structure types are composite types that are composed of basic types.

2.4.1 Declaration, initialization and reference of structure variables

Taking the pos type as an example, the coordinates of any point in space are composed of coordinate components x, y & z, so the pos type (space point coordinate type) is composed of 3 subcomponents of double type.

The definition of the pos type is as follows:

```
struct pos
{
double x
double y
double z
}
```

The declaration of structure type variables is the same as that of basic type variables. The format is as follows:

Variable type Variable name

Example:

```
pos a
```

It declares a variable of type pos and name a. If the value of a is printed with the print instruction:

```
print a
```

The system will output:

```
{9e+09,9e+09,9e+09}
```

The three numbers separated by commas are the values of the components in the variable a respectively. It can be seen that by default, the components x, y & z of the pos variable a are initialized to 9e+09, 9e+09 and 9e+09, respectively.

The variables can be initialized at the same time of declaration. The structure variable can be initialized in two formats as follows:

- The first n parameters among the structure parameters are initialized in the format as follows:

  Structure type Variable name = {value of component 1, value of component 2, ... value of component n}

- One or more parameters among the structure parameters are initialized:

  Structure type Variable name = {component name Component value, component name Component value, ...}

For the above two initialization methods of the structure variable, the uninitialized components will be initializedby the system to the default values.

Program example:

```
pos a = {1, 2}
print a //output"{1,2,9e+09}"
pos b = {x 3, z 4}
print b //output"{3, 9e+09, 4}"
```

The parameter operator "." is used to access the parameter components of a structure variable in the format as follows:

Structure variable name . Component name

Program example:

```
pos a
a.x = 1
a.y = 2
a.z = 3
double b = a.x
print a //Output"{1,2,3}"
print b //Output"1"
```

The structure supports nesting, that is, a parameter of a structure type may be another structure type. The definition of the following structure type tool is considered:

```
struct tool
{
frame t_frame
bool stationary
```

}

A variable of tool type is declared and output:

```
tool a
print a
```

The system will output:

{{0,0,0,0,0,0},false}

Here the structure variables will be output in the form of nested braces. Similarly, when the structure variables are initialized, they must be in nested braces.

Program example:

```
tool a = {{10,10,0,0,90,0},false}
print a //output "{{10,10,0,0,90,0},false}"
print a.t_frame //output "{10,10,0,0,90,0}"
print a.t_frame.x //output "10"
tool b = ({x 10, b 90},false}
print b //Output "{{10,0,0,0,90,0},false}"
```

2.4.2 pos(Space point coordinates)

Description

The pos structure type is used to describe the coordinates of any point in space.

Definition

```
struct pos
{
double x
double y
double z
}
```

Parameter

The parameters of the pos structure type are shown in Table2-2.

Table2-2 Parameters of pos structure type

Name	Description
x	Type: double
	It refers to x component of any point in space, in mm.
y	Type: double
	It refers to y component of any point in space, in mm.
z	Type: double
	It refers to z component of an point in space, in mm.

Example

```
pos p1 = {0,0,0}
print p1 //Output "{0,0,0}"
p1.x = p1.x +50
print p1 //Output "{50,0,0}"
```

2.4.3 frame(Coordinate System)

Description

The frame structure type is used to describe the transformation relationship between two rectangular coordinate systems in space.

Definition

```
struct frame
{
double x
double y
double z
double a
double b
double c
}
```

Parameter

The parameters of the frame structure type are shown in Table2-3.

Table2-3 Parameters of frame structure type

Name	Description
x	Type: double
	It refers to x component of the coordinate of the transformed coordinate system origin in the original coordinate system, in mm
y	Type: double
	It refers to y component of the coordinate of the transformed coordinate system origin in the original coordinate system, in mm
z	Type: double
	It refers to Z component of the coordinate of the transformed coordinate system origin in the original coordinate system, in mm
a	Type: double
	It refers to a component expressed in Euler angle of the transformed coordinate system pose in the original coordinate system, in deg
b	Type: double
	It refers to b component expressed in Euler angle of the transformed coordinate system pose in the original coordinate system, in deg

Name	Description
c	Type: double
	It refers to c component expressed in Euler angle of the transformed coordinate system pose in the original coordinate system, in deg

There are 12 different expressions of Euler angles, depending on the order of the axis around which they are rotated. The ZYX Euler angles are used here. That is, the order transforming from the initial coordinate system pose to the transformed coordinate system pose is to rotate the an angle around z axis, then rotate the b angle around the new y axis, and finally rotate the c angle around the new x axis.

Example

```
frame f = {10,10,0,0,90,0}
print f //Output "{10,10,0,0,90,0}"
```

2.4.4 pose(Cartesian target point)

Description

The pose structure type uses Cartesian coordinates to describe the positions of the robot axis and the external axis.

Definition

```
struct pose
{
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
}
```

Parameter

The parameters of the pos structure type are shown in Table2-4.

Table2-4 Parameters of pos structure type

Name	Description
x	Type: double

Name	Description
	It refers to x component of the robot's TCP relative to the current workpiece coordinate system coordinates, in mm
y	Type: double
	It refers to y component of the robot's TCP relative to the current workpiece coordinate system coordinates, in mm
z	Type: double
	It refers to x component of the robot's TCP relative to the current workpiece coordinate system coordinates, in mm
a	Type: double
	It refers to a component expressed in Euler angle of the robot's tool pose in the current workpiece coordinate system, in deg
b	Type: double
	It refers to b component expressed in Euler angle of the robot's tool pose in the current workpiece coordinate system, in deg
c	Type: double
	It refers to c component expressed in Euler angle of the robot's tool pose in the current workpiece coordinate system, in deg
cfg	Type: int
	It specifies the configuration of the robot axis. Since there may be several different methods available for the robot to make TCP reach the same pose, only one method is specified through the cfg parameter to uniquely determine the axis positions of a group of robots
turn	Type: int
	It is used together with cfg to determine the axis position of the inverse solution. Only the lower 6 bits of turn are used here, among which bit0 represents J1, bit1 represents J2, and so on. When the turn value is -1, it indicates that the solution closest to the starting point will be selected automatically; when a bit value is 1, it indicates that a solution less than 0 will be selected for the axis; when a bit value is 0, it indicates that a solution more than 0 will be selected for the axis. The parameter may be used to assist the solution selection when the axis's movement range is greater than 360 degrees. In other cases, the parameter will not be set, and the default value of -1 will be taken.
ej1~ej6	Type: double
	It specifies the positions of the external axes J1~J6, in deg

Table 2-5 cfg value and pose relationship

cfg value	Center of wrist relative to axis 1	Center of wrist relative to forearm	5-axis angle
0	on the right	on the right	positive
1	on the right	on the right	negative
2	on the right	on the left	positive
3	on the right	on the left	negative
4	on the left	on the left	positive
5	on the left	on the left	negative
6	on the left	on the right	positive

cfg value	Center of wrist relative to axis 1	Center of wrist relative to forearm	5-axis angle
7	on the left	on the right	negative

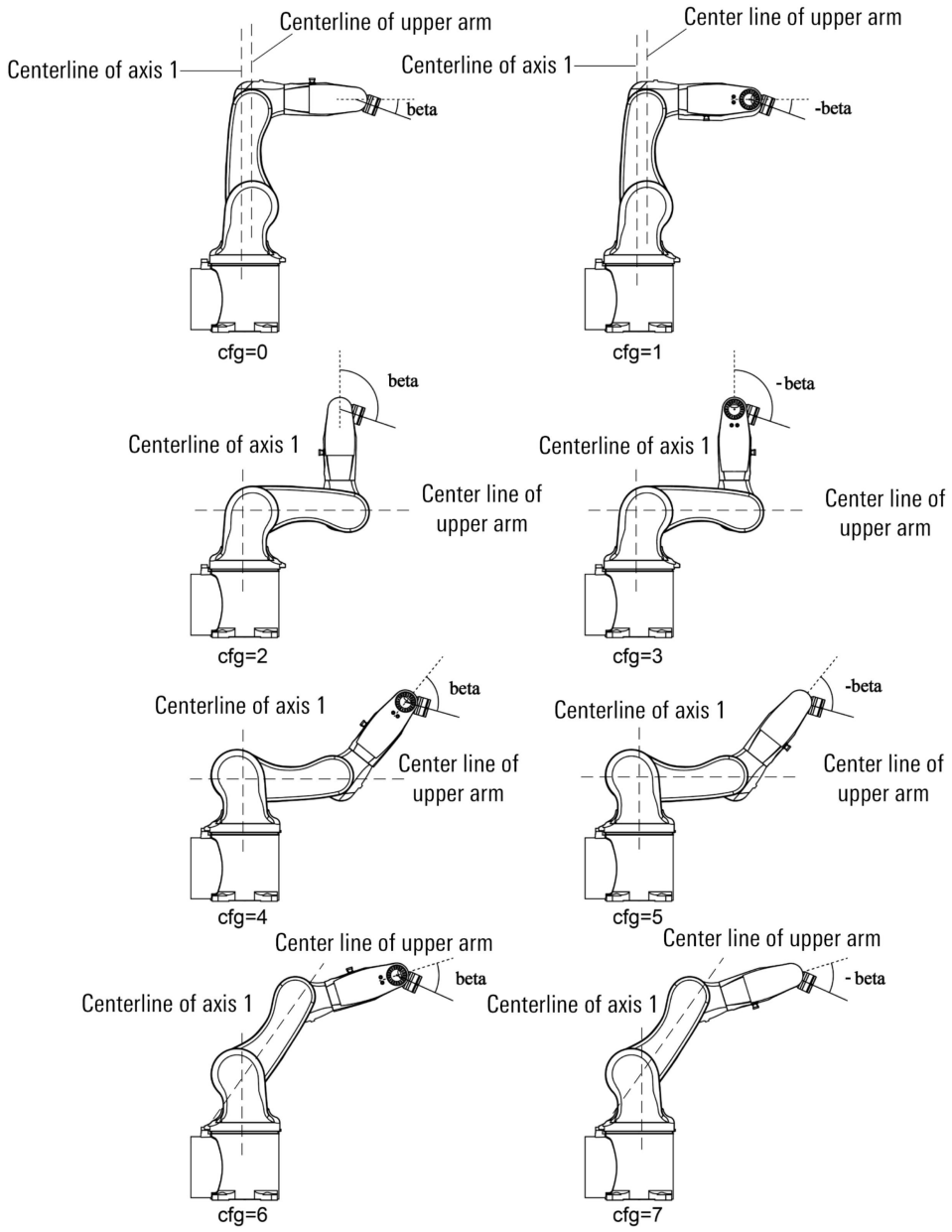


Figure 2-1 Description of cfg parameter value attitude

Notice

For definition of Euler angle, please refer to *Section 2.4.3*.

Example

```
pose p1 = {x 0, y 10, z 15, a 0, b 90, c 0, cfg 0, ej1 20}
print p1 //Output "{0,10,15,0,90,0,0, -1,20,9e+09,9e+09,9e+09,9e+09,9e+09,9e+09}"
pose p2 = {0, -10,15,0,90,0,0, -1,10}
print p2 //Output "{0,-10,15,0,90,0,0, -1,10,9e+09,9e+09,9e+09,9e+09,9e+09,9e+09}"
```

2.4.5 joint(Axis target point)

Description

The joint structure type is used to directly describe the positions of the robot axis and the external axis.

Definition

```
struct joint
{
double j1
double j2
double j3
double j4
double j5
double j6
double ej1
double ej2
double ej3
double ej4
double ej5
double ej6
}
```

Parameter

The parameters of the joint structure type are shown in Table2-6.

Table2-6 Parameters of joint structure type

Name	Description
j1~j6	Type: double
	It specifies the positions of the robot axes J1~J6, in deg
ej1~ej6	Type: double
	It specifies the positions of the external axes J1~J6, in deg

Example

```
joint p1 = {j1 0, j3 15, j5 0, ej1 20}
print p1 //Output "{0,9e+09,15, 9e+09,0,9e+09,20,9e+09,9e+09,9e+09,9e+09,9e+09}"
joint p2 = {0,10,20,30,40,50,10}
print p2 //Output "{0, 10,20,30,40,50,10,9e+09,9e+09,9e+09,9e+09,9e+09,9e+09}"
```


Notice

When the positions of all axes are specified, the parameter names in the joint can be defaulted, but they all must be defaulted. For example, the following statement is wrong: j: {0,0,90,0,0,0, ej1 0}

2.4.6 tool(Tool)

Description

The tool structure type is used to describe a tool.

Definition

```
struct tool
{
frame t_frame
bool stationary
}
```

Parameter

The parameters of the tool structure type are shown in Table2-7.

Table2-7 Parameters of tool structure type

Name	Description
t_frame	Type: frame
	It specifies the tool coordinate system defined on the tool, which is defined relative to the flange coordinate system
stationary	Type: bool
	false: It specifies the tool as the tool installed at the end of the robot flange. At this time, the tool coordinate system is defined with reference to the flange coordinate system. true: It specifies the tool as an external fixed tool. At this time, the tool coordinate system is defined with reference to the world coordinate system.

Example

```
frame f = {10,10,0,0,90,0}
tool t1
t1.t_frame = f
t1.stationary = false
print t1 //Output "{{10,10,0,0,90,0},false}"
tool t2 = {{0,10,20,0,0,0}, true}
print t2 //Output "{{0,10,20,0,0,0}, true}"
```

2.4.7 wobj(Workpiece coordinate system)

Description

The wobj structure type is used to describe a workpiece coordinate system.

Definition

```
struct wobj
{
frame w_frame
bool robhold
string mu_name
```

}

Parameter

The parameters of the wobj structure type are shown in Table2-8.

Table2-8 Parameters of wobj structure type

Name	Description
w_frame	Type: frame
	It specifies the workpiece coordinate system
robhold	Type: bool
	false: It specifies the workpiece coordinate system to be defined with reference to the world coordinate system true: It specifies the workpiece to be grabbed by the robot. At this time, the workpiece coordinate system is defined with reference to the flange coordinate system.
mu_name	string
	Mechanical unit name

Example

```
frame f = {10,10,0,0,90,0}
wobj w1
w1.w_frame = f
print w1 //Output "{{10,10,0,0,90,0},false}"
wobj w2 = {{0,10,20,0,0,0},false}
print w2 //Output "{{0,10,20,0,0,0},false}"
```

2.4.8 weavedata(Swing graphics parameters)

Description

The weavedata structure type is used to describe the weave graphical parameters.

Definition

```
struct weavedata
{
enum weaveshape
double frequency
double amplitude
double dwell_left
double dwell_right
double dwell_middle
bool track
double swing_angle
double radius
enum weaverotaxis
rotation_angle
bool vibrat
```

}

Parameter

The parameters of the weavedata structure type are shown in Table2-9.

Table2-9 Parameters of weavedata structure type

Name	Description
weaveshape	Type: enum
	It specifies the enumeration type, which contains all graphic types supported by the overlay trajectory. For details, please refer to *Section 2.6.8*
frequency	Type: double
	It specifies the swing cycle frequency, in hz
amplitude	Type: double
	It specifies the swing amplitude, in mm
dwell_left	Type: double
	It specifies the left dwell length. When the cycle type is frequency, the parameter represents the dwell time, in s. When the cycle type is wavelength, the parameter represents the dwell distance, in mm. The minimum value of the parameter is 0
dwell_right	Type: double
	It specifies the right dwell length. When the cycle type is frequency, the parameter represents the dwell time, in s. When the cycle type is wavelength, the parameter represents the dwell distance, in mm. The minimum value of the parameter is 0
dwell_middle	Type: double
	It specifies the center dwell length. When the cycle type is frequency, the parameter represents the dwell time, in s. When the cycle type is wavelength, the parameter represents the dwell distance, in mm. The minimum value of the parameter is 0
track	Type: bool
	It indicates whether to perform weld tracking, represents tracking if true, and represents no tracking if false. It can be defaulted, and the default value is false
swing_angle	Type: double
	It specifies the swing angle of the space triangle pendulum and space v-shaped swing, in deg
radius	Type: double
	It specifies the graphic radius, in mm
weaverotaxis	Type: enum
	The enumeration type data contains the coordinate axes to which the swing plane deflections supported by the overlay trajectory are referenced. For details, please refer to Section 3.5.9.
Vibrat	Type: bool
	It indicates whether to start the vibration function.
rotation_angle	Type: double

Name	Description
	It specifies the deflection angle of swing plane, in deg

Example

```
weavedata weavedatalin1 = {2,15,1,1,0, true}
print weavedatalin1 //Output "{2,15,1,1,0,1, true}"
weavedata weavedatalin2 = {2,15}
print weavedatalin2 //Output "{2,15,0,0,0,0, false}"
```

2.4.9 speed(Speed)

Description

The speed structure type is used to describe the speed parameter of a movement instruction.

Definition

```
struct speed
{
double per
double tcp
double ori
double exj
double exl
}
```

Parameter

The parameters of the speed structure type are shown in Table2-10.

Table2-10 Parameters of speed structure type

Name	Description
per	Type: double
	It specifies the speed percentage. The speed parameter is used by the ptp and movej instructions to indicate the max velocity percentage of the axis. The value ranges within 0.001~100.
tcp	Type: double
	It specifies the TCP movement speed. The speed parameter is used by the lin and cir instructions to indicate the TCP translation speed, in mm/s. The value range of the same model is as follows: SOT3/4/6L/7L/10/20：TCP=2500 SOT50、165：TCP=1300
ori	Type: double
	It specifies the tool coordinate system pose rotation speed. The speed parameter is used by the lin and cir instructions to indicate the TCP rotation speed, in deg/s. The range of values for different models is as follows： ■ SOT3/4/6L/7L/10：ORI=500 ■ SOT20：ORI=500 ■ SOT50、165：ORI=250

Name	Description
exj	Type: double
	It specifies the external axis rotating speed. When there is a rotating external axis, the speed parameter will be used during movement, in deg/s.
exl	Type: double
	It specifies the external axis linear speed. When there is a linear external axis, the speed parameter will be used during movement, in mm/s.

Example

```
speed v = {per 10}
print v //Output "{10,9e+09,9e+09,9e+09,9e+09}"
```

2.4.10 slip(Smoothing parameters)

Description

The slip structure type is used to describe the slip parameters of a movement instruction.

Definition

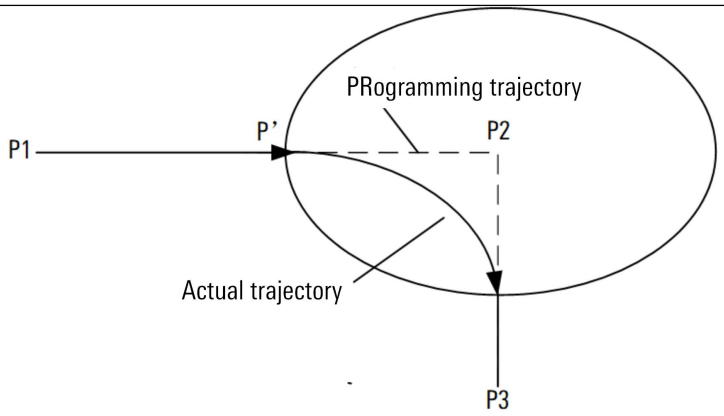
```
struct slip
{
double perdis
double pdis
double odis
double ejdis
double eldis
}
```

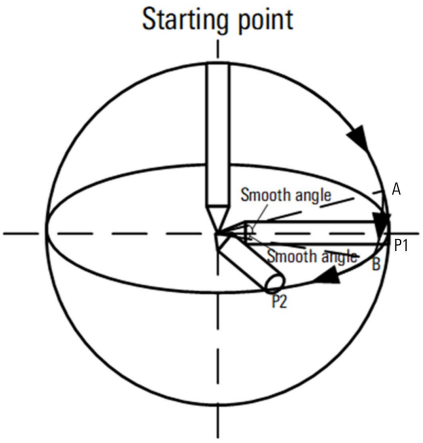
Parameter

The parameters of the slip structure type are shown in Table2-11.

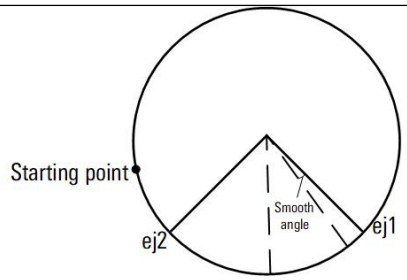
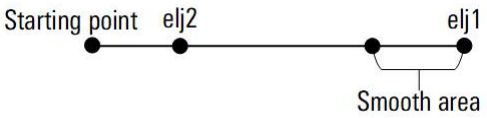
Table2-11 Parameters of slip structure type

Name	Description
perdis	Type: double
	Percentage distance, All motion instructions can use this smoothing parameter. When this smoothing parameter is valid, other smoothing parameters will be invalid. Indicates the percentage of the track to enable smoothing in the smoothable segment, 100% is fully smoothed, and 0% is not smoothed
pdis	Type: double
	The pos distance indicates how far the TCP is from the target point in millimeters (smoothing area radius) when the smoothing starts, in millimeters. As shown in the figure below, the programmed trajectory is "P1→P2→P3". When the TCP enters the smooth area, it moves along the smooth arc trajectory, and after leaving the smooth area, it moves in a straight line to P3.

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td></td><td> 

Tip
The smoothing distance (straight line P'P2) cannot be greater than half of the track length (straight line P1P2). If it is greater than half of the track length on one side, the smoothing distance on that side track will be reduced to half the track length.</td></tr>
<tr><td rowspan="2">odis</td><td>Type: double</td></tr>
<tr><td>Orientation distance, lin and cir instructions use this smoothing parameter to indicate what angle (smooth angle) the posture is from the target point, the smoothing starts, in degrees.
As shown in the figure below, the tool's programming trajectory is "start point→P1→P2". When the tool moves along a circular arc to point A (that is, within a smooth angle), it starts to move smoothly to point B in the smooth area, and then from point B Move to P2.
 

Tip
The smoothing angle cannot be greater than half of the angle between the start and end of the track. If it is greater than half of the angle between the start and end of a certain side track, the smoothing angle of the side track will be reduced to half of the angle between the start and end of the side track.</td></tr>
<tr><td rowspan="2">ejdis</td><td>Type: double</td></tr>
<tr><td>ex-joint distance. This smoothing parameter is used for motion instructions with a rotating external axis. It indicates how many degrees (smooth angle) the slowest external axis is from the target point, when smoothing begins. When the external axis is a rotating axis, the unit is degree; When the axis is a linear axis, the unit is mm.
As shown in the figure below, the programmed trajectory of the external axis is "start point→ej1→ej2". When the external axis rotates to the smooth angle range, it starts to move smoothly and starts to move smoothly in the direction of point ej2.</td></tr>
</table>

Name	Description
	
eldis	Type: double exlinearjoint distance. This smoothing parameter is used for motion instructions with a movable external axis, which indicates how many millimeters (smoothing distance) the slowest external axis is from the target point when it starts to level off. When the external axis is a rotary axis, the unit is degree; when the external axis is a linear axis, the unit is millimeter. As shown in the figure below, the programmed trajectory of the external axis is "start point→elj1→elj2". When the external axis moves linearly to the smooth area, it starts to move smoothly and starts to move smoothly towards the ej2 point. 

Example

slip s = {100,2,3,4,5}

print s//Output "{100,2,3,4,5}"

Notice

- When the parameter jdis or pdis is less than or equal to 0, it indicates that the movement statement with such parameter is not slip.
- Please try to complete all parameters of the structure when initializing the structure variables, so as to prevent the actual movement trajectory or movement speed from being inconsistent with the user's expectations due to the user's negligence, even causing unnecessary loss or injury.
- Perdis has the highest priority. When perdis is available, other parameters will be invalid. When the pdis, odis, ejdis, or jdis, ejdis entered are valid simultaneously, the value of the slip point that is closer to the target point will be valid.

2.4.11 jvel(Joint speed)

Description

The jvel structure type is used to describe the speed value of each joint of the robot.

Definition

struct jvel

{

double jv1

double jv2

double jv3

double jv4

double jv5

double jv6

double ev1

double ev2

double ev3

double ev4

double ev5

double ev6

}

Parameter

The members of the jvel structure type are shown in Table 2-12.

Table 2-12 The members of jvel structure type

Name	Description
jv1~jv6	Type: double
	The axis speed of the robot 1~6 axis, unit: rad/s
ev1~ev6	Type: double
	The axis speed of the robot's external 1 axis ~ external 6 axis, unit: rad/s

Example

```
jvel jvel1 = {jv1 1, jv3 2, ev1 3}
print jvel1 //Output "{1,9e+09,2,9e+09,9e+09,9e+09,3,9e+09,9e+09,9e+09,9e+09,9e+09}"
jvel jvel2 = {0,1,2,3,4,5,1}
print jvel2 //Output "{0,1,2,3,4,5,1,9e+09,9e+09,9e+09,9e+09,9e+09}"
```

2.4.12 Centroid_Pos(Tool load centroid)

Description

The Centroid_Pos structure type is used to describe the position of the centroid of the tool load.

Definition

struct Centroid_Pos

{

double x

double y

```
double z

}
```

Parameter

The parameters of the Centroid_Pos instruction are shown in Table 2-13.

Table 2-13 Centroid_Pos instruction parameters

Name	Description
x	Type: double
	The x component of the coordinate of the tool load centroid in the reference coordinate system, in millimeters.
y	Type: double
	The y component of the coordinate of the tool load centroid in the reference coordinate system, in millimeters.
z	Type: double
	The z component of the coordinate of the tool load centroid in the reference coordinate system, in millimeters.

Example

```
Centroid_Pos cen_pos = {10, 20, 30}
print cen_pos //Output "{10, 20, 30}"
```

2.4.13 Inertia_Tensor(Tool load inertia principal axis direction)

Description

The Inertia_Tensor structure type is used to describe the inertia parameters of the tool load.

Definition

```
struct Inertia_Tensor

{

double Ixx

double Ixy

double Ixz

double Iyy

double Iyz

double Izz

}
```

Parameter

The parameters of the Inertia_Tensor instruction are shown in Table 2-14.

Table 2-14 Parameters of the Inertia_Tensor instruction

Name	Description
Ixx	Type: double
	The xx component of the load inertia matrix, the unit is g*mm^2。
Ixy	Type: double
	The xy component of the load inertia matrix, the unit is g*mm^2。
Ixz	Type: double
	The xz component of the load inertia matrix, the unit is g*mm^2。
Iyy	Type: double
	The yy component of the load inertia matrix, the unit is g*mm^2。
Iyz	Type: double
	The yz component of the load inertia matrix, the unit is g*mm^2。
Izz	Type: double
	The zz component of the load inertia matrix, the unit is g*mm^2。

Example

```
Inertia_Tensor I_T = {10, 20, 30,40,50,60}
print I_T //Output "{10, 20, 30,40,50,60}"
```

2.4.14 ToolInertiaPara(Tool load type)

Description

ToolInertiaPara structure type includes load mass, center of mass, direction of inertia principal axis and moment of inertia.

Definition

```
struct ToolInertiaPara
{
double m
Centroid_Pos centroidpos
Inertia_Tensor inertiatensor
```

}

Parameter

The parameters of ToolInertiaParah instruction are shown in Table 2-15.

Table 2-15 ToolInertiaPara instruction parameters

Name	Description
m	Type: double
	Tool load mass, unit g.
centroidpos	Type: Centroid_Pos
	Tool load centroid position
inertiatensor	Type: Inertia_Tensor
	Describe the inertia parameters of the tool load

Example

```
ToolInertiaPara tip //Define tool load
tip.m = 30 //Quality
tip.centroid_pos = {15, 25, 100} //Define the centroid location
tip. Inertia_Tensor = {10, 20, 30,40,50,60} //Define the direction of the principal axis of inertia
print tip //Output "{30,{15,25,100},{10, 20, 30,40,50,60}}"
```

2.5 Structure type（Suitable for SCARA robot）

ARL supports the system-defined structure types. The structure types are composite types that are composed of basic types.

2.5.1 Declaration, initialization and reference of structure variables

Taking the pos type as an example, the coordinates of any point in space are composed of coordinate components x, y, z, so the pos type (space point coordinate type) is composed of 3 subcomponents of double type.

The definition of the pos type is as follows:

```
struct pos
{
double x
double y
double z
}
```

The declaration of structure type variables is the same as that of basic type variables. The format is as follows:

Variable type Variable name

Example:

```
pos a
```

It declares a variable of type pos and name a. If the value of a is printed with the print instruction:

```
print a
```

The system will output:

{9e+09,9e+09,9e+09}

The three numbers separated by commas are the values of the components in the variable a respectively. It can be seen that by default, the components x, y & z of the pos variable a are initialized to 9e+09, 9e+09 and 9e+09, respectively.

The variables can be initialized at the same time of declaration. The structure variable can be initialized in two formats as follows:

- The first n parameters among the structure parameters are initialized in the format as follows:

  Structure type Variable name = {value of component 1, value of component 2, ... value of component n}

- One or more parameters among the structure parameters are initialized:

  Structure type Variable name = {component name Component value, component name Component value, ...}

For the above two initialization methods of the structure variable, the uninitialized components will be initialized by the system to the default values.

Program example:

```
pos a = {1,2}
print a //Output "{1,2,9e+09}"
pos b = {x 3,z 4}
print b //Output "{3,9e+09,4}"
```

The parameter operator "." is used to access the parameter components of a structure variable in the format as follows:

Structure variable name. Component name

Program example:

```
pos a
a.x = 1
a.y = 2
a.z = 3
double b = a.x
print a //Output "{1,2,3}"
```

The structure supports nesting, that is, a parameter of a structure type may be another structure type. The definition of the following structure type tool is considered:

```
struct tool
{
frame t_frame
bool stationary
}
```

Declare a tool type variable and output:

```
tool a
print a
```

The system will output:

```
{{0,0,0,0,0,0},false}
```

Here structure variables will be output in the form of nested curly braces. Similarly, the structure variable should be initialized in the form of nested curly braces.

Program example:

```
tool a = {{0,0,0,0,0,0},false}
print a //Output "{{0,0,0,0,0,0},false}"
print a.t_frame //Output "{0,0,0,0,0,0}"
print a.t_frame.x //Output "0"
tool b = {{x 10},false}
print b //Output "{{10,0,0,0,0,0},false}"
```

2.5.2 pos(Space point coordinates)

Description

The pos structure type is used to describe the coordinates of any point in space.

Definition

```
struct pos
{
double x
double y
double z
}
```

Parameter

The members of the pos structure type are shown in Table 2-16.

Table 2-16 member of pos structure type

Name	Description
x	Type: double
	It refers to x component of any point in space, in mm
y	Type: double
	It refers to y component of any point in space, in mm
z	Type: double
	It refers to z component of an point in space, in mm

Example

```
pos p1 = {0,0,0}
print p1//Output "{0,0,0}"
p1.x = p1.x +50
print p1//Output "{50,0,0}"
```

2.5.3 frame(Coordinate System)

Description

The frame structure type is used to describe the transformation relationship between two rectangular coordinate systems in space.

Definition

```
struct frame
{
double x
double y
double z
double a
double b
double c
}
```

Parameter

The parameters of the frame structure type are shown in Table 2-17.

Table 2-17 Parameters of frame structure type

Name	Description
x	Type: double
	It refers to x component of the coordinate of the transformed coordinate system origin in the original coordinate system, in mm
y	Type: double
	It refers to y component of the coordinate of the transformed coordinate system origin in the original coordinate system, in mm
z	Type: double
	It refers to Z component of the coordinate of the transformed coordinate system origin in the original coordinate system, in mm
a	Type: double
	It refers to a component expressed in Euler angle of the transformed coordinate system pose in the original

Name	Description
	coordinate system, in deg
b	Type: double
	It refers to b component expressed in Euler angle of the transformed coordinate system pose in the original coordinate system, in deg
c	Type: double
	It refers to c component expressed in Euler angle of the transformed coordinate system pose in the original coordinate system, in deg

There are 12 different expressions of Euler angles, depending on the order of the axis around which they are rotated. The ZYX Euler angles are used here. That is, the order transforming from the initial coordinate system pose to the transformed coordinate system pose is to rotate an angle around z axis, then rotate the b angle around the new y axis, and finally rotate the c angle around the new x axis.

Example

```
frame f = {10,10,0,0,0,0}
print f //Output "{10,10,0,0,0,0}"
```

2.5.4 pose(Descartes target point)

Description

The pose structure type uses Cartesian coordinates to describe the positions of the robot axis and the external axis.

Definition

```
struct pose
{
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

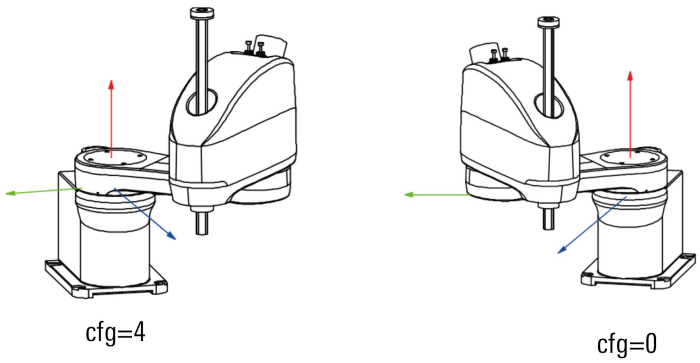
double ej6

}

Parameter

The parameters of the pos structure type are shown in Table 2-18.

Table 2-18 Parameters of pos structure type

Name	Description
x	Type: double
	It refers to x component of the robot's TCP relative to the current workpiece coordinate system coordinates, in mm
y	Type: double
	It refers to y component of the robot's TCP relative to the current workpiece coordinate system coordinates, in mm
z	Type: double
	It refers to x component of the robot's TCP relative to the current workpiece coordinate system coordinates, in mm
a	Type: double
	It refers to a component expressed in Euler angle of the robot's tool pose in the current workpiece coordinate system, in deg
b	Type: double
	It refers to b component expressed in Euler angle of the robot's tool pose in the current workpiece coordinate system, in deg
c	Type: double
	It refers to c component expressed in Euler angle of the robot's tool pose in the current workpiece coordinate system, in deg
cfg	Type: int
	Robot axis configuration. Since the robot may make TCP reach the same posture in two ways: left-handed and right-handed, it is necessary to specify one of them through the cfg parameter (refer to Figure 2 2, 4 means left-handed, 0 means right-handed). Uniquely determine a set of robot axis positions.  cfg=4 cfg=0 Figure 2-2 cfg parameter value attitude description
turn	Type: int

Name	Description
	Cooperate with cfg to determine the axis position of the inverse solution. 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. This parameter may be used to assist solution selection when the axis's motion range is greater than 360 degrees. In most other cases, this value does not need to be set, and the default value is -1.
ej1~ej6	Type: double
	It specifies the positions of the external axes J1~J6, in deg


Notice

For definition of Euler angle, please refer to *Section 2.5.3*.

Example

```
pose p1 = {x 266,y 88,z -50,a -34,b 90,c 0,cfg 0,ej1 20}
print p1 //Output "{266, 88, -50, -34,90,0,0,-1,20,9e+09,9e+09,9e+09,9e+09,9e+09}"
pose p2 = {308, 52, -50, -33,90,0,0,-1,10}
print p2 //Output "{308, 52, -50, -33,90,0,0,-1,10,9e+09,9e+09,9e+09,9e+09,9e+09}"
```

2.5.5 joint(Axis target point)

Description

The joint structure type is used to directly describe the positions of the robot axis and the external axis.

Definition

```
struct joint
{
double j1
double j2
double j3
double j4
double ej1
double ej2
double ej3
double ej4
double ej5
double ej6
}
```

Parameter

The parameters of the joint structure type are shown in Table 2-19.

Table 2-19 Parameters of joint structure type

Name	Description
j1~j4	Type: double
	It specifies the positions of the robot axes J1~J4, in deg
ej1~ej6	Type: double
	It specifies the positions of the external axes J1~J6, in deg

Example

joint j1={j1 0, j2 0, j3 -10, j4 0}

print to:hmi,argtoprint:j1 //Output "{0,0,-10,0,9e+9, 9e+9, 9e+9, 9e+9, 9e+9, 9e+9, 9e+9, 9e+9}"


Notice

When the positions of all axes are specified, the parameter names in the joint can be defaulted, but they all must be defaulted. For example, the following statement is wrong: j:{0,0,0,0,0,0,ej1 0}。

2.5.6 tool(Tool)

Description

The tool structure type is used to describe a tool.

Definition

struct tool

{

frame t_frame

bool stationary

}

Parameter

The parameters of the tool structure type are shown in Table 2-20.

Table 2-20 Parameters of tool structure type

Name	Description
t_frame	Type: frame
	Tool coordinate system defined on the tool
stationary	Type: bool
	■ false: It specifies the tool as the tool installed at the end of the robot flange. At this time, the tool coordinate system is defined with reference to the flange coordinate system. ■ true: It specifies the tool as an external fixed tool. At this time, the tool coordinate system is defined with reference to the world coordinate system.

Example

```
frame f = {10,10,0,0,0,0}
tool t1
t1.t_frame = f
t1.stationary = false
print t1 //Output "{{10,10,0,0,0,0},false}"
tool t2 = {{0,10,0,0,0,0},true}
print t2 //Output "{{0,10,0,0,0,0},true}"
```

2.5.7 wobj(Workpiece coordinate system)

Description

The wobj structure type is used to describe a workpiece coordinate system.

Definition

```
struct wobj
{
frame w_frame
bool robhold
string mu_name
}
```

Parameter

The parameters of the wobj structure type are shown in Table 2-21.

Table 2-21 Parameters of wobj structure type

Name	Description
w_frame	Type: frame
	It specifies the workpiece coordinate system
robhold	Type: bool
	■ false: It specifies the workpiece coordinate system to be defined with reference to the world coordinate system ■ true: It specifies the workpiece to be grabbed by the robot. At this time, the workpiece coordinate system is defined with reference to the flange coordinate system.
mu_name	Type: string
	Mechanical unit name

Example

```
frame f = {10,10,0,0,0,0}
wobj w1
```

```
w1.w_frame = f
print w1 //Output "{{10,10,0,0,0,0},false}"
wobj w2 = {{0,10,0,0,0,0},false}
print w2 //Output "{{0,10,0,0,0,0},false}"
```

2.5.8 speed(Speed)

Description

The speed structure type is used to describe the speed parameter of a movement instruction.

Definition

```
struct speed
{
double per
double tcp
double ori
double exj
double exl
}
```

Parameter

The parameters of the speed structure type are shown in Table 2-22.

Table 2-22 Parameters of speed structure type

Name	Description
per	Type: double
	It specifies the speed percentage. The speed parameter is used by the ptp and movej instructions to indicate the max velocity percentage of the axis. The value ranges within 0.001~100.
tcp	Type: double
	It specifies the TCP movement speed. The speed parameter is used by the lin and cir instructions to indicate the TCP translation speed, in mm/s.
ori	Type: double
	It specifies the tool coordinate system pose rotation speed. The speed parameter is used by the lin and cir instructions to indicate the TCP rotation speed, in deg/s.
exj	Type: double
	It specifies the external axis rotating speed. When there is a rotating external axis, the speed parameter will be used during movement, in deg/s.
exl	Type: double
	It specifies the external axis linear speed. When there is a linear external axis, the speed parameter will be

Name	Description
	used during movement, in mm/s.

Parameter

speed v = {per 10}

print v //Output "{10,9e+09,9e+09,9e+09,9e+09}"

2.5.9 slip(Smoothing parameters)

Description

The slip structure type is used to describe the slip parameters of a movement instruction.

Definition

struct slip

{

double pdis

double ejdis

double eldis

double odis

double perdis

}

Parameter

The parameters of the slip structure type are shown in Table 2-23.

Table 2-23 Parameters of slip structure type

Name	Description
pdis	Type: double
	The pos distance, lin and cir instructions use this smoothing parameter to indicate how far the TCP is from the target point to start smoothing, the unit is millimeters
ejdis	Type: double
	ex-joint distance, use this smoothing parameter for motion instructions with a rotating external axis to indicate how far the slowest external axis is from the target point when smoothing begins. When the external axis is a rotary axis, the unit is degree; the external axis is a linear axis, The unit is mm
eldis	Type: double
	exlinearjoint distance. Use this smoothing parameter for movement instructions with a movable external axis. It indicates how far the slowest external axis is from the target point to start smoothing. When the external axis is a rotary axis, the unit is degree; when the external axis is a linear axis, Unit is mm
odis	Type: double
	Orientation distance, lin and cir instructions use this smoothing parameter to indicate how far the

Name	Description
	posture is from the target point, and the unit is degree.
perdis	Type: double
	Percentage distance, all motion instructions can use this smoothing parameter. When this smoothing parameter takes effect, other smoothing parameters will be invalid. Indicates the percentage of smoothing on the trajectory in the smoothable segment, 100% is completely smooth, 0% is not smooth

Example

