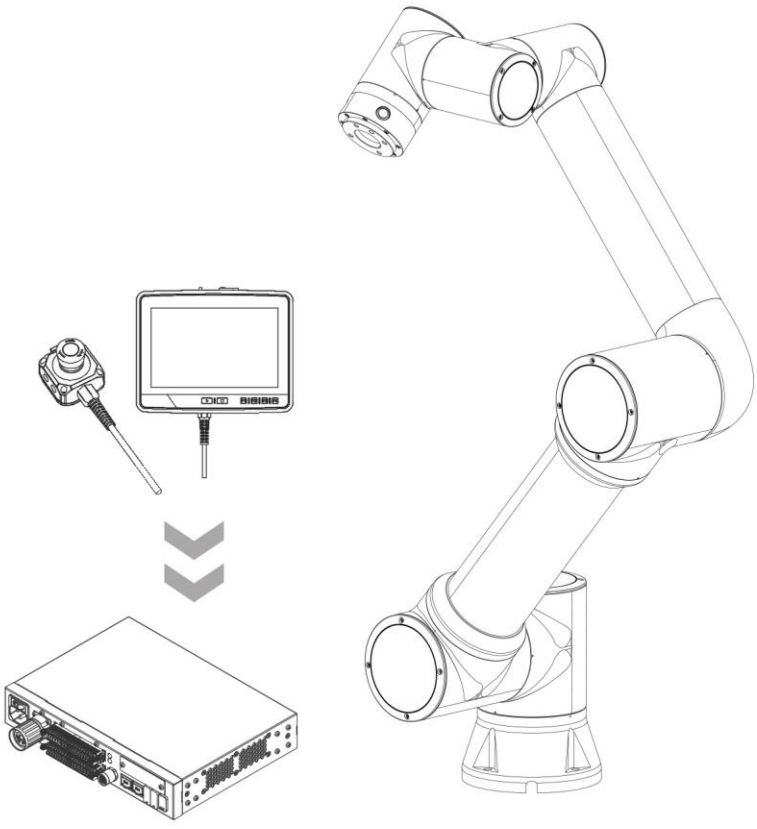
FAIRINO

FR Lua programming script

User Manual



FAIR Innovation (Suzhou) Robotic System Co.,Ltd.

Data Encoding 20200310

Contents

Revision Record

Date	Edition	Description	Author
2024/09/23	V1.0	Unedited English version	xxx
2025/04/23	V1.1	Update welding instructions	xxx
2025/07/17	V1.2	Code examples for verifying and updating Modbus-RTU instructions, collision-detection commands, etc.	xxx

1 Overview

Welcome to the user manual for programming scripts for the FAIRINO collaborative robot FR Lua. This manual aims to provide users with comprehensive guidance on how to proficiently use FR Lua scripts for programming on FAIRINO collaborative robots. Through FR Lua scripts, users can flexibly control robots to perform various tasks.

2 FR Lua Programming Script Fundamentals

2.1 Basic Grammar

2.1.1 FR Lua Annotations

The comments in FR Lua, starting with --, will be ignored by the interpreter. Single line comments are typically used for brief Explanations or annotations of code.

The Explanation for FR Lua comments is as follows.

Code 2-1 FR Lua Annotation Explanation

```
1    --This is a single line comment
```

2.1.2 FR Lua Keywords

The following are the reserved keywords in FR Lua that cannot be used as names. The FR Lua keywords are as follows:

Code 2-2 FR Lua keyword

1	and	break	do	else	elseif	end
2	false	for	function	goto	if	in
3	local	nil	not	or	repeat	return
4	then	true	until	while		

2.1.3 Variables

In the FR Lua environment, variables are identifiers used to store data, which can hold various data types including functions and tables. Variable names consist of letters, numbers, and underscores, and must start with a letter or underscore. FR Lua is case

sensitive.

1) Variable type

FR Lua contains three types of variables:

Global variables: By default, all variables are global variables unless explicitly declared as local variables.

local variable: Declared with the local keyword, the scope starts from the declared position until the end of the current statement block.

Fields in Tables: Tables are very important data structures in FR Lua, and fields in tables can also be used as variables. Examples of FR Lua variable types are shown below.

Code 2-3 FR Lua Variable Type Example

```
1  --Global variables
2  globalVar = 10
3  --local variables
4  local localVar = 20
5  --Fields in the table
6  local tableVar = {key = 30}
```

2) Assignment and multi value assignment

Assignment statement: Used to change the value of a variable or table field. By using "=", the value on the right will be assigned to the variable on the left in sequence. The example of FR Lua assignment is shown below.

Code 2-4 The example of assigning values to FR Lua

```
1  local a = 5
2  local b = 10
3  a = b --The value of a=b ,a is now 10
```

Multi value assignment: FR Lua supports assigning values to multiple variables simultaneously, usually used to swap variable values or assign function Return values to multiple variables. The example of multi value assignment in FR Lua is shown below.

Code 2-5 FR Lua Multi value Assignment Example

```
1  --Exchange variable values
2  local x = 1
3  local y = 2
4  x,y=y, x --x is now 2, y is now 1
5  --Function returns multiple values
```

Code 2-5 (continued）

```
6    local function getvalues()
7    return 5, 10
8    end
9    local a, b=getvalues() -- a is now 5, b is now 10
```

2.1.4 Data Types

FR Lua supports eight basic data types: nil, boolean, number, string, user data, function, thread, and table. The basic data types are described in Table 2-1.

Table 2-1 FR Lua Data Type Description

Value type	Description
Nil	Indicates an invalid value, the default value when the variable is not assigned a value.
Boolean value	Contains only two values, true and false, typically used for conditional judgment.
Number	FR Lua uses double precision floating-point numbers to represent numbers, including integers and floating-point numbers.
String	Used for storing text, it can be represented by single quotes, double quotes, or long strings.
User data	Used to store external data generated by C language.
Function	Functions written in C or RFLua that can be assigned, passed, and returned are the foundation of logical implementation.
Thread	As a coroutine implementation, threads allow concurrent execution, with each thread having an independent execution stack while sharing the global environment and state.

1) Nil data type

Indicates' invalid 'or' null value ', which is the default uninitialized value. Usually used to indicate that there is no value or that a variable's value is undefined, for example:

Code 2-7 nil data type example

```
1.  local a = nil
2.  print (a) -- Output: nil
```

2) Boolean (boolean)

In FR Lua, Boolean types only have two values: true and false. Except for false and nil, all other values (including the number 0) are considered true, as shown in the following example:

Code 2-7 Boolean Data Type Example

```
1.  --Define Boolean values
2.  local a = true
3.  local b = false
4.  --Directly output Boolean values
5.  --Boolean condition judgment
6.  if a then
7.      --code
8.  end
9.  if not b then
10.     --code
11. end
12. --Note: The number 0 is also considered true
13. local c = 0
14. if c then
15.     --code
16. end
17. -False and nil will be considered as false
18. local d = nil
19. if not d then
20.     --code
21. end
```

3) Number

The numeric type is used to store integers or floating-point numbers. FR Lua uses double precision floating-point numbers to represent the number type, so it can accurately represent a wide range of integers and decimals. The example is as follows:

Code 2-8 Numerical Data Type Example

```
1.  local x = 10        --Integer
2.  local y = 3.14      --Floating point number
3.  local z = 1e3       --Scientific notation, representing 1000
```

4) String (string)

Strings are used to store textual data. Strings can be represented by single quotes, double quotes, or [[]] to represent multi line strings, using an example of linking two strings is as follows:

Code 2-9: Example of String Data Type

```
1.  --Example 1: Using single and double quotation marks
2.  local str1 = 'Hello, FR!'       --Use single quotation marks
3.  local str2 = " Hello, FR! "     --Use double quotation marks
4.  --Example 2: [[]], when a string is long or needs to span multiple lines, [[]] can be used to
    represent a multi line string.
```

Code 2-9 (continued)

```
5.  local multi_line_str = [[
6.  I am a FAIRINO collaborative robot.
7.  Thank you for your trust.
8.  May I help you with anything!
9.  ]]
10. --Example 3: String connection
11. --Using FRLua To connect multiple strings.
12. local part1 = "Hello"
13. local part2 = "FR！  "
14. local combined=part1.. "".. Part2- Connect strings using
```

5) User data

User data is a special type used to represent data created by C/C++code. In FR Lua, it is typically used to interact with external programs or libraries.

6) Function

Functions in FR Lua are also a type of data that can be assigned to variables or passed as Parameters. Functions can be named or anonymous (lambda functions), for example:

Code 2-10 Function Data Type Example

```
1.  local function greet()
2.  print("Hello，FR!")
3.  end
4.  Greet() - Output: Hello, FR!
```

7) Thread (thread)

In FR Lua, threads are used to represent coroutine implementations, which allow for non preemptive multitasking between different code blocks, similar to lightweight threads. Examples are as follows:

Code 2-11 Function Thread Data Type Example

```
1.  local co = coroutine.create(function()
2.  print("Running coroutine") end)
3.  Coroutine. sum (co) -- Output: Running coroutine
```

8) Table (Table)

The core data structure of FR Lua is the table, which creates empty tables through {}. It serves as an associative array, supporting numerical and string indexing, providing flexibility in data organization. The table index starts from 1 and automatically expands in length with the content. Unallocated elements default to nil.

The basic syntax for creating and using tables is as follows:

Code 2-12:Basic syntax of table

```
local FR={} -- Create an empty table
local User={"Hello", "FR", "!"} -- Initialize table directly
```

An example of a numerical index (similar to an array) for a table is as follows:

Code 2-13: Numerical Index of Table

```
--Table of numerical indexes
local numbers = {10, 20, 30, 40}
--Accessing the values in the table
print (numbers [1]) -- Output: 10
print (numbers [2]) -- Output: 20
--The length of the table will automatically expand
numbers[5] = 50
print (numbers [5]) --Output: 50
```

An example of a string index (similar to a dictionary) for a table is as follows:

Code 2-14: String Index of Table

```
--Table with string index
local person = {
name = "FR",
age = FR,
city = "China"
}
--Accessing the values in the table
print (person. name) -- Output: FR
print (person ["age"]) -- Output: FR
```

The mixed index of tables, FR Lua tables can use both numeric and string indexes simultaneously, as shown in the following example:

Code 2-15 Mixed Index of Table

```
-Create a mixed index table.
local fr_robots = {
"FR3",
"FR5",
"FR10",
company = "FR",
founded = 2019
}
```

The dynamic growth of a table, for example:

Code 2-16: Dynamic Growth of Tables

```
--Create an empty table to store robot models
local fr_models = {}
--Dynamically add robot product models
fr_models[1] = "FR3"
fr_models[2] = "FR5"
fr_models[3] = "FR10"

--Dynamically add more information
fr_models["total_models"] = 3
fr_models["latest_model"] = "FR10"

--Accessing data in the table
fr_madels [1] -- Output: FR3
fr_madels ["total models"] -- Output: 3
fr_madels ["latest_madel"] -- Output: FR10
```

2.1.5 Operators

1) Arithmetic operator

The commonly used arithmetic operators in FR Lua include addition (+), subtraction (-), multiplication (*), division (/), remainder (%), exponentiation (^), sign (-), and division (//). The following are examples of commonly used arithmetic operators:

Code 2-17 Examples of Common Arithmetic Operators Used in FR Lua

```
--Variable definition
local FR_1 = 10
local FR_2 = 20

--Addition
local addition = FR_1 + FR_2

--Subtraction
local subtraction = FR_1 FR_2
--Multiplication
local multiplication = FR_1 * FR_2
--Division
local division = FR_2 / FR_1
```

Code 2-17 (continued)

```
14. --Take surplus
15. local modulo = FR_2 % FR_1
16.
17. --Multiplying power
18. local power = FR_1 ^ 2
19.
20. --Negative sign
21. local negative = -FR_1
22.
23. –Divisible
24. local integerDivision = 5 // 2
```

2) Relational operator

The commonly used relational operators in FR Lua are equal to (==), not equal to (~=), greater than (>), less than (<), greater than or equal to (>=), and less than or equal to (<=). The following are examples of commonly used relational operators:

Code 2-18: Example of using commonly used relational operators in FR Lua

```
1.  --Variable definition
2.  local FR_1 = 10
3.  local FR_2 = 20
4.
5.  --Equal to
6.  local isEqual = (FR_1 == FR_2)
7.
8.  --Not equal to
9.  local isNotEqual = (FR_1 ~= FR_2)
10.
11. --Greater than
12. local isGreaterThan = (FR_1 > FR_2)
13. local isLessThan = (FR_1 < FR_2)
14.
15. --Greater than or equal to
16. local isGreaterOrEqual = (FR_1 >= FR_2)
17.
18. --Less than or equal to
19. local isLessOrEqual = (FR_1 <= FR_2)
```