```
slip s = {100,2,3,4,5,5}
print s //Output "{100,2,3,4,5,5}"
```

- When the pdis parameter is less than or equal to 0, it means that the motion instruction using it is not smooth.
- Please try to write all the members of the structure when initializing the structure variables to prevent the actual movement track or movement speed from being inconsistent with the user's expectations due to the user's negligence, which may lead to unnecessary losses or injuries.
- Perdis has the highest priority. In the case of perdis, the other parameters are invalid; when the input pdis, odis, and ejdis take effect, the value of the smooth point closer to the target point takes effect.

2.5.10 jvel(Joint speed)

Description

The jvel structure type is used to describe the speed value of each joint of the robot.

Definition

```
struct jvel
{
double jv1
double jv2
double jv3
double jv4
double ev1
double ev2
double ev3
double ev4
double ev5
double ev6
}
```

Parameter

The members of jvel structure type are shown in Table 2-24.

Table 2-24 Members of jvel structure type

Name	Description
jv1~jv4	Type: double
	The axis speed of robot 1~4 axis, unit: rad/s
ev1~ev6	Type: double
	The axis speed of the robot's external 1 axis ~ 6 axis, unit: rad/s

Example

```
jvel jvel1 = {jv1 1, jv3 2, ev1 3}
print jvel1 //Output "{1,9e+09,2,9e+09,9e+09,9e+09,3,9e+09,9e+09,9e+09,9e+09,9e+09}"
jvel jvel2 = {0,1,2,3,4}
print jvel2 //Output "{0,1,2,3,4, 9e+09,9e+09,9e+09,9e+09,9e+09,9e+09,9e+09}"
```

2.5.11 control(Control parameters of gate type action)

Description

The control structure type is used to describe the speed, acceleration, and deceleration during the straight-up and straight-down phases of the door action.

Definition

```
struct control
{
double rising_vel
double rising_acc
double rising_dec
double falling_vel
double falling_acc
double falling_dec
}
```

Parameter

The members of the control structure type are shown in Table 2-25.

Table 2-25 Members of control structure type

Name	Description
rising_vel	Type: double
	The percentage of the maximum speed in the helicopter stage. If it is not specified, it will refer to the current speed parameter. If specified, it will represent the percentage of the maximum shaft speed.
rising_acc	Type: double
	The maximum acceleration percentage in the heli phase. If not specified, it will refer to the current acceleration parameters. If specified, it will represent the maximum axis acceleration percentage.
rising_dec	Type: double
	The maximum deceleration percentage in the helicopter stage. If not specified, the current acceleration parameter will be referred to. If specified, it will represent the maximum axis acceleration percentage.
falling_vel	Type: double
	The percentage of the maximum speed in the straight-down stage. If not specified, it will refer to the current speed parameter. If specified, it will represent the percentage of the maximum axis acceleration.
falling_acc	Type: double
	The percentage of the maximum acceleration in the straight-down stage. If not specified, the current acceleration parameter will be referred to. If specified, it will represent the percentage of the maximum axis acceleration.
falling_dec	Type: double
	The maximum deceleration percentage in the straight-down stage. If not specified, it will refer to the current acceleration parameters. If specified, it will represent the percentage of the maximum axis acceleration.

Example

```
control ctr1 = {rising_vel 0, rising_acc 0, rising_dec 0, falling_vel 0, falling_acc 0, falling_dec 100}
print ctr1 //Output "{0, 0, 0, 0, 0, 100}"
```

2.6 Enumeration type

ARL supports the system's predefined enumeration types. The values of the enumeration type variables can only be selected from the fixed values.

For example, the enumeration types are defined as follows:

```
enum $VEL_PROFILE
{
T_type,
S_type,
O_type
}
```

The enumeration type defines the different types of velocity profiles, where T_type, S_type and O_type represent T-type velocity profile, S-type velocity profile and O-type velocity profile respectively. The enumeration type is a kind of special integer. T_type, S_type and O_type actually correspond to the integers 0, 1 & 2 respectively.

The format of the reference enumeration variable is as follows:

Enumeration type name::Enumeration constant name

Example:

```
print $VEL_PROFILE :: S_TYPE //Output "S_TYPE"
```

In general, you can omit the enumeration type name in the program and refer to the enumeration constant directly with the enumeration constant name.

Example:

```
print S_TYPE //Output "S_TYPE"
```

However, when the same variable name as the enumeration constant name is defined in the program, the system will preferentially identify the name as the variable name rather than the enumeration constant name.

Example:

```
int S_TYPE = 6
print S_TYPE //Output "6"
```

In this case, the enumeration type name cannot be omitted when referring to the enumeration constant.

Example:

```
int S_TYPE = 6
print $VEL_PROFILE :: S_TYPE //Output "S_TYPE"
```

2.6.1 $VEL_PROFILE(Speed curve)

Description

The $VEL_PROFILE enumeration type contains all velocity profile types supported by the movement trajectory.

Definition

```
enum $VEL_PROFILE
{
T_type,
S_type,
O_type
}
```

Parameter

The parameters of the $VEL_PROFILE enumeration type are shown in Table2-26.

Table2-26 Parameters of $VEL_PROFILE enumeration type

Name	Description
T_type	It specifies the T-type velocity profile
S_type	It specifies the S-type velocity profile
O_type	It specifies the O-type velocity profile

Example

```
$VEL_PROFILE = S_type //The following movement instructions are planned with S-type velocity
movej j:{j1 10}
```

For details about velocity profile, please refer to the relevant movement instructions.

2.6.2 printto(Output orientation)

Description

The printto enumeration type contains all output orientations that the print instructions can set.

Definition

```
enum printto
{
off,
hmi,
file,
console
}
```

Parameter

The parameters of printto enumeration type are shown in Table2-27.

Table2-27 Parameters of printto enum type

Name	Description
off	It specifies the switch off of output, that is, no output to anywhere
hmi	It specifies the output to the message bar on HMI (Human-machine interface)
file	It specifies the output to the specified file
console	It specifies the output to the terminal, and used for development mode. Usually, the setting will not be used by the user.

Example

```
print to:hmi, "hello world" //HMI message bar outputs "hello world"
```

For details about output orientation, please refer to Section *3.2*.

2.6.3 num_base(Output number system)

Description

The base enumeration type contains all output number systems that the print instructions can set.

Definition

enum num_base

{

hex,

dec

}

Parameter

The parameters of the base enumeration type are shown in Table2-28.

Table2-28 Parameters of base enum type

Name	Description
hex	Hexadecimal
dec	Decimal

Example

print hex,ffh //Output "ff"

print dec,ffh //Output "255"

For details about output number system, please refer to *Section 3.5*.

2.6.4 stoptype(Stop type)

Description

The stoptype enumeration type contains all stop methods of the movement trajectory.

Definition

enum stoptype

{

general,

fast

}

Parameter

The parameters of the stoptype enumeration type are shown in Table2-29.

Table2-29 Parameters of stoptype enum type

Name	Description
general	It specifies the general deceleration stop method

Name	Description
fast	It specifies the fast deceleration stop method

Example

stopmove fast //Stop the current movement with fast stop method

For details, please refer to *stopmove instruction*.

2.6.5 controlmode(Control mode)

Description

The controlmode enumeration type defines the control mode of the system.

Definition

enum controlmode

{

MANUAL,

AUTO,

MANUFAST

}

Parameter

The parameters of the controlmode enumeration type are shown in Table2-30.

Table2-30 Parameters of controlmode enumeration type

Name	Description
MANUAL	Manual low-speed mode
AUTO	Auto mode
MANUFAST	Manual high-speed mode

2.6.6 stopbits(Stop bits)

Description

The stopbits enumeration type defines the type of stop bits in serial communication.

Definition

enum stopbits

{

one,

two

}

Parameter

The parameters of the stopbits enumeration type are shown in Table2-31.

Table2-31 Parameters of stopbits enumeration type

Name	Description
one	It refers to 1 stop bit
two	It refers to 2 stop bits

2.6.7 parity(Parity type)

Description

The parity enumeration type defines the parity type in serial communication.

Definition

enum parity

{

none,

odd,

even

}

Parameter

The parameters of the parity enumeration type are shown in Table2-32.

Table2-32 Parameters of parity enumeration type

Name	Description
none	It indicates that no parity bit is used.
odd	It indicates that the odd parity is used.
even	It indicates that the even parity is used.

2.6.8 weaveshape(Superimposed trajectory type)

Description

The weaveshape enumeration type contains all graphic types supported by the superimposed trajectory.

Definition

enum weaveshape

```
{
simple,
V_shape,
triangle,
ellipse,
eight
}
```

Parameter

The parameters of the weaveshape enumeration type are shown in Table2-33.

Table2-33 Parameters of weaveshape enumeration type

Name	Description
simple	It specifies the yaw
V_shape	It specifies the V-shaped swing
triangle	It specifies the space triangle swing
ellipse	It specifies the ellipse swing
eight	It refers to "8" swing

Example

```
weavedata weavedata = {V_shape, 2,15,90} //The overlay graphic is specified as V-shaped swing
```

2.6.9 weaverotaxis(Superimposed trajectory swing plane deflection axis)

Description

The weaverotaxis enumeration type contains the coordinate axes to which the swing plane deflections supported by the overlay trajectory are referenced.

Definition

```
enum weaverotaxis
{
rot_x,
rot_y,
rot_z
}
```

Parameter

The parameters of the weaverotaxis enumeration type are shown in Table2-34.

Table2-34 Parameters of weaverotaxis enumeration type

Name	Description
rot_x	Refere to x-axis deflection
rot_y	Refere to y-axis deflection
rot_z	Refere to z-axis deflection

Example

weavedata weavedata = {V_shape, 2,15,90, rot_x, 10} //The swing plane of the overlay trajectory is specified to deflect with reference to x-axis

2.7 Other types

2.7.1 socket(Socket type)

Description

The socket type is used to communicate with external equipment via the internet access. The data type is used together with *connect*, *accept*, *write*, *read*, *readuntil* and other functions.

Example

Refer to the functions such as *connect*, *accept*, *write*, *read*, *readuntil*.

2.7.2 iodev(IO device type)

Description

The io device type is used for io read and write, such as communicating with external devices via serial read and write. The data type is used together with *open*, *write*, *read*, *readuntil* and other functions.

Example

Refer to the functions such as open, write, read, readuntil.

2.8 Array

It is convenient to process a collection of variables of the same type together. In this case, an array can be used. An array is a collection of elements of the same data type that are arranged in a certain order. The array can be one-dimensional or multi-dimensional:

- One-dimensional array

  It specifies the array in which the elements are not arrays.

- Multi-dimensional array

  It specifies the array which has the array equal to or above two-dimensional array. A two-dimensional array uses the arrays as its elements, and a three-dimensional array uses the two-dimensional arrays as its elements. Of course, the arrays of more dimensions can also be generated.

2.8.1 Declaration of array (preparation before using array)

The format in which an array variable is declared:

Variable type name Array variable name [size of 1st dimension] [size of 2nd dimension] ...

Example:

double a [10]//A one-dimensional array of double type is declared, and its size is 10

double b [2] [3]//A two-dimensional array of double type is declared, the size of 1st dimension is 2 and the size of 2nd dimension is 3.

2.8.2 Initialization of array

The array can be initialized at the same time of declaration. The format of data initialization is as follows:

{1st data value of 1st dimension, 2nd data value of 1st dimension, ... 1st data value of nth dimension, 2nd data value of nth dimension ... nth data value of nth dimension}

Example:

double a [3] = {1,2,3}

double b [2][3] = {1,2,3,1,2,3}

It is allowed to only initialize the first m elements of the array. The uninitialized elements will be the default values.

2.8.3 Access to array (application method of array)

The subscript is used to access the elements in the array. The subscript of each dimension of the array starts from 0.

Example:

double a [3] = {1,2,3}

double b [2][3] = {1,2,3,4,5,6}

print a [0] //Output "1"

print b [0] [1] //Output "2"


Notice

The subscript of one-dimensional array is limited to 10000, and the subscript product of multi-dimensional array is limited to 10000.

2.8.4 Structure variables for array management

Structure variables can also be managed as arrays.

Example:

pos p [3] = {{0,0,0}, {10,10,10},{20,20,20}}

print p [0] //Output"{0,0,0}"

print p [1].x//Output"10"

2.9 Operators

2.9.1 Arithmetic operator

The classification and role of arithmetic operators are shown in Table2-35:

Table2-35 Arithmetic operators

Operator	Role	Applicable data types
+	plus	int,byte,double,string,joint
-	Minus/negate	int, byte, double, joint
*	Multiply	int,byte,double,frame,pose
/	Divide	int,byte,double,
%	Modulo	int,byte,double,
-	Decrease by 1	int
++	Increase by 1	int

The addition of string type data represents string concatenation.

The multiplying of frame type data represents coordinate system transformation.

Examples of using arithmetic operators are as follows:

```
int sa = 1
int sb = 2
int sc = -sb //negate
int sac = sa*sc //multiply
sac = -2
```

Examples of using ++,- are as follows:

x = m++ indicates that m plus 1 after the value of m is assigned to x;

x = ++m indicates that m is increased by 1, which is then assigned to x.

-- operator is used in the same way as ++.

2.9.2 Bitwise operator

The classification and role of bitwise operators are shown in Table2-36:

Table2-36 Bitwise operators

Operator	Role	Applicable data types
<<	Shift left	int,byte
>>	Arithmetic shift right	int,byte
&	Bitwise AND	int,byte
|	Bitwise OR	int,byte
^	Bitwise XOR	int,byte
~	Bitwise negation	int,byte

Examples of using bitwise operators are as follows:

```
int A = 00111100b
int B = 00001101b
```

print A," ",A&B," ",A|B," ",A^B," ",~A," ",A<<2," ",A>>2 //Output "60 12 61 49 -61 240 15 "

2.9.3 Logical operators

The classification and role of logical operators are shown in Table2-37:

Table2-37 Logical operators

Operator	Role	Applicable data types
&&	Logical AND	int,byte,bool
||	Logical OR	int,byte,bool
<	Less than	int,byte,double,string
>	More than	int,byte,double,string
<=	Less than or equal to	int,byte,double,string
> =	More than or equal to	int,byte,double,string
==	Equal to	int,byte,double,bool,string
!=	Not equal to	int,byte,double,bool,string
!	Logical negation	int,byte, bool

The condition that the logic and "&&" expression is true is that the two results on both sides are true, while the condition that the logic or "||" is true is that only one result on both sides is true.

Examples of using logical operators are as follows:

```
int i = 0
while(getdi(1) && !getdi(2))
if(i<100)
i++
endif
endwhile
```

2.9.4 Assignment operator

The classification and role of the assignment operators are shown in Table2-38:

Table2-38 Assignment operators

Operator	Role
=	Assignment
+=	Plus & equal to
-=	Minus & equal to
*=	Multiply & equal to
/=	Divide & equal to
%=	Modulo & equal to

Operator	Role
<<=	Shift left & equal to
>>=	Arithmetic right shift & equal to
&=	Bitwise and equal to
|=	Bitwise or equal to

All data types support assignment operations. The data types supported by the XX= operator are the same as those supported by the XX operator.

a += b is equivalent to a = a+b, and so on.

Examples of using various assignment operators are as follows:

```
int a = 1
print a //Output "1"
a += 1
print a //Output "2"
a << = 2
print a //Output "8"
```

2.9.5 Other operators

The classification and role of other operators are shown in Table2-39:

Table2-39 Other operators

Operator	Role
[ ]	Array index For the application method of the array subscript operator, please refer to Section *"2.8 Array"*.
( )	Parentheses For the structure parameter variable operator, please refer to Section *"2.4 Structure"*.
.	Structure parameter variable.

The parentheses are used to specify the priority of operations, such as:

```
int num = (1+2)*3
print num //Output "9"
```

2.9.6 Operator priority

The priority order of different operators is shown in Table2-40. The lower the priority number in the table, the higher the priority. For operators of the same priority, the operation order depends on the combined direction.

Table2-40 Operator priority

Priority	Operator	Name or meaning	Use form	Combined direction	Description
1	[ ]	Array index	Array name [constant expression]	From left to right	-
	( )	Parentheses	(Expression)/function name		-

Priority	Operator	Name or meaning	Use form	Combined direction	Description
			(formal parameter list)		
	.	Structure variable parameter selection	Structure variable. Parameter name		--
2	-	Minus operator	-expression	From right to left	Unary operator
	++	Increment operator	++variable name/variable name ++		Unary operator
	-	Decrement operator	--variable name/variable name--		Unary operator
	!	Logical NOT operator	!expression		Unary operator
	~	Bitwise negation operator	~expression		Unary operator
3	/	Divide	Expression/expression	From left to right	Binocular operator
	*	Multiply	Expression*expression		Binocular operator
	%	Remainder (modulo)	Integer expression/integer expression		Binocular operator
4	+	plus	Expression+expression	From left to right	Binocular operator
	-	Minus	Expression-expression		Binocular operator
5	<<	Shift left	Variable<<expression	From left to right	Binocular operator
	>>	Shift right	Variable>>expression		Binocular operator
6	>	More than	Expression>expression	From left to right	Binocular operator
	> =	More than or equal to	Expression>=expression		Binocular operator
	<	Less than	Expression<expression		Binocular operator
	<=	Less than or equal to	Expression<=expression		Binocular operator
7	==	Equal to	Expression==expression	From left to right	Binocular operator
	!=	Not equal to	Expression!=expression		Binocular operator
8	&	Bitwise AND	Expression&expression	From left to right	Binocular operator
9	^	Bitwise XOR	Expression^expression		Binocular operator
10	|	Bitwise OR	Expression|expression		Binocular

Priority	Operator	Name or meaning	Use form	Combined direction	Description
					operator
11	&&	Logical AND	Expression&&expression	From left to right	Binocular operator
12	||	Logical OR	Expression||expression	From left to right	Binocular operator
13	=	Assignment operator	Variable=expression	From right to left	-
	/=	Assignment after division	Variable/=expression		
	*=	Assignment after multiplying	Variable*=expression		
	%=	Assignment after modulo	Variable%=expression		
	+=	Assignment after addition	Variable+= expression		
	-=	Assignment after subtraction	Variable-=expression		
	<<=	Assignment after shift left	Variable<<=expression		
	>>=	Assign after shift right	Variable>>=expression		
	&=	Assignment after bitwise AND	Variable&=expression		
	^=	assignment after bitwise XOR	Variable^=expression		
	|=	Assignment after bitwise OR	Variable|=expression		


Notice

In practice, if you are not sure of the priority of the operator, please use parentheses to show the desired order of expression as much as possible, otherwise the calculation result of the expression may not be as expected by the user.

2.10 Scope of variable

According to the scope, variables can be divided into local variables, global variables and system variables. In ARL, the variables that are declared inside a function are local variables, and the variables that are declared outside a function are global variables by default. The system variables refer to the system-predefined variables, which are not declared and can be directly referenced via "$" symbol.

Local variables are only valid locally.

For example, in the following program:

```
func void myfunc()
for(int i=1;i<2;i++)
int a = 1 //The variable is only valid within the range of for and endfor
print a //Here you can access a
endfor
print a //Here it will be reported that the variable a does not exist
```

```
endfunc
```

The variable names that are declared within the same scope cannot be repeated; otherwise the system will report an error that the variable is defined repeatedly. However, the variable names that are declared within different scopes can be repeated. At this time, the variable that is accessed by this name refers to the variable within the nearest scope. For example:

```
func void myfunc()
int a = 6
for(int i=1;i<2;i++)
int a = 1 //The variable is only valid within the range of for and endfor
print a //Output "1" here
endfor
print a //Output "6" here
endfunc
```

Global variables are accessible in all functions after they are declared. For example:

```
int a = 1
func void myfunc()
a = 2
print a //Output "2" here
endfunc
func void main()
print a //Output "1" here
myfunc()
endfunc
```

The system variables are accessible at any time, for example:

```
func void main()
$DFSPPED = {10,50,5,5,5} //Here the default speed is set as:{10,50,5,5,5}
print $DFSPPED //Output "{10,50,5,5,5}" here
……
Endfunc
```

3 Sequential instructions

3.1 General format of sequential instruction

Except flow control instructions, ARL sequential instructions are in general format as follows:

Instruction name Parameter name: parameter value, parameter name: parameter value, ……

Among them, some parameters are mandatory. If not provided, the system will report an error in instruction format. Some parameters are optional. Optional parameters can be defaulted. In this case, the default values will be taken or the previously set values will be maintained. The parameter in [ ] in the following instruction format indicates that the parameter is optional.

The names of some parameters can also be defaulted and only the parameter values are provided. However, the names of some parameters may not be defaulted. The form of "parameter name:" is used in the following instruction format to indicate that the name of the parameter may not be defaulted. There is a special case of syntax rules in which the parameter name is allowed to be defaulted. When the parameter value is a structure constant expressed in the form of braces "{}" rather than a variable, the parameter name may not be defaulted, otherwise the system will report an error in instruction format.

3.2 Motion instructions（Suitable for six-axis robots）

3.2.1 movej(Moving axis)

Description

The movej instruction is used to move the robot axis or external axis to a specified axis position. All axes will reach the target axis position simultaneously.

Format

movej j:,[v: | vp:],[s: | sp: | sl:],[t:],[dura:]

Parameter

The parameters of the movej instruction are shown in Table3-1.

Table3-1 Parameters of movej instruction

Name	Description
j	Data type: joint
	It specifies the target point positions of the robot axes and external axes. Among them, the structure component with the value of 9e+09 will remain at the starting position. If the first movement instruction executed in the program is movej, the target points j1~j6 of the movej instruction and the external axis may not be defaulted.
v	Data type: speed
	It specifies the movement speed. The movej instruction uses the components per and exj in the speed structure variable, which are used to specify the percentage of axis movement speed and the external axis movement speed respectively. If the parameter v is defaulted here, the system variable $DFSPEED will be used as the value of the parameter v by default; if the parameter v is specified, the value of the parameter component will be updated to the component corresponding to $DFSPEED.

Name	Description
	In simplified programming, v can be replaced by the velocity percentage parameter vp. See the application example for the format.
vp	Data type: double
	It specifies the percentage of movement speed. It can replace the speed parameter v, and can be used in the scenarios where it is unnecessary to specify the speed precisely. The format is in vp:10%, which indicates that the speed of the current line is 10% of the max speed of the robot.
s	Data type: slip
	It specifies the slip distance. The Movej instruction uses the components perdis, jdis and ejdis of the slip structure to describe the point where the original trajectory is disengaged and the slip trajectory is entered. Among them, perdis is used to specify the percentage of the distance from the target point, jdis is used to specify the axis position angle of the distance from the target point, and ejdis is used to specify the position angle of external axis from the target point. Here if the parameter s is defaulted, the system variable $DFSLIP will be used as the value of the parameter s by default; if the parameter s is specified, the value of the parameter component will be updated to the component corresponding to $DFSLIP In simplified programming, s can be replaced by the slip percentage parameter sp. See the application example for the format. Note: ■ When perdis> = 0, other components will be ignored; ■ When jdis and ejdis are specified simultaneously, the component that is closer to the target point will be selected; ■ When the above components are all 0, it represents non-slip, but the interpolation point does not necessarily coincide with the target point; ■ When the above components are all less than 0, it represents non-slip, and the target point will definitely have the interpolation point, ie orientation
sp	Data type: double
	It specifies the slip percentage. It can replace the slip parameter s, and can be used in the scenarios where it is unnecessary to specify the slip size precisely. The format is in sp:10%, which indicates that the slip distance of the target point of the current line is 10% of the max slip distance.
sl	Data type: double
	This parameter specifies the smoothing distance. It can replace the smoothing parameter s and is used in situations where the smoothing value needs to be precisely specified. The format is sl:10mm, which means that the smoothing distance of the current line is 10mm
t	Data type: tool structure
	It specifies the tool. The user must customize the tool coordinate system on the tool that is installed at the end of the flange. The tool coordinate system is fixed to the tool. The target point p specifies the position of the tool coordinate system in the coordinate system that is specified by the parameter w. Here if the t parameter is defaulted, the system variable $DFTOOL will be used as the value of the parameter t by default; if the parameter t is specified, the value of the parameter component will be updated to the component corresponding to $DFTOOL. The parameter is to ensure that the TCP speed is not greater than 250mm/s when running the program at T1.
dura	Data type: double
	It specifies the trajectory duration. The user can directly specify the movement duration rather than the movement speed. If the user specifies the parameter dura, the system will ignore the parameter speed, and will adjust the speed automatically to meet the time requirements specified by the parameter dura. Dura parameter, the unit is seconds.

Example

```
//The velocity and slip size are specified in a fuzzy manner
joint j1 = {j1 10,j2 20,j3 30,j4 40,j5 50,j6 60}
movej j:j1,vp:5%,sp:5%
movej j:{j1 10,j2 20,j3 30,j4 40,j5 50,j6 60}
movej j:{j1 20,ej1 30}
joint p = {0,20,30,40,50,60}
speed v = {per 50}
movej p,v
```

3.2.2 ptp(Point to point)

Description

The ptp instruction is used to quickly move the robot from one point to another without any requirements for the shape of the trajectory of the TCP. All axes will reach the target point simultaneously.

Format

ptp p:,[v: | vp:],[s: | sl: | sp:],[t:],[w:],[dura:]

Parameter

The parameters of the ptp instruction are shown in Table3-2.

Table3-2 Parameters of ptp instruction

Name	Description
p	Data type: pose
	It specifies the TCP target pose of the robot and the target point position of the external axis. Among them, the structure component with the value of 9e+09 will remain at the starting position. The default value of the component cfg is 0, and the default value of the component turn is -1. If the first movement instruction executed in the program is ptp, the target points X, Y, Z, A, B, C of the ptp instruction and the external axis may not be defaulted.
v	Data type: speed
	It specifies the movement speed. The ptp instruction uses the components per and exj in the speed structure variable, which are used to specify the percentage of axis movement speed and the movement speed of the external axis respectively. If the parameter v is defaulted here, the system variable $DFSPEED will be used as the value of the parameter v by default; if the parameter v is specified, the value of the parameter component will be updated to the component corresponding to $DFSPEED. In simplified programming, v can be replaced by the velocity percentage parameter vp. See the application example for the format.
vp	Data type: double
	It specifies the percentage of movement speed. It can replace the speed parameter v, and can be used in the scenarios where it is unnecessary to specify the speed precisely. The format is in vp:10%, which indicates that the speed of the current line is 10% of the max speed of the robot.
s	Data type: slip
	It specifies the slip distance. The PTP instruction uses the components perdis, jdis and ejdis of the slip structure to describe the point where the original trajectory is disengaged and the slip trajectory is entered. Among them, perdis is used to specify the percentage of the distance from the target point, jdis is used to specify the axis position angle

Name	Description
	of the distance from the target point, and ejdis is used to specify the position angle of external axis from the target point. Here if the parameter s is defaulted, the system variable $DFSLIP will be used as the value of the parameter s by default; if the parameter s is specified, the value of the parameter component will be updated to the component corresponding to $DFSLIP In simplified programming, s can be replaced by the slip percentage parameter sp. See the application example for the format. Note: ■ When perdis> = 0, other components will be ignored; ■ When jdis and ejdis are specified simultaneously, the component that is closer to the target point will be selected; ■ When the above components are all 0, it represents non-slip, but the interpolation point does not necessarily coincide with the target point; ■ When the above components are all less than 0, it represents non-slip, and the target point will definitely have the interpolation point, ie orientation
sp	Data type: double
	It specifies the slip percentage. It can replace the slip parameter s, and can be used in the scenarios where it is unnecessary to specify the slip size precisely. The format is in sp:10%, which indicates that the slip distance of the target point of the current line is 10% of the max slip distance.
sl	Data type:： double
	This parameter specifies the smoothing distance. It can replace the smoothing parameter s and is used in situations where the smoothing value needs to be precisely specified. The format is sl:10mm, which means that the smoothing distance of the current line is 10mm
t	Data type: tool structure
	It specifies the tool. The user must customize the tool coordinate system on the tool that is installed at the end of the flange. The tool coordinate system is fixed to the tool. The target point p specifies the position of the tool coordinate system in the coordinate system that is specified by the parameter w. Here if the t parameter is defaulted, the system variable $DFTOOL will be used as the value of the parameter t by default; if the parameter t is specified, the value of the parameter component will be updated to the component corresponding to $DFTOOL. The parameter is to ensure that the TCP speed is not greater than 250mm/s when running the program at T1.
w	Data type: wobj structure
	It specifies the workpiece coordinate system. The user can customize the workpiece coordinate system. The target point p specifies the pose of the tool coordinate system in the coordinate system that is specified by the parameter w. In some cases, it is convenient for the user to program in a specific coordinate system. In addition, when the workpiece moves, you only need to re-calibrate the workpiece coordinate system without modifying the user program. Here if the parameter w is defaulted, the system variable $DFWOBJ will be used as the value of the parameter w by default; if the parameter w is specified, the value of the parameter component will be updated to the component corresponding to $DFWOBJ.
dura	Data type: double
	It specifies the trajectory duration. The user can directly specify the movement duration rather than the movement speed. If the user specifies the parameter dura, the system will ignore the parameter speed, and will adjust the speed automatically to meet the time requirements specified by the parameter dura. Dura parameter, in s

For details about above structure data, please refer to *Section 2*.

Example

```
pose p = {x 1500,y 500,z 500,a 0,b 90,c 0,cfg 0}
//the velocity and slip size are specified in a fuzzy manner
```

```
ptp p:p1,vp:5%,sp:5%
ptp p,v:{per 10}
ptp p:{x 1000,y 500,z 500,a 0,b 90,c 0}
ptp p,dura:10
tool t = {{x 0,y 0,z 10,a 0,b 0,c 0}}
wobj w = {{x 1000,y 0,z 500,a 0,b 10,c 0}}
speed v = {per 10,tcp 50,ori 5,exj 10}
pose p2 = {x 10,y 10,z 10,a 0,b 90,c 0,cfg 0}
ptp p2,t,w
```

3.2.3 lin(Linear motion)

Description

The lin instruction is used to move the robot TCP along a linear path to the target point pose; the position movement and pose rotation are synchronized.

Format

lin p:,[v: | vl: | vp:],[s: | sl: | sp:],[t:],[w:],[dura:]

Parameter

The parameters of the lin instruction are shown in Table3-3.

Table3-3 Parameters of lin instruction

Name	Description
p	Data type: pose
	It specifies the TCP target pose of the robot and the target point position of the external axis. Among them, the structure component with the value of 9e+09 will remain at the starting position. The parameter cfg will be ignored and be consistent with the parameter cfg of the starting point. The default value of the turn component is -1.
v	Data type: speed
	It specifies the movement speed. The lin instruction uses the components tcp, ori and exj in the speed structure to specify the TCP movement speed, TCP pose change speed, and movement speed of external axis respectively. If the parameter v is defaulted here, the system variable $DFSPEED will be used as the value of the parameter v by default; if the parameter v is specified, the value of the parameter component will be updated to the component corresponding to $DFSPEED. In simplified programming, v can be replaced by the velocity percentage parameter vp or velocity absolute value parameter vl
vp	Data type: double
	It specifies the percentage of movement speed. It can replace the speed parameter v, and can be used in the scenarios where it is unnecessary to specify the speed precisely. The format is in vp:10%, which indicates that the speed of the current line is 10% of the max speed of the robot.
vl	Data type: double
	It specifies the linear velocity of the movement. It can replace the speed parameter v, and can be used in the scenarios where it is necessary to specify the velocity size precisely. The format is in vl:200mm/s, which indicates that the max TCP speed of the current line is 200mm/s

Name	Description
s	Data type: slip
	It specifies the slip distance. The lin instruction uses the components perdis, pdis, odis and ejdis of the slip structure to describe the point where the original trajectory is disengaged and the slip trajectory is entered. Among them, perdis is used to specify the percentage of the distance from the target point, pdis is used to specify the axis path from the target point, odis is used to specify the pose angle from the target point, and ejdis is used to specify the position angle of external axis from the target point. Here if the parameter s is defaulted, the system variable $DFSLIP will be used as the value of the parameter s by default; if the parameter s is specified, the value of the parameter component will be updated to the component corresponding to $DFSLIP In simplified programming, s can be replaced by slip percentage parameter sp or slip absolute value parameter sl. See the application example for the format. Note: ■ When perdis> = 0, other components will be ignored; ■ When pdis, odis and ejdis are specified simultaneously, the component that is closer to the target point will be selected; ■ When the above components are all 0, it represents non-slip, but the interpolation point does not necessarily coincide with the target point; ■ When the above components are all less than 0, it indicates non-slip, and the target point will definitely have the interpolation point, ie orientation; ■ When the Cartesian space trajectory (ie, LIN, CIR & HELIX) slip is used, it is recommended to use pdis slip, which can make the slip trajectory even on the forward and backward trajectories.
sp	Data type: double
	It specifies the slip percentage. It can replace the slip parameter s, and can be used in the scenarios where it is unnecessary to specify the slip size precisely. The format is in sp:10%, which indicates that the slip distance of the target point of the current line is 10% of the max slip distance.
sl	Data type: double
	It specifies the slip distance. It can replace the slip parameter s, and can be used in the scenarios where it is necessary to specify the slip size precisely. The format is in sl:10mm, which indicates that the slip distance of the current line is 10mm
t	Data type: tool structure
	It specifies the tool. The user must customize the tool coordinate system on the tool that is installed at the end of the flange. The tool coordinate system is fixed to the tool. The target point p specifies the position of the tool coordinate system in the coordinate system that is specified by the parameter w. Here if the t parameter is defaulted, the system variable $DFTOOL will be used as the value of the parameter t by default; if the parameter t is specified, the value of the parameter component will be updated to the component corresponding to $DFTOOL. The parameter is to ensure that the TCP speed is not greater than 250mm/s when running the program at T1.
w	Data type: wobj structure
	It specifies the workpiece coordinate system. The user can customize the workpiece coordinate system. The target point p specifies the pose of the tool coordinate system in the coordinate system that is specified by the parameter w. In some cases, it is convenient for the user to program in a specific coordinate system. In addition, when the workpiece moves, you only need to re-calibrate the workpiece coordinate system without modifying the user program. Here if the parameter w is defaulted, the system variable $DFWOBJ will be used as the value of the parameter w by default; if the parameter w is specified, the value of the parameter component will be updated to the component corresponding to $DFWOBJ.
dura	Data type: double
	It specifies the trajectory duration. The user can directly specify the movement duration rather than the movement speed. If the user specifies the parameter dura, the system will ignore the parameter speed, and will adjust the speed

Name	Description
	automatically to meet the time requirements specified by the parameter dura. Dura parameter, in s

For details about above structure data, please refer to Section 2.

Example

```
pose p = {x 1500,y 500,z 500,a 0,b 90,c 0,cfg 0}
ptp p,v:{per 10}
//the velocity and slip size are specified in a fuzzy manner
lin p:p1,vp:5%,sp:5%
//the speed and slip distance are specified precisely
lin p:p1, vl:100mm/s,sl:5mm
lin p:{x 1000,y 500,z 500,a 0,b 90,c 0}
lin p,dura:10
//the velocity and slip structures are specified precisely
tool t = {{x 0,y 0,z 10,a 0,b 0,c 0}}
wobj w = {{x 1000,y 0,z 500,a 0,b 10,c 0}}
speed v = {per 10,tcp 50,ori 5,exj 10}
slip s = { pdis 10, ejdis 10, odis 10 }
pose p2 = {x 10,y 10,z 10,a 0,b 90,c 0,cfg 0}
lin p2,t,w,v,s
```

3.2.4 spl(Spline motion)

Description

The spl instruction makes the robot go through the teach point smoothly and without pause.


Notice

If the distance between the two teaching points is too small, it is easy to cause a large deviation of the spl track. In this case, please add teaching points appropriately according to the alarm information during operation to ensure that the trajectory is as expected.

Format

```
spl p: , [ v: | vp: ],[ s: | sl: | sp: ],[ t: ],[ w: ]
```

Parameter

The parameters of the spl instruction are shown in Table 3-4.

Table 3-4 Parameters of the spl instruction

Name	description
p	Data type: pose
	This parameter specifies the robot TCP target pose and the target point position of the external axis. Structure components with a value of 9e+09 in the parameter will keep the starting position unchanged. The cfg parameter is ignored and is consistent with the starting point cfg parameter. The default value of the turn component is -1
v	Data type: speed

Name	description
	This parameter specifies the motion speed. The lin instruction uses the tcp, ori, and exj components in the speed structure, which are used to specify the TCP point movement speed, the TCP attitude change speed, and the external axis movement speed, respectively. Here, if the v parameter is defaulted, the system variable $DFSPEED is used as the value of the v parameter by default; if the v parameter is specified, the value of the member component will be updated to the component corresponding to $DFSPEED In simplified programming, v can be replaced by the speed percentage parameter vp or the speed absolute value parameter vl
vp	Data type: double
	This parameter specifies the movement speed percentage. It can be used as an alternative to the speed parameter v, which is used for occasions where it is not necessary to specify the speed value precisely. The format is vp:10%, indicating that the current line speed is 10% of the maximum speed of the robot
s	Data type: slip
	This parameter specifies the smoothing distance. The lin instruction uses the perdis, pdis, odis, and ejdis components of the slip structure to describe the points that leave the original trajectory and enter the smooth trajectory. Among them, perdis is used to specify the percentage of the distance from the target point, pdis is used to specify the path distance from the target point axis, odis is used to specify the attitude angle from the target point, and ejdis is used to specify the position angle of the external axis from the target point axis. Here, if the s parameter is defaulted, the system variable $DFSLIP is used by default as the value of the s parameter; if the s parameter is specified, the value of the member component will be updated to the component corresponding to $DFSLIP In simplified programming, s can be replaced by the smoothing percentage parameter sp or the smoothing absolute value parameter sl. See the usage example for the format. Notice: ■ When perdis>=0, other components will be ignored; ■ When pdis, odis and ejdis are specified at the same time, the component closer to the target point will be selected; ■ When the above components are all 0, it means that it is not smooth, but the interpolation point does not necessarily coincide with the target point; ■ When the above components are all less than 0, it means that it is not smooth, and the target point will definitely have an interpolation point, that is, exact stop; When using Cartesian space trajectory (ie lin, cir) smoothing, it is recommended to use pdis smoothing, which can make the smooth trajectory uniform on the front and rear trajectories
sl	Data type: double
	This parameter specifies the smoothing distance, which can replace the smoothing parameter s, and is used when the smoothing value needs to be precisely specified. The format is sl:10mm, indicating that the smoothing distance of the current line is 10mm
sp	Data type: double
	This parameter specifies the smoothing percentage, which can replace the smoothing parameter s and is used in the occasions where the smoothing value does not need to be specified precisely. The format is sp:10%, indicating that the smoothing distance of the current line target point is 10% of the maximum smoothing distance
t	Data type: tool structure
	This parameter specifies the tool. The user needs to customize the tool coordinate system on the tool installed at the end of the flange. The tool coordinate system is fixed to the tool. The target point p specifies the pose of the tool coordinate system in the coordinate system specified by the w parameter. Here, if the t parameter is defaulted, the system variable $DFTOOL is used as the value of the t parameter by default; if the t parameter is

Name	description
	specified, the value of the member component will be updated to the component corresponding to $DFTOOL This parameter is to ensure that the TCP speed is limited to not more than 250mm/s when the program is manually run at low speed.
w	Data type: wobj structure
	This parameter specifies the workpiece coordinate system. The user can customize the workpiece coordinate system. The target point p specifies the pose of the tool coordinate system in the coordinate system specified by the w parameter. In some cases, it may be more convenient for the user to program in a specific coordinate system. In addition, when the workpiece moves, it is only necessary to re-calibrate the workpiece coordinate system without modifying the user program. Here, if the w parameter is defaulted, the system variable $DFWOBJ is used as the value of the w parameter by default; if the w parameter is specified, the value of the member component will be updated to the component corresponding to $DFWOBJ

Example

Sample program:

```
lin P:P1
lin P:P2
spl P:P3
spl P:P4
spl P:P5
spl P:P6
lin P:P7
```

The program runs as shown in Figure 3-1, moving smoothly from program point 2 to program point 6.

- Run to program point 3 along the path formed by the teaching points of program points 2, 3 and 4.
- Along the trajectory of interval 3 and 4 formed by the teaching points of program points 2, 3 and 4, and the trajectory of interval 3 and 4 formed by the teaching points of program points 3, 4 and 5, the synthesized trajectory runs to Program point 4.
- Along the trajectory of interval 4 and 5 formed by the teaching points of program points 3, 4, and 5, and the trajectory of interval 4 and 5 formed by the teaching points of program points 4, 5, and 6, the synthesized trajectory runs to Program point 5.
- Run to program point 6 along the path formed by the teaching points of program points 4, 5 and 6.

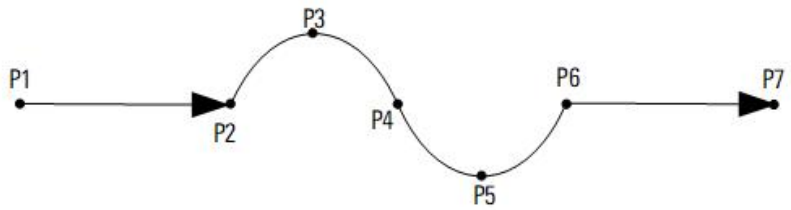


Figure 3-1 Diagram of program execution

3.2.5 cir(Circular movement)

Description

The cir instruction is used to move the TCP of the robot along the circular path to the target point; the translation movement and rotation movement are synchronized.

Arc Path Coordinate System

The arc path coordinate system is shown in Figure 3-2 below. The X-axis is the tangent direction of the path, the Y-axis points to the center of the circle, and the Z-axis direction is determined by the right-hand rule.

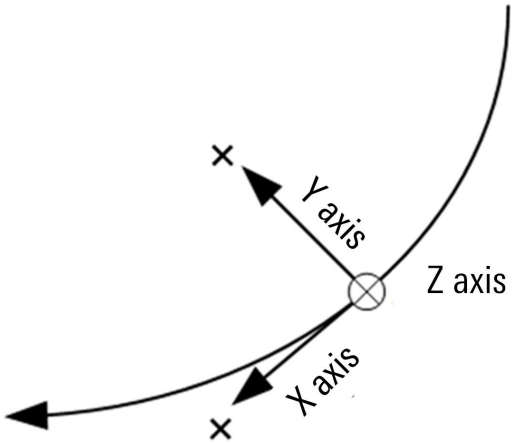


Figure 3-2 Diagram of the arc path coordinate system

Format

cir m:,p:,[v: | vl: | vp:],[s: | sl: | sp:],[t:],[w:],[CA:],[dura:]

Parameter

The parameters of the cir instruction are shown in Table3-5.

Table3-5 Parameters of cir instruction

Name	Description
m	Data type: pose
	It specifies the circular auxiliary point. Starting point, auxiliary point and target point work together to define a circle uniquely. Only the components x, y and z of the auxiliary point are used. Among them, the component with the value of 9e+09 will remain at the position value of the starting point, and the other components will be ignored.
p	Data type: pose
	It specifies the TCP target pose of the robot and the target point position of the external axis. Among them, the structure component with the value of 9e+09 will remain at the starting position. The parameter cfg will be ignored and be consistent with the parameter cfg of the starting point. The default value of the turn component is -1.
v	Data type: speed
	It specifies the movement speed. The cir instruction uses the components tcp, ori and exj in the speed structure to specify the TCP movement speed, TCP pose change speed, and movement speed of external axis respectively. If the parameter v is defaulted here, the system variable $DFSPEED will be used as the value of the parameter v by default; if the parameter v is specified, the value of the parameter component will be updated to the component corresponding to $DFSPEED. In simplified programming, v can be replaced by the velocity percentage parameter vp or velocity absolute value parameter vl. See the application example for the format
vp	Data type: double
	It specifies the percentage of movement speed. It can replace the speed parameter v, and can be used in the scenarios where it is unnecessary to specify the speed precisely. The format is in vp:10%, which indicates that the

Name	Description
	speed of the current line is 10% of the max speed of the robot.
vl	Data type: double
	It specifies the linear velocity of the movement. It can replace the speed parameter v, and can be used in the scenarios where it is necessary to specify the velocity size precisely. The format is in vl:500mm/s, which indicates that the max TCP speed of the current line is 500mm/s
s	Data type: slip
	It specifies the slip distance. The lin instruction uses the components perdis, pdis, odis and ejdis of the slip structure to describe the point where the original trajectory is disengaged and the slip trajectory is entered. Among them, perdis is used to specify the percentage of the distance from the target point, pdis is used to specify the axis path from the target point, odis is used to specify the pose angle from the target point, and ejdis is used to specify the position angle of external axis from the target point. Here if the parameter s is defaulted, the system variable $DFSLIP will be used as the value of the parameter s by default; if the parameter s is specified, the value of the parameter component will be updated to the component corresponding to $DFSLIP In simplified programming, s can be replaced by slip percentage parameter sp or slip absolute value parameter sl. See the application example for the format. Note: ■ When perdis> = 0, other components will be ignored; ■ When pdis, odis and ejdis are specified simultaneously, the component that is closer to the target point will be selected; ■ When the above components are all 0, it represents non-slip, but the interpolation point does not necessarily coincide with the target point; ■ When the above components are all less than 0, it indicates non-slip, and the target point will definitely have the interpolation point, ie orientation; ■ When the Cartesian space trajectory (ie, LIN, CIR & HELIX) slip is used, it is recommended to use pdis slip, which can make the slip trajectory even on the forward and backward trajectories.
sp	Data type: double
	It specifies the slip percentage. It can replace the slip parameter s, and can be used in the scenarios where it is unnecessary to specify the slip size precisely. The format is in sp:10%, which indicates that the slip distance of the target point of the current line is 10% of the max slip distance.
sl	Data type: double
	It specifies the slip distance. It can replace the slip parameter s, and can be used in the scenarios where it is necessary to specify the slip size precisely. The format is in sl:10mm, which indicates that the slip distance of the current line is 10mm.
w	Data type: wobj structure
	It specifies the workpiece coordinate system. The user can customize the workpiece coordinate system. The target point p specifies the pose of the tool coordinate system in the coordinate system that is specified by the parameter w. In some cases, it is convenient for the user to program in a specific coordinate system. In addition, when the workpiece moves, you only need to re-calibrate the workpiece coordinate system without modifying the user program. Here if the parameter w is defaulted, the system variable $DFWOBJ will be used as the value of the parameter w by default; if the parameter w is specified, the value of the parameter component will be updated to the component corresponding to $DFWOBJ.
CA	Data type: double
	It specifies the circle angle. The user can, instead of specifying the target point directly, specify the target point by specifying the circle angle at which the circle rotates. If the user specifies the parameter CA, the parameter p will be only used to determine the geometry of the circle together with the auxiliary point, rather than the real target point. The system will automatically calculate the real target point with the help of the user-specified circle angle. CA

Name	Description
	parameter, in deg
dura	Data type: double
	It specifies the trajectory duration. The user can directly specify the movement duration rather than the movement speed. If the user specifies the parameter dura, the system will ignore the parameter speed, and will adjust the speed automatically to meet the time requirements specified by the parameter dura. Dura parameter, in s

For details about above structure data, please refer to *Section 2*.

Example

```
pose p3={x 1000,y 200,z 500,a 90,b 0,c 90}
pose p4={x 800,y 0,z 500,a 90,b 0,c 90}
tool tool1 = {{x 0,y 10,z 15,a 0,b 90,c 0}}
cir p3,p4,tool1
//The circle angle is specified as 360 degrees
cir p3,p4,CA:360
//The velocity and slip size are specified in a fuzzy manner
cir m:p1,p:p2,vp:5%,sp:5%
//The speed and slip distance are specified precisely
cir m:p3,p:p4,vl:500mm/s,sl:5mm
//The velocity and slip structures are specified precisely
tool t1 = {{x 0,y 0,z 10,a 0,b 0,c 0}}
wobj w1 = {{x 1000,y 0,z 500,a 0,b 10,c 0}}
speed v2 = {per 10,tcp 50,ori 5,exj 10}
slip s = { pdis 10, ejdis 10, odis 10 }
cir p3,p4,t1,w1,v2, s
```

- In single step or segment debugging mode, the cir instruction is executed in two steps, the first step runs to the auxiliary point, and the second step runs to the target point.
- In the program debugger, when executing <jump> to the cir instruction, the robot will move to the target point by following the lin instruction, instead of following the arc trajectory.Although the user can simplify the program using the parameter default, it is still recommended that the user write the parameters as much as possible when writing the movement instruction to prevent the actual movement from being inconsistent with the user's expectations due to the programmer's negligence, even causing unnecessary loss or injury.

3.2.6 ccir(Continuous circular motion)

Description

In the cir instruction, the user needs to teach the two positions of the passing point and the end point. In the ccir instruction, only one point needs to be taught, but at least two ccir instructions need to be taught continuously to determine the arc path.

Compared with the cir instruction, the ccir instruction has the following characteristics:

- Speed and smoothing can be specified respectively at the passing point and end point of the arc action.
- Logic instructions can be taught between the passing point and the end point. However, the logic instructions that can be taught are limited.
- A ccir can be inserted between two ccirs to fine-tune the arc trajectory.

When the following situations occur, the arc cannot be created and the system reports "[12002][0] illegal arc plane".

- When the number of arc points created is less than 3, the arc cannot be formed.
- When the 3 points on the created arc form a straight line, the arc cannot be created.
- When consecutive identical points appear in the ccir instruction, an arc cannot be created.
- It is not allowed to stop the forward look between ccir instructions, for example: when there is waittime 0 between ccir instructions, an illegal arc plane will be alarmed

If the distance between the two teaching points is too short, it is easy to cause a large deviation of the ccir track. In this case, please add teaching points appropriately according to the alarm information during operation to ensure that the trajectory is as expected.

Format

ccir p:,[v: | vl: | vp:],[s: | sl: | sp:],[t:],[w:]

Parameter

The parameters of the ccir instruction are shown in Table 3-6.

Table 3-6 ccir instruction parameters

Name	Description
p	Data type: pose
	This parameter specifies the target pose of the robot TCP and the target point position of the external axis. The structural component with the value of 9e+09 in the parameter will keep the starting position unchanged. The cfg parameter is ignored and is consistent with the starting point cfg parameter. The default value of the turn component is -1
v	Data type: speed
	This parameter specifies the movement speed. The lin instruction uses the tcp, ori, and exj components in the speed structure to specify the TCP point motion speed, TCP attitude change speed, and external axis motion speed, respectively. Here if the v parameter is defaulted, the system variable $DFSPEED is used as the value of the v parameter by default; if the v parameter is specified, the value of the member component will be updated to the component corresponding to $DFSPEED In simplified programming, v can be replaced by speed percentage parameter vp or speed absolute value parameter vl
vp	Data type: double
	This parameter specifies the movement speed percentage. It can replace the speed parameter v, which is used in occasions that do not need to specify the speed accurately. The format is vp:10%, which means that the speed of the currently executed instruction is 10% of the maximum speed of the robot
vl	Data type: double
	This parameter specifies the linear velocity of motion. It can replace the speed parameter v, which is used in occasions where the speed needs to be precisely specified. The format is vl:200mm/s, which means the maximum TCP speed of the current line is 200mm/s
s	Data type: slip
	This parameter specifies the smoothing distance. The LIN instruction uses the perdis, pdis, odis, and ejdis components of the slip structure to describe points that leave the original trajectory and enter the smooth trajectory. Among them, perdis is used to specify the percentage of distance from the target point, pdis is used to

Name	Description
	specify the axis path distance from the target point, odis is used to specify the attitude angle from the target point, and ejdis is used to specify the position angle of the external axis from the target point. Here, if the default s parameter is used, the system variable $DFSLIP is used as the value of the s parameter by default; if the s parameter is specified, the value of the member component will be updated to the component corresponding to $DFSLIP. In simplified programming, s can be replaced by the smoothing percentage parameter sp or the smoothing absolute value parameter sl. See usage examples for the format. Note: ■ When perdis>=0, other components will be ignored; ■ When pdis, odis and ejdis are specified at the same time, the component closer to the target point will be selected; ■ When the above components are all 0, it means that it is not smooth, but the interpolation point does not necessarily coincide with the target point; ■ When the above components are all less than 0, it means that it is not smooth, and the target point will definitely have an interpolation point, that is, exact stop; ■ When using Cartesian space trajectory (LIN, CIR) smoothing, it is recommended to use pdis smoothing, which can make the smooth trajectory uniform on the front and back trajectories
sp	Data type: double
	This parameter specifies the smoothing percentage. It can replace the smoothing parameter s, and is used in situations where the smoothing value does not need to be precisely specified. The format is sp:10%, which means that the smoothing distance of the target point of the current execution instruction is 10% of the maximum smoothing distance
sl	Data type: double
	This parameter specifies the smoothing distance. It can replace the smoothing parameter s and is used in situations where the smoothing value needs to be precisely specified. The format is sl:10mm, which means that the smoothing distance of the currently executed instruction is 10mm
t	Data type: tool structure
	This parameter specifies the tool. The user needs to customize the tool coordinate system on the tool installed at the end of the flange. The tool coordinate system is fixedly connected to the tool. The target point p specifies the pose of the tool coordinate system in the coordinate system specified by the w parameter. Here, if the t parameter is defaulted, the system variable $DFTOOL is used as the value of the t parameter by default; if the t parameter is specified, the value of the member component will be updated to the component corresponding to $DFTOOL This parameter is to ensure that the TCP speed is not more than 250mm/s when running the program manually at low speed
w	Data type: wobj structure
	This parameter specifies the workpiece coordinate system. The user can customize the workpiece coordinate system. The target point p specifies the pose of the tool coordinate system in the coordinate system specified by the w parameter. In some cases, it may be more convenient for users to program in a specific coordinate system. In addition, when the workpiece moves its position, only need to re-calibrate the workpiece coordinate system without modifying the user program. If the w parameter is defaulted, the system variable $DFWOBJ is used as the value of the w parameter by default; if the w parameter is specified, the value of the member component will be updated to the corresponding component of $DFWOBJ

Please refer to *Chapter 2* for details of the above structure data.

Notice

In the program debugger, when executing the action of <jump> to ccir instruction, the robot moves to the target point by following the lin instruction, instead of following the arc trajectory.

Example

Example 1: Instruction method

Program example:

```
lin P:P1
lin P:P2
ccir P:P3
ccir P:P4
ccir P:P5
lin P:P6
```

The schematic diagram of program execution is shown in Figure 3-3.

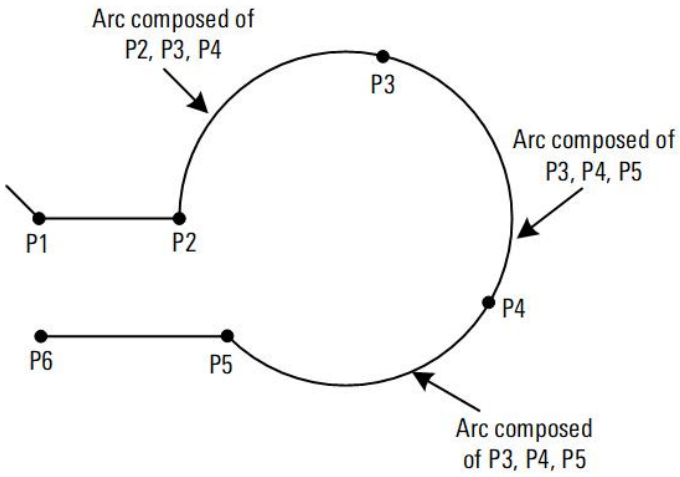


Figure 3-3 Program execution diagram

The initial lin instruction is linear motion, as shown in Figure 3-4(a). When the first ccir instruction is executed, the robot moves on a circle composed of three points: the current position, the target position of the instruction, and the target position of the next ccir (as shown in Figure 3-4(b)). When the program executes until the next action instruction is not a ccir motion instruction, the target point of the last ccir motion instruction is regarded as the end point of the arc. When moving toward the end point, the robot moves on a circle of 3 points that pass the target position of the previous ccir instruction, the current position, and the target position of the instruction (as shown in Figure 3-4(c)).

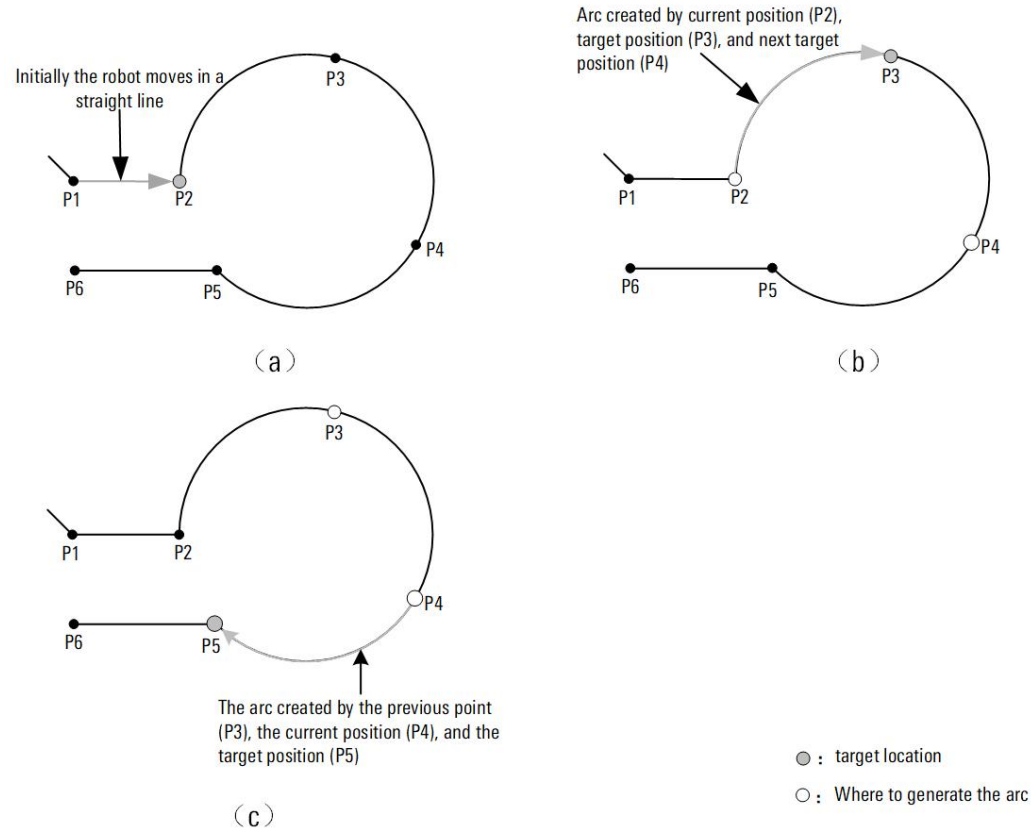


Figure 3-4 Schematic diagram of program distribution execution

Example 2: Judgment of arc direction

Program example:

lin P:P1

lin P:P2

ccir P:P3

ccir P:P4

When executing the third line of the above program, the robot moves toward P3 through the arc of 3 points through P2, P3, and P4. The path toward P3 through this arc is shown in Figure 3-5. The robot moves along the arc from P2→P3→P4 to P3.

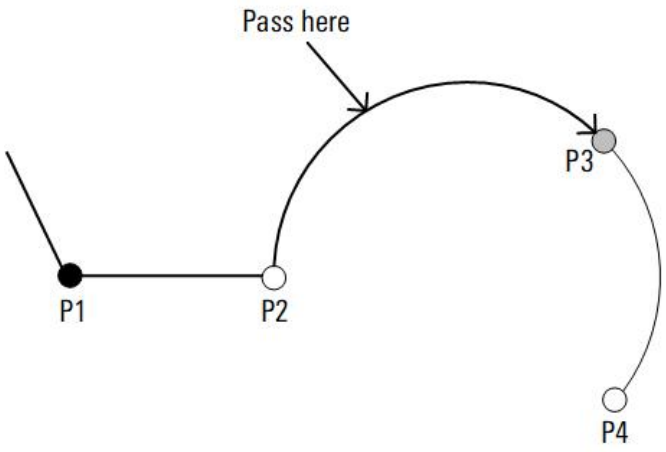


Figure 3-5 Schematic diagram of arc direction judgment program

Example 3: Pause and jog before continuing to move

The robot pauses in the middle of the ccir instruction. When it continues to move in this state, it will continue to move along the previous motion track. If the robot pauses and jogs during the execution of the ccir instruction, when it continues to move, the robot returns to the paused position in a straight line, and then continues to move according to the previous path.

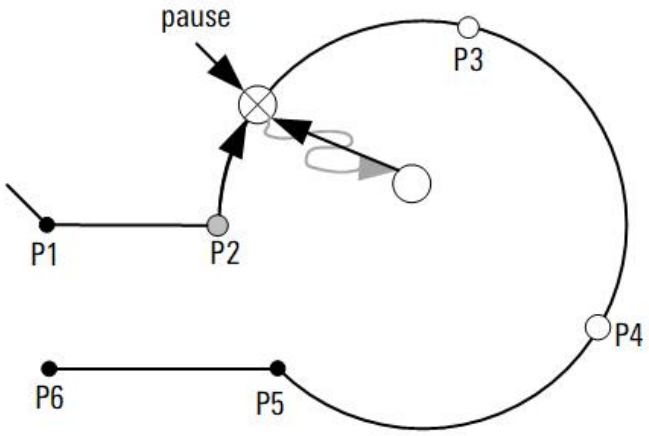


Figure 3-6 Pause and jog before continuing to move the diagram

Example 4: Correction of the next target point

When P4 in Example 1 is changed, the robot continues to move along the arc that passes through the corrected P4.

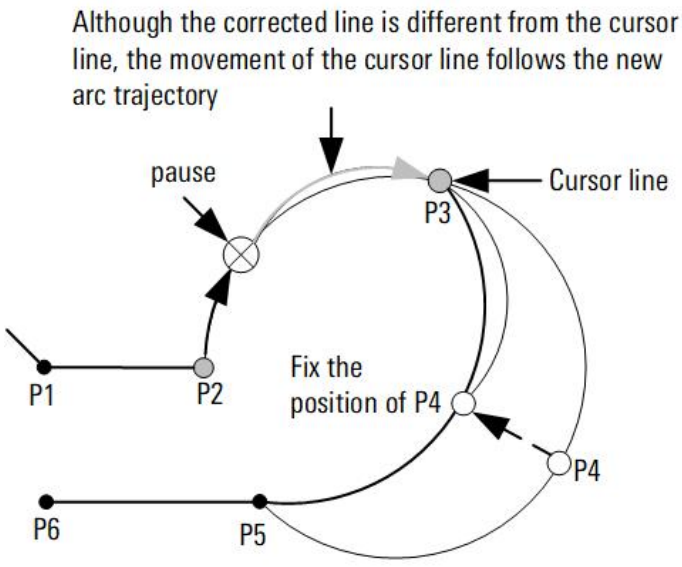


Figure 3-7 Correction diagram of the next target point

Example 5: Jump after pause

Figure 3-8 shows an example of programming to pause halfway toward P2 and continue moving from P3. The robot jumps straight to P3 from the pause position, and then moves in the order of P3→P4→P5.

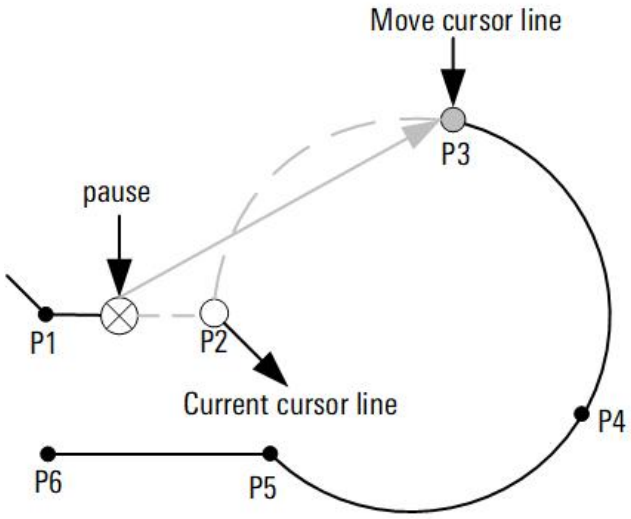


Figure 3-8 Schematic diagram of jump after pause

3.2.7 Superimposed swing instructions

3.2.7.1 startweave(Turn on superimposed swing)

Description

Use the startweave instruction to turn on the superimposed swing.

Format

startweave weave:

Parameter

The parameters of startweave instruction are shown in Table 3-7.

Table 3-7 Parameters of startweavelin instruction

Name	Description
weave	Data type: weavedata
	The parameters that are used to specify the swing trajectory, including: ■ weave_type ■ frequency ■ amplitude ■ dwell_left ■ dwell_right ■ dwell_middle ■ swing_angle ■ radius ■ axis ■ rotation_angle For detailed description see *Section 2.4.8*.

Tip

- "axis" (deflection axis) can be defaulted.
- "rotation_angle" (deflection angle) can be defaulted.
- When the amplitude is large, an overspeed warning may occur. At this time, try to reduce the expected speed or amplitude of the trajectory to avoid the warning.

Example

Yaw:

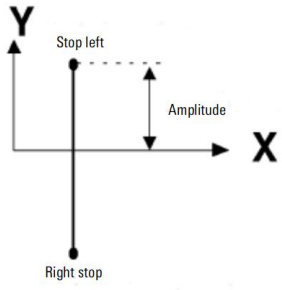
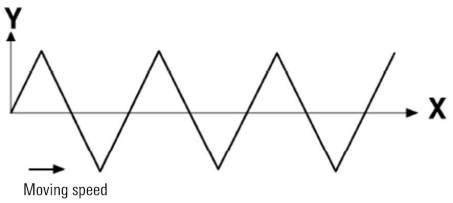
```
weavedata weavedata1
startweave weavedata1={simple,10,15,0,45}
tool t = {{x 0,y 0,z 10,a 0,b 0,c 0}}
wobj w = {{x 1000,y 0,z 500,a 0,b 10,c 0}}
speed v = {per 10,tcp 50,ori 5,exj 10}
pose p2 = {x 10,y 10,z 10,a 0,b 90,c 0,cfg 0}
lin p2,t,w,v
endweave
```

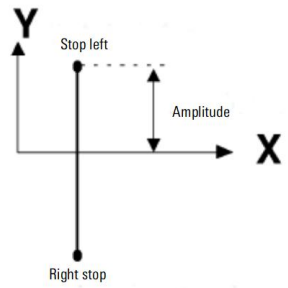
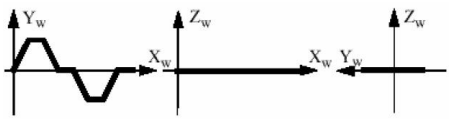
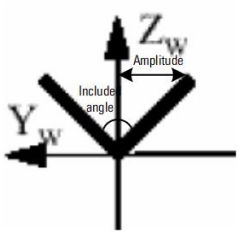
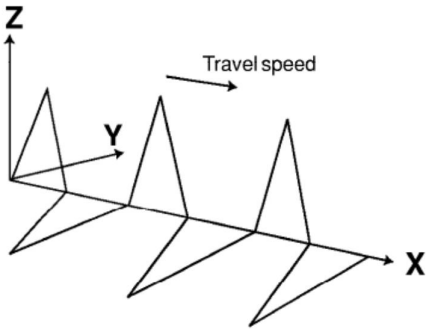
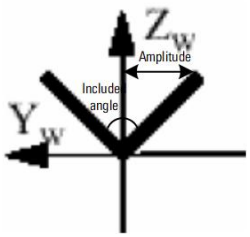
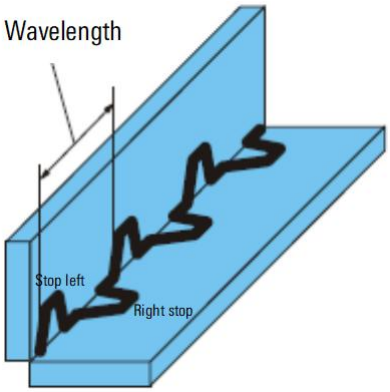
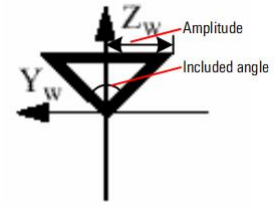
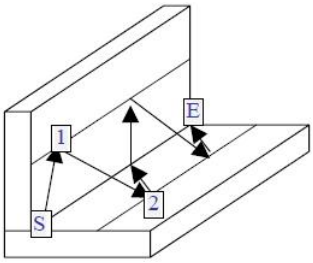
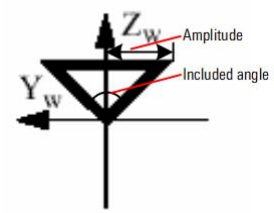
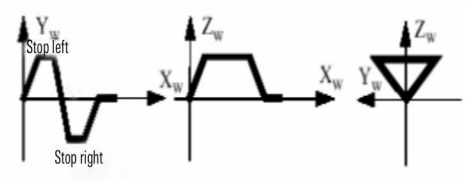
Triangle swing:

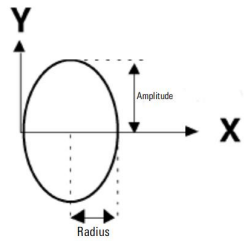
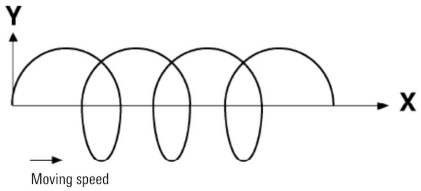
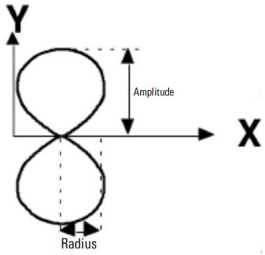
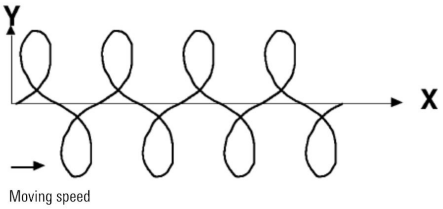
```
weavedata weavedata1 = {V_shape,2,15,90,1,1, 0,1}
startweavespa weavedata1
tool t = {{x 0,y 0,z 10,a 0,b 0,c 0}}
wobj w = {{x 1000,y 0,z 500,a 0,b 10,c 0}}
speed v = {per 10,tcp 50,ori 5,exj 10}
pose p2 = {x 10,y 10,z 10,a 0,b 90,c 0,cfg 0}
lin p2,t,w,v
endweave
```

The superimposed trajectory swing needs to support the adjustment of the graphics parameters of the superimposed pattern. Refer to Table 3-8 for details.

Table 3-8 Graphical parameters of superimposed trajectory

Name	Superimposed pattern	Combination pattern	Graphics parameters	Remarks
Yaw			Amplitude	

Name	Superimposed pattern	Combination pattern	Graphics parameters	Remarks
Linear zigzag swing			Amplitude Left stay length/left stay time Right stay length/right stay time Intermediate stay length/intermediate stay time	If it is based on the frequency cycle, the dwell time is adopted, if it is based on the wavelength, the dwell length is adopted
V-shaped swing			Amplitude Included angle	When the included angle is 0, the swing direction coincides with the z axis
V-shaped sawtooth swing			Amplitude Included angle Left stay length/left stay time Right stay length/right stay time Intermediate stay length/intermediate stay time	If it is based on the frequency cycle, the dwell time is adopted, if it is based on the wavelength, the dwell length is adopted
Spatial triangle swing			Amplitude Included angle	-
Spatial triangle sawtooth swing			Amplitude Included angle Left stay length/left stay time Right stay length/right stay time Intermediate stay length/intermediate	If it is based on the frequency cycle, the dwell time is adopted, if it is based on the wavelength, the dwell

Name	Superimposed pattern	Combination pattern	Graphics parameters	Remarks
			stay time	length is adopted
Spiral curve			Amplitude Radius	-
"8" shape			Amplitude Radius	-

3.2.7.2 endweave(End superimposed swing)

Description

The endweave instruction is used to end the superimposed swing.

Definition

endweave

Example

```
weavedatalin weavedatalin1 = {2,15}
startweavelin weavedatalin1
lin p:{x 1000,y 500,z 500,a 0,b 90,c 0}
endweave
```

3.2.8 Combination instructions

Please refer to our company's "Multi-machine Linkage Operation Manual" for the use of "combined instructions".

3.2.9 Conveyor belt

Please refer to the "Conveyor Belt Tracking Manual" of our company for the usage of the related instructions of "Conveyor Belt".

3.2.10 Soft move

Please refer to our company's "Soft Move Operation Manual" for the use of related instructions of "soft move".

3.2.11 Trajectory compensation

3.2.11.1 startcompen(Start trajectory compensation)

Description

The startcompen instruction is used to enable the trajectory compensation function of the robot.


Tip

Angle compensation is used for trajectory fine-tuning, and the accumulated maximum value should not exceed 90°.

Format

startcompen data:data1

Parameter

The parameters of the startcompen instruction are shown in Table 3-9.

Table 3-9 Parameters of startcompen instruction

Name	Description
data	Data type: compendata
	It specifies max speed, acceleration, jerk, angular velocity, angular acceleration, angular jerk of TCP in the robot trajectory compensation process.

Example

const compendata data3 = {tcp_max_vel 200 ,tcp_max_acc 600 ,tcp_max_jerk 1000 ,ori_max_vel 200 ,ori_max_acc 50 ,ori_max_jerk 1000}

startcompen data:data3

3.2.11.2 compen(Trajectory Compensation)

Description

The compen instruction is used between startcompen and endcompen to compensate the trajectory of the robot.

Format

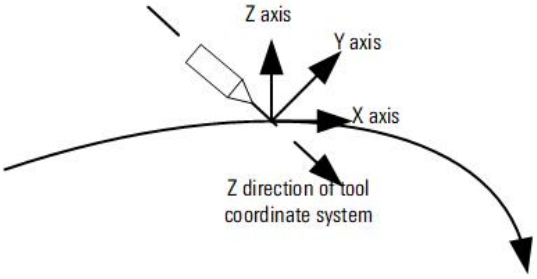
compen [x:],[y:],[z:],[a:],[b:],[c:],type:

Parameter

The parameters of the compen instruction are shown in Table 3-10.

Table 3-10 Parameters of compen instruction

Name	Description
type	Data Type: Enumeration
	You can select TOOL, WOBJ, TOOL_PATH, MODIFY_PATH or WORLD, which represents the coordinate system to which the trajectory compensation amount is referenced. The description of each coordinate system type is as follows:

Name	Description
	■ TOOL：Tool coordinate system ■ WOBJ：Workpiece coordinate system ■ TOOL_PATH：The tool-path coordinate system is defined as shown in the figure below.  The x direction of the coordinate system is the tangent direction of the main track; The y direction of the coordinate system is determined by the cross product of the x direction of the coordinate system and the z direction of the tool coordinate system; The z direction of the coordinate system is determined by the cross product of the x direction of the coordinate system and the y direction of the coordinate system. Tip: If the tangent direction of the main trajectory is parallel to the z direction of the tool coordinate system, the y direction of the coordinate system cannot be obtained, and an alarm is required at this time. ■ WORLD：World coordinate system ■ MODIFY_PATH：In general, MODIFY_PATH and TOOL_PATH are defined in the same way. When there is a turning point in the trajectory, according to the tool-path coordinate system definition, the x-axis will be reversed, and the y-axis will also be reversed. At this time, the y-axis of MODIFY_PATH is not reversed, so as to ensure that when performing compensation in the y-direction, the compensation direction remains unchanged after the return trajectory (a, b, and c are not compensated in this coordinate system)
x	The amount of translation compensation of the TCP along the x-axis of the reference coordinate system
y	The amount of translation compensation of the TCP along the y-axis of the reference coordinate system
z	The amount of translation compensation of the TCP along the z-axis of the reference coordinate system
a	Rotation compensation amount of TCP along the z-axis of the reference coordinate system
b	Rotation compensation amount of TCP along the y-axis of the reference coordinate system
c	Rotation compensation amount of TCP along the x-axis of the reference coordinate system

Example

The example shown in Figure 3-9 is that the robot's trajectory is offset by 100 mm in the X direction of the workpiece coordinate system.

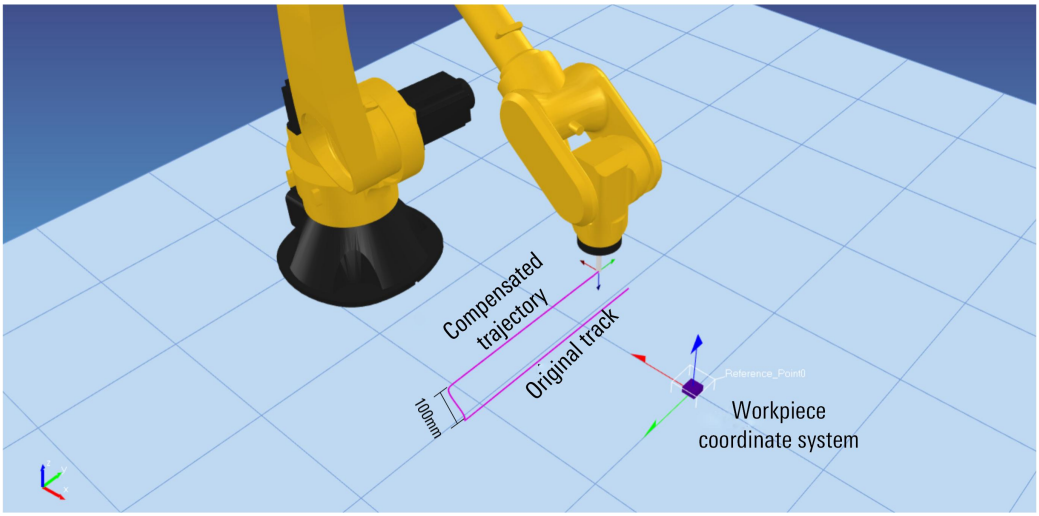


Figure 3-9 Schematic diagram of trajectory compensation

The corresponding program example in the picture is:

const compendata data1 = { tcp_max_vel 250 , tcp_max_acc 900 , tcp_max_jerk 6000 , ori_max_vel 50 , ori_max_acc 75 , ori_max_jerk 375 }

const pose p1 = { x 911.138, y -5.704, z 739.959, a -178.159, b 43.881, c 2.201, cfg 1, turn 000000b, ej1 9.000e+09, ej2 0.000, ej3 9.000e+09, ej4 9.000e+09, ej5 9.000e+09, ej6 9.000e+09 }

startcompen data:data1

compen x:100,y:0,z:0,a:0,b:0,c:0,type:"WOBJ" // It is offset by 100mm from the X direction of the workpiece coordinate system.

lin p:p1,vl:50mm/s,sl:0mm,t:$tool0,w:$WORLD

endcompen

Notice

In single step or segment debugging mode, the trajectory compensation is not effective.

3.2.11.3 endcompen(End trajectory compensation)

Description

The endcompen instruction is used to end the trajectory compensation function of the robot.

Format

endcompen

Example

Refer to *Chapter 3.2.11.2*.

Notice

After endcompen, directly use the compensated point as the starting point to start the next instruction without compensation. If you need to return to the original path first, you can add a lin instruction after endcompen to return to the original path.

3.3 Motion instructions（Suitable for SCARA robot）

3.3.1 movej(Moving axis)

Description

The movej instruction is used to move the robot axis or external axis to a specified axis position. All axes will reach the target axis position simultaneously.

Format

movej j:,[v: | vp:],[s: | sl: | sp:],[dura:]

Parameter

The parameters of the movej instruction are shown in Table 3-11.

Table 3-11 Parameters of movej instruction

Name	Description
j	Data type: joint
	It specifies the target point positions of the robot axes and external axes. Among them, the structure component with the value of 9e+09 will remain at the starting position. If the first movement instruction executed in the program is movej, the target points j1~j4 of the movej instruction and the external axis may not be defaulted.
v	Data type: speed
	It specifies the movement speed. The movej instruction uses the components per and exj in the speed structure variable, which are used to specify the percentage of axis movement speed and the external axis movement speed respectively. If the parameter v is defaulted here, the system variable $DFSPEED will be used as the value of the parameter v by default; if the parameter v is specified, the value of the parameter component will be updated to the component corresponding to $DFSPEED. In simplified programming, v can be replaced by the velocity percentage parameter vp. See the application example for the format.
vp	Data type: double
	It specifies the percentage of movement speed. It can replace the speed parameter v, and can be used in the scenarios where it is unnecessary to specify the speed precisely. The format is in vp:10%, which indicates that the speed of the current line is 10% of the max speed of the robot.
s	Data type: slip
	It specifies the slip distance. The Movej instruction uses the components perdis and ejdis of the slip structure to describe the point where the original trajectory is disengaged and the slip trajectory is entered. Among them, perdis is used to specify the percentage of the distance from the target point, and ejdis is used to specify the position angle of external axis from the target point. Here if the parameter s is defaulted, the system variable $DFSLIP will be used as the value of the parameter s by default; if the parameter s is specified, the value of the parameter component will be updated to the component corresponding to $DFSLIP In simplified programming, s can be replaced by the slip percentage parameter sp. See the application example for the format. Note: ■ When perdis> = 0, other components will be ignored; ■ When ejdis is specified simultaneously, the component that is closer to the target point will be selected; ■ When the above components are all 0, it represents non-slip, but the interpolation point does not necessarily coincide with the target point; ■ When the above components are all less than 0, it represents non-slip, and the target point will definitely have the interpolation point, ie orientation
sp	Data type: double

Name	Description
	It specifies the slip percentage. It can replace the slip parameter s, and can be used in the scenarios where it is unnecessary to specify the slip size precisely. The format is in sp:10%, which indicates that the slip distance of the target point of the current line is 10% of the max slip distance.
sl	Data type: double
	This parameter specifies the smoothing distance. It can replace the smoothing parameter s and is used in situations where the smoothing size needs to be precisely specified. The format is sl:10mm, which means that the smoothing distance of the current line is 10mm
dura	Data type: double
	It specifies the trajectory duration. The user can directly specify the movement duration rather than the movement speed. If the user specifies the parameter dura, the system will ignore the parameter speed, and will adjust the speed automatically to meet the time requirements specified by the parameter dura. Dura parameter, in s

Example