3) Logical operator

The commonly used logical operators in RLua are (and), or (or), and non (not). The following are examples of commonly used logical operators:

Code 2-19 Examples of Common Logical Operators Used in FR Lua

```
--Example:
local FR = true
local NFR = false
local result1 = FR and NFR
print ("FR and NFR:", result1) -- Output: FR and NFR: false
local result2 = FR or NFR
print ("FR or NFR:", result2) -- Output: FR or NFR: true
local result3 = not FR
print ("not FR:", result3) -- Output: not FR: false
```

4) Other operators

Other commonly used operators in FR Lua include the join operator (..), table index ([]), assignment (=), table constructor ({}), and variable length operator (#). Examples of commonly used other operators are as follows:

Code 2-20 Example of using other commonly used operators in FR Lua

```
--Examples of using other operators:
local str1 = "Hello"
local str2 = "FR! "
local result = str1 .. " " .. str2
local myTable = {a = 1, b = 2}
local valueA = myTable["a"]
local x = 5
local myTable = {1, 2, 3, 4}
```

2.2 Control structure

2.2.1 Conditional statements

In FR Lua, the if, elseif, and else keywords are used to execute different code blocks, and the execution path is determined based on the authenticity of the conditions. The following is the working mechanism of FR Lua conditional statements:

If conditional statement: Used to specify a condition. If the condition is true, execute the code in the condition block.

Elseif conditional statement: When the if condition is false, another condition can be provided for checking.

If all if and elseif conditions are false, execute the code in the else block.

Condition evaluation: FR Lua assumes that Boolean values true and non nil values are true. Boolean values of false or nil are considered false.

Note: In FR Lua, 0 is considered true, unlike some other programming languages.

The basic structure of conditional statements:

Code 2-21 Basic structure of conditional statements in FR Lua

```
if condition1 then
--Code block executed when condition 1 is true

elseif condition2 then
--The code block executed when condition 1 is false and condition 2 is true

else
--Code block executed when all conditions are false

end
```

The following is an example of using conditional statements:

Code 2-22 Example of using conditional statements

```
--Example 1: Judging the Positive and Negative of Numbers
local number = 10

if number > 0 then
    --Number is positive
elseif number < 0 then
    --Number is negative
else
    --Number is zero

end
--Output result: The number is a positive number

--Example 2: Check if the variable is nil
local value = nil

if value then
    --variable not nil
else
    --variable is nil

end·
--Output result: The variable is nil
```

Code 2-22 (continued)

```
--Example 3: 0 is considered true in FRLua
local num = 0

if num then
    --0 is considered true
else
    --0 is considered false

end
        --Output result: 0 is considered true
```

2.2.2 Loop statements

In FR Lua programming, it is often necessary to repeatedly execute certain code segments, which is called a loop. A loop consists of two parts: the loop body and the termination condition of the loop. FR Lua provides various loop control structures for repeatedly executing a certain piece of code when conditions are met.

A loop consists of a loop body and termination conditions, where the loop body refers to a set of statements that are repeatedly executed; The termination condition refers to the condition that determines whether the loop continues. When the condition is false, the loop ends.

1) While loop

The while loop will repeatedly execute the code block when the specified condition is true, and will not terminate until the condition is false. The basic structure and example of a while loop are as follows:

Code 2-23 The basic structure of the while loop in FR Lua

```
while condition do
--Circular body

end
```

Code 2-24 Example of using while

```
--Example: Calculate the sum of 1 to 5
local sum = 0
local i = 1
while i <= 5 do
sum = sum + i
i = i + 1
end

```

2. For numerical loop: The for loop is used to iterate over a range of numbers and execute the loop body. The basic structure and example of a for numerical loop are as follows:

Code 2-25 Basic structure of for numerical loop in FR Lua

```
1.  for i = start, end, step do
2.  --Circular body
3.
4.  end
```

Code 2-26 Example of using for numerical loop

```
1.  Example: Output numbers from 1 to 5
2.  for i = 1, 5 do
3.      --code
4.
5.  end
6.
```

3) For Generic Loop

Generic loops are used to traverse tables or iterators. The basic structure and example of a for generic loop are as follows:

Code 2-27 Basic structure of for generic loop in FR Lua

```
1.  for key, value in pairs(table) do
2.  --Circular body
3.
4.  end
```

Code 2-28 Example of using for generic loops

```
1.  --Example: Traverse keys and values in a table
2.  local myTable = {a = 1, b = 2, c = 3}
3.  for key, value in pairs(myTable) do
4.      -- code
5.  end
6.
```

4）repeat... Until loop

Repeat... until loop is similar to while loop, but it first executes the loop body and then checks the conditions. Only when the condition is false, will it continue to execute. The basic structure and example of the repeat... until loop are as follows:

Code 2-29 The basic structure of the until loop

```
repeat
--Circular body

until condition
```

Code 2-30 repeat Example of using the until loop

```
--Example: Calculate the sum of 1 to 5
local sum = 0
local i = 1
repeat
sum = sum + i
i = i + 1
until i > 5
-- Sum : 15
```

5) Nested loop

Nested loop refers to a loop structure being contained within another loop structure. This is typically used in scenarios where multidimensional data is processed or multiple sets of similar operations need to be repeated. In nested loops, the outer loop controls larger operations such as the number of rows, while the inner loop controls smaller operations such as the specific content of each row.

Code 2-31 Nested Loop Example

```
--Example: print a 5x5 star matrix.
for i = 1, 5 do -- outer loop, control line
    for i = 1, 5 do -- Inner loop, control column
        io. write ("*") -- Output asterisks and keep them on the same line
    end
end

for i = 1, 5 do -- outer loop, control line
    for i = 1, 5 do -- Inner loop, control column
        io.write("* ") -- Output asterisks and keep them on the same line
    end
    -- Line break after completing each line of output
end

```

2.2.3 Control statements

FR Lua provides two special statements for controlling loops, namely break and goto.

Break statement: Used to jump out of the current loop in advance. When the loop encounters a break statement, it will immediately end the loop, jump out of the current loop body, and no longer execute subsequent iterations. It can avoid unnecessary loops and improve efficiency.

Code 2-32 Example of break statement in FR Lua

```
1.  --Example: Exit the loop when the counter reaches 3
2.  for i = 1, 5 do
3.      if i==3 then -- When i equals 3, exit the loop
4.          break -- prematurely terminate the loop
5.      end
6.  end
7.
```

goto statement: It can unconditionally jump to the specified tag position. It is possible to simplify code logic in certain complex situations.

Code 2-33 Example of goto statement in FR Lua

```
1.  Example: Using goto to skip certain statements
2.  local i = 1
3.  :: loop_start:: -- Define a label
4.  i = i + 1
5.  if i <= 5 then
6.      goto loop_start -- Jump back to the tag and continue the loop
7.  end
```

2.3 Functions

A function is an abstraction of a set of instructions in a program, used to perform specific tasks or calculations and return results. The functions in FR Lua language are very flexible in writing and use, they can have or not have Parameters and can return one or multiple values.

2.3.1 Definition and Use of FR Lua Functions

The functions in FR Lua are an important component of programming, supporting encapsulation of duplicate code, modular programming, and handling complex logical

operations. The basic structure of FR Lua functions is as follows:

Code 2-34 Basic Structure of FR Lua Functions

```
optional_function_scope function function_name(argument1, argument2, ...)
    --Function Body: Operation of Function
    return result_params_comma_separated
end
```

Function structure analysis:

optional_function-scope: optional scope setting. If the function needs to be used only in a specific module or block, it can be defined as a local function, and if not set, it defaults to a global function.

• Function_name: The function name used to identify the function for easy calling.

• Argument1, Argument2,...: Parameters of the function, data passed to the function.

• Function_fody: The function body contains the specific code that needs to be executed.

• The Return value of the function 'return result_crams_comma_separated' can return multiple values.

The two main uses of functions are:

Complete tasks: for example, call functions in the cooperative robot to perform operations such as moving and grabbing. Functions are used as call statements.

Calculate and Return values: When calculating loads or coordinates, functions are used as expressions for assignment statements.

2.3.2 Function Parameters

FR Lua functions support different types of Parameter passing methods, allowing the passing of basic data types, tables, and functions.

1) Passing basic data types can include numerical, string, and boolean values as Parameters:

Code 2-35 Function Parameter Passing Example

```
function set_speed(speed)
    print ("Set speed to:", speed)
end

set_speed(15)
```

2) A table, as a Parameter table and a complex data structure, is commonly used to pass multiple related values:

Code 2-36 The function takes a table as a Parameter and passes it

```
1.  function set_position(pos)
2.      --
3.  end
4.
5.  local position = {x = 100, y = 200, z = 300}
6.  set_position(position)
```

3) Varargs, FR Lua supports mutable Parameters, meaning functions can accept any number of Parameters. Through Grammar implementation:

Code 2-37 The function takes a table as a Parameter and passes it

```
1.  function print_args(...)
2.      local args = {...}    -- Package all Parameters into a table
3.      for i, v in ipairs(args) do
4.          --code
5.      end
6.  end
7.
8.  print_args("Hello，FR!", 123, true)
```

2.3.4 Return value of function

The FR Lua function can return any type of value, including a single value, multiple values, or a table. The Return value is used to provide the caller with the operation result.

1) Return a single value: The function can return a calculation result or status information:

Code 2-38 returns a single value

```
1.  function square(x)
2.      return x * x
3.  end
4.
5.  local result = square(5)
```

2) If a function needs to return multiple results, you can use commas to separate multiple returns:

Code 2-39 returns multiple values

```
function calculate(a, b)
    local sum = a + b
    local diff = a b
    return sum, diff
end

local sum_result, diff_result = calculate(10, 5)
print ("sum:", sum_desult, "difference:", diff_desult)
--[[
Output
Sum: 15 Difference: 5
]]
```

3) return table: Complex data can be returned through a table, especially when multiple values need to be passed:

Code 2-40 returns the table

```
function get_robot_status()
   return {speed = 10, position = {x = 100, y = 200, z = 300}}
end

local status = get_robot_status()
```

2.3.5 Functions as Parameters and Return values

FR Lua is a functional language that allows functions to be passed as Parameters to other functions and also allows functions to return to other functions. This feature can be used to create flexible callback mechanisms and higher-order functions.

1) Function as Parameter: One function can be passed as a Parameter to another function to implement callbacks

Code 2-41 function passed as Parameter

```
function operate_on_numbers(a, b, operation)
    return operation(a, b)
end

local result = operate_on_numbers(5, 10, function(x, y)
    return x * y
end)
```

2) Functions as Return values: Functions can also return another function, which is very useful when creating custom logic:

Code 2-42 function passed as Parameter

```
1.  function multiplier(factor)
2.      return function(x)
3.          return x * factor
4.  end
5.  end
6.
7.  local double = multiplier(2)
8.  local triple = multiplier(3)
```

2.3.6 Recursive Functions

FR Lua supports recursion, which means that the function calls itself. Recursive functions are commonly used to handle tasks such as decomposition problems and traversing tree structures.

The basic structure of recursive functions usually consists of two parts:

Base Case: This is the termination condition of recursion to prevent infinite recursion of the function.

Recursive Case: A function recursively calls itself to gradually reduce the size of the problem until it meets the baseline conditions.

Code 2-43 Basic Structure of Recursive Functions

```
1.  function recursive_function(param)
2.      if benchmark condition then
3.          --Terminate recursion and return result
4.          return result
5.      else
6.          --Recursive call
7.          return recursive _function (reduced Parameter)
8.      end
9.  end
```

Simple example: factorial calculation, factorial is a classic example of recursive functions. The definition of factorial is: n= n * (n-1) * (n-2) * ... * 1, and 0= 1. We can calculate factorial through recursive functions.

Stepwise recursive formula: n! = n * (n 1)!

The benchmark condition is 0! = 1

Code 2-44 Recursive implementation of factorial

```
function factorial(n)
    if n == 0 then
        return 1-- Benchmark condition: The factorial of 0 is 1
    else
        return n * factorial (n 1) -- Recursive call: n * (n-1)!
    end
end

```

2.4 Character string

In FR Lua language, string is a basic data type used to store textual data. Strings in FR Lua can contain various characters, including but not limited to letters, numbers, symbols, spaces, and other special characters.