```
//The velocity and slip size are specified in a fuzzy manner
movej j:j1,vp:5%,sp:5%
movej j:{j1 0,j20,j3 0,j4 0, }
movej j:{j1 0}
movej j:{j1 20,ej1 30}
movej j:{ej1 10}
joint p = {0,0,0,0}
speed v = {per 50}
movej p,v
```

3.3.2 ptp(Point to point)

Description

The ptp instruction is used to quickly move the robot from one point to another without any requirements for the shape of the trajectory of the TCP. All axes will reach the target point simultaneously.

Format

ptp p:,[v: | vp:],[s: | sl: | sp:],[t:],[w:],[dura:]

Parameter

The parameters of the ptp instruction are shown in Table 3-12.

Table 3-12 Parameters of ptp instruction

Name	Description
p	Data type: pose
	It specifies the TCP target pose of the robot and the target point position of the external axis. Among them, the structure component with the value of 9e+09 will remain at the starting position. The default value of the component cfg is 0, and the default value of the component turn is -1. If the first movement instruction executed in the program is ptp, the target points X, Y, Z, A, B, C of the ptp instruction and the external axis may not be defaulted.

Name	Description
v	Data type: speed
	It specifies the movement speed. The ptp instruction uses the components per and exj in the speed structure variable, which are used to specify the percentage of axis movement speed and the movement speed of the external axis respectively. If the parameter v is defaulted here, the system variable $DFSPEED will be used as the value of the parameter v by default; if the parameter v is specified, the value of the parameter component will be updated to the component corresponding to $DFSPEED. In simplified programming, v can be replaced by the velocity percentage parameter vp. See the application example for the format.
vp	Data type: double
	It specifies the percentage of movement speed. It can replace the speed parameter v, and can be used in the scenarios where it is unnecessary to specify the speed precisely. The format is in vp:10%, which indicates that the speed of the current line is 10% of the max speed of the robot.
s	Data type: slip
	It specifies the slip distance. The PTP instruction uses the components perdis and ejdis of the slip structure to describe the point where the original trajectory is disengaged and the slip trajectory is entered. Among them, perdis is used to specify the percentage of the distance from the target point, and ejdis is used to specify the position angle of external axis from the target point. Here if the parameter s is defaulted, the system variable $DFSLIP will be used as the value of the parameter s by default; if the parameter s is specified, the value of the parameter component will be updated to the component corresponding to $DFSLIP In simplified programming, s can be replaced by the slip percentage parameter sp. See the application example for the format. Note: ■ When perdis> = 0, other components will be ignored; ■ When ejdis is specified simultaneously, the component that is closer to the target point will be selected; ■ When the above components are all 0, it represents non-slip, but the interpolation point does not necessarily coincide with the target point; ■ When the above components are all less than 0, it represents non-slip, and the target point will definitely have the interpolation point, ie precise stop
sp	Data type: double
	It specifies the slip percentage. It can replace the slip parameter s, and can be used in the scenarios where it is unnecessary to specify the slip value precisely. The format is in sp:10%, which indicates that the slip distance of the target point of the current line is 10% of the max slip distance.
sl	Data type: double
	This parameter specifies the smoothing distance. It can replace the smoothing parameter s and is used in situations where the smoothing value needs to be precisely specified. The format is sl:10mm, which means that the smoothing distance of the current line is 10mm
t	Data type: tool structure
	It specifies the tool. The user must customize the tool coordinate system on the tool that is installed at the end of the flange. The tool coordinate system is fixed to the tool. The target point p specifies the position of the tool coordinate system in the coordinate system that is specified by the parameter w. Here if the t parameter is defaulted, the system variable $DFTOOL will be used as the value of the parameter t by default; if the parameter t is specified, the value of the parameter component will be updated to the component corresponding to $DFTOOL. The parameter is to ensure that the TCP speed is not greater than 250mm/s when running the program at T1.
w	Data type: wobj structure
	It specifies the workpiece coordinate system. The user can customize the workpiece coordinate system. The target

Name	Description
	point p specifies the pose of the tool coordinate system in the coordinate system that is specified by the parameter w. In some cases, it is convenient for the user to program in a specific coordinate system. In addition, when the workpiece moves, you only need to re-calibrate the workpiece coordinate system without modifying the user program. Here if the parameter w is defaulted, the system variable $DFWOBJ will be used as the value of the parameter w by default; if the parameter w is specified, the value of the parameter component will be updated to the component corresponding to $DFWOBJ.
dura	Data type: double
	It specifies the trajectory duration. The user can directly specify the movement duration rather than the movement speed. If the user specifies the parameter dura, the system will ignore the parameter speed, and will adjust the speed automatically to meet the time requirements specified by the parameter dura. Dura parameter, in s

For details about above structure data, please refer to *Section 2*.

Example

```
pose p = {x 266,y 88,z -50,a -34,b 0,c 0,cfg 4}
//The velocity and slip size are specified in a fuzzy manner
ptp p:p1,vp:5%,sp:5%
ptp p,v:{per 10}
ptp p:{x 266,y 88,z -50,a 0,b 0,c 0}
ptp p,dura:10
tool t = {{x 0,y 0,z 0,a 0,b 0,c 0}}
wobj w = {{x 0,y 0,z 500,a 0,b 0,c 0}}
speed v = {per 10,tcp 50,ori 5,exj 10}
pose p2 = {x 266,y 88,z -50,a 0,b 0,c 0,cfg 0}
ptp p2,t,w
```

3.3.3 lin(Linear motion)

Description

The lin instruction is used to move the robot TCP along a linear path to the target point pose; the position movement and pose rotation are synchronized.

Format

lin p:,[v: | vl: | vp:],[s: | sl: | sp:],[t:],[w:],[dura:]

Parameter

The parameters of the lin instruction are shown in Table 3-13.

Table 3-13 Parameters of lin instruction

Name	Description
p	Data type: pose
	It specifies the TCP target pose of the robot and the target point position of the external axis. Among them, the structure component with the value of 9e+09 will remain at the starting position. The parameter cfg will be ignored and be consistent with the parameter cfg of the starting point. The default value of the turn component is -1.

Name	Description
v	Data type: speed
	It specifies the movement speed. The lin instruction uses the components tcp, ori and exj in the speed structure to specify the TCP movement speed, TCP pose change speed, and movement speed of external axis respectively. If the parameter v is defaulted here, the system variable $DFSPEED will be used as the value of the parameter v by default; if the parameter v is specified, the value of the parameter component will be updated to the component corresponding to $DFSPEED. In simplified programming, v can be replaced by speed percentage parameter vp or speed absolute value parameter vl
vp	Data type: double
	It specifies the percentage of movement speed. It can replace the speed parameter v, and can be used in the scenarios where it is unnecessary to specify the speed precisely. The format is in vp:10%, which indicates that the speed of the current line is 10% of the max speed of the robot.
vl	Data type: double
	It specifies the linear velocity of the movement. It can replace the speed parameter v, and can be used in the scenarios where it is necessary to specify the velocity value precisely. The format is in vl:500mm/s, which indicates that the max TCP speed of the current line is 500mm/s
s	Data type: slip
	It specifies the slip distance. The lin instruction uses the components perdis, pdis, odis and ejdis of the slip structure to describe the point where the original trajectory is disengaged and the slip trajectory is entered. Among them, perdis is used to specify the percentage of the distance from the target point, pdis is used to specify the axis path from the target point, odis is used to specify the pose angle from the target point, and ejdis is used to specify the position angle of external axis from the target point. Here if the parameter s is defaulted, the system variable $DFSLIP will be used as the value of the parameter s by default; if the parameter s is specified, the value of the parameter component will be updated to the component corresponding to $DFSLIP In simplified programming, s can be replaced by slip percentage parameter sp or slip absolute value parameter sl. See the application example for the format. Note: ■ When perdis> = 0, other components will be ignored; ■ When pdis, odis and ejdis are specified simultaneously, the component that is closer to the target point will be selected; ■ When the above components are all 0, it represents non-slip, but the interpolation point does not necessarily coincide with the target point; ■ When the above components are all less than 0, it indicates non-slip, and the target point will definitely have the interpolation point, ie precise stop; ■ When the Cartesian space trajectory (ie, LIN, CIR) slip is used, it is recommended to use pdis slip, which can make the slip trajectory even on the forward and backward trajectories.
sp	Data type: double
	It specifies the slip percentage. It can replace the slip parameter s, and can be used in the scenarios where it is unnecessary to specify the slip size precisely. The format is in sp:10%, which indicates that the slip distance of the target point of the current line is 10% of the max slip distance.
sl	Data type: double
	It specifies the slip distance. It can replace the slip parameter s, and can be used in the scenarios where it is necessary to specify the slip size precisely. The format is in sl:10mm, which indicates that the slip distance of the current line is 10mm
t	Data type: tool structure

Name	Description
	It specifies the tool. The user must customize the tool coordinate system on the tool that is installed at the end of the flange. The tool coordinate system is fixed to the tool. The target point p specifies the position of the tool coordinate system in the coordinate system that is specified by the parameter w. Here if the t parameter is defaulted, the system variable $DFTOOL will be used as the value of the parameter t by default; if the parameter t is specified, the value of the parameter component will be updated to the component corresponding to $DFTOOL.
w	Data type: wobj structure
	It specifies the workpiece coordinate system. The user can customize the workpiece coordinate system. The target point p specifies the pose of the tool coordinate system in the coordinate system that is specified by the parameter w. In some cases, it is convenient for the user to program in a specific coordinate system. In addition, when the workpiece moves, you only need to re-calibrate the workpiece coordinate system without modifying the user program. Here if the parameter w is defaulted, the system variable $DFWOBJ will be used as the value of the parameter w by default; if the parameter w is specified, the value of the parameter component will be updated to the component corresponding to $DFWOBJ.
dura	Data type: double
	It specifies the trajectory duration. The user can directly specify the movement duration rather than the movement speed. If the user specifies the parameter dura, the system will ignore the parameter speed, and will adjust the speed automatically to meet the time requirements specified by the parameter dura. Dura parameter, in s

For details about above structure data, please refer to *Section 2*.

Example

```
pose p = {x 266,y 88,z -50,a 0,b 0,c 0,cfg 0}
ptp p,v:{per 10}
//The velocity and slip size are specified in a fuzzy manner.
lin p:p1,vp:5%,sp:5%
//The speed and slip distance are specified precisely.
lin p:p1, vl:100mm/s,sl:5mm
lin p:{x 266,y 88,z -50,a 0,b 90,c 0}
lin p,dura:10
//The velocity and slip structures are specified precisely.
tool t = {{x 0,y 0,z 10,a 0,b 0,c 0}}
wobj w = {{x 0,y 0,z 0,a 0,b 0,c 0}}
speed v = {per 10,tcp 50,ori 5,exj 10}
slip s = { pdis 10, ejdis 10, odis 10 }
pose p2 = {x 266,y 88,z -50,a 0,b 0,c 0,cfg 0}
lin p2,t,w,v,s
```

3.3.4 cir(Circular movement)

Description

The cir instruction is used to move the TCP of the robot along the circular path to the target point; the translation movement and rotation movement are synchronized.

Format

```
cir m:,p:,[ v: ],[ s: ],[ t: ],[ w: ],[ CA: ],[ dura: ]
```

Parameter

The parameters of the cir instruction are shown in Table 3-14.

Table 3-14 Parameters of cir instruction

Name	Description
m	Data type: pose
	It specifies the circular auxiliary point. Starting point, auxiliary point and target point work together to define a circle uniquely. Only the components x, y and z of the auxiliary point are used. Among them, the component with the value of 9e+09 will remain at the position value of the starting point, and the other components will be ignored.
p	Data type: pose
	It specifies the TCP target pose of the robot and the target point position of the external axis. Among them, the structure component with the value of 9e+09 will remain at the starting position. The parameter cfg will be ignored and be consistent with the parameter cfg of the starting point. The default value of the turn component is -1.
v	Data type: speed
	It specifies the movement speed. The cir instruction uses the components tcp, ori and exj in the speed structure to specify the TCP movement speed, TCP pose change speed, and movement speed of external axis respectively. If the parameter v is defaulted here, the system variable $DFSPEED will be used as the value of the parameter v by default; if the parameter v is specified, the value of the parameter component will be updated to the component corresponding to $DFSPEED. In simplified programming, v can be replaced by the velocity percentage parameter vp or velocity absolute value parameter vl. See the application example for the format
vp	Data type: double
	It specifies the percentage of movement speed. It can replace the speed parameter v, and can be used in the scenarios where it is unnecessary to specify the speed precisely. The format is in vp:10%, which indicates that the speed of the current line is 10% of the max speed of the robot.
vl	Data type: double
	It specifies the linear velocity of the movement. It can replace the speed parameter v, and can be used in the scenarios where it is necessary to specify the velocity value precisely. The format is in vl:500mm/s, which indicates that the max TCP speed of the current line is 500mm/s
s	Data type: slip
	It specifies the slip distance. The lin instruction uses the components perdis, pdis, odis and ejdis of the slip structure to describe the point where the original trajectory is disengaged and the slip trajectory is entered. Among them, perdis is used to specify the percentage of the distance from the target point, pdis is used to specify the axis path from the target point, odis is used to specify the pose angle from the target point, and ejdis is used to specify the position angle of external axis from the target point. Here if the parameter s is defaulted, the system variable $DFSLIP will be used as the value of the parameter s by default; if the parameter s is specified, the value of the parameter component will be updated to the component corresponding to $DFSLIP In simplified programming, s can be replaced by slip percentage parameter sp or slip absolute value parameter sl. See the application example for the format. Note: ■ When perdis> = 0, other components will be ignored; ■ When pdis, odis and ejdis are specified simultaneously, the component that is closer to the target point will be selected; ■ When the above components are all 0, it represents non-slip, but the interpolation point does not necessarily coincide with the target point; ■ When the above components are all less than 0, it indicates non-slip, and the target point will definitely have the

Name	Description
	interpolation point, ie precise stop; ■ When the Cartesian space trajectory (ie, LIN, CIR) slip is used, it is recommended to use pdis slip, which can make the slip trajectory even on the forward and backward trajectories.
sp	Data type: double
	It specifies the slip percentage. It can replace the slip parameter s, and can be used in the scenarios where it is unnecessary to specify the slip size precisely. The format is in sp:10%, which indicates that the slip distance of the target point of the current line is 10% of the max slip distance.
sl	Data type: double
	It specifies the slip distance. It can replace the slip parameter s, and can be used in the scenarios where it is necessary to specify the slip size precisely. The format is in sl:10mm, which indicates that the slip distance of the current line is 10mm.
w	Data type: wobj structure
	It specifies the workpiece coordinate system. The user can customize the workpiece coordinate system. The target point p specifies the pose of the tool coordinate system in the coordinate system that is specified by the parameter w. In some cases, it is convenient for the user to program in a specific coordinate system. In addition, when the workpiece moves, you only need to re-calibrate the workpiece coordinate system without modifying the user program. Here if the parameter w is defaulted, the system variable $DFWOBJ will be used as the value of the parameter w by default; if the parameter w is specified, the value of the parameter component will be updated to the component corresponding to $DFWOBJ.
CA	Data type: double
	It specifies the circle angle. The user can, instead of specifying the target point directly, specify the target point by specifying the circle angle at which the circle rotates. If the user specifies the parameter CA, the parameter p will be only used to determine the geometry of the circle together with the auxiliary point, rather than the real target point. The system will automatically calculate the real target point with the help of the user-specified circle angle. CA parameter, in deg
dura	Data type: double
	It specifies the trajectory duration. The user can directly specify the movement duration rather than the movement speed. If the user specifies the parameter dura, the system will ignore the parameter speed, and will adjust the speed automatically to meet the time requirements specified by the parameter dura. Dura parameter, in s

For details about above structure data, please refer to *Section 2*.

Example

```
pose p3={x 266,y 88,z -50,a 0,b 0,c 0}
pose p4={x 308,y 52,z -50,a 0,b 0,c 0}
tool tool1 = {{x 0,y 0,z 0,a 0,b 0,c 0}}
cir p3,p4,tool1
// The circle angle is specified as 360 degrees
cir p3,p4,CA:360
// The velocity and slip value are specified in a fuzzy manner
cir m:p3,p:p4,vp:5%,sp:5%
// The speed and slip distance are specified precisely
cir m:p3,p:p4,vl:500mm/s,sl:5mm
// The velocity and slip structures are specified precisely
tool t1 = {{x 0,y 0,z 0,a 0,b 0,c 0}}
```

```
wobj w1 = {{x 0,y 0,z 0,a 0,b 0,c 0}}
speed v2 = {per 10,tcp 50,ori 5,exj 10}
slip s = { pdis 10, ejdis 10, odis 10 }
cir p3,p4,t1,w1,v2, s
```


Notice

- In single step or segment debugging mode, the cir instruction is executed in two steps, the first step runs to the auxiliary point, and the second step runs to the target point.
- Although the user can simplify the program using the parameter default, it is still recommended that the user write the parameters as much as possible when writing the movement instruction to prevent the actual movement from being inconsistent with the user's expectations due to the programmer's negligence, even causing unnecessary loss or injury.

3.3.5 spl(spline motion)

Description

The spl instruction makes the robot go through the teach point smoothly and without pause.

Format

spl p: , [v: | vp:],[s: | sl: | sp:],[t:],[w:]

Parameter

The parameters of the spl instruction are shown in Table 3-5.

Table 3-15 Parameters of the spl

Name	Description
p	Data type: pose
	This parameter specifies the robot TCP target pose and the target point position of the external axis. Structure components with a value of 9e+09 in the parameter will keep the starting position unchanged. The cfg parameter is ignored and is consistent with the starting point cfg parameter. The default value of the turn component is -1
v	Data type: speed
	This parameter specifies the motion speed. The lin instruction uses the tcp, ori, and exj components in the speed structure, which are used to specify the TCP point movement speed, the TCP attitude change speed, and the external axis movement speed, respectively. Here, if the v parameter is defaulted, the system variable $DFSPEED is used as the value of the v parameter by default; if the v parameter is specified, the value of the member component will be updated to the component corresponding to $DFSPEED In simplified programming, v can be replaced by the speed percentage parameter vp or the speed absolute value parameter vl
vp	Data type: double
	This parameter specifies the movement speed percentage. It can be used as an alternative to the speed parameter v, which is used for occasions where it is not necessary to specify the speed value precisely. The format is vp:10%, indicating that the current line speed is 10% of the maximum speed of the robot
s	Data type: slip
	This parameter specifies the smoothing distance. The lin instruction uses the perdis, pdis, odis, and ejdis components of the slip structure to describe the points that leave the original trajectory and enter the smooth trajectory. Among them, perdis is used to specify the percentage of the distance from the target point, pdis is used to specify the path distance from the target point axis, odis is used to specify the attitude angle from the

Name	Description
	target point, and ejdis is used to specify the position angle of the external axis from the target point axis. Here, if the s parameter is defaulted, the system variable $DFSLIP is used by default as the value of the s parameter; if the s parameter is specified, the value of the member component will be updated to the component corresponding to $DFSLIP In simplified programming, s can be replaced by the smoothing percentage parameter sp or the smoothing absolute value parameter sl. See the usage example for the format. Notice: ■ When perdis>=0, other components will be ignored; ■ When pdis, odis and ejdis are specified at the same time, the component closer to the target point will be selected; ■ When the above components are all 0, it means that it is not smooth, but the interpolation point does not necessarily coincide with the target point; ■ When the above components are all less than 0, it means that it is not smooth, and the target point will definitely have an interpolation point, that is, exact stop; When using Cartesian space trajectory (ie lin, cir) smoothing, it is recommended to use pdis smoothing, which can make the smooth trajectory uniform on the front and rear trajectories
sl	Data type: double
	This parameter specifies the smoothing distance, which can replace the smoothing parameter s, and is used when the smoothing value needs to be precisely specified. The format is sl:10mm, indicating that the smoothing distance of the current line is 10mm
sp	Data type: double
	This parameter specifies the smoothing percentage, which can replace the smoothing parameter s and is used in the occasions where the smoothing value does not need to be specified precisely. The format is sp:10%, indicating that the smoothing distance of the current line target point is 10% of the maximum smoothing distance
t	Data type: tool structure
	This parameter specifies the tool. The user needs to customize the tool coordinate system on the tool installed at the end of the flange. The tool coordinate system is fixed to the tool. The target point p specifies the pose of the tool coordinate system in the coordinate system specified by the w parameter. Here, if the t parameter is defaulted, the system variable $DFTOOL is used as the value of the t parameter by default; if the t parameter is specified, the value of the member component will be updated to the component corresponding to $DFTOOL This parameter is to ensure that the TCP speed is limited to not more than 250mm/s when the program is manually run at low speed.
w	Data type: wobj structure
	This parameter specifies the workpiece coordinate system. The user can customize the workpiece coordinate system. The target point p specifies the pose of the tool coordinate system in the coordinate system specified by the w parameter. In some cases, it may be more convenient for the user to program in a specific coordinate system. In addition, when the workpiece moves, it is only necessary to re-calibrate the workpiece coordinate system without modifying the user program. Here, if the w parameter is defaulted, the system variable $DFWOBJ is used as the value of the w parameter by default; if the w parameter is specified, the value of the member component will be updated to the component corresponding to $DFWOBJ

Example

Sample program:

```
lin P:P1
lin P:P2
```

```
spl P:P3
spl P:P4
spl P:P5
spl P:P6
lin P:P7
```

The program runs as shown in Figure 31, moving smoothly from program point 2 to program point 6.

- Run to program point 3 along the path formed by the teaching points of program points 2, 3 and 4.
- Along the trajectory of interval 3 and 4 formed by the teaching points of program points 2, 3 and 4, and the trajectory of interval 3 and 4 formed by the teaching points of program points 3, 4 and 5, the synthesized trajectory runs to Program point 4.
- Along the trajectory of interval 4 and 5 formed by the teaching points of program points 3, 4, and 5, and the trajectory of interval 4 and 5 formed by the teaching points of program points 4, 5, and 6, the synthesized trajectory runs to Program point 5.
- Run to program point 6 along the path formed by the teaching points of program points 4, 5 and 6.

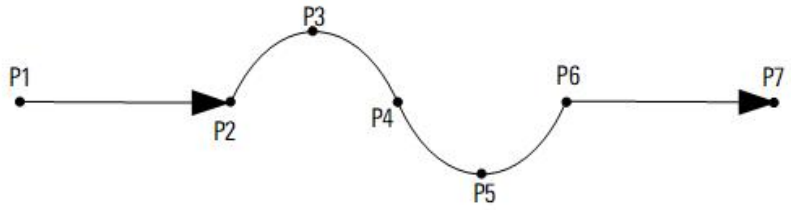


Figure 3-10 Diagram of program execution

3.3.6 jump(Gate movement)

Description

The jump instruction is used to quickly lift the robot from one point and move to another point without requiring the shape of the trajectory followed by the TCP point. All axes reach the target point at the same time.

Format

```
jump p:,[ v: | vp: ],[ t: ],[ w: ],[ a: ],[ b: ],[ Z: ],[ ctr: ],[ s: ],[ sig: ]
```

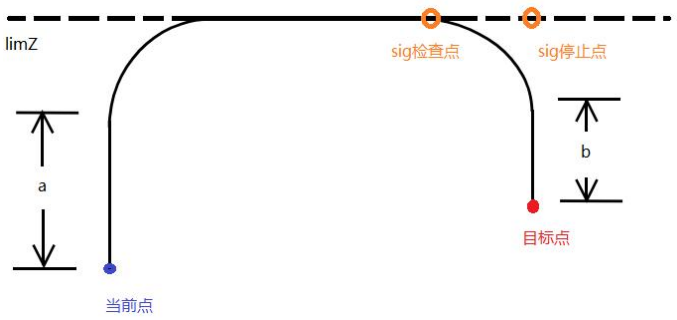
Parameter

See Table 3-16 for the parameters of jump instruction.

Table 3-16 Parameters of jump instruction

Name	Description
p	Data type: pose
	This parameter specifies the target pose of the robot TCP and the target point position of the external axis. The structural component with the value of 9e+09 in the parameter will keep the starting position unchanged. The default value of the cfg component is 0, which means right-handed system, and the default value of the turn component is -1, which means running to the nearest turn value. If the first movement instruction executed in the program is jump, the target point X, Y, Z, A, B, C of the jump instruction and the configured external axis are not allowed to default
v	Data type: speed
	This parameter specifies the movement speed. The ptp instruction uses the per and exj components in the speed structure variable to specify the axis movement speed percentage and external axis movement speed respectively.

Name	Description
	Here if the v parameter is defaulted, the system variable $DFSPEED is used as the value of the v parameter by default; if the v parameter is specified, the value of the member component will be updated to the component corresponding to $DFSPEED In simplified programming, v can be replaced by the speed percentage parameter vp, see usage examples for the format
vp	Data type: double
	This parameter specifies the movement speed percentage. It can replace the speed parameter v, which is used in occasions that do not need to specify the speed accurately. The format is vp:10%, which means that the current line speed is 10% of the maximum speed of the robot
t	Data type: tool structure
	This parameter specifies the tool. The user needs to customize the tool coordinate system on the tool installed at the end of the flange. The tool coordinate system is fixedly connected to the tool. The target point p specifies the pose of the tool coordinate system in the coordinate system specified by the w parameter. Here, if the t parameter is defaulted, the system variable $DFTOOL is used as the value of the t parameter by default; if the t parameter is specified, the value of the member component will be updated to the component corresponding to $DFTOOL This parameter is to determine the TCP position of the robot, while ensuring that the TCP speed is not more than 250mm/s when the program is manually run at low speed.
w	Data type: wobj structure
	This parameter specifies the workpiece coordinate system. The user can customize the workpiece coordinate system. The target point p specifies the pose of the tool coordinate system in the coordinate system specified by the w parameter. In some cases, it may be more convenient for users to program in a specific coordinate system. In addition, when the workpiece moves its position, only need to re-calibrate the workpiece coordinate system without modifying the user program. Here, if the w parameter is defaulted, the system variable $DFWOBJ is used as the value of the w parameter by default; if the w parameter is specified, the value of the member component will be updated to the component corresponding to $DFWOBJ
a	Data type: double
	The relative distance of the arch instruction ascending vertically from the starting point, in mm, the default value is 25mm
b	Data type: double
	Arch instruction the distance to descend vertically before reaching the target point, in mm, the default value is 25mm
Z	Data type: double
	The maximum absolute height allowed in the Z direction of the arch instruction, in mm, the default value is 0 (the maximum height limit)
s	Data type: bool
	Whether it is smooth (do not reach the target point, but turn to the next sentence before deceleration),the true means smooth, the default value is false
ctr	Data type: control
	Set the speed, acceleration and deceleration of the vertical ascending section and descending section of the arch instruction. When you need to specify the speed and stability of the ascent or descent section separately, you need to specify it. For details, see Chapter 2 Control Structure
sig	Data type: bool
	Before the jump instruction drops (the sig checkpoint in the figure), the system will check the sig parameter, and True will stop directly above the target point (the sig stop point in the figure), the three-axis will not drop, and false

Name	Description
	means the target point has been reached. The default is false  This must be a bool type or an expression that can be implicitly converted to a bool type. This expression is the same as the judgment statement in if and while statements For example, the following expressions are all of this type of expression: ■ 1 ■ true ■ getdi(5) ■ getdi(5) == true ■ getdi(5) && getdi(6) ■ counter>=20 ■ T(3.4)

Please refer to *Chapter 2* for details of the above structure data.

Example

```
pose p3={x 266,y 88,z -50,a 0,b 0,c 0}
jump p:p1,vp:5%,t:$FLANGE,w:$WORLD,ctr:ctr1,s:false
```

Notice

- The jump instruction is only applicable to scara robots, and invalid to six-axis robots.
- The jump instruction pursues the fastest tempo, and the trajectory during low-speed operation will be lower than that during high-speed operation. Therefore, even if it is confirmed that there is no collision at high-speed, a collision may occur in low-speed operation.
- If the starting point and target point exceed the Z limit height, it will alarm "12042: Jump trajectory start and end points exceed Cartesian Z axis limit".
- If the vertical ascent distance a or the vertical descent distance b is set to exceed the Z height limit, the Z height limit is met, a or b will not be met, and the trajectory degenerates into a right-angle gate trajectory.

3.3.7 Conveyor belt

Please refer to the "Conveyor Belt Tracking Manual" of our company for the usage of the related instructions of "Conveyor Belt".

3.4 Process control

3.4.1 waittime(Delayed waiting)

Description

It specifies the specified waiting time of the program. When the waittime instruction is executed, the advance planning of the movement instruction will be paused automatically.

Format

waittime time

Parameter

The parameters of the waittime parameter are shown in Table3-17.

Table3-17 Parameters of waittime instruction

Name	Description
time	Data type: double
	This parameter specifies the waiting time, the parameter value should be greater than 0, the unit is seconds

Example

waittime time:5

//It represents waiting for 5 s.

3.4.2 waituntil(Condition wait)

Description

The program waits until the value of a conditional expression is true or times out, and then continues to execute.

Format

waituntil cond:,[maxtime:],[timeoutflag:]

Parameter

The parameters of the waituntil instruction are shown in Table3-18.

Table3-18 Parameters of waituntil instruction

Name	Description
cond	Data type: bool
	It is a conditional expression of bool, and specifies the conditions that the program waits for.
maxtime	Data type: double
	It specifies the max waiting time, in s. When the waiting time exceeds the time set by the parameter, the program will continue to execute even if the value of the cond expression does not become true. The parameter is infinity by default, that is, if the parameter is not written in the program, the program will wait until the value of the cond expression becomes true.
timeoutflag	Data type: bool
	When the parameter maxtime is specified in the waituntil instruction, the timeoutflag parameter can also be specified at the same time. The value of the parameter must be a bool variable. After the waituntil instruction is executed, you can determine whether the waituntil instruction has timed out according to the value of the variable. If the value of the variable is true, it indicates that the value of the cond expression has not changed to true; if the value of the variable is false, it indicates that the value of the cond expression has become true before the timeout.

Example

```
waituntil getdi(6) //Wait for the di signal value of channel 6 to be true

waituntil getdi(6),maxtime:5 //Wait for the 6th channel di signal value to be true before executing the following instructions. The maximum waiting time is 5s

bool time_flag

waituntil getdi(6),maxtime:2,timeoutflag:time_flag //Wait for the 6th channel di signal value to be true before executing the following instructions. The maximum waiting time is 2s

if(time_flag)

print "timeout." //Waiting time exceeds 2s

else

print "cond satisfy within 2 sec." //Waiting time is less than 2s

endif
```

3.4.3 pause(Pause)

Description

Pause the program. The program will enter the pause state after the previous instruction of the pause statement is executed, and the program must be continued through the start of the teach pendant or the external control button.

Format

```
pause
```

Example

```
func void main()

movej j:j1,vp:5%,sp:-1%

pause //The program will be paused when it is running, and the program execution will continue through the start button of the teach pendant

movej j:j2,vp:5%,sp:-1%

endfunc
```


Notice

When there is a motion instruction after the pause instruction, and the program runs to the pause instruction, the program will not enter the pause state. The program will enter the pause state when all the forward-looking trajectories are running. So when the robot needs to stop immediately, please do not use the pause instruction.

3.4.4 exit(Exit the program)

Description

Exit the program running.

The difference between exit and return is as follows: return means to go back to the previous function for continuous execution; exit means to exit the entire program directly no matter which function the current program is executing.

Format

```
exit
```

Example

```
func void main()
```

```
movej j:j1,vp:5%,sp:-1%
//If the di signal of the 6th channel is true, exit the program execution
if(getdi(6))
exit
endif
movej j:j2,vp:5%,sp:-1%
endfunc
```

3.4.5 restart(Restart the program)

Description

The program is executed again. When the program is executed to the statement, the program pointer will return to the first line of the main function and restart execution.

Format

```
restart
```

Example

```
func void main()
movej j:{j1 10}
movej j:{j1 20}
restart
endfunc
```

3.4.6 stopmove(Stop the current movement)

Description

Stop forward motion instructions. This instruction is generally used in interrupt processing functions to stop the motion being executed in the interrupted function. Used in conjunction with startmove, it is necessary to determine the number of tracks to jump after starting according to the parameters following startmove.

Format

```
stopmove [ type: ]
```

For details about stopmove instruction, please refer to *Section 6.7*.

Parameter

The parameters of the stopmove instruction are shown in Table3-19.

Table3-19 Parameters of stopmove instruction

Name	Description
type	Data type: stoptype
	It specifies the type of deceleration for movement stop, including general stop and quick stop. Please refer to *Section 2.6.4* "Definition of stoptype". The parameter can be defaulted. The default value is general.

Name	Description
	■ Normal stop is to stop according to the current maximum acceleration and jerk. ■ Quick stop is faster than normal stop, using 5 times acceleration and jerk.

3.4.7 startmove(Restart the stopped movement)

Description

Restart the stopped movement. This instruction is generally used in interrupt processing functions and used in conjunction with stopmove to continue to execute the movement stopped by the stopmove instruction. The number of trajectories to be jumped after starting needs to be determined according to the parameters following startmove.

Format

startmove [skip:]

For details about startmove instruction, please refer to *Section 6.7.*

Parameter

The parameters of the startmove instruction are shown in Table3-20.

Table3-20 Parameters of startmove instruction

Name	Description
skip	Data type: int
	The number following jump indicates the number of tracks to jump after restart, which is calculated from the number of stop line. Among them: 0 represents no jump, that is, first return to the stop point (return to CP trajectory in a straight line, and return to PTP trajectory in a PTP trajectory), and then continue the original trajectory. 1 represents 1 jump, that is, return to the interrupted trajectory target point directly (return to CP trajectory in a straight line, and return to PTP trajectory in a PTP trajectory). 2 represents 2 jumps, and so on. The parameter is 0 by default. Notice: When the skip value is greater than the number of current motion instructions, it will move to the starting point and then run the subsequent program, that is, restart the program.

3.5 Auxiliary instructions

3.5.1 print(Printout)

Description

The print instruction is used to printout to a certain location. You can use the function to print the value of one or more expressions to HMI message bar, USB, a specified file, or a string. The instruction is mostly used for program debugging, can also be used by the user for log output.

Format

print [to:],[to:],[to:],[to:],[tostr:],[filepath:],[precision:],[numbase:],{ argtoprint: }

Parameter

The parameters of the print instruction are shown in Table3-21.

Table3-21 Parameters of print instruction

Name	Description
to	Data type: printto
	It specifies the output orientation, which can be off, hmi, file or console. ■ off refers to closing any printout d; ■ hmi refers to printout to the message bar on HMI; ■ file refers to printout to the specified file, and the file path is specified by the filepath parameter; ■ console refers to printout to the terminal. Usually, the option will not be used by the user. It is allowed to have multiple parameters to (up to 4) simultaneously, which are used to output to hmi and file simultaneously. Provided that the parameter to with a value of off exists in the print instruction, the parameter specified in front of off will be invalid. The parameter to is a modal parameter. Provided that the parameter to is specified in a print instruction, the output orientation setting will be maintained until the parameter to is specified in the next time. After the program is loaded, the default configuration is to output to hmi only
tostr	Data type: string
	It specifies the output to a string variable. You can use the parameter to output the string corresponding to the variable to a variable of string type
filepath	Data type: string
	When the value of the parameter to is file, the parameter can be used to specify the path to which the file is output. The path takes the script folder as the root folder. If the specified file path is "mylog/mylog.log", it will be actually output to "script/mylog/mylog.log". In addition, the user must ensure that the directory "script/mylog" exists, otherwise an error will be reported during runtime
precision	Data type: int
	It specifies the number of significant digits in which the variable of double type is output
numbase	Data type: num_base
	It specifies the number system format in which the variable of int type is output. It can be hex (hexadecimal) or dec (decimal)
argtoprint	Data type: anytype
	The expressions to be printed can have one or more argtoprint parameters. Finally the system will convert the values of the expressions into strings and concatenate them to printout to the specified position. There is a special constant endl to represent the linefeed character. Printing endl will change the printed string to a new line.

Example

```
print to:hmi, "hello world!"
int i = 1
while(i <= 3)
print to:file,filepath:"mylog","loop ",i,endl
i++
endwhile
print precision:3,"PI:",PI
print "255 =",hex,255,"h"
```

Program running will output:

HMI message bar:

hello world!

In mylog file:

loop 1

loop 2

loop 3

PI:3.14

255 = ffh

3.5.2 scan(Scan input)

Description

The scan instruction is used to scan a string, and read a series of substrings separated by a delimiter into a series of variables according to their types.

Format

scan from:,delimiter:,{ argtosave: }

Parameter

The parameters of the scan instruction are shown in Table3-22.

Table3-22 Parameters of scan instruction

Name	Description
from	Data type: string
	String to be scanned
delimiter	Data type: string
	It referse to the delimiter
argtosave	Data type: int/double/bool/string
	The variable to be saved can have one or more argtosave parameters. The variables must be previously defined. The system will parse a series of strings in the parameter from into corresponding values according to the types of the variables and store them in the variables

Example

double x,y,z

bool b

scan from:"1.1,1.2,1.3,true ",delimiter:",",x,y,z,b

print x, " " ,y, " " ,z, " " ,b

Program running will output:

1.1 1.2 1.3 true

3.5.3 import(Import ARL module)

Description

Import an ARL module.

Format

import modpath

Parameter

The parameters of the import instruction are shown in Table3-23.

Table3-23 Parameters of import instruction

Name	Description
modpath	Data type: string
	It specifies the path of the imported ARL file. If it is in the same directory as the current file, you can directly write the file name.

For details about import instruction, please refer to *Section 8*.

3.5.4 velset(Speed adjustment)

Description

The velset instruction can be used to decrease or increase the speed planned in the program override of all subsequent movement instructions, and can also be used to set the max speed of the movement segment.

Format

velset override:,max:

Parameter

The parameters of the velset instruction are shown in Table3-24.

Table3-24 Parameters of velset instruction

Name	Description
override	Data type: int
	It specifies the percentage value of the speed planned in the program override, including the axis speed, TCP speed, and ORI speed. 100% indicates that the speed set in the program is used. The parameter value range is 0~100 The speed planned in the program override is used for movement planning. The speed override adjustment is operated on the teach pendant during program running, and the actual speed override of the robot is the product of the planning override and the teach pendant speed override.
max	Data type: int
	It specifies the program planning max TCP speed, in mm/s The max speed in the instruction limits the speed planned in the program, rather than the actual running speed. That is, when the speed set in the program is greater than the max speed set in the instruction, the system will plan the speed using the max value in the instruction, but the actual running speed will be also affected by the teach pendant speed override. Since the speed override is definitely less than 100%,

Name	Description
	the actual running speed will not be greater than max.

Example

velset 50,800

All speeds planned in the program are reduced to 50% of the speed set in the program. Under no circumstances can TCP speed be greater than 800mm/s.

3.5.5 accset(Acceleration adjustment)

Description

The accset instruction adjusts the acceleration and jerk of the robot's movement. It is often used when the robot is holding a fragile load. Low acceleration and deceleration are allowed. As a result, the robot's movement is more flexible and the speed can be increased appropriately.

Format

accset acc:,ramp:

Parameter

The parameters of the accset instruction are shown in Table3-25.

Table3-25 Parameters of accset instruction

Name	Description
acc	Data type: int
	It specifies the percentage of the actual acceleration and deceleration of the robot relative to max value. 100% indicates the max acceleration of the system. Max value: 300%. When the instruction input is less than 20%, 20% will be taken as the actual value.
ramp	Data type: int
	It specifies the percentage of the actual jerk of the robot relative to max value. 100% indicates the max jerk of the system. Max value: 300%. When the instruction input is less than 20%, 20% will be taken as the actual value.

Example

accset 50, 300

//The acceleration is limited to 50% of max value.

accset 300, 50

//The jerk is limited to 50% of max value.

3.5.6 toolload(Tool load setting)

Description

The toolload instruction is used to specify the tool load of the robot.

Format

toolload toolinertia:

Parameter

The parameters of the toolload instruction are shown in Table 3-26.

Table 3-26 Parameters of the toolload instruction

Name	Description
toolinertia	Data type: ToolInertiaPara
	This parameter specifies the mass, center of mass, direction of inertia principal axis and moment of inertia of the robot tool load. You can customize ToolInertiaPara type data as toolload parameters. You can also use the system variable $TOOL_INERTIA[i] directly as a parameter to switch to the i-th tool load

Example