2.4.1 String Definition

In FR Lua, strings can be represented using single quotes, double quotes, or square brackets [[]]. Square brackets are commonly used to represent multi line strings.

Code 2-45 String Definition Example

```
--Define a string using single quotation marks
local str1 = 'Hello FR3'

--Define a string using double quotation marks
local str2 = "Welcome to FR "

--Define a multiline string using square brackets
local str3 = [[
This is a multi-line
string for FR5 product.
]]
--          string for FR5 product.
```

2.4.2 Escaping Characters

Lua supports the use of backslashes to represent escape characters. Escaped characters and their corresponding meanings:

Table 2-2 Escaped Characters and Their Corresponding Meanings

Escaping characters	Significance	ASCII code value (decimal)
\a	Bell ringing (BEL)	007
\b	Backspace (BS), move the current position to the previous column	008
\f	Page change (FF), move the current position to the beginning of the next page	012
\n	Line break (LF), move the current position to the beginning of the next line	010
\r	Enter (CR) to move the current position to the beginning of the line	013
\t	Horizontal Tab (HT) (Jump to the next TAB position)	009
\v	Vertical Tabulation (VT)	011
\\	Represents a reverse slash character " \ '	092
\'	Represents a single quotation mark (apostrophe) character	039
\"	Represents a double quotation mark character	034
0	Empty character (NULL)	000
\ddd	Any character represented by 1 to 3 octal digits	Three digit octal
\xhh	Any character represented by 1 to 2 hexadecimal digits	Two digit hexadecimal system

2.4.3 String Operations

Lua provides various built-in functions for string operations, including some commonly used functions:

Table 2-3 Common Functions for String Operations

Serial sumber	Method&Application
1	String.upper (argument) converts all strings to uppercase letters.
2	String. power (argument) converts all strings to lowercase letters.
3	String.gsub (mainString, findString, replaceString, num) replaces a specified character in a string.
4	String.find (str, substr, [init, [plain]) searches for substrings in a specified string and returns substrings the start and end indexes.

Table 2-3 (Continued)

Serial sumber	Method&Application
5	String. reverse (arg) inverts the string.
6	string.format (...) Format a string.
7	String.char (arg) and string.byte (arg) String.char: Convert integer numbers to characters. String.byte: Convert characters to integer values.
8	String. len (arg) calculates the length of a string.
9	String.rep (string, n) returns n copies of a string.
10	String.match (str, pattern, init) searches for the first substring from the specified string str that matches the pattern pattern.
11	String. sub (s, i [, j]) performs string truncation operations.

1) String.upper: Convert lowercase letters to uppercase letters

Table 2-4 Detailed Parameters of string.upper

Attribute	Explanation
Prototype	local upper_str = string.upper(argument)
Description	Convert all strings to uppercase letters
Parameter	·Argument: The string to be converted
Return value	·Upper_str: The converted output string

Code 2-46 string.upper Example

```
1.  local original_str = "Hello, FR!"
2.  local upper_str = string.upper(original_str)
```

2) String.lower: Convert lowercase letters to uppercase letters

Table 2-5 Detailed Parameters of string.lower

Attribute	Explanation
Prototype	local lower_str = string.lower(str)
Description	Convert all uppercase letters in a string to lowercase letters
Parameter	• Str: The string to be converted
Return value	• Lower_str: The converted output string

Code 2-47 string.upper Example

```
1.  local original_str = "HELLO, FR!"
2.  local lower_str = string.lower(original_str)
```

3) String.gsub: Global substitution in strings

Table 2-6 Detailed Parameters of string.gsub

Attribute	Explanation
Prototype	local newString = string.gsub(mainString, findString, replaceString, num)
Description	Used for global replacement in a string, i.e. replacing all matching substrings.
Parameter	• MainString: The original string to be replaced; • FindString: The substring or pattern to be searched for; • (Point): Match any individual character. • .%Escaping special characters or Lua patterns. For example,% Representing dot characters in literal sense; • %a: Match any letter ([A-Za-Z]); • %c: Match any control character; • %d: Match any number ([0-9]); • %l: Match any lowercase letter; • %u: Match any uppercase letter; • %x: Match any hexadecimal digit ([0-9A-Fa-f]); • %p: Match any punctuation mark; • %s: Match any blank character (space, tab, line break, etc.); • %w: Match any alphanumeric character (equivalent to% a% d); • %b: Match any word boundary; • %f: Match any file name character; • %[: Match any character class; • %]End character class; • %*: Indicates that the preceding character or sub pattern can appear zero or multiple times; • %+: Indicates that the preceding character or sub pattern appears at least once. • %-: Indicates that the preceding character or sub pattern appears zero or once. • %?: Indicates that the preceding character or sub pattern appears zero or once. • %n: Represents the nth captured sub pattern, where n is a number. • %%Match the percentage sign% itself. • ReplaceString: a string used to replace the found substring;
Return value	• NewString: The replaced string.

Code 2-52 string.gsub Example

```
1.  local mainString = "Hello, FR! Hello, Lua!"
2.  local findString = "Hello"
3.  local replaceString = "Hi"
4.  --Replace all matching substrings
5.  local newString = string.gsub(mainString, findString, replaceString)
```

4) String.find: Search for substrings in a string.

Table 2-7 Detailed Parameters of string.lind

Attribute	Explanation
Prototype	string.find (str, substr, init, plain)
Description	Search for substrings in a string and return the start and end indices of the substring. If a substring is found, it returns the starting and ending positions of the substring in the string; If not found, it returns nil
Parameter	• Str: The string to search for; • Substr: The substring to be searched for.
Return value	• Init: The starting position of the search, default is 1. If specified, the search will start from this location.

Code 2-49 string.find Example

```
1.  local str = "Hello, FR
2.  local substr = "world"
3.
4.  --Find the position of substring 'FR'
5.  local start, end_ = string.find(str, substr)
6.
7.  --Start searching from the specified location
8.  local start, end_ = string.find(str, substr, 6)
9.
10. --Compare using regular strings
11. local start, end_ = string.find(str, "FR", 1, true)
12.
```

5) String.reverse: Inverts the string.

Table 2-8 Detailed Parameters of string.reverse

Attribute	Explanation
Prototype	local reversed_str = string.reverse (arg)
Description	Used to invert strings
Parameter	• arg: The string that needs to be reversed;
Return value	Reversed_str: The reversed string

Code 2-50 string.reverse Example

```
1.  local original_str = "Hello, World!"
2.  local reversed_str = string.reverse(original_str)
3.  print (reversed_str) -- Output "! DlroW, olleH"
```

6) String.f Format: The function is used to create a formatted string.

Table 2-9 Detailed Parameters of string.form

Attribute	Explanation
Prototype	local string.format = string.format(format, arg1, arg2, ...)
Description	Used to create formatted strings
Parameter	• Format: a string containing formatting instructions that start with the% symbol and are followed by one or more characters to specify the format; • %D or% i: integer; • %f: Floating point number; • %g: Automatically select% f or% e based on the size of the value; • %E or% E: floating point number represented by Scientific notation; • %X or% X: hexadecimal integer; • %o: Octal integer; • %p: Pointer (usually displayed as a hexadecimal number); • %s: String; • %q: A string enclosed in double quotation marks, used for program output; • %c: Characters; • %b: Binary numbers; • %%Output% symbol; • Arg1, arg2,...: Parameters to be inserted into the formatted string.
Return value	• Reversed_str: The reversed string.

Code 2-51 string.f Format Example

```
1.  --Format numbers
2.  local num = 123
3.  local formatted_num = string.format("Number: %d", num)
4.
5.  --Format floating-point numbers
6.  local pi = 3.14159
7.  local formatted_pi = string.format("Pi: %.2f", pi)
8.
9.  --Format string
10. local name = "Kimi"
11. local greeting = string.format("Hello, %s!", name)
12.
13. --Format multiple values
14. local name = "Kimi"
15. local age = 30
16. local greeting = string.format("Hello, %s. You are %d years old.", name, age)
17.
18. --Format as hexadecimal
19. local num = 255
20. local formatted_hex = string.format("Hex: %x", num)
```

Code 2-51 (continued)

```
21.  --Format as Scientific notation
22.  local large_num = 123456789
23.  local formatted_scientific = string.format("Scientific: %e", large_num)
24.
```

7) String.char: Convert one or more integer Parameters into corresponding strings

Table 2-10 Detailed Parameters of string.char

Attribute	Explanation
Prototype	string.char(arg1, arg2, ...)
Description	Convert one or more integer Parameters into corresponding strings, where each integer represents the ASCII or Unicode encoding of a character
Parameter	• Arg1, arg2,...: integer sequence to be converted to characters;
Return value	• Str: A string composed of converted characters.

Code 2-52 string.char Example

```
1.  local str = string.char(72, 101, 108, 108, 111)
2.
```

8) String.byte: Convert one or more characters in a string to an integer.

Table 2-11 Detailed Parameters of string.byte

Attribute	Explanation
Prototype	local byte1, byte2 = string.byte (s, i, j)
Description	Convert one or more characters in a string to their corresponding ASCII or Unicode encoded integers.
Parameter	• s: The string to be converted. • i: The position of the first character to be converted in a string is set to 1 by default; • j: The position of the last character to be converted in a string is set to i by default.
Return value	• Byte1, byte2: The encoded values corresponding to the converted characters.

Code 2-53 string.byte Example

```
1.  local str = "Hello"
2.  local byte1 = string.byte(str, 1)
3.  local bytes = string.byte(str, 1, 5)
4.
5.  for i, v in ipairs(bytes) do
6.      -- code
7.  end
```

9) String.len: Calculate the length of a string.

Table 2-12 Detailed Parameters of string.len

Attribute	Explanation
Prototype	local length = string.len (arg)
Description	The function is used to calculate the length of a string, which is the number of characters contained in the string.
Parameter	• Arg: The string to calculate its length.
Return value	• Length: The length of a string, which is the number of characters in the string.

Code 2-54 string.len Example

```
1. local str = "Hello, FR!"
2. local length = string.len(str)
```

10) String.rep: Copy a string.

Table 2-13 Detailed Parameters of string.rep

Attribute	Explanation
Prototype	local repeated_str = string.rep (string, n)
Description	Used to repeat a string a specified number of times, that is, to copy and concatenate a string multiple times.
Parameter	• Arg: The string to calculate its length.
Return value	• Length: The length of a string, which is the number of characters in the string.

Code 2-55 string.rep Example

```
1. local str = "FR "
2. local n = 3
3. local repeated_str = string.rep(str, n)
```

11) String.match: searches for substrings in the str that match the specified pattern.

Table 2-14 Detailed Parameters of string.match

Attribute	Explanation
Prototype	local match_result = string.match (str, pattern， init)
Description	The function is used to search for substrings in a given string str that match a specified pattern.
Parameter	• Str: The string to search for; • Pattern: a string that defines the search pattern and can contain special pattern matching characters; • Init: The starting position of the search, default is 1. If specified, the search will start from this location.
Return value	• Match_desult: If a matching substring is found, return the matching string. If captures (sub patterns enclosed in parentheses) are defined in the pattern, return the values of these captures. If no match is found, return nil.

Code 2-56 string.match Example

```
1.  local text = "Hello, 1234 world! "
2.  local pattern="% d+" -- matches one or more numbers
3.  local start = 1
4.
5.  --Match numbers
6.  local match = string.match(text, pattern, start)
7.
8.  --Match and return
9.  local pattern_with_capture = "(%d+) world"
10. local match, number = string.match(text,    pattern_with_capture, start)
11.
```

12) String.sub: Extract substrings from a string.

Table 2-15 Detailed Parameters of string.sub

Attribute	Explanation
Prototype	string.sub (s, i, j)
Description	Extract a substring from the given string s. It determines the range of substrings to be truncated based on the specified starting position i and optional ending position j.
Parameter	• s: To extract the original string of a substring; • i: The index position at the beginning of a substring can be negative, indicating that the calculation starts from the end of the string; • j: End position (optional), can also be negative, indicating calculation starts from the end of the string. If j is omitted, it will be truncated to the end of the string by default.
Return value	• Sub-result: The extracted substring.

Code 2-57 string.sub Example

```
1.  --Simple excerpt
2.  local text = "FR3 Robotics"
3.  local subText = string.sub(text, 1, 3)
4.
5.  --Cut from the starting position
6.  local text = "FRs FR5"
7.  local subText = string.sub(text, 5)
8.
9.  --Use negative index to extract from the end
10. local text = "Welcome to FR10"
11. local subText = string.sub(text, -4, -1)
12.
```

2.5 Arrays

In FR Lua, arrays are implemented using the table type. In fact, there is no dedicated array type in FR Lua, but tables can be used as arrays to process elements. The index of an array generally starts from 1, rather than 0 as in other languages. You can use {} to create an empty array and store various types of elements in it.