```
CentroidPos cen_pos = {x 15, y 25, z 100} //Define the center of mass position
InertiaTensor inertia_tensor = {Ixx 30, Ixy 40, Ixz 50, Iyy 60, Iyz 70, Izz 80} //Define the inertia tensor
ToolInertiaPara tip //Define tool load
tip.m = 30 //mass
tip.centroid_pos = cen_pos
tip.centroid_dir = cen_dir
tip.moment_inertia = inertia
toolload tip //Switch tool load
//Define tool load and initialize
ToolInertiaPara tip1 = {20, {30, 35, 40}, {45, 50, 55}, {60, 65, 70}}
toolload toolinertia:tip1
toolload $TOOL_INERTIA[3] //Switch to the load of tool [3] in the system
```

3.5.7 toolswitch(Tool load switch)

Description

The toolswitch instruction is used to switch the tool load serial number of the current robot.

Format

toolswitch toolindex: ,mu:

Parameter

The parameters of the toolswitch instruction are shown in Table 3-27.

Table 3-27 Parameters of the toolswitch instruction

Name	Description
toolindex	Data type:int
	This parameter specifies the mass, center of mass, direction of inertia principal axis and moment of inertia of the robot tool load. Use the system variable $TOOL_INERTIA[i] directly as a parameter to switch to the i-th tool load

Name	Description
mu	Switch the mechanical unit name of the load

Example

toolswitch toolindex:2

movej j:{j1 10,j2 20,j3 30,j4 40,j5 50,j6 60}

3.5.8 startdetect (Enable collision detection)

Description

Enable collision detection.

Format

startdetect cid: ,mu:

Parameter

Parameter	Name	Description
cid	Collision detection condition number	The value range is 0~16, as follows: ■ 0: Represents the condition number corresponding to manual jog ■ 1-16: Represents the condition number corresponding to the program running
mu	Mechanical unit name	The name of the mechanical unit that starts the collision detection function.

Example

startdetect cid:1 ,mu:R1

3.5.9 enddetect (Disable collision detection)

Description

Disable collision detection.

Format

enddetect mu:

Parameter

Parameter	Name	Description
mu	Mechanical unit name	The name of the mechanical unit for which collision detection is disabled.

Example

enddetect mu:R1

3.6 Function package

For related instructions on arc welding, stacking, bending and other functions, please refer to the corresponding manuals of each function package of our company for specific usage.


Tip

SCARA robot currently does not support the above-mentioned function package related instructions.

4 Logic control instructions

4.1 if(Conditional statements)

Format

if (bool expression)

……

elseif (bool expression)

....

elseif (bool expression)

……

else

....

endif

Description

It specifies the conditional execution instructions.

The system will calculate the value of the bool expression following if from top to bottom, until the value of an expression is true, then execute the instruction between if and the next elseif or else, and go to endif for continuous execution.

The number of elseif will not be limited, and there may be no elseif and/or else.

Example

```
int count = 1
if(count == 1)
setdo(5,true)
endif
if(count > 2)
setdo(5,true)
elseif(count < 2)
setdo(6,true)
else
setdo(7,true)
endif
if(count > 2)
setdo(5,true)
else
setdo(6,true)
endif
```

4.2 compact if(Compact conditional statement)

Format

if (bool expression) ...

Description

It specifies the compact conditional instruction.

When the conditional instruction has only one if, and the value of the conditional expression is true, and when there is only one instruction line being executed, the compact conditional instruction can be used to reduce 3 lines to 1 line.

Example

```
if(getdi(3))    setdo(3,true)
```

4.3 while(while loop)

Format

while (bool expression)

……

endwhile

Description

It specifies the loop execution instructions.

When the value of the bool expression following while is true, the instructions between while and endwhile will be executed, and the value of the bool expression will be recalculated after execution. If true, the instruction between while and endwhile will be re-executed, until the value of the bool expression behind while is false. At this time, go to endwhile for continuous execution.

Example

```
int a = 0
while(a <3)
a++
endwhile
print a //While loop will be executed 3 times, with output "3"
```

4.4 repeat(repeat loop)

Format

repeat

……

until (bool expression)

Description

It specifies the loop execution instructions.

The instructions in the repeat and until loops will be executed at least once. When the value of bool expression following until is true, the loop will be exited; if the value of bool expression following until is false, the repeat loop will be continued.

Example

```
int a = 0
repeat
a++
until(a >= 3)
print a //repeat loop will be executed 3 times, with output "3"
```

4.5 loop(Infinite loop)

Format

```
loop
……
endloop
```

Description

It specifies the infinite loop execution instructions.

The loop instruction is equivalent to while(1) or while(true). The loop will never stop unless the break/exit/restart instruction is executed internally.

Example

```
int a = 5
loop
if(a-- == 0) break
endloop
print a //output "-1"
```

4.6 for(for loop)

Format

```
for (initialization statement; bool expression; iteration expression)
……
endfor
```

Description

It specifies the loop execution instructions.

Execute the initialization instruction, and then judge whether the bool expression is true. If true, execute the instruction between for and endfor, then execute the iteration expression, and judge whether the bool expression is true again. If true, execute the instruction between for and endfor. When the value of bool expression is false, go to endfor for continuous execution.

Example

```
int b = 0
for(int i = 0;i<5;i++)
b++
endfor
print b //for loop will be executed 5 times, with output "5"
```


Notice

- The while and for loops are not exactly equivalent to the loop running mode. When using a while or for loop, all system variables and parameters allowed to be defaulted will not be reset by the system. When using the loop mode, the program will reset after a loop is executed. This means that the system variables to be reset when the program is reset and the parameters allowed to be defaulted will be reset to their default values.

  Example:

  ptp P0

  lin P1,s:{pdis 10}

- If the loop running mode is set, the program will reset after the first loop is executed. When the PTP statement is executed for the second time, the slip parameter has been reset, so the PTP movement will not have slip behavior. If you want the target point of the PTP statement to be slip during loop running, please specify the slip parameter of the PTP explicitly instead of defaulting it.

  Example:

  ptp P0,s:{jdis 10}

  lin P1,s:{pdis 10}

4.7 break(Out of the loop)

Format

break

Description

It refers to breaking out of the loop. In the while, for, loop and repeat cycles, if the instruction is executed, the program will exit the last loop for continuous execution.

Example

```
int counter = 0
while(1)
if(counter == 5)
break //Jump out of the while loop.
else
counter++
endif
endwhile
print count // Output "5"
```

4.8 continue(Continue to the next cycle)

Format

continue

Description

It specifies the ending the current loop and continue the next loop. In the while, for, loop and repeat cycles, if the instruction is executed, the program will end the current loop and continue to execute the next loop of the last loop.

Example

```
int count = 0
while(1)
count++
if(count == 1)
Continue //When executed here, go back directly to continue count++
else
Break //When executed here, exit the while loop, and continue to execute the instructions following endwhile
endif
endwhile
```

4.9 switch(Conditional branch)

Format

switch (expression)

case constant expression:

……

case constant expression:

……

……

default:

……

endswitch

Description

The switch instruction is another writing method of if statement. The system will calculate the value of the expression following switch, and then compare it with the constant expression following each case in turn from top to bottom, until they are equal to each other. Then the instruction between the case and the next case will be executed. After execution, go to endswitch for continuous execution. If failed to match any of the cases, the instruction following default will be executed.

Example

```
func bool TestSwitch()
int j = 3
int i = 0
switch(j)
case 0:
i = 0
case 1:
i = 1
case 2:
i = 2
case 3:
i = 3 //The program will be executed here
default:
i = 4
endswitch
endfunc
```

4.10 goto(Jump)

Format

```
label:

goto label
```

Description

You can define a label on any line in a function. The instruction rules for the label name are the same as those of variables. When executed to a line of goto label, the program will go to the next line of the line where the label is located and continue execution.

Example

```
func bool TestGoto()
int i = 0
next:
i++
if(i<5)
goto next
else
goto end
endif
end:
print i //Output "5"
endfunc
```

4.11 return(Function return)

Format

return

or

return expression

Description

It specifies the function return.

When the program encounters a return instruction, if the program is currently in the called function, the program will return to the previous function. If the program is currently in the main function, the program will end directly.

In the function with a return value, return should be followed with an expression of the function return value type. The system will calculate the value of the expression and return the result to the expression that calls the function.

Example

```
func int add (int x,int y)
return x+y //Return statement
endfunc
func void main()
print add(2,3) //Output "5"
endfunc
```

5 System-defined Functions

The system has defined many functional functions, which can be called directly in the program.

5.1 Mathematical function

5.1.1 abs(Find the absolute value)

Function prototype

double abs(double x)

Or int abs(int x)

Description

It calculates the absolute value of the parameter x.

Parameter

Table5-1 Parameters of abs function

Name	Description
x	Type: double or int
	input value

Return value

Type: double or int

It returns the absolute value of the parameter x.

Example

```
double x = -11.11
int y = -10
double resultx = abs(x)
int resulty = abs(y)
print resultx //Output "11.11"
print resulty //Output "10"
```

5.1.2 sin(Sine function)

Function prototype

double sin(double x)

Description

It calculates the sine value of the parameter x.

Parameter

Table5-2 Parameters of sin function

Name	Description
x	Type: double
	It specifies the angle value, in rad

Return value

Type: double

It returns the sine value of the parameter x, with a range of [-1,1].

Example

```
double x = PI/6
double y = sin(x)
print y //Output "0.5"
```

5.1.3 cos(Cosine function)

Function prototype

```
double cos(double x)
```

Description

It calculates the cosine value of the parameter x.

Parameter

Table5-3 Parameters of cos function

Name	Description
x	Type: double
	It specifies the angle value, in rad

Return value

Type: double

It returns the cosine value of the parameter x, with a range of [-1,1].

Example

```
double x = PI/3
double y = cos(x)
print y //Output "0.5"
```

5.1.4 tan(Tangent function)

Function prototype

double tan(double x)

Description

It is used calculate the tangent value of the parameter x.

Parameter

Table5-4 Parameters of tan function

Name	Description
x	Type: double
	It specifies the angle value, in rad

Return value

Type: double

It returns the tangent value of the parameter x.

Example

```
double x = PI/4
double y = tan(x)
print y //Output "1"
```

5.1.5 asin(Arc sine function)

Function prototype

double asin(double x)

Description

It calculates the anti-sine value of the parameter x.

Parameter

Table5-5 Parameters of asin function

Name	Description
x	Type: double
	It specifies the parameter range [-1,1]

Return value

Type: double

It returns the anti-sine value of the parameter x, in rad, with a range of [-PI/2, PI/2].

Example

```
double x = 0.5
double y = asin(x)
print y //Output "0.523599"
```

5.1.6 acos(Arccosine Function)

Function prototype

```
double acos(double x)
```

Description

It calculates the anti-cosine value of the parameter x.

Parameter

Table5-6 Parameters of acos function

Name	Description
x	Type: double
	It specifies the parameter range [-1,1]

Return value

Type: double

It returns the anti-cosine value of the parameter x, in rad, with a range of [0, PI].

Example

```
double x = 0.5
double y = acos(x)
print y //Output "1.0472"
```

5.1.7 atan(Arctangent function)

Function prototype

```
double atan(double x)
```

Description

It calculates the anti-tangent value of the parameter x.

Parameter

Table5-7 Parameters of atan function

Name	Description
x	Type: double
	Input value

Return value

Type: double

It returns the anti-tangent value of the parameter x, in rad, with a range of (-PI/2,+PI/2).

Example

```
double x = 1
double y = atan(x)
print y //Output "0.785398"
```

5.1.8 atan2(Arctangent function)

Function prototype

```
double atan2(double y, double x)
```

Description

It calculates the anti-tangent value of the parameter y/x.

Parameter

Table5-8 Parameters of atan2 function

Name	Description
x	Type: double
	Numerator value
y	Type: double
	Denominator value

Return value

Type: double

It returns the anti-tangent value of the parameter y/x, in rad, with a range of [-PI,+PI].

Example

```
double x = 1
double y = 1
double z = atan2(y, x)
print z //Output "0.785398"
```

5.1.9 sinh(Hyperbolic sine function)

Function prototype

```
double sinh(double x)
```

Description

It calculates the hyperbolic sine value of the parameter x, where sinh(x)=(exp(x)-exp(-x))/2.

Parameter

Table5-9 Parameters of sin function

Name	Description
x	Type: double
	Enter a value in radians

Return value

Type: double

It returns the hyperbolic sine value of the parameter x.

Example

```
double x = 1
double y = sinh(x)
print y //Output "1.1752"
```

5.1.10 cosh(Hyperbolic cosine function)

Function prototype

double cosh(double x)

Description

It calculates the hyperbolic cosine value of the parameter x, where cosh(x)=(exp(x)+exp(-x))/2.

Parameter

Table5-10 Parameters of cosh function

Name	Description
x	Type: double
	Enter a value in radians

Return value

Type: double

It returns the hyperbolic cosine value of the parameter x.

Example

```
double x = 1
double y = cosh(x)
print y //Output "1.54308"
```

5.1.11 tanh(Hyperbolic tangent function)

Function prototype

double tanh(double x)

Description

It calculates the hyperbolic tangent value of the parameter x, where tanh(x)=sinh(x)/cosh(x).

Parameter

Table5-11 Parameters of tanh function

Name	Description
x	Type: double
	Enter the value in radians

Return value

Type: double

It returns the hyperbolic tangent value of the parameter x, with a range of (-1,+1).

Example

```
double x = 1
double y = tanh(x)
print y //Output "0.76159"
```

5.1.12 exp(Exponential function)

Function prototype

double exp(double x)

Description

It calculates the x power value of the parameter x.

Parameter

Table5-12 Parameters of exp function

Name	Description
x	Type: double
	Exponent value.

Return value

Type: double

It returns the e-based x power value of the parameter x, with a range of (0,+∞).

Example

```
double x = 1
double y = exp(x)
print y //Output "2.71828"
```

5.1.13 pow(Exponential function)

Function prototype

```
double pow(double x,double y)
```

Description

It calculates the x-based y power value.

Parameter

Table5-13 Parameters of pow function

Name	Description
x	Type: double
	Base
y	Type: double
	Exponent

Return value

Type: double

It returns the x-based y power value.

Example

```
double x = 2
double y = 3
double z = pow(x,y)
print z //Output "8"
```

5.1.14 pow10(Exponential function)

Function prototype

```
double pow10(double x)
```

Description

It calculates the 10-based x power value of the parameter x.

Parameter

Table5-14 Parameters of pow10 function

Name	Description
x	Type: double
	Exponent value

Return value

Type: double

It returns the 10-based x power value of the parameter x.

Example

```
double x = -1
double y = pow10(x)
print y //Output "0.1"
```

5.1.15 log(Logarithmic function)

Function prototype

double log(double x)

Description

It calculates the e-based x log value.

Parameter

Table5-15 Parameters of log function

Name	Description
x	Type: double
	It specifies the parameter range x>0

Return value

Type: double

It returns the e-based x log value, ie ln(x).

Example

```
double x = 2
double y = log(x)
print y //Output "0.69315"
```

5.1.16 log10(Logarithmic function)

Function prototype

double log10(double x)

Description

It calculates the 10-based x loga value.

Parameter

Table5-16 Parameters of log10 function

Name	Description
x	Type: double
	It specifies the parameter range x>0

Return value

Type: double

It returns the 10-based x log value, ie $\log_{10}(x)$.

Example

```
double x = 100
double y = log10(x)
print y //Output "2"
```

5.1.17 sqrt(Square root function)

Function prototype

double sqrt(double x)

Description

It calculates the square root value of the parameter x.

Parameter

Table5-17 Parameters of sqrt function

Name	Description
x	Type: double
	It specifies the parameter range x> = 0

Return value

Type: double

It returns the square root value of the parameter x, ie $\sqrt{x}$.

Example

```
double x = 2
double y = sqrt(x)
print y //Output "1.41421"
```

5.1.18 floor(Rounded down)

Function prototype

```
double floor(double x)
```

Description

It calculates the max integer value that is not greater than the parameter x.

Parameter

Table5-18 Parameters of floor function

Name	Description
x	Type: double
	Input value

Return value

Type: double

It returns the max integer value that is not greater than the parameter x.

Example

```
double x = 2.36
double y = floor(x)
print y //Output "2"
```

5.1.19 ceil

Function prototype

```
double ceil(double x)
```

Description

It calculates the min integer value that is not less than the parameter x.

Parameter

Table5-19 Parameters of ceil function

Name	Description
x	Type: double
	Input value

Return value

Type: double

It returns the min integer value that is not less than the parameter x.

Example

```
double x = 2.36
double y = ceil(x)
print y //Output "3"
```

5.1.20 frexp(Decompose floating point numbers)

Function prototype

double frexp(double val,int & exp)

Description

The parameter val is broken down into the numerical part x and the 2-based exponent part n, that is, val = $x*2^n$, where n is stored in the variable to which exp points.

Parameter

Table5-20 Parameters of frexp function

Name	Description
val	Type: double
	Input value
exp	Type: int
	Output exponent value

Return value

Type: double

It returns the value of the parameter val that is broken down into the numerical part x, with a range of [0.5,1).

Example

```
double val = 64
int exp
double y = frexp(val,exp)
print y //Output "0.5"
print exp //Output "7"
```

5.1.21 ldexp(Load floating point number)

Function prototype

double ldexp(double val,int exp)

Description

It calculates the product of the exp power value of the parameters val and 2, ie val*2exp.

Parameter

Table5-21 Parameters of the ldexp function

Name	Description
val	Type: double
	Input value
exp	Type: int
	Enter exponent value

Return value

Type: double

It returns the product of the exp power value of the parameters val and 2.

Example

```
double val = 3
int exp = 3
double y = ldexp (val,exp)
print y //Output "24"
```

5.1.22 fmod(Modulo)

Function prototype

double fmod(double x,double y)

Description

It calculates the modulus of the parameters x versus y, ie remainder of x/y.

Parameter

Table5-22 Parameters of fmod function

Name	Description
val	Type: double
	Molecular parameters
exp	Type: double
	Denominator parameter, y cannot be 0

Return value

Type: double

It returns the remainder of the parameters x/y.

Example

```
double x = 12.58
double y = 2.6
double z = fmod(x,y)
print z //Output "2.18"
```

5.1.23 modf(Decompose floating point numbers)

Function prototype

```
double modf(double x,int &y)
```

Description

It breaks down the double number x into integer and decimal parts, and stores the integer part into the variable y.

Parameter

Table5-23 Parameters of modf function

Name	Description
x	Type: double
	It specifies the number of doubles broken down
y	Type: int
	It specifies the integer part of the number of doubles returned

Return value

Type: double

It returns the decimal part of the double-precision number x, with a range of (0,1).

Example

```
double x = 123.456
double y
double z = modf (x,y)
print y //Output "123"
print z //Output "0.456"
```

5.1.24 hypot(Calculate the length of the hypotenuse of a right triangle)

Function prototype

```
double hypot(double x,double y)
```

Description

It calculates the length of the hypotenuse of the right triangle when the two sides are x and y respectively.

Parameter

Table5-24 Parameters of hypot function

Name	Description
x	Type: double
	The length of a right side of a right triangle
y	Type: double
	The length of the other right side of a right triangle

Return value

Type: double

It returns the square root of the sum of x-square and y-square, ie $\sqrt{x^2 + y^2}$.

Example

```
double x = 6
double y = 8
double z = hypot(x,y)
print z //Output "10"
```

5.1.25 rand(Generate random number)

Function prototype

```
int rand(void)
```

Description

It randomly generates an integer.

Parameter

None

Return value

Type: int

It returns a random integer, with a range of [0,+2147483647].

Example

```
int x = rand()
print x //Randomly output an integer, such as "15"
```

5.1.26 norm(Calculate vector length)

Function prototype

```
double norm(pos p)
```

Description

It calculates the length of the current position point from the origin of the coordinate system.

Parameter

Table5-25 Parameters of norm function

Name	Description
p	Type: pos
	Vector coordinate value

Return value

Type: double

It returns the length of the current position p from the origin of the coordinate system.

Example

```
pos p = {x 300,y 400,z 500}
double len = norm(p)
print len //Output "707.107"
```

5.1.27 trunc(Truncated floating number)

Function prototype

double trunc(double num,int n,bool round)

Description

It truncates a value into a specified number of digits following the decimal point according to the rounding requirements.

Parameter

Table5-26 Parameters of trunc function

Name	Description
num	Type: double
	It specifies the number of doubles truncated
y	Type: double
	It specifies the number of decimal places reserved. n>=0
round	Type: bool
	"true" represents rounding and "false" represents truncation

Return value

Type: double

It returns the result after truncation.

Example

```
double num = 3.1415926
int n = 4
double result1 = trunc(num,n,true)
print result1 //Output "3.1416"
double result2 = trunc(num,n,false)
print result2 //Output "3.1415"
```

5.2 Bit manipulation function

5.2.1 bitclear(Bit cleared to 0)

Function prototype

void bitclear(byte &data,int pos)

Or void bitclear(int & data,int pos)

Description

It clears the bit pos of an integer or byte data to 0.

Parameter

Table5-27 Parameters of bitclear function

Name	Description
data	Type: byte or int
	Input value
pos	Type: int
	It specifies the parameter range [0,7] (when the data type is byte) or [0,31] (when the data type is int)

Return value

None

Example

```
byte data1= 00001111b
bitclear(data1,2)
print data1 //Output "0bh"
int data2= 15
bitclear(data2,2)
print data2 //Output "11"
```

5.2.2 bitset(Bits set to 1)

Function prototype

void bitset(byte & data,int pos)

Or void bitset(int & data,int pos)

Description

It sets the bit pos of an integer or byte data to 1.

Parameter

Table5-28 Parameters of bitset function

Name	Description
data	Type: byte or int
	The number of bits to be set.
pos	Type: int
	It specifies the parameter range [0,7] (when the data type is byte) or [0,31] (when the data type is int)

Return value

None

Example

byte data1= 00001000

bitset(data1,2)

print data1 //Output "0ch"

int data2 = 8

bitset(data2,2)

print data2 //Output "12"

5.2.3 bitcheck(Bits check)

Function prototype

bool bitcheck(byte & data,int pos)

Or bool bitcheck(int & data,int pos)

Description

It checks whether the bit pos of an integer or byte data is 1. If yes, the function will return true. If not, the function will return false.

Parameter

Table5-29 Parameters of bitcheck function

Name	Description
data	Type: byte or int

Name	Description
	The number of bits to be checked.
pos	Type: int
	It specifies the parameter range [0,7] (when the data type is byte) or [0,31] (when the data type is int)

Return value

Type: bool

It returns whether the bit pos of the parameter data is equal to 1. If not, the function will return true.

Example

```
byte data1= 00001000b
bool result = bitcheck(data1,2)
print result //Output "false"
int data2= 8
print bitcheck(data2,3) //Output "true"
```

5.2.4 bitlcs(Cyclic shift multiple bits to the left)

Function prototype

int bitlcs(int data,int n)

Or byte bitlcs(byte data,int n)

Description

It shifts left an integer or byte data by n bits.

Parameter

Table5-30 Parameters of bitlcs function

Name	Description
data	Type: byte or int
	The number to be shifted.
pos	Type: int
	n >= 0. The parameter n can be defaulted. The default value is 1

Return value

Type: int or byte

It returns the result of the parameter data after left shift.

Example

```
byte data= 11000000b
print bitlcs (data, 2) //output "03h"
```

print bitlcs (data) //Output "81h"

5.2.5 bitrcs(Cyclic shift multiple bits to the right)

Function prototype

int bitrcs(int data,int n)

Or byte bitrcs(byte data,int n)

Description

It shifts right an integer or byte data by n bits.

Parameter

Table5-31 Parameters of bitrcs function

Name	Description
data	Type: byte or int
	The number to be shifted.
pos	Type: int
	n >= 0. The parameter n can be defaulted. The default value is 1

Return value

Type: int or byte

It returns the result of the parameter data after right shift.

Example

byte data= 00000011b

print bitrcs(data, 2) //Output "c0h"

print bitrcs(data) //Output "81h"

5.3 Clock function

5.3.1 clock(Clock type)

Description

The clock type is used for timing in the program. This data type is used with *clkreset*, *clkstart*, *clkstop* and *clkread* functions.

Example

clock c

clkreset(c)

clkstart(c)

waittime 3

clkstop(c)

print clkread(c) //The output "3.00098" means that it takes about 3 seconds from clkstart to clkstop.

5.3.2 clkstart(Start clock timing)

Function prototype

void clkstart(clock &c)

Description

It starts the clock time. After a clock is started, you can perform operations such as clkread, clkstop, and clkreset.

Parameter

Table5-32 Parameters of clkstart function

Name	Description
c	Type: clock
	Refer to *clock* data type

Return value

None

Example

Refer to *clkread* function.

5.3.3 clkstop(Stop clock timing)

Function prototype

void clkstop(clock &c)

Description

It stops the clock time. After a clock is stopped, you can perform operations such as clkread, clkstart, and clkreset.

Parameter

Table5-33 Parameters of clkstop function

Name	Description
c	Type: clock
	Refer to *clock* data type

Return value

None

Example

Refer to *clkread* function.

5.3.4 clkreset(Clock cleared)

Function prototype

void clkreset(clock &c)

Description

It resets the clock variable to 0.

Parameter

Table5-34 Parameters of clkreset function

Name	Description
c	Type: clock
	Refer to *clock* data type

Return value

None

Example

Refer to *clkread* function.

5.3.5 clkread(Read clock)

Function prototype

double clkread(clock &c)

Description

It reads the value of the clock variable c.

Parameter

Table5-35 Parameters of clkread function

Name	Description
c	Type: clock
	Refer to *clock* data type

Return value

Type: double

It returns the value of the clock variable c, in s

Example

```
clock c
clkstart(c)
byte data= 00001100b
int n = 2
```

```
byte result = bitrcs1(data, n)

clkstop(c)

print clkread (c) //Output the time to run from the previous clkstart function to clkstop function. For example, if the running time is 0.001s, the output will be "0.001"

clkreset(c)

print clkread (c) //Output "0"
```

5.4 String-related functions

5.4.1 strlen(Get string length)

Function prototype

```
int strlen(string s)
```

Description

It calculates the length of the string s.

Parameter

Table5-36 Parameters of strlen function

Name	Description
s	Type: string
	The number of characters is a string to be counted.

Return value

Type: int

It returns the length of the string s.

Example

```
string s = "hello world!"

int len = strlen(s)

print len //Output "12"
```

5.4.2 substr(Intercept string)

Function prototype

```
string substr(string s,int startpos,int len)
```

Description

Truncate a substring of a specific length from a specified position in the string.

Parameter

Table5-37 Parameters of the substr function

Name	Description
s	Type: string
	The string to be intercepted.
startpos	Type: int
	It truncates the starting position of the string. startpos starts from 0. If startpos is negative or greater than or equal to the total length of the string, there may have an operating error
len	Type: int
	It truncates the length of the string. If the parameter is not specified, the substring will continue to the end of the string. If len is negative, there may have an operating error

Return value

Type: string

It returns the substring in which the length is truncated from the specified position startpos of the string s.

Example

```
s1 = "hello world! "
int statpos = 0
int len = 5
string s2 = substr(s1,startpos,len)
print s2 //Output "hello"
```

5.4.3 toascii(Get the ASCII code corresponding to the character)

Function prototype

int toascii(string s)

Description

It returns the ASCII code corresponding to the character s.

Parameter

Table5-38 Parameters of toascii function

Name	Description
s	Type: string
	It enters the string s, which can only contain 1 character

Return value

Type: int

ASCII code value corresponding to the character s.

Example

print toascii("a") //Output "97"

5.5 IO-related functions and instructions

5.5.1 setdo(Asynchronous output single DO)

Function prototype

void setdo(int chan,bool value)

Description

It sets a DO signal to true or false.

Parameters

Table5-39 Parameters of setdo function

Name	Description
chan	Type: int
	The parameter range is [1, DO port number]. Such as: [1,32*DO slave station number] (for MIII bus version control system) or [1, 40*PLC_MF slave station number] (for RTEX bus version control system)
value	Type: bool
	It specifies the output value

Return value

None

Example

Setdo(3,true) //Set the 3rd channel to true

5.5.2 setdo(Asynchronous output multiple DO)

Function prototype

void setdo(int from_chan,int to_chan,int value)

Description

It sets continuous multiple DO signals.

Parameter

Table5-40 Parameters of setdo function

Name	Description
from_chan	Type: int
	The parameter range is [1, DO port number]. Such as: [1,32*DO slave station number] (for MIII bus version control system) or [1, 40*PLC_MF slave station number] (for RTEX bus version control system)

Name	Description
to_chan	Type: int
	It specifies the ending channel number. it specifies the parameter range [number of 1,32*DO slaves] (for MIII bus control system) or [number of 1,40*PLC_MF slaves] (for RTEX bus control system) from_chan <= to_chan Even if to_chan exceeds the actual controllable address range, provided that the above conditions are met, DOs within the controllable address range can still be output normally without any warning.
value	Type: int
	Since the length of int data is 32 bits, only a maximum of 32 continuous DOs can be set simultaneously.

Return value

None

Example

setdo(5,8,1100b) //Set the 5th and 6th DO signals to false, and set the 7th and 8th signals to true

5.5.3 setdoinmv(Non-stop forward-looking asynchronous output single DO)

Function prototype

setdoinmv channel:int chan,value:bool value

Description

It sets a DO signal to true or false and outputs it before the next line of movement instruction starts.

Parameter

Table5-41 Parameters of setdoinmv instruction

Name	Description
chan	Type: int
	The parameter range is [1, DO port number]. Such as: [1,32*DO slave station number] (for MIII bus version control system) or [1, 40*PLC_MF slave station number] (for RTEX bus version control system)
value	Type: bool
	It specifies the output value

Return value

None

Example

setdoinmv channel:3,value:true //Set the 3rd channel to true

5.5.4 syncdo(Sync output single DO)

Function prototype

void syncdo(int chan,bool value)

Description

It synchronously outputs a DO signal to be true or false. The difference between synchronous output and asynchronous output is that the synchronous output ensures that the IO port signal has been output before continuing to execute the program, while the asynchronous output continues to execute the program after sending the data.

Parameter

Table5-42 Parameters of syncdo function

Name	Description
chan	Type: int
	The parameter range is [1, DO port number]. Such as: [1,32*DO slave station number] (for MIII bus version control system) or [1, 40*PLC_MF slave station number] (for RTEX bus version control system)
value	Type: bool
	It specifies the output value

Return value

None

Example

syncdo (3, true) //Set the 3rd channel to true

5.5.5 syncdo(Synchronous output multiple DO)

Function prototype

void syncdo(int from_chan, int to_chan,int value)

Description

It synchronously outputs continuous multiple DO signals. The difference between synchronous output and asynchronous output is that the synchronous output ensures that the IO port signal has been output before continuing to execute the program, while the asynchronous output continues to execute the program after sending the data.

Parameter

Table5-43 Parameters of syncdo function

Name	Description
from_chan	Type: int
	The parameter range is [1, DO port number]. Such as: [1,32*DO slave station number] (for MIII bus version control system) or [1, 40*PLC_MF slave station number] (for RTEX bus version control system)
to_chan	Type: int

Name	Description
	End channel number. The parameter range is [1, DO port number]. Such as: [1,32*DO slave station number] (for MIII bus version control system) or [1, 40*PLC_MF slave station number] (for RTEX bus version control system) from_chan <= to_chan
value	Type: int
	Since the length of int data is 32 bits, only a maximum of 32 continuous DOs can be set simultaneously.

Return value

None

Example

syncdo(5,8,1100b)//Set the 5th and 6th DO signals to false, and set the 7th and 8th signals to true

5.5.6 pulsedo(Output single pulse signal)

Function prototype

void pulsedo(int chan, bool value,double width)

Description

The function is used to output the single pulse signal with a specified pulse width. Compared with the pulse signal that is implemented by the setdo function in combination with the waittime instruction, using the pulsedo function is more efficient, because it will continue to execute the program without waiting for the pulse signal to complete execution. That is, the execution of the pulse output is in parallel with the execution of the program.

Parameter

Table5-44 Parameters of pulsedo function

Name	Description
chan	Type: int
	The parameter range is [1, DO port number]. Such as: [1,32*DO slave station number] (for MIII bus version control system) or [1, 40*PLC_MF slave station number] (for RTEX bus version control system)
value	Type: bool
	True represents output of high pulse, and false represents output of low pulse.
width	Type: double
	It specifies the pulse width, in s. The value must be greater than or equal to 0, and the minimum output pulse width is about 25ms.

Return value

None

Example

pulsedo(5,true,0.2)//The 5th DO outputs a high pulse signal with a pulse width of 200ms

Precautions

- When a pulse output function has not yet been executed and another pulsedo function is executed in the DO, the pulsedo signal that has not yet been executed will terminate.

  For example, for the following program:

  pulsedo(5,true,3)

  waittime 2

  pulsedo(5,false,1)

  The waveform in Figure5-1will be output in #5 DO:

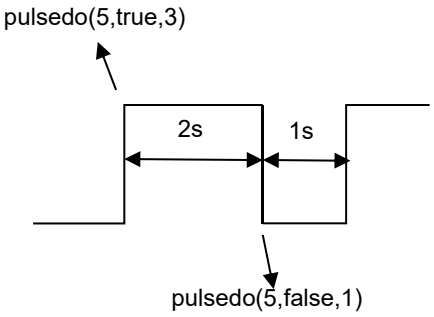


  Figure5-1 Schematic diagram of program waveform

- If there is a pulse signal in a channel that has not yet been executed, the channel will be in the running or paused state always and will be in the stopped state until all the pulse output signals are executed.
- The pulse output signal will not stop execution when the program is paused.
- When the program is reset, if there are pulse output signals that have not yet been executed, these signals will be stopped from execution immediately.

5.5.7 getdo(Get single DO)

Function prototype

bool getdo(int chan)

Description

It gets the value of a single DO signal.

Parameter

Table5-45 Parameters of getdo function

Name	Description
chan	Type: int
	The parameter range is [1, DO port number]. Such as: [1,32*DO slave station number] (for MIII bus version control system) or [1, 40*PLC_MF slave station number] (for RTEX bus version control system)

Return value

Type: bool

The acquired DO signal value of channel chan.

Example

```
int chan = 3
setdo(chan,true)
bool value = getdo(chan)
print value //Output "true"
```

5.5.8 getdo(Get multiple DO)

Function prototype

int getdo(int from_chan,int to_chan)

Description

It gets the values of continuous multiple DO signals.

Parameter

Table5-46 Parameters of getdo function

Name	Description
from_chan	Type: int
	Starting channel number. The parameter range is [1, DO port number]. Such as: [1,32*DO slave station number] (for MIII bus version control system) or [1, 40*PLC_MF slave station number] (for RTEX bus version control system)
to_chan	Type: int
	End channel number. The parameter range is [1, DO port number]. Such as: [1,32*DO slave station number] (for MIII bus version control system) or [1, 40*PLC_MF slave station number] (for RTEX bus version control system) from_chan <= to_chan

Return value

Type: int

Since the length of the int data is 32 bits, only a maximum of the values of 32 continuous DO signals can be obtained simultaneously.

Example

```
int from = 5
int to = 8
setdo(from,to,1001b)
print getdo(from,to) //Output "9"
```

5.5.9 getdi(Get single DI)

Function prototype

bool getdi(int chan)

Description

It gets value of a single DI signal.

Parameter

Table5-47 Parameters of getdi function

Name	Description
chan	Type: int
	The parameter range is [1, DO port number]. Such as: [1,32*DO slave station number] (for MIII bus version control system) or [1, 40*PLC_MF slave station number] (for RTEX bus version control system)

Return value

Type: bool

The obtained value of #chan DI signal

Example

```
//Provided that #4 channel DI inputs high level
print getdi(4) //Output "true"
```

5.5.10 getdi(Get multiple DI)

Function prototype

int getdi(int from_chan,int to_chan)

Description

It gets the values of continuous multi-channel DI signals.

Parameter

Table5-48 Parameters of getdi function

Name	Description
from_chan	Type: int
	Starting channel number. The parameter range is [1, DO port number]. Such as: [1,32*DO slave station number] (for MIII bus version control system) or [1, 40*PLC_MF slave station number] (for RTEX bus version control system)
to_chan	Type: int
	End channel number. The parameter range is [1, DO port number]. Such as: [1,32*DO slave station number] (for MIII bus version control system) or [1, 40*PLC_MF slave station number] (for RTEX bus version control system) from_chan <= to_chan

Return value

Type: int

Since the length of the int data is 32 bits, only a maximum of the values of 32 continuous DO signals can be obtained simultaneously.

Example

```
//Assuming that the DI inputs of channels 6 to 9 are high
int value = getdi(6,9)
print value //Output "15"
```

5.5.11 getai(Get analog input signal)

Function prototype

double getai(int chan)

Description

Get the input value of an analog signal. Among them, the AI signal type includes: current type and voltage type, the AI signal type is configured through the PLC slave type. When the input signal is current type, the return value unit is milliampere; when the input signal is voltage type, the return value unit is volts.

Parameter

Table 5-49 Parameters of getai function

Name	Description
chan	Type:int
	The parameter range is [1, DO port number]. Such as: [1,32*DO slave station number] (for MIII bus version control system) or [1, 40*PLC_MF slave station number] (for RTEX bus version control system)

Example

```
print getai(2) //Output "analog input value of the second channel" (the unit is related to the configured AI signal type)
```

5.5.12 getnostopdi(Non-stop forward-looking asynchronous acquisition of single-channel DI)

Function prototype

bool getnostopdi(int chan)

Description

Get the value of a single DI signal, before the start of the next line of motion instruction.


Tip

Compared with getdi, this instruction keeps looking forward, so it will not cause the robot to stop stuck. However, if there are too many look-aheads, the robot may trigger the instruction before executing the previous motion statement, resulting in undesirable actions.

Parameter

Table 5 50 Parameters of getnostopdi function

Name	Description
chan	Type:int
	The parameter range is [1, DO port number]. Such as: [1,32*DO slave station number] (for MIII bus version control system) or [1, 40*PLC_MF slave station number] (for RTEX bus version control system)

Return value

Type: bool

The acquired DI signal value of channel chan

Example

```
//Assuming that the 4th channel DI input high level
print getnostopdi(4) //Output "true"
```

5.5.13 setao(Asynchronous output analog signal)

Function prototype

void setao(int chan,double value)

Description

Set a channel of analog signal output. Among them, AO signal types include: current type and voltage type, AO signal type is configured through PLC slave type.

Parameter

Table 5-50 Parameters of setao function

Name	Description
chan	Type:int
	Analog output port number, parameter range [1,2*AO slave station number] (for MIII bus version control system) or [1,2*PLC_MF slave station number] (for RTEX bus version control system)
value	Type: double
	When the output signal is current type, the parameter range is [4,20], the unit is mA; when the output signal is voltage type, the parameter range is [-10,10], and the unit is volt

Example

```
setao(2,5.2) //If the second AO is configured as current type, set the second channel to 5.2mA
setao(2,5.2) //If the second AO is configured as voltage type, set the second channel to 5.2V
```

5.5.14 syncao(Synchronous output analog signal)

Function prototype

void syncao(int chan,double value)

Description

Simultaneously output one analog signal. The difference between synchronous output and asynchronous output is that synchronous output ensures that the IO port signal has been output before continuing to execute the program, while asynchronous output continues to execute the program after sending the data.

Parameter

Table 5-51 Parameters of the syncaov function

Name	Description
chan	Type:int
	Analog output port number, parameter range [1, 2*AO slave station number] (for MIII bus version control system) or [1, 2*PLC_MF slave station number] (for RTEX bus version control system)
value	Type: double
	When the output signal is current type, the parameter range is [4, 20], the unit is mA; when the output signal is voltage type, the parameter range is [-10,10], and the unit is volt

Parameter

None

Example

syncao(2,5.2) //If the second AO is configured as current type, set the second channel to 5.2mA

syncao(2,5.2) //If the second AO is configured as voltage type, set the second channel to 5.2V

5.5.15 getao(Get analog output signal)

Function prototype

double getao(int chan)

Description

Get the output value of an analog signal. Among them, AO signal types include: current type and voltage type, AO signal type is configured through PLC slave type. When the output signal is configured as current type, the return value unit is milliampere; when the output signal is configured as voltage type, the return value unit is volt.

Parameter

Table 5-52 Parameters of getao function

Name	Description
chan	Type:int
	Analog output port number, parameter range [1,2*AO slave station number] (for MIII bus version control system) or [1,2*PLC_MF slave station number] (for RTEX bus version control system)

Parameter

Type: double

Returns the value of the chan analog port signal.

Example

```
int chan = 1
setao(chan,5.2)
double value = getao(chan)
print value //Output "5.2"
```

5.5.16 getintdo(Read the DO signal value of a single PLC_INT)

Function prototype

```
bool getintdo(int chan)
```

Description

It gets the value of the single DO signal on INT.

Parameter

Table5-53 Parameters of getintdo function

Name	Description
chan	Type: int
	The parameter range is [1, DO port number]. Such as: [1,32*DO slave station number] (for MIII bus version control system) or [1, 40*PLC_MF slave station number] (for RTEX bus version control system)

Return value

Type: bool

The acquired INT channel DO signal value chan.

Example

```
bool value = getintdo(chan)
print value //Output the state of #chan DO on INT
```

5.5.17 getintdi(Read the DI signal value of a single PLC_INT)

Function prototype

```
bool getintdi(int chan)
```

Description

It gets the value of the single DI signal on INT.

Parameter

Table5-54 Parameters of getintdi function

Name	Description
chan	Type: int
	The parameter range is [1, DO port number]. Such as: [1, 32*DO slave station number] (for MIII bus version control system) or [1, 40*PLC_MF slave station number] (for RTEX bus version control system)

Return value

Type: bool

The acquired DI signal value of INT channel chan.

Example

```
bool value = getintdi(chan)
print value //Output the state of #Output the DI status of channel chan on INT
```

5.5.18 setpwm(Set the frequency and duty cycle of a PWM channel)

Function prototype

bool setpwm(int channel, int freq, int ratio)

Description

Set the frequency and duty cycle of a PWM channel.

Parameter

Table 5-55 Parameters of setpwmi function

Name	Description
channel	Type: int
	PWM channel number, parameter range: [1,3]
freq	Type: int
	PWM frequency, unit: Hz, parameter range: [10,10000]
chan	Type: int
	PWM duty cycle, unit: %, parameter range: [0,100]

Return value

Type: bool

Example

```
setpwm(1,20,50) //Set the first PWM channel, the frequency is set to 20Hz, and the duty cycle is set to 0.5
```

5.6 Communication-related functions

5.6.1 socket communication-related functions

5.6.1.1 setip(Set the external network port IP)

Function prototype

bool setip(string ip,string gate,string mask,string if_name)

bool setip(string ip,string gate,string mask)

Description

If you want to communicate with external devices via external internet access, you must set the internet access IP, gateway, and subnet mask. They can be set by the function.

Parameter

Table5-56 Parameters of setip function

Name	Description
ip	Type: string
	It specifies the IP string to be set
gate	Type: string
	It specifies the gateway to be set
mask	Type: string
	It specifies the subnet mask to be set
if_name	Type: string
	The name of the network port to be set. Note: For the SCARA control cabinet, there are three user network ports: eth1, eth2, and eth3. For other control cabinets, there is only one user network port: eth1 (when this parameter defaults, the default is eth1)

Return value

Type: bool

true indicates that the setting succeeds, false indicates that the setting fails.

When setting the IP, the gateway address must also be a real IP, otherwise it will return false.

Example

setip("10.20.210.93","10.20.210.255","255.255.255.0","eth1") //The set user network port name is eth1

setip("10.20.210.93","10.20.210.255","255.255.255.0") //The set user network port name is eth1

5.6.1.2 getip(Get the external network port IP)

Function prototype

bool getip(string& ip,string if_name)

bool getip(string& ip)

Description

If you want to obtain the IP string of the external network port. It can be set through this function.

Parameter

Table5-57 Parameters of getip function

Name	Description
ip	Type: string
	It specifies the IP string to be set
if_name	Type: string
	Specify which user network port IP to obtain. Note: For the SCARA control cabinet, there are three user network ports: eth1, eth2, and eth3. For other control cabinets, there is only one user network port: eth1 (when this parameter defaults, the default is eth1)

Return value

Type: bool

True indicates that the getting succeeds, and false indicates that the getting fails.

Example

```
string ip
getip(ip,"eth2") //Get the IP of the user network port named "eth2"
getip(ip) //Get the IP of the user network port named "eth1"
```

5.6.1.3 connect(Initiate socket connection)

Function prototype

bool connect(socket s,string ip,int port)

Description

If you want the robot controller to be the party that initiates the connection, ie the client, you must initiate the socket connection with the function. The synchronous connection must be implemented in conjunction with waituntil instruction, that is, the program will continue to run until the connection is successful.

Parameter

Table5-58 Parameters of connect function

Name	Description
s	Type: socket
	It specifies the predefined socket variables
ip	Type: string
	It specifies the IP string of the other party’ s device to be connected

Name	Description
port	Type: int
	It specifies the port of the other party’ s device to be connected

Return value

Type: bool

When the connection fails, it will return false; when it returns true, the connection is successful.

Example

```
socket s
//Wait until the connection is successful, you can also use the maxtime of the waituntil instruction here
//Wait until the parameter implements the timeout mechanism
waituntil connect(s, "192.168.0.40" ,2888)
```

5.6.1.4 accept(Accept socket connection)

Function prototype

bool accept(socket s, string ip, int port)

Description

If you want the robot controller to be the party that accepts connection passively, ie the server, you must accept the socket connection with the function. The synchronous connection must be implemented in conjunction with waituntil instruction, that is, the program will continue to run until the other party initiates the connection.

Parameter

Table5-59 Parameters of accept function

Name	Description
s	Type: socket
	It specifies the predefined socket variables
ip	Type: string
	It specifies the local IP string
port	Type: int
	It specifies the local port

Return value

Type: bool

When no connection is established, it will return false; when it returns true, the connection is established successfully.

Example

```
socket s
```

```
//Wait until the connection is established, you can also use the maxtime of the waituntil instruction here
//Wait until the parameter implements the timeout mechanism
waituntil accept(s, "192.168.0.40" ,2888)
```

5.6.1.5 write(Send data through socket)

Function prototype

bool write(socket s,string data) or

bool write(socket s,byte[] data,int len)

Description

Both the server and client send data via the function. The synchronous sending must be implemented in conjunction with waituntil instruction, that is, the program will continue to run until the data is sent.

Parameter

Table5-60 Parameters of write function

Name	Description
s	Type: socket
	It specifies the predefined socket variables
data	Type: string/byte[]
	It specifies the string or binary data to be sent
len	Type: int
	When the data sent is a byte array, the parameter represents the length of the byte written

Return value

Type: bool

When the sending is not completed, it will return false; when it returns true, it indicates that the sending is completed.

Example

```
socket s
waituntil connect(s, "192.168.0.40" ,2888)
waituntil write(s, "hello" )
byte bdata[3] = {1,2,3}
waituntil write(s,bdata,3)
```

Socket disconnection and reconnection function

When the socket is disconnected in the write process, ARCS will send an alarm and stop running. If you do not want the program to stop at this time, you can try to reconnect it after disconnection. You can set it in the ARL program, $DETECT_SOCK_CLOSE = true.

After disconnection, the current socket error status will be set to $ERR_SOCK_CLOSED. The user can get the error status of socket s via geterror(s).

Tip

- When you need to use the disconnection reconnection function, the use of timer interrupt (timer) and clear buffer (clearbuff) will affect the detection of disconnection status. It is recommended to use while and waittime instead of timer interrupt timer.
- When the robot is Server, in the disconnection and reconnection sub-function, you need to execute close first, and then execute accept.

The method to implement disconnection and reconnection is as follows:

The user can declare an interrupt. When the error status of a socket is $ERR_SOCK_CLOSED, an interrupt processing function will be triggered, and the socket connection will be re-established in the interrupt processing function.

Example:

```
socket s
func void handler()
close(s)
waituntil connect(s,"10.20.220.1",8080)
endfunc
func viod main()
init()
$DETECT_SOCK_CLOSE=true
interrupt 1, when:geterror(s)==$ERR_SOCK_CLOSED, do:handler()
setip(.....)
waituntil connect(s,.......)
waituntil write (s,......)
endfunc
```

5.6.1.6 read(Receive data through socket)

Function prototype

bool read(socket s,string data,int len)

bool read(socket s,byte[] data,int len)

Description

Both the server and the client receive the data of a specified length via the function. The synchronous receiving must be implemented in conjunction with waituntil instruction, that is, the program will continue to run until the data is received.

Parameter

Table5-61 Parameters of read function

Name	Description
s	Type: socket
	It specifies the predefined socket variables
data	Type: string/byte[]

Name	Description
	It specifies the predefined string or byte array variables
len	Type: int
	It specifies the length of the data received. When the data parameter is a string, the length refers to the number of characters in the string. When the data parameter is a byte array, the length refers to the number of bytes received.

Return value

Type: bool

When the string of a specified length is not received, it will return false; when it returns true, the receiving is completed.

Example

```
socket s
waituntil connect(s, "192.168.0.40" ,2888)
string data
waituntil read(s, data,5)
print data
byte bdata[3]
waituntil read(s, bdata,3)
```

Socket disconnection and reconnection function

When the socket is disconnected in the read process, ARCS will send an alarm and stop running. If you do not want the program to stop at this time, you can try to reconnect it after disconnection. You can set it in the ARL program, $DETECT_SOCK_CLOSE=true.

After disconnection, the current socket error status will be set to $ERR_SOCK_CLOSE. The user can get the error status of socket s via geterror(s).

The method to implement disconnection and reconnection is as follows:

The user can declare an interrupt. When the error status of a socket is $ERR_SOCK_CLOSED, an interrupt processing function will be triggered, and the socket connection will be re-established in the interrupt processing function.

Example:

```
socket s
func void handler()
close(s)
waituntil connect(s, "10.20.220.1", 8080)
endfunc
func viod main()
init()
$DETECT_SOCK_CLOSE=true
interrupt 1, when:geterror(s)== $ERR_SOCK_CLOSED, do:handler()
setip(.....)
```

```
waituntil connet(s,.....)
waituntil read (s,.....)
endfunc
```

5.6.1.7 readuntil(Receive data through socket)

Function prototype

```
bool readuntil(socket s,string data,string cut)
```

Description

Both the server and the client receive data via the function until the data of a specified character or string is read. The synchronous receiving must be implemented in conjunction with waituntil instruction, that is, the program will continue to run until the data of a specified character or string is received.

Parameter

Table5-62 Parameters of readuntil function

Name	Description
s	Type: socket
	It specifies the predefined socket variable
data	Type: string
	It specifies the predefined string variable
cut	Type: string
	It specifies the truncated character or string. If you want the character to be a linefeed, \n must be used as the cut parameter here.

Return value

Type: bool

When the specified character or string is not received, it will return false; when it returns true, the specified character or string is received.

Example

```
socket s
waituntil connect(s, "192.168.0.40" ,2888)
string data
//If the data sent by the other party is "hello", the data here will print he
waituntil readuntil(s, data, "hello" )
print data
```

5.6.1.8 clearbuff(Empty the input buffer of the socket)

Function prototype

```
bool clearbuff(socket s)
```

Description

It clears the socket's input buffer(When the input buffer is empty, no data can be received through the read function).

Parameter

Table5-63 Parameters of clearbuff function

Name	Description
s	Type: socket
	It specifies the predefined socket variables

Return value

Type: bool

When it returns true, the operation is successful. When it returns false, the operation fails.

Example

```
socket s
waituntil accept(s,"192.168.1.1",8080), maxtime:10,timeoutflag:time_flag
waituntil read(s,data,3), maxtime:10,timeoutflag:time_flag
clearbuff(s)
.....
```

5.6.2 Serial communication related functions

5.6.2.1 open(Open the serial device)

Function prototype

bool open(iodev& dev,string devname,int baud_rate,int character_size,stopbits stop_bits,parity prt)

Description

It opens and configures a serial device.

Parameter

Table5-64 Parameters of open function

Name	Description
dev	Type: iodev
	It specifies the predefined IO device
devname	Type: string
	■ When the cabinet type is ARCC series: It specifies the device name. "/dev/ttyS0" represents com1, and "/dev/ttyS1" represents com2 ■ When the cabinet type is ARCCD series: It specifies the device name. "/dev/ttyS0" represents com1

Name	Description
baud_rate	Type: int
	It specifies the baud rate, which can be set to 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, in bps
character_size	Type: int
	It specifies the character size, which can be set to 5, 6, 7, 8
stop_bits	Type: stopbits
	It specifies the stop bit type (see stopbits enumeration type), which can be set to 1 and 2
prt	Type: parity
	It specifies the parity type (see parity enumeration type), which can be set to no parity, odd check or even check

Return value

Type: bool

When it returns true, the device is opened successfully.

Example

```
iodev s
open(s, "/dev/ttyS0" ,9600,8,one,none)
waituntil write(s, " hello")
```

5.6.2.2 close(Close the serial device)

Function prototype

void close(iodev& dev)

Description

It closes a serial device that is opened previously.

Parameter

Table5-65 Parameters of close function

Name	Description
dev	Type: iodev
	It specifies the device that is opened previously

Return value

None

Example

```
iodev s
```

```
open(s, "/dev/ttyS0" ,9600,8,one,none)
waituntil write(s, "hello" )
close(s)
```

5.6.2.3 read/write/readuntil(Read and write serial device)

Description

The application of the above 3 functions is exactly the same as that of read/write/readuntil in the socket communication function, provided that the first parameter is replaced by iodev.

Example

```
iodev s
//open the port com1
open(s,"/dev/ttyS0",9600,8,one,none)
waittime 1
//write the string
write(s,"hello")
write(s,"world")
string data
//read the string by terminator
waituntil readuntil(s,data,"o")
print "data:",data
//Read the string by number
waituntil read(s,data,4)
//write the binary data
byte sdata[4]={1,2,3,4}
write(s,sdata,4)
close(s)
```

5.6.2.4 clearbuff(Clear the input buffer of the serial device)

Function prototype

```
bool clearbuff(iodev& dev)
```

Description

It clears the input buffer of the serial device.

Parameter

Table5-66 Parameters of clearbuff function

Name	Description
dev	Type: iodev
	It specifies the device that is opened previously

Return value

Type: bool

When it returns true, the operation is successful. When it returns false, the operation fails.

Example

```
iodev s
open(s, "/dev/ttyS0" ,9600,8,one,none)
waituntil write(s, "hello")
clearbuff(s)
……
close(s)
```

5.6.3 devicenet bus communication related functions

5.6.3.1 dnwrite(Send data to the devicenet bus)

Function prototype

int dnwrite(int offset,int len, byte[] data)

Description

It writes data to the devicenet bus buffer.

Parameter

Table5-67 Parameters of dnwrite function

Name	Description
offset	Type: int
	It specifies the offset address of the starting byte
len	Type: int
	It specifies the length of the write data, in byte
data	Type: byte []
	It specifies the binary data array to be sent

Return value

Type: int

When it returns 0, the write is successful. Non-zero represents the write failure, and the value represents the error code.

Example

```
byte bdata[3] = {1,2,3}
dnwrite (0,3,bdata) //Starting from byte 0, write 3 byte data to the devicenet device
```

5.6.3.2 dnread(Read data from the devicenet bus)

Function prototype

int dnread(int offset,int len, byte[] data)

Description

It reads data from the devicenet bus buffer.

Parameter

Table5-68 Parameters of dnread function

Name	Description
offset	Type: int
	It specifies the offset address of the starting byte
len	Type: int
	It specifies the length of the read data, in byte
data	Type: byte []
	It specifies the binary data array to be read

Return value

Type: int

When it returns 0, the read is successful. Non-zero represents the read failure, and the value represents the error code.

Example

byte bdata[3]

dnread (0,3,bdata) //Starting from byte 0, read 3 byte data from the devicenet device

5.6.4 ModBusTCP related functions

5.6.4.1 readregisters (read data from modbus register)

Function prototype

bool readregisters(string data, int start, int len)

bool readregisters(byte[] data, int start, int len)

Description

In the modbus register with start as the starting address and length as len (unit: register), read and convert to byte or string data, and store it in a predefined variable.

Parameter

Name	Type	Description
data	string/byte[]	Predefined byte/string type variables
start	int	The starting register address of the read data

len	int	Number of read registers

Return value

Type: bool

When the read is successful, it returns true; when the read fails, it returns false.

Example

byte bdata[4]

waituntil readregisters(bdata,100,2) // In the modbus register with a starting address of 100 and a length of 2 (unit: register), read and convert to byte data and store it in the bdata variable.

print bdata

string s

waituntil readregisters(s,200,2) // In the modbus register with a starting address of 200 and a length of 2 (unit: register), it is read and converted into string data and stored in the s variable.

print s

5.6.4.2 writeregisters (Write data to modbus register)

Function prototype

bool writeregisters(string data, int start)

bool writeregisters(byte[] data, int start, int len)

Description

Write the string or byte data into the modbus register whose starting address is start and whose length is len (unit: register).

Parameter

Name	Type	Description
data	string/byte[]	Predefined byte/string type variables
start	int	Start register address of write data
len	int	Number of write registers

Return value

Type: bool

Returns true when writing is successful; returns false when writing fails.

Example

byte bdata[4] = {1,2,3,4}

waituntil writeregisters(bdata,100,2) // Write the byte data bdata into the modbus register with the starting address of 100 and the length of 2 (unit: register).

string s = "abcd"

waituntil writeregisters(s,200,2) // Write the string data s into the modbus register with the starting address of 200 and the length of 2 (unit: register).

5.6.4.3 readcoils (Read data from modbus coil)

Function prototype

bool readcoils(byte[], int start, int length)

bool readcoils(byte[], int start, int length)

Description

The binary data to be sent is read from the modbus coil whose starting address is start and whose length is length (unit: bit).

Parameter

Name	Type	Description
byte[]	byte[]	Binary data to be sent
start	int	The starting register address of the read data
length	int	Read the number of coils, expressed as a decimal number

Return value

Type: bool

When the read is successful, it returns true; when the read fails, it returns false.

Example

```
byte bdata[2]
waituntil readcoils(bdata,100,16) // The bdata represents the binary data to be sent, 100 represents the starting address of the read coil, here it means reading from the 100th coil, and 16 means reading 16 coils.
print bdata
```

5.6.4.4 writecoils (Write data to modbus coil)

Function prototype

bool writecoils(byte[], int start, int length)

bool writecoils(byte[], int start, int length)

Description

Write the binary data into the modbus coil whose starting address is start and whose length is length (unit: bit).

Parameter

Name	Type	Description
byte[]	byte[]	Binary data to be sent
start	int	Start register address of write data

Name	Type	Description
length	int	Write the number of coils, expressed in decimal

Return value

Type: bool

Returns true when writing is successful; returns false when writing fails.

Example

byte bdata[2] = {1,2}

waituntil writecoils(bdata,100,16) // The bdata represents the binary array to be sent, 100 represents the starting address of the write coil, here it means writing from the 100th coil, and 16 means writing 16 coils.

5.6.5 ARL functions related to Melsec communication

5.6.5.1 open(Open the melsec slave device)

Function prototype

bool open(melsec_dev dev, string ip, int port, int slave_id)

Description

Open and configure a Melsec protocol PLC device.

Parameter

Table 5-69 Parameters of the open function

Name	Description
dev	Type: melsec_dev
	Pre-defined melsec socket variables
ip	Type: string
	Device ip, such as the value "192.168.1.101"
port	Type: int
	Port of the partner device to be connected
slave_id	Type: int
	Melsec slave station number, value range: 0~247 and 255; where 0 is the Melsec broadcast address; in TCP mode, 255 can be used to restore the default value.

Return value

Type: bool

True indicates successful opening and false indicates failed opening.

Example

melsec_dev mc //Define socket, similar to socket function

```
waituntil open(mc,"10.20.220.92",8080,255)
```

5.6.5.2 close(Turn off the melsec slave device)

Function prototype

```
void close(melsec_dev dev)
```

Description

Close an opened Melsec protocol PLC device

Parameter

Table 5-70 Parameters of the close function

Name	Description
dev	Type: melsec_dev
	Pre-defined melsec socket variable

Return value

Type: void

Example

```
close(mc) //Close the Melsec protocol PLC device
```

5.6.5.3 read(Receive data from the master station through melsec)

Function prototype

```
bool read(melsec_dev dev, string data, in start, int len)

bool read(melsec_dev dev, byte[] data, in start, int len)
```

Description

With start as the starting address and len as the length (unit: byte), read the contents of the plc register to a predefined string variable or byte array variable.

Both the server side and the client side receive data of the specified length through this function. It must cooperate with the waituntil instruction to achieve synchronous reception, that is, wait until the data reception is completed before continuing to run.

Parameter

Table 5-71 Parameters of the read function

Name	Description
dev	Type: melsec_dev
	Pre-defined melsec socket variable

Name	Description
data	Type: string or byte[]
	Pre-defined string variable or byte array variable
start	Type: int
	Start offset address
len	Type: int
	The received data length. When the data parameter is a string type, the length refers to the number of characters in the string; when the data parameter is a byte array type, the length refers to the number of bytes received.

Return value

Type: bool

When the string of the specified length is not received, false is returned; when it returns true, the reception is completed.

Example

```
melsec mc
waituntil open(mc,"10.20.220.92",8080,255)
string data
waituntil read(mc,data,6000,10)
byte bdata24]
waituntil read(mc,bdata,6000,24)
```

5.6.5.4 write(Send data to the master station through melsec)

Function prototype

bool write(melsec_dev dev, string data, in start, int len)

bool write(melsec_dev dev, string data, in start).

bool write(melsec_dev dev, byte[] data, in start, int len)

Description

With start as the starting address and len as the length (unit: byte), write the input string variable or byte array variable into the plc register.

Both the server side and the client side send data through this function. Must cooperate with waituntil instruction to realize synchronous transmission, that is to say, wait until data transmission is completed before continuing to run.

Parameter

Table 5-72 Parameters of the write function

Name	Description
dev	Type: melsec_dev

Name	Description
	Pre-defined melsec socket variable
data	Type: string or byte[]
	Pre-defined string variable or byte array variable
start	Type: int
	Start offset address
len	Type: int
	The length of the sent data. When the data parameter is of string type, the length refers to the number of characters in the string (the parameter defaults to the length of string); when the data parameter is of byte array type, the length refers to the received bytes Number.

Return value

Type: bool

When sending is not completed, it returns false; when it returns true, it means sending is completed.

Example

```
melsec mc
waituntil open(mc,"10.20.220.92",8080,255)
string data = "abcd"
waituntil write(mc1,data,6000)
byte bdata[4] = {1,2,3,4}
waituntil write (mc,bdata,6000,4)
```

5.6.6 Modbus-rtu communication related functions

5.6.6.1 open(Open modbus slave device)

Function prototype

bool open(modbus_dev& dev, string devname, int slave_id, int baud_rate, ParityType parity, int databits, StopBitType stop_bits)

Description

Turn on and configure a modbus slave device.

Parameter

Table 5-73 Parameter Description

Name	Parameter Type	Description
dev	modbus_dev	Pre-defined modbus devices
devname	string	Device name, for the second-generation cabinet, the device name is "rtserMB0"
slave_id	int	Modbus slave station number, value range 1-247

baud_rate	int	The baud rate can be set to 4800, 9600, 19200, 38400, 57600, 115200, the unit is bps
parity	ParityType	Parity check type, see modbus_parity enumeration type, can be set to no check, odd check or even check
databits	int	Data bit, can be set to 5, 6, 7, 8
stop_bits	StopBitType	Stop bit type, see modbus_stopbits enumeration type, it can be set to 1 bit or 2 bit

Return value

Type: bool

Return true to indicate that the operation was successful, and false to indicate that the operation failed.

Example

```
modbus_dev m
open(m,"rtserMB0",1,115200,one,8,none)
```

5.6.6.2 close(Turn off the modbus slave device)

Function prototype

void close(modbus_dev& dev)

Description

Close a previously opened modbus slave device.

Parameter

Table 5-74 Parameter Description

Name	Parameter Type	Description
dev	modbus_dev	Modbus device opened before

Return value

None

Example

```
modbus_dev m
open(m,"rtserMB0",1,115200,one,8,none)
close(m)
```

5.6.6.3 read(Receive data from the master station via modbus)

Function prototype

bool read(modbus_dev dev, string data, in start, int len)

bool read(modbus_dev dev, byte[] data, in start, int len)

Description

With start as the starting address and len as the length (unit: byte), read the contents of the modbus slave device register into a predefined string variable or byte array variable.

Parameter

Table 5-75 Parameter description

Name	Parameter Type	Description
dev	modbus_dev	Modbus device that has been opened
data	string/byte[]	Pre-defined string variable or byte array variable
start	int	The starting address of the data to be read
len	int	The length of the data to be read

Return value

Type: bool

When the string of the specified length is not received, false is returned; when it returns true, the reception is completed.

Example

```
modbus_dev m
open(m,"rtserMB0",1,115200,one,8,none)
string data
read(m, data, 0, 4)
close(m)
```

5.6.6.4 write(Send data to the master station via modbus)

Function prototype

bool write(modbus_dev dev,string data,in start)

bool write(modbus_dev dev,byte[] data,in start,int len)

Description

With start as the starting address and len as the length (unit: byte), write the string variable or byte array variable to be sent into the modbus slave device register.

Parameter

Table 5-76 Parameter Description

Name	Parameter Type	Description
dev	modbus_dev	Modbus device that has been opened
data	string/byte[]	String variable or byte array variable to be sent

start	int	The starting address of the data to be written
len	int	The length of the data to be written, this parameter is not required for writing string type data

Return value

Type: bool

When sending is not completed, it returns false; when it returns true, it means sending is completed.

Example

```
modbus_dev m
open(m,"rtserMB0",1,115200,one,8,none)
byte sdata[4]={31h,32h,33h,34h}
write(m, sdata, 0, 4)
close(m)
```

5.6.6.5 clearbuff(Clear the data buffer of the modbus slave)

Function prototype

bool clearbuff(modbus_dev dev)

Description

Clear the data buffer of the modbus slave (when the input buffer is empty, no data can be received through the read function).

Parameter

Table 5-77 Parameter Description

Name	Parameter Type	Description
dev	modbus_dev	Modbus device that has been opened

Return value

Type: bool

Return true to indicate that the operation was successful, and false to indicate that the operation failed.

Example

```
modbus_dev m
open(m,"rtserMB0",1,115200,one,8,none)
string data
read(m, data, 0, 4)
clearbuff(m)
close(m)
```

5.6.6.6 setcycle(Set the read and write cycle of communication with modbus devices)

Function prototype

void setcycle(modbus_dev& dev,int cycle)

Description

Set the read and write cycle of the communication between the control cabinet and the modbus device.

Parameter

Table 5-78 Parameter Description

Name	Parameter Type	Description
dev	modbus_dev	Modbus device that has been opened
cycle	int	Read and write cycle, unit: μs, parameter range: >500000

Return value

None

Example

```
modbus_dev m
open(m,"rtserMB0",1,115200,one,8,none)
setcycle(m,1000000)
clearbuff(m)
close(m)
```

5.7 Data type conversion functions

5.7.1 toint(Converted to integer)

Function prototype

int toint(double d) or

int toint(byte[] data,int start)

int toint(string number_string, base number_base)

Description

It converts the double data type, byte array, or string data to the integer. For byte array, the system will convert 4 bytes starting from the #start byte of the data array into the integer data.

Parameter

Table5-79 Parameters of toint function

Name	Description
d	Type: double
	It specifies the double data converted

Name	Description
data	Type: byte []
	It specifies the byte array converted
start	Type: int
	It specifies the starting offset address
number_string	Type: string
	It specifies the string data converted
number_base	Type: base
	It specifies the display number system of the string data

Return value

Type: int

It specifies the integer data converted.

Example

```
double value =3.1415
int s = toint(value)
print s //Output "3"
byte data[4] = {01h,02h,03h,04h}
print hex,toint(data,0) //Output "4030201h"
```

5.7.2 todouble(Converted to floating point)

Function prototype

double todouble(byte[] data,int start)

Description

It converts the byte array to the double type. The system will convert 8 bytes starting from the #start byte of the data array into the double data.

Parameter

Table5-80 Parameters of todouble function

Name	Description
data	Type: byte []
	It specifies the byte array converted
start	Type: int
	It specifies the starting offset address

Return value

Type: double

It specifies the double data converted.

Example

```
byte data[8] = {00h,00h,00h,00h,00h,00h,08h,40h}
print data //Output 3
byte data[8] = {00h,00h,00h,00h,00h,00h,08h,C0h}
print data //Output -3
byte data[8] = {00h,00h,00h,00h,00h,00h,00h,40h}
print data //Output 2
```

5.7.3 tobytes(Converted to byte array)

Function prototype

void tobytes(string/int/double src,byte[] dest,int start)

Description

It converts the string, int, or double data type to the byte array. int data will be converted to 4 bytes, and double data will be converted to 8 bytes. It is impossible to determine the length in which the string type is converted into byte. According to the strings to be converted, different strings have different lengths.

Parameter

Table5-81 Parameters of tobytes function

Name	Description
dest	Type: byte []
	It specifies the byte array in which the converted data is stored
start	Type: int
	It specifies the starting offset address stored

Return value

Type: void

Example

```
byte result[4]
tobytes(4030201h,result,0)
print hex,result //Output "{01h,02h,03h,04h}"
```

5.7.4 tostr(Forced conversion to string)

Function prototype

string tostr(anytype v)

string tostr(double v,int precision)

string tostr(int v, enum number_base) //Convert the integer to the string that is displayed in decimal/hexadecimal

Description

It converts the data of any type to the string type.

Parameter

Table5-82 Parameters of tostr function

Name	Description
v	Type: It can be any data type
	It specifies the expression of the string converted ■ When v is double, you can specify the significant digits that are reserved in the second parameter precision. If the parameter is defaulted, 6 significant digits will be reserved by default. ■ When v is int, you can specify the number system in which the second parameter number_base is displayed. If the parameter is defaulted, it will be displayed in decimal by default.
precision	Type: int
	Precision, the number of significant digits reserved.
number_base	Type: enum
	Base conversion type, the values are as follows: ■ hex：Hexadecimal ■ dec：Decimal

Return value

Type: string

It returns the string converted.

Example

```
double value =3.1415
string s = tostr(value)
print s //Output "3.1415"
```

5.7.5 ftobytes (Convert floating point to bytes)

Function prototype

void ftobytes(double src, byte[] data,int start)

Description

Convert the double data type to a byte array with a length of 4 bytes. This conversion will lose numerical precision.

Parameter

Name	Description
src	Type: double Double data to be converted

data	Type: byte[] The byte array into which the converted data is stored
start	Type: int The starting offset address of the storage

Return value

Type: void

Example

```
byte result[4]
double b = 1.23
ftobytes(b,result,0)
```

5.7.6 tofloat (Convert bytes to floating point)

Function prototype

double tofloat(byte[] data,int start)

Description

Convert the byte array to float. The system will convert the 4 bytes starting from the start byte of the data array to double data.

Parameter

Name	Description
data	Type: byte[] The byte array to be converted
start	Type: int Start offset address

Return value

Type: double

Converted floating point data.

Example

```
byte result[4]
double b = 1.23
ftobytes(b,result,0)
print tofloat(result, 0) // Output 1.23
```

5.8 Robot pose function

5.8.1 cjoint(Get current axis position)

Function prototype

joint cjoint()

Description

It gets the current axis position, including the external axis. Cannot be used in the background.

Parameter

None

Return value

Type: joint

It returns the current axis position.

Example

```
joint a = {j1 0,j2 10,j3 20,j4 30,j5 40,j6 50}
movej a
waittime 0
print cjoint() //Output "{ 0, 10, 20, 30, 40, 50, 9e+09, 9e+09, 9e+09, 9e+09, 9e+09, 9e+09}"
//The actual print result may be slightly different from the set value due to the error (within the position accuracy range).
```

5.8.2 cpose(Get current TCP pose)

Function prototype

pose cpose(tool t,wobj w)

Description

It gets the current TCP pose and the external axis position.

Parameter

Table5-83 Parameters of cpose function

Name	Description
t	Type: tool
	It specifies the current tool
w	Type: wobj
	It specifies the workpiece coordinate system referenced

Return value

Type: pose

It returns the current TCP pose and the external axis position.

Example

```
pose p = { x 763.869,y 207.323,z 1422.841,a 129.538,b 0.480,c 92.084,cfg 0}
ptp p
waittime 0
wobj wobj1 = {{10,0,0,0,0,0}}
print cpose($FLANGE,wobj1) //Output "{753.869,207.323,1422.841,129.538,0.480,92.084,0,-1,9e+09, 9e+09, 9e+09, 9e+09, 9e+09, 9e+09}"
```

5.8.3 getpose(Get the TCP pose corresponding to a certain group of axis positions, that is, the positive kinematics solution)

Function prototype

```
pose getpose(joint j,tool t,wobj w)
```

Description

It gets the TCP pose corresponding to the axis position j and the external axis position.

Parameter

Table5-84 Parameters of getpose function

Name	Description
j	Type: joint
	It specifies the robot axis position and the external axis position
t	Type: tool
	It specifies the specified tool
w	Type: wobj
	It specifies the workpiece coordinate system referenced

Return value

Type: pose

It returns the corresponding TCP pose and the external axis position.

Example

```
joint j = {j1 0,j2 10,j3 20,j4 30,j5 40,j6 50}
wobj wobj1 = {{10,0,0,0,0,0}}
print getpose(j,$FLANGE,wobj1)
```

5.8.4 getjoint(Get the robot axis position corresponding to a certain TCP pose, that is, the inverse kinematics solution)

Function prototype

```
joint getjoint(pose p,tool t,wobj w)
```

Description

It gets the robot axis position corresponding to the pose p.

Parameter

Table5-85 Parameters of getjoint function

Name	Description
p	Type: pose
	It specifies the robot pose, Refer to *Chapter 2.4.4*
t	Type: tool
	It specifies the specified tool, Refer to *Chapter 2.4.6*
w	Type: wobj
	It specifies the workpiece coordinate system referenced, Refer to *Chapter 2.4.7*

Return value

Type: joint

It returns the corresponding robot axis position data.

Example

```
pose p = { x 763.869,y 207.323,z 1422.841,a 129.538,b 0.480,c 92.084,cfg 0}
print getjoint(p,$FLANGE,$WORLD)
```

5.8.5 poseinv(Calculate the inverse pose of a pose)

Function prototype

pose poseinv(pose p)

Description

It calculates the inverse pose of a pose.

Parameter

Table5-86 Parameters of poseinv function

Name	Description
p	Type: pose
	It enters the pose of the coordinate system A in the coordinate system B

Return value

Type: pose

It returns the pose of the coordinate system B in the coordinate system A.

Example

```
pose p = { x 1,y 2,z 3,a 0,b 0,c 0 }
print poseinv(p)
//Output "{-1,-2,-3,0,0,0,0,-1,9e+09, 9e+09, 9e+09, 9e+09, 9e+09, 9e+09}"
```

5.8.6 offset(The displacement function of the target point relative to the workpiece coordinate system)

Function prototype

pose offset(pose p,double dx,double dy,double dz,double rz,double ry,double rx)

Description

The offset function is used to translate the specified target point relative to the workpiece coordinate system and get the new target point after translation.

Parameter

Table5-87 Parameters of offset function

Name	Description
p	Type: pose
	It specifies the current target
dx	Type: double
	It specifies the amount of translation along the x direction of the workpiece coordinate system, in mm
dy	Type: double
	It specifies the amount of translation along the y direction of the workpiece coordinate system, in mm
dz	Type: double
	It specifies the amount of translation along the z direction of the workpiece coordinate system, in mm
rz	Type: double
	It specifies to the amount of rotation around the z direction of the workpiece coordinate system, in degrees (°)
ry	Type: double
	It specifies to the amount of rotation around the y direction of the workpiece coordinate system, in degrees (°)
rx	Type: double
	It specifies to the amount of rotation around the workpiece coordinate system in the x direction, in degrees (°)

If multiple values of rz, ry, rx are specified at the same time, the ZYX type Euler angle is used for rotation, that is, the rotation sequence is first to rotate the rz angle around the z axis, then rotate the ry angle around the new y axis, and finally Rotate by rx angle around the new x axis.

Return value

Type: pose

It specifies the target point after translation.

Example

```
pose p = {x 1500,y 500,z 500,a 0,b 0,c 0}
double dx = 10
double dy = 20
double dz = 30
double rz = 60
double ry = 50
double rx = 40
p=offset (p,dx,dy,dz,rz,ry,rx)
ptp p:p,vp:5%,sp:5%
waittime 0
print cpose() //Output "{1510, 520, 530, 40, 50, 60, 9e+09, 9e+09, 9e+09, 9e+09, 9e+09, 9e+09}".The actual printing result may slightly deviate from the set value due to errors (within the range of position accuracy).
```

5.8.7 reltool(The displacement function of the target point relative to the tool coordinate system)

Function prototype

pose reltool(pose p,double dx,double dy,double dz,double rz,double ry,double rx)

Description

The reltool function is used to translate and rotate the specified target point relative to the tool coordinate system, and get the new target point after shift.

Parameter

Table5-88 Parameters of reltool function

Name	Description
p	Type: pose
	It specifies the current target
dx	Type: double
	It specifies the amount of translation along the x direction of the tool coordinate system, in mm
dy	Type: double
	It specifies the amount of translation along the y direction of the tool coordinate system, in mm
dz	Type: double
	It specifies the amount of translation along the z direction of the tool coordinate system, in mm
rz	Type: double
	It specifies the amount of rotation around the z direction of the tool coordinate system, in deg
ry	Type: double
	It specifies the amount of rotation around the y direction of the tool coordinate system, in deg
rx	Type: double
	It specifies the amount of rotation around the x direction of the tool coordinate system, in deg


Notice

The rotation in ZYX Euler angle is used here, that is, the rotation order is to rotate the rz angle around the z axis, then rotate the ry angle around the new y axis, and finally rotate the rx angle around the new x axis.

Return value

Type: pose

It specifies the target point after shift.

Example

```
pose p1= {x 1500,y 500,z 500,a 0,b 0,c 0}
double dx = 10
double dy = 20
double dz = 30
double rz = 40
double ry = 50
double rx = 60
pose p2 = reltool(p1, dx, dy, dz, rz, ry, rx)
ptp p:p2,vp:5%,sp:5%
waittime 0
print cpose($FLANGE,WORLD) // Output "{1510,520,530,40,50,60,0,-1,9e+09,9e+09, 9e+09, 9e+09, 9e+09, 9e+09}".The actual printing result may deviate slightly from the set value due to errors (within the range of position accuracy)
```

5.8.8 cjttq(Get the output torque of each axis of the robot)

Description

The jttq structure type is used to directly describe the output torque of the robot axis. The function is used to get the output torque (in N*m) of the current axis of the specified foreground channel. It can be used in the foreground channel and the background channel.

Format

```
jttq cjttq(int chan_no, bool ls_cmd)
```

Definition

```
struct jttq
{
double jt1
double jt2
double jt3
double jt4
```

double jt5

double jt6

double et1

double et2

double et3

double et4

double et5

double et6

}

Parameters

The parameters of the cjttq instruction are shown in Table5-89.

Table5-89 Parameters of cjttq instruction

Name	Description
j1~j6	Type: double
	It specifies the output torque (in N*M) of the robot J1~J6 axis
et1~et6	Type: double
	It specifies the output torque (in N*M) of the external axes J1~J6

Parameter

The parameters of the cjttq instruction are shown in Table5-90.

Table5-90 Parameters of cjttq instruction

Name	Description
chan_no	Type: int
	It specifies the front channel number
ls_cmd	Type: bool
	It indicates whether the instruction torque is output: true - output of instruction torque, false - output of feedback torque

Return value

Type: jttq

It returns the output torque (in N*m) of the current axis in the specified foreground channel.

Example

//Foreground channel #1 executes the program

```
pose p = { x 763.869,y 207.323,z 1422.841,a 129.538,b 0.480,c 92.084,cfg 0}
ptp p
print cjttq (1, true) //Output "{*, *, *, *, *, *, *, *, *, *, *, *, *, *}". Where * indicates that the torque is unknown. The specific value of * reflects the torque value of the axis at the moment.
```

5.8.9 cjtci(Get the motor current of each axis of the robot)

Description

The jtci structure is used to directly describe the motor current Iq of the robot axis. The function is used to get the motor current (in A) of the current axis in the specified foreground channel. It can be used in the foreground channel and the background channel.

Format

```
jtci cjtci(int chan_no, bool Is_cmd)
```

Definition

```
struct jtci
{
double ji1
double ji2
double ji3
double ji4
double ji5
double ji6
double ei1
double ei2
double ei3
double ei4
double ei5
double ei6
}
```

Parameters

The parameters of the cjtci instruction are shown in Table5-91.

Table5-91 Parameters of cjtci instruction

Name	Description
ji1~ji6	Type: double
	It specifies the motor current Iq of the robot axes J1~J6, in A
ei1~ei6	Type: double
	It specifies the motor current Iq of the external axes J1~J6, in A

Parameter

The parameters of the cjtci instruction are shown in Table5-92.

Table5-92 Parameters of cjtci instruction

Name	Description
chan_no	Type: int
	It specifies the front channel number
ls_cmd	Type: bool
	It indicates whether the instruction current is output: true - output of instruction current, false - output of feedback current

Return value

Type: jtci

It returns the motor current (in A) of the current axis in the specified foreground channel.

Example

```
//Foreground channel #1 executes the program
pose p = { x 763.869,y 207.323,z 1422.841,a 129.538,b 0.480,c 92.084,cfg 0}
ptp p
print cjtci (1, true) //Output "{*, *, *, *, *, *, *, *, *, *, *, *, *}". Where * indicates that the current is unknown. The specific value of * reflects the motor current of the axis at the moment.
```

5.8.10 channeltojoint(Get the axis position of the target point of the robot in a channel)

Description

It gets the axis position of the operating target point and the external axis position in the specified foreground channel, which can be used in the foreground channel and the background channel.

Format

joint channeltojoint(int chan_no)

Parameter

The parameters of the channeltojoint instruction are shown in Table5-93.

Table5-93 Parameters of channeltojoint instruction

Name	Description
chan_no	Type: const int
	It specifies the front channel number

Return value

Type: joint

It returns the position of the axis of the operating target point in the specified channel.

Example