2.5.1 One dimensional array

1) Create an array

Code 2-58 Create Array Example

```
1.  --Create an empty array
2.  local array = {}
3.  --Initialize array
4.  local robotmodels = {"FR3", "FR5", "FR10", "FR20"}
```

2) Accessing array elements: Accessing array elements through indexes, starting from 1.

Code 2-59 Example of accessing array elements

```
1.  robotmodels [1]   --Accessing array elements
```

3) Modifying array elements: You can modify elements in an array by indexing them.

Code 2-60 Example of modifying array elements

```
1.  --Modify an element in an array
2.  robotmodels [2] = "FE5_1"
```

2.5.2 Multidimensional array

In FR Lua, multidimensional arrays are implemented through nested tables, meaning that each element in the array itself is also an array. Through this method, two-dimensional arrays, three-dimensional arrays, and even higher dimensional arrays can be created.

1) Create multidimensional array

To create a two-dimensional array, you can store the data for each row in a separate table, and then store the tables for these rows in a larger table.

Here is a simple example of a two-dimensional array that stores different robot models and their Parameters.

Code 2-61 Example of 2D Array

```
--Create a two-dimensional array
local robots = {
    {"FR3", 3, "Lightweight"},
    {"FR5", 5, "Standard"},
    {"FR10", 10, "Heavy-duty"}
}--Accessing elements in a two-dimensional array
--robots [1] [1] -- Output: FR3
--robots [2] [2] -- Output: 5
--robots [3] [3] -- Output: Heavy duty
```

2) Traverse a two-dimensional array

Nested loops can be used to traverse multidimensional arrays. The following example demonstrates how to traverse a two-dimensional array of robots.

Code 2-62 Traverse 2D Array Example

```
for i = 1, #robots do
    for j = 1, #robots[i] do
        -- robots[i][j])
    end
end
```

2.6 Table

In FR Lua, table is a powerful and flexible data structure. It can be used to represent arrays, dictionaries, collections, and other complex data types. Due to the lack of built-in arrays or object systems in FR Lua, tables are one of the most core data structures in FR Lua.

2.6.1 Basic Usage of Table

Table is an associative array that uses any type of key (but not nil) to index. That is to say, both numbers and strings can be used as key values.

Code 2-63 Table Index Example

```
--Example 1: Using numbers as indexes
local fruits = {"FR3", "FR5", "FR10"}
--fruits[1]   -- output: FR3
--Example 2: Using a string as an index
local person = {name = "FR", age = 5}
--person["name"] –output: FR
```

2.6.2 Table Operation Functions

In FR Lua, the table module provides some commonly used functions to manipulate table data.

Table 2-16 Detailed Parameters of Common Functions in Table Module

Serial sumber	Method&Application
1	table.concat (table , sep , start , end): All elements from the start position to the end position are separated by the specified delimiter (sep) and reconnected.
2	table.insert (table, pos, value): Insert value elements at specified positions in the array section of the table
3	table.remove (table , pos) return the elements in the table array that are partially located at the pos position
4	table.sort (table , comp) Sort the given table in ascending order.

Here are detailed Explanations and usage examples of these functions:

1) Table.cncat: Connecting strings in a table

Table 2-17 Detailed Parameters of table.cncat

Attribute	Explanation
Prototype	local concatenated = table.concat (table , sep , start , end)
Description	This function is used to concatenate strings in a table and can specify the delimiter sep, as well as concatenate from the start item to the end item of the table.
Parameter	• Table: a table containing the string elements to be connected; • Sep: The delimiter used when connecting strings. If not provided or nil, do not use the • delimiter. The default value is nil. • Start: Specify which index in the table to start the connection from. The default value is 1, starting from the first element of the table. • End: Specify which index in the table to connect to. If not provided or nil, connect to the end of the table.
Return value	• Concatenated: The concatenated string.

Code 2-65 table.cncat example

```
1.  local FRuser = {" Welcome ", " to ", " FR "}
2.
3.  local result = table.concat(FRuser, " ")
```

2) Table-in-insert: inserts a value at a specified location in a table

Table 2-18: Detailed Parameters of Table-INsert

Attribute	Explanation
Prototype	table.insert (table, pos, value)
Description	Used to insert a value at a specified location in a table
Parameter	·Table: The table where new elements need to be inserted; ·value: The value of the new element to be inserted; ·Pos: Index of the position where the new element is inserted. If pos equals the length of the table plus one, the new element will be added to the end of the table. If pos is greater than the length of the table, the new element will be inserted at the end of the table and the length of the table will increase.
Return value	null

Code 2-66 Table. insert Example

```
1.  local robots = {"FR3", "FR5"}
2.
3.  -- Insert at the second position
4.  table.insert(robots, 2, "FR10")
5.  -- robots [2]-- Output: FR10
```

3) Table.remove: Elements removed from a table

Table 2-19 Detailed Parameters of table.remove

Attribute	Explanation
Prototype	local removed = table.remove (table, pos)
Description	Delete the element at the specified position from the table. If the position pos is not specified, delete the last element.
Parameter	·Table: The table from which elements need to be removed; ·POS: To remove the positional index of an element. The default value is nil, indicating the removal of the last element. If pos is greater than the length of the table, nil will be returned and the table will not change.
Return value	·Removed: The value of the removed element. If no pos is specified or pos is out of range, return nil.

Code 2-67 table.remove example

```
1.  local robots = {"FR3", "FR5", "FR10"}
2.  table.remove(robots, 2)
```

4) Table.sort: Sort the elements in the table.