```
//Foreground channel #1 executes the program
joint a = {j1 0,j2 10,j3 20,j4 30,j5 40,j6 50}
movej a
print channeltojoint(1) //Output "{ 0, 10, 20, 30, 40, 50, 9e+09, 9e+09, 9e+09, 9e+09, 9e+09, 9e+09}"
```

5.8.11 channeltopose(Get the TCP pose of the target point of the robot in a certain channel)

Description

It gets the TCP pose and the external axis position of the operating target point in the specified foreground channel. It can be used in the foreground channel and the background channel.

Format

```
pose channeltopose(int chan_no,tool t,wobj w)
```

Parameter

The parameters of the channeltopose instruction are shown in Table5-94.

Table5-94 Parameters of channeltopose instruction

Name	Description
chan_no	Type: int
	It specifies the front channel number
t	Type: tool
	It specifies the current tool
w	Type: wobj
	It specifies the workpiece coordinate system referenced

Return value

Type: pose

It returns the current TCP pose and the external axis position in the specified foreground channel.

Example

```
//Foreground channel #1 executes the program
pose p = { x 763.869,y 207.323,z 1422.841,a 129.538,b 0.480,c 92.084,cfg 0}
ptp p
wobj wobj1 = {{10,0,0,0,0,0}}
print channeltopose(1,$FLANGE,wobj1) //Output "{753.869,207.323, 1422.841,129.538,0.480,92.084,0,-1,9e+09, 9e+09, 9e+09, 9e+09, 9e+09, 9e+09}"
```

5.8.12 channeljoint(Get the current axis position of the specified foreground channel)

Description

It gets the current axis position and the external axis position of the specified foreground channel, which can be used in the foreground channel and the background channel.

Format

joint channeljoint(int chan_no,bool ls_cmd)

Parameter

The parameters of the channeljoint instruction are shown in Table5-95.

Table5-95 Parameters of channeljoint instruction

Name	Description
chan_no	Type: int
	It specifies the front channel number
ls_cmd	Type: bool
	It indicates whether the instruction position is output: ■ true - output of instruction position, ■ false - output of feedback position

Return value

Type: joint

It returns the current axis position of the specified foreground channel.

Example

```
//Foreground channel #1 executes the program
joint a = {j1 0,j2 10,j3 20,j4 30,j5 40,j6 50}
movej a
waittime 0
//The background channel executes the program
print channeljoint(1,true) //Output "{ 0, 10, 20, 30, 40, 50, 9e+09, 9e+09, 9e+09, 9e+09, 9e+09, 9e+09}"
```

5.8.13 channelpose(Get the current TCP pose of the specified foreground channel)

Description

It gets the current TCP pose and the external axis position of the specified foreground channel, which can be used in the foreground channel and the background channel.

Format

pose channelpose(int chan_no,tool t,wobj w,bool ls_cmd)

Parameter

The parameters of the channelpose instruction are shown in Table5-96.

Table5-96 Parameters of channelpose instruction

Name	Description
chan_no	Type: int
	It specifies the front channel number
t	Type: tool
	It specifies the current tool
w	Type: wobj
	It specifies the workpiece coordinate system referenced
ls_cmd	Type: bool
	It indicates whether the instruction position is output: ■ true - output of instruction position ■ false - output of feedback position

Return value

Type: pose

It returns the current TCP pose and the external axis position in the specified foreground channel.

Example

```
//Foreground channel #1 executes the program
pose p = { x 763.869,y 207.323,z 1422.841,a 129.538,b 0.480,c 92.084,cfg 0}
ptp p
waittime 0
wobj wobj1 = {{10,0,0,0,0,0}}
//The background channel executes the program
print channelpose(1,$FLANGE,wobj1,true) //Output "{753.869,207.323, 1422.841,129.538,0.480,92.084,0,-1,9e+09, 9e+09, 9e+09, 9e+09, 9e+09, 9e+09}"
```

5.8.14 channeljointvel(Get the speed of each axis of the robot)

Description

It gets the speed (in rad/s) of the current axis of the specified foreground channel, including the external axis. It can be used in the foreground channel and the background channel.

Format

jvel channeljointvel(int chan_no,bool ls_cmd)

Parameter

The parameters of the channeljointvel instruction are shown in Table5-97.

Table5-97 Parameters of channeljointvel instruction

Name	Description
chan_no	Type: const int
	It specifies the front channel number
ls_cmd	Type: bool
	It indicates whether the instruction speed is output: true - output of instruction speed, false - output of feedback speed

Return value

Type: jvel

It returns the speed (in rad/s) of the current axis of the specified foreground channel, please refer to *Chapter 2.4.11* for the definition of jvel structure.

Example

//Foreground channel #1 executes the program

joint a = {j1 0,j2 10,j3 20,j4 30,j5 40,j6 50}

movej a

print channeljointvel(1,$FLANGE,wobj1,true) //Output "{ *, *, *, *, *, *, 9e+09, 9e+09, 9e+09, 9e+09, 9e+09, 9e+09}" . Where * indicates that the current axis speed is unknown. The specific value of * reflects the speed at the moment.

5.8.15 channeltcpvel(Get the robot TCP point speed)

Description

It gets the speed of the current TCP of the specified foreground channel, in mm/s. It can be used in the foreground channel and the background channel.

Format

double channeltcpvel(int chan_no, tool t,wobj w,bool ls_cmd)

Parameter

The parameters of the channeltcpvel instruction are shown in Table5-98.

Table5-98 Parameters of channeltcpvel instruction

Name	Description
chan_no	Type: const int
	It specifies the front channel number
t	Type: tool
	It specifies the tool that is used currently
w	Type: wobj
	It specifies the workpiece coordinate system that is referenced currently
ls_cmd	Type: bool
	It indicates whether the instruction speed is output: true - output of instruction speed, false - output of feedback speed

Return value

Type: double

It returns the speed (in mm/s) of the current TCP of the specified foreground channel.

Example

```
//Foreground channel #1 executes the program
joint a = {j1 0,j2 10,j3 20,j4 30,j5 40,j6 50}
movej a
print channeltcpvel(1,$FLANGE,wobj1,true) //Output "*". "*" indicates that the current TCP speed for printing is unknown. The specific value of * reflects the speed at the moment.
```

5.8.16 ctcpforce(Get the six-dimensional force vector of the robot TCP point)

Description

The tcpforce structure describes the robot's TCP output force and torque via Cartesian coordinates. The function is used to get the output force (in N) and output torque (in N*m) of the current TCP of the specified foreground channel. It is used in the foreground channel.

Format

tcpforce ctcpforce(int chan_no, tool t,wobj w)

Definition

```
struct tcpforce
{
double fx
double fy
double fz
```

double tx

double ty

double tz

}

Parameters

The parameters of the ctcpforce instruction are shown in Table5-99.

Table5-99 Parameters of ctcpforce instruction

Name	Description
fx	Type: double
	It specifies the component x of the robot TCP output force relative to the current workpiece coordinate system coordinate, in N
fy	Type: double
	It specifies the component y of the robot TCP output force relative to the current workpiece coordinate system coordinate, in N
fz	Type: double
	It specifies the component z of the robot TCP output force relative to the current workpiece coordinate system coordinate, in N
tx	Type: double
	It specifies the component x of the robot tool output torque in the current workpiece coordinate system coordinate, in N*m
ty	Type: double
	It specifies the component y of the robot tool output torque in the current workpiece coordinate system coordinate, in N*m
tz	Type: double
	It specifies the component z of the robot tool output torque in the current workpiece coordinate system coordinate, in N*m

Parameter

The parameters of the ctcpforce instruction are shown in Table5-100.

Table5-100 Parameters of ctcpforce instruction

Name	Description
chan_no	Type: int
	It specifies the front channel number
t	Type: tool
	It specifies the tool that is used currently

Name	Description
w	Type: wobj
	It specifies the workpiece coordinate system that is referenced currently

Return value

Type: tcpforce

It returns the output force (in N) and output torque (in N*m) of the current TCP of the specified foreground channel.

Example

//Foreground channel #1 executes the program

pose p = { x 763.869,y 207.323,z 1422.841,a 129.538,b 0.480,c 92.084,cfg 0}

ptp p

wobj wobj1 = {{10,0,0,0,0,0}}

print ctcpforce (1, $FLANGE, wobj1) //Output "{*, *, *, *, *, *, *}". "*" indicates that the force and torque are unknown. The specific values of * represent the force and torque values of the TCP at the moment.

5.9 Coordinate system calibration function

5.9.1 getwobj_3p(3-point method to calibrate the workpiece coordinate system)

Function prototype

wobj getwobj_3p(joint j1, joint j2, joint j3,tool t)

Description

The workpiece coordinate system is calibrated through 3 teach points.

Parameter

Table5-101 Parameters of getwobj_3p function

Name	Description
j1	Type: joint
	It specifies the teach point corresponding to the origin of the workpiece coordinate system to be calibrated
j2	Type: joint
	It specifies the teach point corresponding to a positive X-axis point of the workpiece coordinate system to be calibrated
j3	Type: joint
	It specifies the teach point corresponding to a point where the Y axis is positive on the XY plane of the workpiece coordinate system to be calibrated
t	Type: tool
	It specifies the tool that is used during calibration

Return value

Type: wobj

It returns the calculated calibration results of the workpiece coordinate system.

Example

```
joint j1 = { j1 1, j2 0.005,j3 120.001,j4 0.001,j5 30.015, j6 -0.139}
joint j2 = { j1 10,j2 0.005,j3 120.001,j4 0.001,j5 30.015, j6 -0.139}
joint j3 = { j1 15,j2 0.005,j3 120.001,j4 0.001,j5 30.015, j6 -0.139}
print getwobj_3p(j1,j2,j3,$TOOL0)
```

5.9.2 getwobj_indi(Indirect method to calibrate the workpiece coordinate system)

Function prototype

wobj getwobj_indi(joint j1, joint j2, joint j3,pos p1,pos p2,pos p3,tool t)

Description

The workpiece coordinate system is indirectly calibrated through 3 test points on coordinates in the workpiece coordinate system to be calibrated.

Parameter

Table5-102 Parameters of getwobj_indi function

Name	Description
j1	Type: joint
	It specifies the teach point corresponding to the point p1
j2	Type: joint
	It specifies the teach point corresponding to the point p1
j3	Type: joint
	It specifies the teach point corresponding to the point p3
p1	Type: pos
	It specifies the coordinates of point p1 in the workpiece coordinate system to be calibrated
p2	Type: joint
	It specifies the coordinates of point p2 in the workpiece coordinate system to be calibrated
p3	Type: joint
	It specifies the coordinates of point p3 in the workpiece coordinate system to be calibrated
t	Type: tool
	It specifies the tool that is used during calibration

Return value

Type: wobj

It returns the calculated calibration results of the workpiece coordinate system.

Example

```
joint j1 = j:{ j1 1,j2 0.005,j3 120.001,j4 0.001,j5 30.015 ,j6-0.139}
joint j2 = j:{ j1 10,j2 0.005,j3 120.001,j4 0.001,j5 30.015 ,j6-0.139}
joint j3 = j:{ j1 15,j2 0.005,j3 120.001,j4 0.001,j5 30.015 ,j6-0.139}
pos p1 = {10,20,20}
pos p2 = {40,60,20}
pos p3 = {50,20,40}
print getwobj_indi(j1,j2,j3,p1,p2,p3, $TOOL0)
```

5.9.3 getwobj_flange(Flange reference method to calibrate the workpiece coordinate system)

Function prototype

```
wobj getwobj_flange(joint j1, joint j2, tool t,bool xydefault)
```

Description

The function uses a standard tool to calibrate the origin of the workpiece coordinate system or the external fixed tool coordinate system, and calibrates the direction of the workpiece coordinate system or the external fixed tool coordinate system with reference to the robot's flange coordinate system. The calibration method is mainly used for the calibration of the external tool coordinate system.

Calibration method:

- Install the standard tool on the robot's flange.
- Move the TCP of the standard tool to the origin of the coordinate system to be measured, and record the position and angle of the axis at this time.
- Move the Z-direction of the flange coordinate system parallel to the Z+ direction of the coordinate system to be measured.
- If you want to calibrate the X and Y directions of the coordinate system to be measured, you must move the X+ direction of the flange coordinate system parallel to the X+ direction of the coordinate system to be measured. If you take no notice of the X and Y directions of the coordinate system to be measured, this step can be omitted.
- Record the position and angle of the axis at this time.
- Calculate the function to get the workpiece coordinate system to be measured.

Parameter

Table5-103 Parameters of getwobj_flange function

Name	Description
j1	Type: joint
	It specifies the axis position that is recorded during calibration of origin
j2	Type: joint
	It specifies the axis position that is recorded during calibration of direction
t	Type: tool
	It specifies the standard tool that is installed at the flange end during calibration of origin. Only the components x,

Name	Description
	y & z of the tool are used here, and the components a, b & c are ignored
xydefault	Type: bool
	It indicates whether the directions of x & y axes are defaulted ■ true: only the direction of z axis is calibrated, and the directions of x & y axes is defaulted by the system ■ false: the directions of x, y & z axes are calibrated

Return value

Type: wobj

It returns the calculated calibration results of the workpiece coordinate system.

Example

```
tool knowntool = {{0,0,0,0,0,0}}
joint j1 = { j1 4.229 ,j2 42.310 ,j3 84.665, j4 88.315, j5 84.614 ,j6 -36.429}
joint j2 = { j1 3.456, j2 41.824, j3 86.464 ,j4 92.073, j5 84.982, j6 -36.506}
wobj w = getwobj_flange(j1,j2,knowntool,true)
print w
$TOOLS[0].t_frame = w.w_frame
$TOOLS[0].stationary = true
```

5.9.4 gettooltcp_ref(Use the standard tool reference method to calibrate the tool coordinate system xyz)

Function prototype

void gettooltcp_ref(joint j1, joint j2, tool ref,tool& t)

Description

Calibrate x, y & z of the tool coordinate system with a standard tool.

Calibration method:

- Select a reference point and a reference tool with known TCP coordinates.
- Install the reference tool on the robot's flange, manually control the robot to approach the reference point, and record the teach point.
- Remove the reference tool, install the tool to be measured on the robot's flange, manually control the robot to approach the reference point, and record the teach point.

Parameter

Table5-104 Parameters of gettooltcp_ref function

Name	Description
j1	Type: joint
	It specifies the corresponding teach point when installing the reference tool
j2	Type: joint
	It specifies the corresponding teach point when installing the tool to be measured

Name	Description
tool	Type: ref
	It specifies the reference tool. It is only necessary to know the values of x, y & z
tool	Type: tool
	It specifies the tool coordinate system to be calibrated. The calibration result will be saved to x, y & z of the variable.

Return value

None

Example

```
joint j1 = j:{ j1 1,j2 0.005,j3 120.001,j4 0.001,j5 30.015 ,j6-0.139}
joint j2 = j:{ j1 10,j2 0.005,j3 120.001,j4 0.001,j5 30.015 ,j6-0.139}
tool ref = {{x 10,y 10,z 50}}
tool t
gettooltcp_ref(j1,j2,ref,t)
print t
```

5.9.5 gettoolrot_world(Use world coordinate system reference method to measure tool coordinate system abc)

Function prototype

void gettoolrot_world(joint j,tool& t)

Description

Calculate the a, b & c of the tool coordinate system through adjusting the axis of the tool coordinate system to be parallel with the axis of the world coordinate system.

Calibration method:

- Keep $+X_{TOOL}$ to be parallel with $-Z_{WORLD}$, $+Y_{TOOL}$ to be parallel with $+Y_{WORLD}$, $+Z_{TOOL}$ to be parallel with $+X_{WORLD}$, and record the teach points at this time.

Parameter

Table5-105 Parameters of gettoolrot_world function

Name	Description
j	Type: joint
	It specifies the teach point recorded
tool	Type: tool
	It specifies the tool coordinate system to be calibrated. The calibration results will be saved to a, b & c of the variable.

Return value

None

Example

```
joint j = j:{ j1 1,j2 0.005,j3 120.001,j4 0.001,j5 30.015 ,j6-0.139}
tool t
gettoolrot_world (j,t)
print t
```

5.9.6 gettoolrot_3p(Use 3-point method to measure tool coordinate system abc)

Function prototype

void gettoolrot_3p(joint j1,joint j2,joint j3,tool& t)

Description

Calibrate a, b & c of the tool coordinate system through 3-point method.

Calibration method:

- Move the tool TCP to any reference point and record the teach point at this time.
- Move a point in the negative direction of X axis of the tool coordinate system to the reference point, and record the teach point at this time.
- Move the point where Y axis on the XY plane of the tool coordinate system is negative to the reference point, and record the teach point at this time.

Parameter

Table5-106 Parameters of gettoolrot_3p function

Name	Description
j1	Type: joint
	It specifies the teach point that is obtained in step 1
j2	Type: joint
	It specifies the teach point that is obtained in step 2
j3	Type: joint
	It specifies the teach point that is obtained in step 3
tool	Type: tool
	It specifies the tool coordinate system to be calibrated. The components x, y & z of the component t_frame must be provided with the correct values. The calibration results will be saved to the components a, b & c of the component t_frame of the variable.

Return value

None

Example

```
joint j1 = j:{ j1 1,j2 0.005,j3 120.001,j4 0.001,j5 30.015 ,j6-0.139}
```

```
joint j2 = j:{ j1 10,j2 0.005,j3 120.001,j4 0.001,j5 30.015 ,j6-0.139}
joint j3 = j:{ j1 15,j2 0.005,j3 120.001,j4 0.001,j5 30.015 ,j6-0.139}
tool t
gettoolrot_3p (j1, j2, j3, t)
print t
```

5.9.7 gettool_3p(Use the 3-point method to calibrate the tool coordinate system)

Function prototype

tool gettool_3p(joint j1, joint j2, joint j3,wobj w)

Description

The function calibrates the tool coordinate system defined relative to the flange coordinate system or the robot gripping workpiece coordinate system using a reference point with known coordinates. The calibration method is mainly used for calibrating the robot gripping workpiece coordinate system.

Calibration method:

- Move the origin of the coordinate system to the known reference point, and record the teach point at this time.
- Move a point in the positive direction of X axis of the coordinate system to the known reference point, and record the teach point at this time.
- Move a point where Y axis on the XY plane of the coordinate system is positive to the known reference point, and record the teach point at this time.

Parameter

Table5-107 Parameters of gettool_3p function

Name	Description
j1	Type: joint
	It specifies the teach point that is obtained in step 1
j2	Type: joint
	It specifies the teach point that is obtained in step 2
j3	Type: joint
	It specifies the teach point that is obtained in step 3
w	Type: wobj
	It specifies the reference point with the known coordinate values in the world coordinate system. Only the components x, y & z of the workpiece coordinate system are used here, and the components a, b & c are ignored

Return value

Type: tool

It returns the calculated calibration result of the tool coordinate system.

Example

```
wobj w = {{935,0,1300,0,0,0}}
```

```
joint j1 = {0,0,90,0,90,0}
joint j2 = {0,0,90,0,-90,0}
joint j3 = {0,0,90,90,90,0}
tool t = gettool_3p(j1,j2,j3,w)
print t
$WOBJS[0].w_frame = t.t_frame
$WOBJS[0].robhold = true
```

5.9.8 getbase_3p(Use the 3-point method to calibrate the basic coordinate system)

Function prototype

void getbase_3p(joint j1, joint j2, joint j3,tool t, int index)

Description

The base coordinate system is calibrated through 3 teaching points.

Parameter

Table5-108 Parameters of getbase_3p function

Name	Description
j1	Type: joint
	It specifies the teach point corresponding to the origin of the world coordinate system
j2	Type: joint
	It specifies the teach point corresponding to a positive point of X axis in the world coordinate system
j3	Type: joint
	It specifies the teach point corresponding to a point where Y axis on the XY plane of the world coordinate system is positive
t	Type: tool
	It specifies the tool that is used during calibration
index	Type:int
	The serial number of the mechanical unit, the parameter range: [1,n]. Where n is the total number of mechanical units of the current channel

Return value

None

Example

```
joint j1 = j:{ j1 1,j2 0.005,j3 120.001,j4 0.001,j5 30.015 ,j6-0.139}
joint j2 = j:{ j1 10,j2 0.005,j3 120.001,j4 0.001,j5 30.015 ,j6-0.139}
joint j3 = j:{ j1 15,j2 0.005,j3 120.001,j4 0.001,j5 30.015 ,j6-0.139}
getbase_3p(j1,j2,j3, $TOOL0,1)
print $BASE
```

5.10 Trajectory trigger related function

5.10.1 T(Whether the current trajectory reaches a certain point in time)

Function prototype

bool T(double t)

Description

It returns whether the current movement trajectory has been t seconds away from the starting point. The function is mainly used for definition of the event in trajectory trigger.

Parameter

Table5-109 Parameters of T function

Name	Description
t	Type: double
	in s

Return value

Type: bool

- false: The current movement trajectory has not reached t.
- true: The current movement trajectory has reached t.

Example

trigger 1, when: T (1), do: setdo (1, true) //declare a trajectory trigger on the following trajectory: When the time from the starting point is 1 s, #1 DO output signal is true

movej j:{j1 30},v:{per 2}

5.10.2 S(Whether the current trajectory reaches a certain distance point)

Function prototype

bool S (double s)

Description

It returns whether the current movement trajectory has moved s mm away from the starting point. The function is mainly used for definition of the event in trajectory trigger.

Parameter

Table5-110 Parameters of S function

Name	Description
s	Type: double
	in mm

Return value

Type: bool

- false: The current movement trajectory has not reached the position point away from the starting point s.
- true: The current movement trajectory has reached the position point from the starting point s.

Example

trigger 1, when: S (100), do: setdo (1, true) //Declare a trajectory trigger on the following trajectory: When the distance from the starting point is 100 mm, #1 DO output signal is true

lin p1,v:{tcp 20,ori 10}

5.10.3 StoEnd(Whether the current motion trajectory reaches a certain distance from the target point)

Function prototype

bool StoEnd(double s)

Description

It returns whether the current trajectory is s mm away from the target point. The function is mainly used for definition of the event in trajectory trigger.

Parameter

Table5-111 Parameters of StoEnd function

Name	Description
s	Type: double
	in mm

Return value

Type: bool

- false: The current trajectory has not reached the point where s mm is left before reaching the target point.
- true: The current trajectory has reached a point where s mm is left before reaching the target point.

Example

trigger 1,when:StoEnd(100),do:setdo(1,true) //declare a trajectory trigger on the following trajectory: When the distance from the target point is 100 mm, #1 DO output signal is true

lin p1,v:{tcp 20,ori 10}

5.11 Relevant functions for point formula modification

5.11.1 savearl(Copy arl file)

Function prototype

bool savearl(const string& file_src, const string& file_dst, bool mode)

Description

Copy the arl file with the specified name in the current folder and name the new arl file

Parameter

Table 5-112 Parameters of the savearl function

Name	Description
file_src	Type: string
	Original arl file name
file_dst	Type: string
	Copy the arl file name
mode	Type: bool
	Whether to overwrite the file with the same name. true: overwrite the file with the same name; false: do not overwrite the file with the same name

Return value

Type: bool

It returns true if the copy is successful, false if it fails.

Example

print savearl("Recipe1.arl",Recipe2.arl",true) // If there is already a Recipe2.arl file in the current folder, output true and overwrite the Recipe2.arl file

print savearl("Recipe1.arl","Recipe2.arl",false) // If there is already a Recipe2.arl file in the current folder, it will output false and do nothing

5.11.2 savefilepose(Save point information to data file)

Function prototype

bool savefilepose(const string& file_name,const string& pose_name,const pose& p)

Description

Save the pose point information to the specified point of the specified arl file (valid after reloading)

Parameter

Table 5-113 Parameters of savefilepose function

Name	Description
file_name	Type: string
	The name of the _data.arl file stored in the point
p_name	Type: string
	Name of pose variable to store point information
p	Type: pose
	Pose variable to be stored

Return value

Type: bool

It returns true if the deposit is successful, and false if the deposit fails.

Example

```
pose a = {x 0,y 10,z 15,a 0,b 90,c 0,cfg 0,ej1 20}
savefilepose("Recipe1_data.arl", p1, a) // Save {x 0,y 10,z 15,a 0,b 90,c 0,cfg 0,ej1 20} to p1 of Recipe1_data.arl.
```

5.11.3 savefilejoint(Save point information to data file)

Function prototype

bool savefilejoint(const string& file_name,const string& pose_name,const joint& j)

Description

Save the joint point information to the specified point of the specified arl file (valid after reloading)

Parameter

Table 5-114 Parameters of the savefilejoint function

Name	Description
file_name	Type: string
	The name of the _data.arl file stored in the point
joint_name	Type: string
	Name of joint variable to store point information
j	Type: joint
	Joint variable to be saved

Return value

Type: bool

It returns true if the deposit is successful, and false if the deposit fails.

Example

```
joint a = {j1 0,j2 10,j3 20,j4 30,j5 40,j6 50}
savefilejoint("Recipe1_data.arl", j1, a) //Save {j1 0, j2 10, j3 20, j4 30, j5 40, j6 50} to point j1 in Recipe1_data.arl.
```

5.11.4 saveposenow(Program to save point information to a channel)

Function prototype

bool saveposenow(unsigned int channel, const string& file_name, const string& pose_name, const pose& p)

Description

Save the pose type point information to the specified point of the specified program under the specified foreground channel (effective immediately).

Parameter

Table 5-115 Parameters of saveposenow function

Name	Description
channel	Type: unsigned int
	Channel number of deposit point
file_name	Type: string
	Arl program name to save points
pose_name	Type: string
	Name of pose variable to store point information
p	Type: pose
	Pose variable to be stored

Return value

Type: bool

It returns true if the deposit is successful, and false if the deposit fails.

Example

```
pose a = {x 0,y 10,z 15,a 0,b 90,c 0,cfg 0,ej1 20}
saveposenow(1,"prog1.arl","p1", a) //Save {x 0,y 10,z 15,a 0,b 90,c 0,cfg 0,ej1 20} to p1 of prog1.arl under foreground channel 1.
```

5.11.5 savejointnow(Program to save point information to a channel)

Function prototype

bool savejointnow(unsigned int channel,const string& file_name,const string& pose_name,const joint& j)

Description

Save the joint type point information to the designated point of the designated program under the designated foreground channel (effective immediately).

Parameter

Table 5-116 Parameters of savejointnow function

Name	Description
channel	Type: unsigned int
	Channel number of deposit point

Name	Description
file_name	Type: string
	Arl program name to save points
joint_name	Type: string
	Name of joint variable to store point information
j	Type: joint
	Joint variable to be saved

Return value

Type: bool

It returns true if the deposit is successful, and false if the deposit fails.

Example

```
joint a = {j1 0,j2 10,j3 20,j4 30,j5 40,j6 50}
savejointnow(1,"prog1.arl","j1", a) //Save {j1 0, j2 10, j3 20, j4 30, j5 40, j6 50} to j1 of prog1.arl under foreground channel 1.
```

5.11.6 switcharl(Switch the arl program loaded by the foreground channel)

Function prototype

bool switcharl(unsigned int channel,const string& file_name)

Description

Pause, reset, and unload the program of the specified channel, and load the arl file with the specified name into the channel and run it. Only allowed in the background channel.

Parameter

Table 5-117 Parameters of the switcharl function

Name	Description
channel	Type: unsigned int
	Foreground channel number, range: [1,6]
file_name	Type: string
	File name of the arl program to be loaded

Return value

Type: bool

If the switch succeeds, it returns true; if the switch fails, it returns false.

Example

```
print switcharl(1,"Recipe1.arl") //Output true and load Recipe1.arl to the foreground channel 1
```

5.12 Other functions

5.12.1 typeof(Get the parameter type name)

Function prototype

string typeof(anytype v)

Description

It returns the name of any data type.

Parameter

Table5-118 Parameters of typeof function

Name	Description
v	Type: It can be an expression of any data type
	Expression of the type to be obtained

Return value

Type: string

It returns the name string of the type.

Example

```
double value =3.1415
print typeof(value) //Output "double"
```

5.12.2 ctime(Get the current time string)

Function prototype

string ctime()

Description

It expresses the current time in a string.

Parameter

None

Return value

Type: string

It returns the current time, which is expressed in a string of "hh:mm:ss".

Example

```
print ctime() //If the current time is 10:10:10, "10:10:10" will be output
```

5.12.3 cdate(Get the current date string)

Function prototype

string cdate()

Description

It expresses the current date in a string.

Parameter

None

Return value

Type: string

It returns the current date, which is expressed in a string of "YY-MM-DD".

Example

print cdate() //If the current date is October 25, 2014, "2014-11-25" will be output

5.12.4 assert(Assertion)

Function prototype

void assert(bool x)

Description

It judges whether the value of x is true. If not, an alarm will be reported, and the file and line number where the assert function is located will be indicated.

Parameter

Table5-119 Parameters of assert function

Name	Description
x	Type: bool
	It specifies the bool expression asserted

Return value

None

Example

int a = 6

assert(a == 5) //The program will exit with this line and report an error

5.12.5 savesv(Storage system variables)

Function prototype

void savesv(string svname)

Description

It saves the system variable named svname.

Parameter

Table5-120 Parameters of savesv function

Name	Description
svname	Type: string
	It specifies the name of the system variable pre-saved. If the variable name entered does not exist, there may have an operating error

Return value

Type: void

Example

savesv ("TOOLS") //Save the tool system variables so that the TOOLS value will not be lost the next time the system is started.

5.12.6 init(Restore system variables to default values)

Function prototype

void init()

Description

It restores the system variables described in *Section10* to the default values.

Parameter

None

Return value

None

Example

$DFSPPED = {10,50,5,5,5}

init()

print $DFSPPED //Output "{5,50,5,5,5}"

5.12.7 gettextstr(Read a line in a text file)

Function prototype

gettextstr(string file_path,int line_no,bool external_file)

Description

It reads a line from the text file.

Parameter

Table5-121 Parameters for gettextstr function

Name	Description
file_path	Type: string
	It specifies the file name. For example, if the file under the script directory is 1.txt, the file name 1.txt will be entered, and if the file under the script directory is subdir/2.txt, the file name subdir/2.txt will be entered
line_no	Type: int
	It specifies the line number, starting from 1. When the file name entered does not exist, there will have the alarm 10064 "The file does not exist or is damaged". When the line number entered is greater than the max line number of the file, there will have the alarm 10065 "The line number entered is greater than the max number of the file".
external_file	Type: bool
	It specifies the file storage position type. The default value is false. true represents the internal storage files, and reads the files under "/home/USB" directory; false represents the external storage files, and reads the files under "/home/ae/script" directory

Return value

Type: string

Example

gettextstr("test1.txt",5, false) //Read the #5 line of test1.txt under /home/ae/script directory;

gettextstr("test/test1.txt",5, true) //Read the #5 line of test1.txt under /home/ae/USB/test directory;

5.12.8 curmpfile (Get the name of the arl file where the motion pointer is located)

Function prototype

string curmpfile(int channel_index)

Description

Get the name of the arl file where the motion pointer is located

When the control cabinet is powered off, the arl file name will be saved. When the control cabinet is restarted, the program name will remain unchanged; after the current channel is running, the program name will be updated with the running of the program.

Parameter

Table 5-122 Parameters of the curmpfile function

Name	Description
channel_index	Type: int
	The number of the foreground channel, the starting value is 1

Return value

Type: string

Example

string s = curmpfile(1) // Read the arl file name where the motion pointer of foreground channel 1 is located.

5.12.9 curmpline (Get the movement pointer line number)

Function prototype

int curmpline(int channel_index)

Description

Used to get the motion pointer line number.

When the control cabinet is powered off, the pointer line number will be saved. When the control cabinet is restarted, the pointer line number will remain unchanged; after the current channel is running, the pointer line number will be updated with the running of the program.

Parameter

Table 5-123 Parameters of the curmpline function

Name	Description
channel_index	Type: int
	The number of the foreground channel, the starting value is 1

Return value

Type: int

Example

int i = curmpline(1) // Read the motion pointer line number of foreground channel 1.

5.12.10 curppfile (Get the name of the arl file where the program pointer is located)

Function prototype

string curppfile(int channel_index)

Description

Used to get the name of the arl file where the program pointer is located.

When the control cabinet is powered off, the arl file name will be saved. When the control cabinet is restarted, the program name will remain unchanged; after the current channel is running, the program name will be updated with the running of the program.

Parameter

Table 5-124 Parameters of the curppfile function

Name	Description
channel_index	Type: int
	The number of the foreground channel, the starting value is 1

Return value

Type: string

Example

string s = curppfile(1) // Read the arl file name where the program pointer of foreground channel 1 is located

5.12.11 curppline (Get the program pointer line number)

Function prototype

int curppline(int channel_index)

Description

Used to get the program pointer line number.


Notice

When the control cabinet is powered off, the pointer line number will be saved. When the control cabinet is restarted, the pointer line number will remain unchanged; after the current channel is running, the pointer line number will be updated with the running of the program.

Parameter

Table 5-125 Parameters of the curppline function

Name	Description
channel_index	Type: int
	The number of the foreground channel, the starting value is 1

Return value

Type: int

Example

int i = curppline(1) // Read the program pointer line number of foreground channel 1.

5.12.12 filesize (Query file size)

Function prototype

int filesize(string file_name)

Description

Used to query file size.

Parameter

Table 5-126 Parameters of the filesize function

Name	Description
file_name	Type: string
	The name of the file being queried

Return value

Type: int

Example

int i = filesize(prog1.arl) // Read the size of the script/prog1.arl file, in bytes.

5.12.13 renamefile(File rename)

Function prototype

bool renamefile(string file_name, string new_file_name)

Description

Used to rename files.

Parameter

Table 5-127 Parameters of the renamefile function

Name	**Description**
file_name	Type: string
	Original file name
new_file_name	Type: string
	New file name

Return value

Type: bool

Returns true to indicate that the operation was successful, and false to indicate that the operation failed.

Example

bool result = renamefile(prog1.arl, prog2.arl) // Change the name of the script/prog1.arl file to prog2.arl.

5.12.14 removefile (Delete Files)

Function prototype

bool removefile(string file_name)

Description

Used to delete files.

Parameter

Table 5-128 Parameters of the removefile function

Name	Description
file_name	Type: string
	The name of the deleted file

Return value

Type: bool

Returns true to indicate that the operation was successful, and false to indicate that the operation failed.

Example

bool result = removefile(prog2.arl) // Delete the file named script/prog2.arl.

6 Interrupt

When the robot is operated in a complex application, the robot must immediately respond to certain external or internal events, and must interrupt the running robot program to start the interrupt processing function. After the interrupt processing function is executed, the interrupted program will be executed continuously.

6.1 Interrupt declaration

Format

interrupt [name],[priority],when:,do:

Parameter

Table6-1 Parameter description of interrupt

Name	Description
name	Data type: string
	It specifies the name of the interrupt. If the name is specified, the name may not be null or be the same as other declared interrupts. The user can then enable or disable the interrupt through the interrupt name The parameter can be defaulted. The default value is a null string
priority	Data type: int
	It specifies the interrupt priority The interrupt priority must be an integer within 0~255 The parameter can be defaulted. The default value is 0. For details about interrupt priority, please refer to *Section 6.2*.
when	Data type: bool
	It defines the interrupt event The interrupt event is a bool expression. Provided that the value of the bool expression is true, the interrupt time will be executed, indicating that an interrupt event has occurred. For details about interrupt event, please refer to *Section 6.3*.
do	Data type: any
	It defines the interrupt processing action When an interrupt event occurs, the expression action will be executed For details about interrupt processing action, please refer to *Section 6.4*.

6.2 Interrupt priority

The interrupt priority is an integer within 0~255, among which 0 represents the highest priority and 255 represents the lowest priority.

The interrupt priority indicates that when two events occur simultaneously during the same interrupt scan cycle, the event with the higher priority will be responded first, that is, the interrupt processing action of the interrupt of higher priority will be executed first, then the Interrupt processing action of lower priority will be executed, and finally the

interrupted function will be executed continuously. If the two events have the same priority, they will be responded in the declared order.

The program example is as follows:

```
func void inthandler1
print "-->inthandler1"
endfunc
func void inthandler2
print "-->inthandler2"
endfunc
func void main
//Declare the interrupt of priority 1
interrupt 1,when:getdi(5),do:inthandler1()
//Declare the interrupt of priority 0
interrupt 0,when:getdi(6),do:inthandler2()
while(1)
waittime 1
endwhile
endfunc
```

Provided that signals are generated on #5 channel and #6 channel of the DI board simultaneously, the system will respond to the interrupt with a priority of 0 first, and then respond to the interrupt with a priority of 1.

Therefore, the final output is:

```
->inthandler2
->inthandler1
```

Nesting is allowed for interrupts. During the execution of the interrupt processing function, if an interrupt event of high priority occurs, the system will suspend the execution of the current interrupt function and enter the execution of a new interrupt processing function. Then the previous interrupt processing function will be executed continuously.

The program example is as follows:

```
func void inthandler1
print "-->inthandler1"
endfunc
func void inthandler2
print "-->inthandler2"
movej j:{j2 -20}
movej j:{j2 -30}
waittime 0
endfunc
func void main
//Declare the interrupt of priority 0
interrupt 0,when:getdi(5),do:inthandler1()
//Declare the interrupt of priority 1
interrupt 1,when:getdi(6),do:inthandler2()
```

```
while(1)
waittime 1
endwhile
endfunc
```

Provided that #6 channel of DI generates a signal at the moment of t1, the program will enter inthandler2 for execution. During execution, if #5 channel of DI generates a signal, the program will directly enter inthandler1 for execution. Then the interrupted program in inthandler2 will be executed continuously.

6.3 Interrupt event

The interrupt event must be a bool expression or an expression that can be implicitly converted to bool. The expression is the same as the judgment statement in the statements if and while.

For example, the following expressions are all such types:

- 1
- true
- getdi(5)
- getdi(5) == true
- getdi(5) && getdi(6)
- counter>=20
- T(3.4)

An interrupt event is defined as follows:

In the previous interrupt scan cycle, the value of the interrupt event expression is false, and in the local interrupt scan cycle, the value of the interrupt event expression is true. That is, the interrupt in ARL is edge-triggered. It indicates that the interrupt event has occurred only when the value of the interrupt event expression changes from false to true.

6.4 Interrupt processingaction

The interrupt processing action is an expression. When an interrupt event occurs, the expression will be executed if the execution conditions are satisfied.

The do parameter expression in the interrupt declaration instruction is usually a calling function, which can be either a user-defined interrupt processing function or a system-predefined function.

If the interrupt processing action is simple, such as output of 1 or multiple DOs only, the interrupt can be declared as follows:

```
interrupt 1,when:getdi(6),do:setdo(1,true) //Declare an interrupt with priority 1. When #6 channel of DI has a signal, #1 channel signal of DO will be set to true.
```

If the interrupt processing action is complicated, you can only define the interrupt processing function. There is no difference between the definition of an interrupt processing function and an ordinary function, except that the ordinary interrupt processing function usually has no parameters or return values.

6.5 Interrupt enable, disable and delete

When an interrupt is declared, the interrupt is enabled by default.

If you want to disable the interrupt in some cases, you can disable it with the disableint instruction. If you want to enable the interrupt again, you can enable it with the enableint instruction. If you want to delete a declared interrupt, you can delete it with the delint instruction.

The disableint, enableint and delint instructions are used as follows:

Format

disableint/enableint/delint [name:],[priority:]

Parameter

When the disableint/enableint/delint instruction is used alone, that is, without any parameter, it indicates that all interrupts are disabled/enabled/deleted.

Table6-2 Parameter description of disableint/enableint/delint

Name	Description
name	Data type: string
	It specifies the name of the interrupt to be disabled/enabled/deleted If you want to disable/enable/delete an interrupt by the name, the name parameter must be executed to give the interrupt a proper name when the interrupt is declared. If the interrupt with the specified name does not exist, there may have an operating error
priority	Data type: int
	It specifies the interrupt that disables/enables/deletes a certain priority The disableint instruction allows to disable/enable/delete the interrupt of a certain priority If the interrupt with the specified priority does not exist, there may have an operating error

If the parameters name and priority are specified at the same time, it indicates that both the interrupt with a name of name and all interrupts with a priority of priority are enabled.

Examples of disabling and enabling interrupt are as follows:

```
func void inthandler1
disalbeint//Disable any interrupt, that is, any other interrupts will not be responded during the execution of the interrupt function
……
enableint//Enable all interrupts
endfunc
func void main
//Declare the interrupt of priority 0
interrupt 0,when:getdi(5),do:inthandler1()
while(1)
waittime 1
endwhile
endfunc
```

6.6 It stops the current robot action in the interrupt processing function

By default, when an interrupt event occurs, the trajectory planned in the main function will still be executed before the movement in the interrupt processing function is executed.

If you want to stop the current movement when an interrupt event occurs, you must use the stopmove instruction.

If you want to continue the interrupted movement when exiting the interrupt processing function, you must use the startmove instruction.

Application examples of stopmove and startmove instructions are as follows:

```
func void inthandler1
stopmove fast //Stop the current movement quickly
waituntil getdi(6) //Wait for #6 channel DI signal
startmove true //Skip the stopped block and restart the current movement
endfunc
func void main
//Declare the interrupt of priority 0
interrupt 0,when:getdi(5),do:inthandler1()
pose p = {x 1500,y 500,z 500,a 0,b 90,c 0,cfg 0}
ptp p,v:{per 10}
while(true)
lin p:{x 1000,y 500,z 500,a 0,b 90,c 0},v:{tcp 10}
lin p:{x 1100,y 500,z 500,a 0,b 90,c 0},v:{tcp 10}
endwhile
endfunc
```


Tip

"stopmove" will decelerate and stop along the path set by programming, and it needs a certain stopping time and stopping distance. When the speed is faster, it may even stop to the next trajectory. Pay attention when programming.

6.7 Timer interrupt

Description

The timer instruction is a special interrupt declaration. It uses the clock as the interrupt source, and can be used in the scenarios where an interrupt needs to be triggered after a while, or an interrupt is triggered every while.

Format

timer [name:],[priority:],interval:,[rmode:],do:

Parameter

Table6-3 Parameter description of timer

Name	Description
name	Data type: string
	It specifies the name of the interrupt. If the name is specified, the name may not be null or be the same as other declared interrupts.

Name	Description
	The user can then enable or disable the interrupt by the interrupt name. The parameter can be defaulted. The default value is a null string
priority	Data type: int
	It specifies the interrupt priority The interrupt priority must be an integer within 0~255 The parameter can be defaulted. The default value is 0. For details about interrupt priority, please refer to Section 6.2
interval	Data type: double
	It defines the time interval to trigger the interrupt, in s When the value of the parameter rmode is true, the min interval value can be specified as 0.001. When the value of the parameter rmode is false, the min interval value can be specified as 0.
rmode	Data type: bool
	It defines the repeat mode ■ false indicates that the interrupt will be triggered only after interval ■ true indicates that the interrupt is triggered repeatedly after interval
do	Data type: any
	It defines the interrupt processing action When an interrupt event occurs, the expression action will be executed

Example

timer name:"t1",priority:1,interval:1,rmode:true,do:setdo(1,true) //Execute setdo(1,true) every 1s

7 Trajectory trigger

In some cases, the user wants to trigger some events at a certain point in the middle of a movement trajectory without destroying the dynamic characteristics of the movement trajectory. At this time, the trajectory trigger instruction can be used. The ARL track trigger instruction can trigger multiple trigger functions at multiple time points or multiple distance points on a movement track to achieve the user's desired behavior.

7.1 Trajectory trigger declaration

The trajectory trigger in ARL is actually a special form of interrupt, so the format of the trajectory trigger declaration is similar to that of the interrupt declaration. The only difference is that there is no name parameter in the trajectory trigger declaration.

Instruction format

trigger [priority:],when:,do:

Parameter

The parameters of the trajectory trigger declaration are shown in Table7-1.

Table7-1 Parameters of trajectory trigger declaration

Name	Description
priority	Data type: int
	It specifies the trajectory trigger priority The definition of the trajectory trigger priority is the same as that of the interrupt priority. When a trigger event and an interrupt event occur simultaneously in an interrupt scan cycle, the interrupt or trigger event with higher priority will be responded first The parameter can be defaulted. The default value is 10.
when	Data type: bool
	It defines the trigger event The definition of the trigger event is the same as that of the interrupt event. Since the track trigger is the interrupt of scan in the trajectory movement process, the trajectory trigger event is usually related to the trajectory information, such as the running time and running distance of the trajectory For details about trajectory trigger event, please refer to *Section 6.3*.
do	Data type: any
	It defines the trajectory trigger action The expression action will be executed when a trajectory trigger event occurs. The definition of the trajectory trigger action is the same as that of the interrupt trigger action.

The declared trajectory trigger will act on the last movement instruction under the trigger declaration statement.

Example:

```
trigger 1,when:T(1),do:setdo(2,true) //The declaration acts on the next movej instruction
movej j:{j1 30},v:{per 2}
ptp p0,v:{per 50}
```

```
trigger 1,when:S(100),do:setdo(3,true) //The declaration acts on the next lin instruction
lin p1,v:{tcp 20,ori 10}
```

7.2 Trajectory trigger event

Generally, trigger event is defined through 4 system predefined functions:

- Trajectory elapsed time (T)
- Trajectory elapsed distance (S)
- Trajectory left distance (StoEnd)
- Trajectory passing percentage (P)

Example:

```
trigger 0,when:T(0.2),do:setdo(2,true)
```

The declaration means that #2 signal of DO is set to be true when the trajectory of the next movement statement is 0.2s from the starting point of the trajectory.

Example:

```
trigger 0,when:StoEnd(10),do:setdo(2,true)
```

The declaration means that #2 signal of DO is set to be true when the trajectory of the next movement statement is 10 mm from the target point.

The more complicated trajectory trigger events can be implemented through the system variables $TRAJ_ ELAPSE _TIME, $TRAJ_LEFT_TIME, $TRAJ_ ELAPSE _DIS and $TRAJ_LEFT_DIS.

E.g:

```
trigger 0,when: P(50),do:setdo(2,true)
```

The meaning of this statement is to set the second signal of DO to true when the trajectory of the next motion statement moves to 50% of the trajectory length.

7.3 Precautions for trajectory trigger

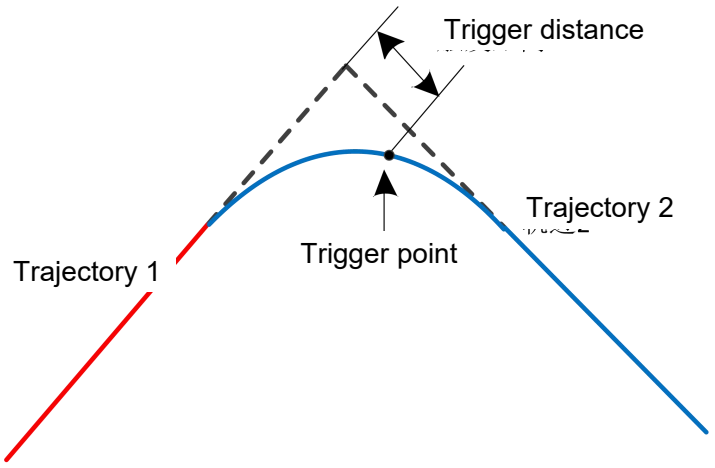


Figure7-1 Trajectory trigger

The precautions for trajectory trigger are as follows:

- Since the PTP and MOVEJ instructions are independent of the TCP trajectory, the trigger function will not be triggered if the trajectory trigger event is specified with the S and StoEnd functions.
- Due to the problems of interpolation accuracy and signal output speed, the specified trigger time or trigger distance may differ from the actual trigger event or trigger distance by several milliseconds or several millimeters.
- When the specified time or distance parameter does not range from of 0 to the total time or total distance of the trajectory, the declared trajectory trigger will not be triggered.
- If the slip behavior is specified, the trajectory of the slip part will be considered to be the next trajectory, so if you want to trigger an event during the slip trajectory, you must declare the trigger event before the second trajectory. The trajectory distance will still be calculated from the target point when the previous track is not slipped, as shown in Figure7-1.
- The trigger of the T() function is for the magnification of 100%. When the magnification is not 100%, the absolute position of the trigger point will not change, so the trigger time is different from the specified actual time.

7.4 Parallel processing

Description

During the execution of the motion instruction, some actions are processed in parallel, which is functionally consistent with the track trigger. The parallel processing format cannot specify the priority (the default is 10), and the other functions are exactly the same as the track trigger.

Function prototype

Motion instruction; trigger event, trigger action

Motion instruction; trigger event 1, trigger action 1;...; trigger event i, trigger action i;...; trigger event n, trigger action n

Parameter

Name	Description
trigger event	Define trigger event The definition of trigger event and interrupt event is the same. Just because the trajectory trigger is the interruption of the scan during the trajectory movement, the trajectory trigger event is generally related to the trajectory information, such as the running time and running distance of the trajectory. For more detailed information about the trajectory trigger event, see Chapter 7.3.
trigger action	Define track trigger action When the track trigger event occurs, the expression action will be executed. The definition of trajectory trigger action is the same as that of interrupt trigger action

Example

Examples are as follows:

```
movej j:{j1 30},v:{per 2};T(1)，setdo(2,true)
//The function is exactly equivalent to the following 2 instructions
trigger 10,when:T(1),do:setdo(2,true)
movej j:{j1 30},v:{per 2}
lin p1,v:{tcp 20,ori 10};T(1)，setdo(2,true);S(100),setdo(3,true)
//The function is exactly equivalent to the following 3 instructions
trigger 10,when:T(1),do:setdo(2,true)
trigger 10,when:S(100),do:setdo(3,true)
lin p1,v:{tcp 20,ori 10}
```

8 Modular Programming

ARL supports modular programming, that is, one arl file can call the functions that are defined in another arl file or access the global variables that are defined in another arl file.

The function is implemented by the "::" scope operator.

Example:

a.arl file

```
double gc = 1
func double add(double x,double y)
return x+y
endfunc
```

The file defines a global variable gc and a function add.

If you want to call the function add or refer to the variable gc in the b.arl file, you can use the following method:

b.arl file

```
func void main()
print a::add(1,2) + a::gc
endfunc
```

After running the program, 4 will be output.

The above procedure assumes that the a.arl file is in the same directory as the b.arl file.

If the a.arl file is not in the same directory as the b.arl file, you must use the import instruction:

b.arl file

```
import "/XXX/XXX/a.arl"
func void main()
print a::add(1,2) + a::gc
endfunc
```

Where /XXX/XXX/a.arl refers to the absolute path of the a.arl file.


Notice

- When loading an arl file, the system will automatically import a module that is named by the arl file+_data.arl (if the file exists). For example, for the b.arl file, if a file named b_data.arl exists in the same directory, the system will import it automatically. At this time, when referring to the global variables in the b_data.arl module, "module name ::" can be defaulted and you can refer to the global variables by their names.
- The module name refers to the file name without .arl.
- The file name where the called function is located must conform to the variable naming rules, and the name of the called arl file may not be pure numbers or Chinese.
- When referring to the functions or variables of other modules, you must refer to them through "module name ::".

9 External auto control

The robot system can be controlled by other controllers (such as machine tool PLC) via external auto control function.

To activate the external automatic control function, you must do as follows:

- Configure the external auto control function.
- Connect the external controller to the robot control system port according to the configuration.
- Write an external auto control main program and test run it.

9.1 Configuration of external auto control function

ecause the external automatic control function uses the user IO port to realize the interaction with the external controller, it is necessary to configure the corresponding port numbers for various external automatic control functions.

including the configuration of IO signals. Here, IO is described from the perspective of the robot control system.

9.1.15 Configuration of input signal

$EXT_CTL_ACT_DI

- Variable type: int
- Variable function: To define the port number of the external auto control activation signal DI.

  If the system variable is defined as an integer between [1~40*, the number of MFs connected to the system], the external controller must activate the external auto control function of the robot control system by setting the port signal to true; when defined as 0, it indicates that the activation function is not used, that is, the robot control system activates the external auto control function by default.

$SERVO_ON_DI

- Variable type: int
- Variable function: To define the port number of the servo-on signal DI.

  If the system variable is defined as a valid port number in the form of integer and a high pulse with a pulse width of at least 30ms is applied to the input port, the robot control system will perform servo-on action; when defined as 0, it indicates that the function is not activated.

$SERVO_OFF_DI

- Variable type: int
- Variable function: To define the port number of the servo-off signal DI.

  If the system variable is defined as a valid port number in the form of integer and a high pulse with a pulse width of at least 30ms is applied to the input port, the robot control system will perform servo-off action; when defined as 0, it indicates that the function is not activated.

$START_PROG_DI

- Variable type: int
- Variable function: To define the port number of the start program signal DI.

  If the system variable is defined as a valid port number in the form of integer and a rising edge signal is applied to the input port, the robot control system will start the execution of the program loaded in the channel (the

program is usually an external auto control main program written by the user); when defined as 0, it indicates that the function is not activated.

$PAUSE_PROG_DI

- Variable type: int
- Variable function: To define the port number of the pause program signal DI.

  If the system variable is defined as a valid port number in the form of integer and a rising edge signal is applied to the input port, the robot control system will pause the execution of the program loaded in the channel (the program is usually an external auto control main program written by the user); when defined as 0, it indicates that the function is not activated.

$RESET_PROG_DI

- Variable type: int
- Variable function: To define the port number of the reset program signal DI.

  If the system variable is defined as a valid port number in the form of integer and a true signal is applied to the input port, the robot control system will reset the program loaded in the channel; when defined as 0, it indicates that the function is not activated. It should be noted that if you use the signal (for example, you want to start the program), you must input a false signal to the DI signal, otherwise the program will not be started.

$CLEAR_ALARM_DI

- Variable type: int
- Variable function: To define the port number of the clear alarm signal DI.

  If the system variable is defined as a valid port number in the form of integer and a rising edge signal is applied to the input port, the robot control system will clear the current alarm; when defined as 0, it indicates that the function is not activated.

$PGNO_TYPE

- Variable type: int
- Variable function: To define the format of the program number. The value ranges from 0, 1 and 2. The meaning is shown in Table9-1:

Table9-1 Meaning of value range of $PGNO_TYPE variable

Value	Description	Example
0	To read in binary format	0 0 0 0 0 1 1 1 $PGNO = 7
1	To read in BCD encoding format. 4 binary bits represent a BCD code. In this format, the decimal number corresponding to 4 binary digits may not be greater than 9	0 0 0 1 0 1 0 0 $PGNO = 14
2	To read in "1 out of N" format. In this format, only 1 bit in the N-bit length must be 1, and all other bits are 0.	0 0 0 0 0 0 0 1 $PGNO = 0 0 0 0 1 0 0 0 0 $PGNO = 4

$PGNO_LENGTH

- Variable type: int
- Variable function: To define the width of the program number.

  The value ranges from 1 to 16. When the value of $PGNO_TYPE is 1, that is, to read the program number in BCD encoding format, the value of $PGNO_LENGTH can only be 4, 8, 12, 16.

$PGNO_FBIT_DI

- Variable type: int
- Variable function: To define the port number of DI where the first digit of the program number is located.

  For example, when the variable is set to 21 and the bit width of the program number is set to 4, the program number will depend on #21, #22, #23 and #24 DI signals.

$PGNO_PARITY_DI

- Variable type: int
- Variable function: To define the port number of the parity signal DI.

  The value of the variable can be positive or negative. The absolute value of the variable represents the port number used. A negative value indicates that the odd parity is used, and a positive value indicates that the even parity is used. Other invalid values indicate that the program number is not checked for parity. When the value of $PGNO_TYPE is 2, that is, to read the program number in "1 out of N" format, no parity will be performed regardless of the value of $PGNO_PARITY_DI.

$PGNO_VALID_DI

- Variable type: int
- Variable function: To define the port number of valid signal DI of the program number.

  When the external controller inputs the signal, the robot controller will start to read the program number. The value of the variable is positive. The value represents the port number used, and the rising edge of the signal is valid.

9.1.16 Configuration of output signal

$EXT_CTL_ACT_CONF_DO

- Variable type: int
- Variable function: To define the port number of the confirmation signal DO when the external auto control function is activated.

  The signal will output true when the external auto control is activated, otherwise it will output false. To activate the external auto control function, you must do as follows:

  The external auto control function is enabled on the setting interface of HMI; the external controller inputs an activation signal; when the variable is set to a non-existed channel number, it indicates that the function is not enabled.

$SERVO_ON_DO

- Variable type: int
- Variable function: To define the port number of the servo-on signal DO.

The signal will output true when the robot control system is in the servo-on state, otherwise it will output false. When the variable is set to a non-existed channel number, it indicates that the function is not enabled.

$PGNO_REQ_DO

- Variable type: int
- Variable function: To define the port number of the program number request signal DO.

When the user sets the system variable $PGNO_REQ to true, the system will set the DO signal defined by $PGNO_REQ_DO to true, indicating that a program number is requested from the external controller. When the variable is set to a non-existed channel number, it indicates that the function is not enabled.

$AT_HOME_DO

- Variable type: int[5]
- Variable function: To define the port number of the signal DO at the HOME point.

The system can set 5 home points, each of which indicates that the position can be modified on the real-time position interface. The DO signal indicates whether the robot is currently at a home position.

$AT_T1_DO

- Variable type: int
- Variable function: Define the signal DO port number when the system is in manual low-speed mode.

When the system is in manual low speed mode, the signal outputs true, otherwise the signal outputs false. Setting this variable to a channel number that does not exist means that the function is not enabled.

$AT_T2_DO

- Variable type: int
- Variable function: Define the signal DO port number when the system is in manual high-speed mode.

When the system is in manual high-speed mode, the signal outputs true, otherwise the signal outputs false. Setting this variable to a channel number that does not exist means that the function is not enabled.

$AT_AUT_DO

- Variable type: int
- Variable function: To define the port number of signal DO when the system is in AUT mode.

The signal will output true when the system is in AUT mode, otherwise it will output false. When the variable is set to a non-existed channel number, it indicates that the function is not enabled.

$PGNO_ACK_FBIT_DO

- Variable type: int
- Variable function: To define the port number of DO where the first digit of the program number confirmation signal is located.

After the robot system receives the program number applied by the external DI signal, if the program number is legal, the DO signal group defined by the system variable will output the program number to the external controller. The external controller will, through program number feedback, check whether the robot controller

receives the program number. When the variable is set to a non-existed channel number, it indicates that the function is not enabled.

$CHAN_STATE_DO

- Variable type: int
- Variable function: The starting logical address number in the current channel state.

  If the value is 5, it indicates that the address numbers 5 and 6 are used as the channel state output. 00: Not loaded; 10: Run; 01: Pause; 11: Stop. When the variable is set to a non-existed channel number, it indicates that the function is not enabled.

10 System variables and constants

There are three types of variables in ARL (refer to *Section 2*):

- Local variable
- Global variable
- System variable

The system variable refers to the system-defined variable with special meaning. The user can use it directly without declaration. There is no difference between using the system variables and using the user-defined variables.

Each system variable has its own variable type, default value, and meaning. Some system variables also have restrictions on the max and min values. For these system variables, the user can only write the value between max value and min value. If you attempt to assign an excess value to these system variables, the system will report an error.

The system variables that can be referenced in the user program are described below.

10.1 Data type system variable

10.1.1 $I(Integer system variable)

Table10-1 $I variable

Property	Description
Variable name	It specifies the integer system variable
Data type	int array [100]
Max value	Max value of integer data
Min value	Min value of integer data
Default value	Factory default values are all 0
When to restore the default value	After the user assigns new values to certain elements of the array in the program, the values of all elements of the array can be saved in the configuration file via savesv("I") function, so that the data will not be lost when servo-off. If not saved, the data will be restored to the last saved value the next time the system is started
Function description	It specifies the system-defined integer variable array, the meaning of which can be defined by the user
Value meaning	User-defined

10.1.2 $I_NAME(Integer system variable name)

Table10-2 $I_NAME variable

Property	Description
Variable name	It specifies the name of integer system variable
Data type	string array [100]
Max value	\
Min value	\
Default value	

Property	Description
When to restore the default value	The variable will take effect immediately after modification, and will remain the last modified value after servo-off and restart
Function description	It specifies the system-defined integer variable name array, the meaning of which can be defined by the user
Value meaning	User-defined

10.1.3 $B(Boolean system variable)

Table10-3 $B variable

Property	Description
Variable name	It specifies the Boolean system variables
Data type	bool array [100]
Max value	-
Min value	-
Default value	Factory default values are all false
When to restore the default value	After the user assigns new values to certain elements of the array in the program, the values of all elements of the array can be saved in the configuration file via savesv("B") function, so that the data will not be lost when servo-off. If not saved, the data will be restored to the last saved value the next time the system is started
Function description	It specifies the system-defined Boolean variable array, the meaning of which can be defined by the user
Value meaning	User-defined

10.1.4 $B_NAME(Boolean system variable name)

Table10-4 $B_NAME variable

Property	Description
Variable name	It specifies the name of Boolean system variable
Data type	string array [100]
Max value	\
Min value	\
Default value	
When to restore the default value	The variable will take effect immediately after modification, and will remain the last modified value after servo-off and restart
Function description	It specifies the system-defined Boolean variable name array, the meaning of which can be defined by the user
Value meaning	User-defined

10.1.5 $D(Floating point system variable)

Table10-5 $D variable

Property	Description
Variable name	It specifies the double system variable
Data type	double array [100]
Max value	Max value of double data
Min value	Min value of double data
Default value	Factory default values are all 0
When to restore the default value	After the user assigns new values to certain elements of the array in the program, the values of all elements of the array can be saved in the configuration file via savesv("D") function, so that the data will not be lost when servo-off. If not saved, the data will be restored to the last saved value the next time the system is started
Function description	It specifies the system-defined double variable array, the meaning of which can be defined by the user
Value meaning	User-defined

10.1.6 $D_NAME(Floating point system variable name)

Table10-6 $D_NAME variable

Property	Description
Variable name	It specifies the name of the double system variable
Data type	string array [100]
Max value	\
Min value	\
Default value	
When to restore the default value	The variable will take effect immediately after modification, and will remain the last modified value after servo-off and restart
Function description	It specifies the system-defined double variable name array, the meaning of which can be defined by the user
Value meaning	User-defined

10.1.7 $P(Structure type system variable P)

Table10-7 $P variable

Property	Description
Variable name	It specifies the structure system variable P
Data type	pose array [100]
Max value	-
Min value	-
Default value	Factory default values are all false
When to restore the	After the user assigns new values to certain elements of the array in the program, the values of all elements of the array can be saved in the configuration file via savesv("P") function, so that the data

Property	Description
default value	will not be lost when servo-off. If not saved, the data will be restored to the last saved value the next time the system is started
Function description	It specifies the system-defined structure variable array, the meaning of which can be defined by the user
Value meaning	User-defined

10.1.8 $J(Structure type system variable J)

Table10-8 $J variable

Property	Description
Variable name	It specifies the structure system variable J
Data type	joint array [100]
Max value	-
Min value	-
Default value	Factory default values are all false
When to restore the default value	After the user assigns new values to certain elements of the array in the program, the values of all elements of the array can be saved in the configuration file via savesv("J") function, so that the data will not be lost when servo-off. If not saved, the data will be restored to the last saved value the next time the system is started
Function description	It specifies the system-defined structure variable array, the meaning of which can be defined by the user
Value meaning	User-defined

10.1.9 $TOOLS(Tool coordinate system)

Table10-9 $TOOLS variable

Property	Description
Variable name	It specifies the tool coordinate system
Data type	tool structure array [32]
Max value	-
Min value	-
Default value	-
When to restore the default value	The system variable will be permanently saved after modification on HMI, and if modified through program, will be restored to the previously saved value after system restart.
Function description	It stores the user-defined tool coordinate system
Value meaning	It specifies the user-defined tool coordinate system

10.1.10 $TOOLS_NAME(Name of tool coordinate system)

Table10-10 $TOOLS_NAME variable

Property	Description
Variable name	It specifies the name of the tool coordinate system
Data type	tool structure array [32]
Max value	-
Min value	-
Default value	-
When to restore the default value	The variable will take effect immediately after modification, and will remain the last modified value after servo-off and restart
Function description	It stores the user-defined tool coordinate system
Value meaning	It specifies the user-defined tool coordinate system

10.1.11 $WOBJS(Workpiece coordinate system)

Table10-11 $WOBJS variable

Property	Description
Variable name	Workpiece coordinate system
Data type	wobj structure array [32]
Max value	-
Min value	-
Default value	-
When to restore the default value	The system variable will be permanently saved after modification on HMI, and if modified through program, will be restored to the previously saved value after system restart.
Function description	It stores the user-defined workpiece coordinate system
Value meaning	It specifies the user-defined workpiece coordinate system

10.1.12 $WOBJS_NAME(Workpiece coordinate system name)

Table10-12 $WOBJS_NAME variable

Property	Description
Variable name	It specifies the name of the workpiece coordinate system
Data type	wobj structure array [32]
Max value	-
Min value	-
Default value	-
When to restore the default value	The variable will take effect immediately after modification, and will remain the last modified value after servo-off and restart
Function description	It stores the user-defined workpiece coordinate system
Value meaning	It specifies the user-defined workpiece coordinate system

10.1.13 $BASE(Basic coordinate system)

Table10-13 $BASE [] Variable

Property	Description
Variable name	It specifies the base coordinate system
Data type	wobj
Value range	0,1,2
Max value	-
Min value	-
Default value	{{0,0,0,0,0,0},false} {{0,0,0,0,0,0},false} {{0,0,0,0,0,0},false}
When to restore the default value	The system variable will be permanently saved after modification on HMI, and if modified through program, will be restored to the previously saved value after system restart.
Function description	It defines the base coordinate system that is located at the bottom of the robot.
Value meaning	It specifies the base coordinate system, which is defined relative to world coordinate system.

10.1.14 $FLANGE(Flange coordinate system)

Table10-14 $FLANGE constant

Property	Description
Variable name	It specifies the flange coordinate system
Data type	tool
Value	{{0,0,0,0,0,0,},false,}
When to restore the default value	-
Function description	It stores the system-defined special tool coordinate system: flange coordinate system
Value meaning	It specifies the system-defined flange coordinate system

10.1.15 $WORLD(World coordinate system)

Table10-15 $WORLD constant

Property	Description
Variable name	It specifies the world coordinate system
Data type	wobj
Value	{{0,0,0,0,0,0,0,},false, WORLD}
When to restore the default value	-
Function description	It stores the system-defined special workpiece coordinate system: world coordinate system
Value meaning	It specifies the system-defined world coordinate system

10.2 Function Type System Variable

10.2.1 $WRIST(Open wrist singularity avoidance)

Table 10-16 $WRIST variable

Property	Description
Variable name	Boolean system variable
Data type	bool
Max value	\
Min value	\
Defaults	The factory default value is false
When to restore the default value	When the software is started, when the program is reset
Function description	Theoretically, the robot trajectory cannot pass through the singularity, and will alarm for speeding when approaching the singularity. Among them, the singular point of the wrist refers to the position where the 5 axis of the robot is 0. When you need to traverse the position where the 5-axis is 0 for Cartesian trajectories such as lin and cir, you can set the system variable to true to enable the singularity avoidance function of the wrist. At this time, the robot will partially sacrifice attitude accuracy to ensure TCP accuracy, thereby passing through the singularity. After completing the traversal, set the system variable to false to continue normal movement.
Value meaning	■ true：Turn on ■ false：Shut down

10.2.2 $Congfig_check (Axis configuration check enable)

Table 10-17 $ check_congfig variable

Property	Description
Variable name	Axis configuration check enable
Data type	bool
Default value	true
When to restore the default value	When the software is started, when the program is reset
Function description	When the TCP of the robot arrives at a point in the same space, the corresponding axis position may be different, please refer to the description of the turn value in the pose structure in *Section 2.4.4*. If during teaching, the axis positions of the two points are different (for example, there is a situation where the six axes differ by 1 circle), this parameter needs to be used in actual operation to determine whether to move to the axis position obtained during teaching.
Value meaning	■ true：Move according to the axis configuration during teaching ■ false：Configure movement according to the axis closest to the current axis position

10.2.3 $DFSPEED(Default speed parameter)

Table10-18 $DFSPEED variable

Property	Description
Variable name	Default speed parameter
Data type	speed
Max value	-
Min value	-
Default value	{5,50,400,5,5}
When to restore the default value	When the software is started or the program is reset
Function description	The default parameter will be used when the velocity parameter v is not specified in the instruction.
Value meaning	Refer to the definition of speed type

10.2.4 $DFSLIP(Default smoothing parameter)

Table10-19 $DFSLIP variable

Property	Description
Variable name	It specifies the default slip parameter
Data type	slip
Max value	-
Min value	-
Default value	{-1.0, -1.0}
When to restore the default value	When the software is started or the program is reset
Function description	The default parameter will be used when the slip parameter s is not specified in the instruction
Value meaning	Refer to the definition of slip type

10.2.5 $DFTOOL(Default tool parameters)

Table10-20 $DFTOOL variable

Property	Description
Variable name	It specifies the default tool parameter
Data type	tool
Max value	-
Min value	-
Default value	{{0,0,0,0,0,0},false}
When to restore the default value	When the software is started or the program is reset
Function description	The default parameter will be used when the tool parameter s is not specified in the instruction
Value meaning	Refer to the definition of tool type

10.2.6 $DFWOBJ(Default workpiece coordinate system parameters)

Table10-21 $DFWOBJ variable

Property	Description
Variable name	It specifies the default workpiece coordinate system parameter
Data type	wobj
Max value	-
Min value	-
Default value	{{0,0,0,0,0,0},false}
When to restore the default value	When the software is started or the program is reset
Function description	The default parameter will be used when the workpiece coordinate system parameter is not specified in the instruction
Value meaning	Refer to the definition of wobj type

10.2.7 $IGNORE_ORI(Direction ignore enable)

Table10-22 $IGNORE_ORI variable

Property	Description
Variable name	It specifies the orientation ignore enable
Data type	bool
Max value	-
Min value	-
Default value	false
When to restore the default value	When the software is started or the program is reset
Function description	When the value is true, the parameters in A, B and C direction of the target point in the movement statement will be ignored, and the direction of the target point will remain the same as the starting point.
Value meaning	true: enable

10.2.8 $ORI_REF_PATH(The arc direction refers to the path coordinate system)

Table10-23 $ORI_REF_PATH variable

Property	Description
Variable name	It specifies the circle orientation reference path coordinate system
Data type	bool
Max value	-
Min value	-
Default value	false

When to restore the default value	When the software is started or the program is reset
Function description	It sets whether the circle trajectory orientation refers to the circle path coordinate system
Value meaning	true: The trajectory orientation of the cir statement refers to the circle path coordinate system

10.2.9 $VEL_PROFILE(Speed profile type)

Table10-24 $VEL_PROFILE variable

Property	Description	
Variable name	It refers to the velocity profile type	
Data type	$VEL_PROFILE	
Max value	-	
Min value	-	
Default value	Different models (corresponding to the mechanical unit model in the teach pendant) have different default values	The default value of 3A, 6A, 6L, 10A is O_type
		The default value of other models is S_type
When to restore the default value	When the software is started or the program is reset	
Function description	It sets the velocity profile of the movement trajectory Each movement instruction is planned with the velocity profile type set in the parameter ■ The advantage of S-type velocity planning is that it is smooth without impact, and the disadvantage is that the planned movement time is long ■ The advantage of T-type velocity planning is that the planned movement time is relatively short, and the disadvantage is that it has impact on the machine. ■ The advantages of O-type velocity planning are fast cycle and low mechanical shock	
Value meaning	■ S_type: S-type velocity profile ■ T_type: T-type velocity profile ■ O_type: O-type velocity profile	

10.2.10 $TRAJ_ELAPSE_TIME(Trajectory elapsed time)

Table10-25 $TRAJ_ELAPSE_TIME variable

Property	Description
Variable name	It specifies the trajectory elapsed time
Data type	double
Max value	-
Min value	-
Default value	0
When to restore the default value	It is restored by the system to the default value when the program is reset.
Function description	It stores the trajectory elapsed time, and generally is used in trajectory trigger declaration.

Property	Description
Value meaning	It specifies the trajectory elapsed time, in s.

10.2.11 $TRAJ_LEFT_TIME(Remaining time on trajectory)

Table10-26 $TRAJ_LEFT_TIME variable

Property	Description
Variable name	It specifies the trajectory left time
Data type	double
Max value	-
Min value	-
Default value	0
When to restore the default value	It is restored by the system to the default value when the program is reset.
Function description	It stores the trajectory left time, and generally is used in the trajectory trigger statement.
Value meaning	It specifies the trajectory left time, in s.

10.2.12 $TRAJ_ELAPSE_DIS(Trajectory passing distance)

Table10-27 $TRAJ_ELAPSE_DIS variable

Property	Description
Variable name	It specifies the trajectory elapsed distance
Data type	double
Max value	-
Min value	-
Default value	0
When to restore the default value	It is restored by the system to the default value when the program is reset.
Function description	It stores the trajectory elapsed distance, and generally is used in trajectory trigger declaration.
Value meaning	It specifies the trajectory elapsed distance, in mm.

10.2.13 $TRAJ_LEFT_DIS(Remaining trajectory distance)

Table10-28 $TRAJ_LEFT_DIS variable

Property	Description
Variable name	It specifies the trajectory left distance.
Data type	double
Max value	-
Min value	-

Property	Description
Default value	0
When to restore the default value	It is restored by the system to the default value when the program is reset.
Function description	It stores the trajectory left distance, and generally is used in trajectory trigger declaration.
Value meaning	It specifies the trajectory left distance, in mm.

10.2.14 $CJOINT(Current axis position point)

Table10-29 $CJOINT variable

Property	Description
Variable name	It specifies the current axis position point.
Data type	joint
Max value	-
Min value	-
Default value	-
When to restore the default value	It is restored by the system to the default value when the program is reset.
Function description	The value of this system variable is always assigned by the system to the target point axis position of the previous movement trajectory, so you can use this system variable to implement incremental programming.
Value meaning	It specifies the current axis position point.

10.2.15 $RPP_ENABLE(RPP enabled)

Table10-30 $RPP_ENABLE variable

Property	Description
Variable name	It specifies the RPP enable.
Data type	bool
Max value	-
Min value	-
Default value	false
When to restore the default value	When the software is started or the program is reset.
Function description	RPP (return to path point) trajectory refers to the first trajectory of automatic program operation. ■ When the RPP function is enabled, the program will execute #1 movement instruction at manual speed and will stop. At this time, the speed will be controlled by the override. ■ When the RPP function is disabled, the program will execute #1 movement instruction at the programmed speed and will not stop. At this time, the speed will not be controlled by the override.
Value meaning	■ true: RPP is enabled. ■ false: RPP is disabled

10.2.16 $AT_HOME(Whether it is in HOME position)

Table10-31 $AT_HOME variable

Property	Description
Variable name	It indicates whether the robot is in the HOME position.
Data type	bool array [5]
Max value	-
Min value	-
Default value	{false,false,false,false,false}
When to restore the default value	-
Function description	The system can set 5 home points, each of which indicates that the position can be modified on the real-time position interface. The system variable indicates whether the robot is currently in the home position. The system variable is read-only to the user and modified internally by the system.
Value meaning	■ true: The robot is in the home position ■ false: The robot is not in the home position

10.2.17 $EXT_CTL_ACT(External automatic control is activated)

Table10-32 $EXT_CTL_ACT variable

Property	Description
Variable name	It indicates that the external auto control is activated.
Data type	bool
Max value	-
Min value	-
Default value	-
When to restore the default value	-
Function description	The system variable indicates whether the external auto control is currently activated. The system variable is read-only to the user and modified internally by the system.
Value meaning	■ true: The external auto control is activated ■ false: The external auto control is not activated

10.2.18 $PGNO_REQ(Request program number status)

Table10-33 $PGNO_REQ variable

Property	Description
Variable name	It specifies the request program number state.
Data type	bool
Max value	-

Min value	-
Default value	false
When to restore the default value	When the software is started, or the program is reset.
Function description	The system variable indicates that the system is currently in the request program number state. The system variable is used to enable external auto control. When the value of the system variable is set to true, the DO port defined by the system variable $PGNO_REQ_DO will output true, indicating that the system is currently in the request program number state.
Value meaning	■ true: The system is in the request program number state ■ false: The system is not in the request program number state

10.2.19 $PGNO(Program number obtained from outside)

Table10-34 $PGNO variable

Property	Description
Variable name	It specifies the program number obtained externally.
Data type	int
Max value	-
Min value	-
Default value	-1
When to restore the default value	When the software is started
Function description	When the external auto control is enabled and the external control system inputs a program number valid signal via DI port, the system will read the program number from DI port. If the program number is valid, the system will set the system variable to the corresponding program number. The user can execute different subprograms in the external auto control main program according to the different values of the system variable.
Value meaning	It specifies the valid program number obtained externally.

10.2.20 $PI(PI)

Table10-35 $PI constants

Property	Description
Variable name	PI
Data type	double
Value	3.1415926535897932
When to restore the default value	-
Function description	It specifies the system-defined pi constant.
Value meaning	PI

10.2.21 $CTL_MODE(Current control mode)

Table10-36 $CTL_MODE variable

Property	Description
Variable name	It specifies the current control mode.
Data type	controlmode
Max value	-
Min value	-
Default value	-
When to restore the default value	-
Function description	The system variable indicates what control mode the system is currently in. The system variable is read-only to the user and modified internally by the system.
Value meaning	■ T1: Manual low-speed mode ■ T2: Manual high-speed mode ■ AUT: Auto mode

10.2.22 $WOBJ_OFFSET(Workpiece coordinate system offset)

Table 10-37 $WOBJ_OFFSET variable

Property	Description
Variable name	Workpiece coordinate system offset.
Data type	double array[6]
Max value	-
Min value	-
Default value	{0,0,0,0,0,0}
When to restore the default value	When the software is started, when the program is reset.
Function description	By modifying this system variable, the batch offset of the target point can be realized.
Value meaning	User-defined offset value of workpiece coordinate system.

Examples of usage are as follows:

```
pose p1 = { x 400, y 0, z 800, a 0, b 0, c 00}
$WOBJ_OFFSET={50,50,50,0,0,0}
lin p:p1,vp:50%,sp:-1%,t:$FLANGE,w:$WORLD //At this time, $WORLD = {{50,50,50,0,0,0},false,WORLD}
print cpose($FLANGE,$WORLD) //Output "{ 450, 50, 850, 0, 0, 0, -1, 9e+09, 9e+09, 9e+09, 9e+09, 9e+09, 9e+09}"
```

10.2.23 $TOOL_OFFSET(Tool coordinate system offset)

Table 10-38 $TOOL_OFFSET variable

Property	Description
Variable name	Tool coordinate system offset
Data type	double array[6]

Property	Description
Max value	-
Min value	-
Default value	{0,0,0,0,0,0}
When to restore the default value	When the software is started, when the program is reset
Function description	By modifying this system variable, the batch offset of the target point can be realized
Value meaning	User-defined tool coordinate system offset value

Examples of usage are as follows:

pose p1 = { x 400, y 0, z 800, a 0, b 0, c 00}

$TOOL_OFFSET={50,0,0,0,0,0}

lin p:p1,vp:50%,sp:-1%,t:$FLANGE,w:$WORLD //At this time, $FLANGE = {{50,0,0,0,0,0},false}

print cpose($FLANGE,$WORLD) //Output "{ 350, 0, 800, 0, 0, 0, 0, -1, 9e+09, 9e+09, 9e+09, 9e+09, 9e+09, 9e+09, 9e+09}"

10.2.24 $RESET_POS_TYPE (Position reset method at power-on)

Table 10-39 $RESET_POS_TYPE variable

Property	Description
Variable name	Position reset method at power-on
Data type	string array[100]
Max value	-
Min value	-
Default value	-
When to restore the default value	Changes to this variable take effect immediately, and the last modified value remains after power-off and restart.
Function description	Position reset method when enabled on the system
Value meaning	■ feedback: Indicates that the instruction position is updated with actual feedback ■ cmd: Indicates that when the instruction feedback error threshold requirement is met, the instruction keeps the position unchanged

10.2.25 $RESET_POS_THESHOLD(Position reset judgment threshold at power-on)

Table 10-40 $RESET_POS_THESHOLD variable

Property	Description
Variable name	Position reset judgment threshold at power-on
Data type	double array[100]

Property	Description
Max value	0.5
Min value	0
Default value	0
When to restore the default value	Changes to this variable take effect immediately, and the last modified value remains after power-off and restart.
Function description	Restoration position threshold when enabled
Value meaning	When the reset position threshold value RESET_POS_TYPE when the upper enable is set to cmd, and the deviation between the instruction position and the feedback position of each axis is less than the threshold value, the instruction position before the last lower enable will be restored when the upper enable is enabled. The unit is degree (°)

Appendix A ARL Keywords

Schedule A ARL Keyword List

Num	Name	Num	Name	Num	Name
1	accset	23	joint	45	stopbits
2	bool	24	jvel	46	stopmove
3	break	25	lin	47	stoptype
4	byte	26	loop	48	string
5	byte	27	movej	49	switch
6	ccir	28	num_base	50	timer
7	cir	29	parity	51	tool
8	clock	30	pause	52	ToolInertiaPara
9	compen	31	pos	53	toolload
10	continue	32	pose	54	toolswitch
11	controlmode	33	print	55	trigger
12	double	34	printto	56	uint
13	endcompen	35	ptp	57	velset
14	endweave	36	repeat	58	waittime
15	exit	37	restart	59	waituntil
16	for	38	return	60	weavedata
17	frame	39	scan	61	weaverotaxis
18	goto	40	slip	62	weaveshape
19	if	41	speed	63	while
20	import	42	startcompen	64	wobj
21	int	43	startmove		
22	interrupt	44	startweave		

Appendix B Instructions and variables index table

LIGENT TECH CO., LTD

Website YouTube

Official Website: http://ligentrobot.com
E-Mail: info@sotrobot.com
whatsapp:+86-13551010933