Table 2-20 Detailed Parameters of table.sort

Attribute	Explanation
Prototype	table.sort (table, comp)
Description	Sort the elements in the table. If the comp function is provided, use a custom comparison function to determine the sorting order.
Parameter	·Table: The table to be sorted; ·Comp: A comparison function used to determine the order of two elements. This function takes two Parameters (usually elements from a table) and returns a Boolean value. If the first Parameter should be before the second Parameter, return true; Otherwise, return false.
Return value	table.sort (table, comp)

Code 2-68 table.sort example

```
1.  local numbers = {5, 2, 9, 1, 7}
2.  table.sort(numbers)
3.  for i, v in ipairs(numbers) do
4.      -- code
5.  end
6.  --Output: 1 2 5 7 9
7.
8.  --Example: Custom Sorting
9.  local numbers = {5, 2, 9, 1, 7}
10. table.sort (numbers, function (a, b) return a>b end) -- Sort in descending order
11. for i, v in ipairs(numbers) do
12.     -- code
13. end
14. --Output: 9 7 5 2 1
```

2.7 File operation

File I/O in FR Lua is used for reading and modifying files. It operates in two modes: simple mode and full mode.

2.7.1 Simple mode

The simple mode is similar to file I/O operations in the C language. It maintains a current input file and a current output file, providing operations for these files. It is suitable for basic file operations.

Use the io.open function to open the file, where the value of mode is shown in Table 2-22.

Table 2-21 values of mode

Mode	Description
r	Open the file in read-only mode, the file must exist.
w	Open the write only file. If the file exists, the file length will be reset to 0, which means the content of the file will disappear. If the file does not exist, create it.
a	Open write only files in an attached manner. If the file does not exist, it will be created. If the file exists, the written data will be added to the end of the file, and the original content of the file will be retained. (EOF symbol reserved)
r+	Open the file in read-write mode, the file must exist.
w+	Open the read-write file, and if the file exists, the file length will be reset to zero, meaning the content of the file will disappear. If the file does not exist, create it.
a+	Similar to a, but this file is readable and writable
b	Binary mode, if the file is a binary file, b can be added
+	The number indicates that the file can be read or written

Simple mode uses standard I/O or a current input file and a current output file.

For example, if there is a file named 'example. txt', perform a file read operation

Code 2-69 reads files

```
1.  --Open files in read-only mode
2.  file = io.open("example.txt", "r")
3.
4.  --Set the default input file to 'example. txt'
5.  io.input(file)
6.
7.  --Output file first line
8.  print(io.read())
9.
10. --Close open files
11. io.close(file)
12.
13. --Open write only files in an attached manner
14. file = io.open("example.txt", "a")
15.
16. --Set the default output file to 'example. txt'
17. io.output(file)
18.
19. --Add Lua comments on the last line of the file
20. Io. write ("end of example. txt file comment")
21.
22. --Close open files
23. io.close(file)
```

In the above example, the io. "x" method was used, where io. read() does not have

any Parameters. The Parameters can be one from Table 2-22:

Table 2-22 Parameters of io. read()

Mode	Description
"*n"	Read a number and return it. Example: file.read ("* n")
"*a"	Read the entire file from the current location. Example: file.read ("* a")
* l "(default)	Read the next line and return nil at the end of the file (EOF). Example: file.read ("* l")
number	returns a string with a specified number of characters, or returns nil on EOF. Example: file. read (5)

Other IO methods include:

io. tpfile(): returns a temporary file handle that opens in update mode and is automatically deleted at the end of the program

io. type (file): Check if obj has an available file handle

io. lush(): Write all data in the buffer to the file

io. lines (optional file name): returns an iterative function that retrieves a line of content from the file each time it is called. When it reaches the end of the file, it returns nil but does not close the file.

2.7.2 Full mode

Full mode uses file handles to perform operations, defining all file operations as file handle methods in an object-oriented style. It is suitable for more complex tasks, such as reading multiple files simultaneously. For example, using a file named example.txt, you can perform various file operations.

Code 2-70 operates on files in full mode

```
1.  --1.  Read the entire file content:
2.  local file = io.open("example.txt", "r")
3.  local content=file: read ("* a") -- Read the entire file content
4.  file:close()
5.
6.  --2. Read files line by line:
7.  local file = io.open("example.txt", "r")
8.  for line in file:lines() do
9.      -- code
10. end
11. file:close()
12.
```

```
--3. Read a line
local file = io.open("example.txt", "r")
local line=file: read ("* l") -- Read one line
file:close()

--Write file operation to the example. txt file
--1.  Write string:
local file = io.open("example.txt", "w")
file:write("Hello, Lua!\n")
file:close()

--2. Add write, append content to the end of the file:
local file = io.open("example.txt", "a")
file:write("This is appended text.\n")
file:close()
```

2.8 Modules

Modules are a mechanism used in FR Lua to organize code, providing a better way to encapsulate and reuse code. In large-scale applications, using modules can make the code structure clearer and facilitate maintenance and expansion.

2.10.1 Creating Modules

In FR Lua, tables are used to create modules by directly defining a table and returning it as the module. This method is more intuitive and clear.

Code 2-71: Creating a Module Using Tables

```
--Create modules using tables
--robot_module.lua
local robot_module = {}

robot_module.version = "1.0"

function robot_module.greet()
    – code
end

return robot_module
```

2.10.2 Module Call

FR Lua uses the require function to load modules. Require will execute the module file and return the module's table. Modules are usually saved in the same directory as FR Lua scripts or configured in the path specified by the LUA_PATH environment variable.

The require function will only be executed once when the module is loaded for the first time and the result will be cached. If you call require the same module multiple times, it will only return the table of the first loaded module and will not reload the module. This behavior helps improve performance and avoid duplicate loading and execution.

Code 2-72 records the modules that have been created

```
1.  --Loading module
2.  local robot = require "robot_module"
3.
4.  --Using functions in the module
5.  Robot.green() -- Output: Welcome to use the FAIRINO collaborative robot
```

2.10.3 Search Path

FR Lua uses a search path to find module files. This path can be set through the package. path variable. The path is a string containing patterns, and FR Lua will search for module files in the directories specified by these patterns. Package. path can be set in the script to specify the search path for the module:

Code 2-73 records the modules that have been created

```
1.  package.path = package.path .. ";/ path/to/modules/?.lua"
2.  local mathlib = require("robot_module ")
```

3 FR Lua Script Preset Functions

3.1 Logical instruction

3.1.1 Loop

Please refer to section 2.2.2 for details.

3.1.2 Waiting

This instruction is a delay instruction, divided into four parts: "WaitMs", "WaitDI", "WaitMultiDI", and "WaitAI".

WaitMs: Wait for a specified time

Table 3-1 Detailed Parameters of WaitMs

Attribute	Explanation
Prototype	WaitMs(t_ms)
Description	Wait for the specified time
Parameter	• t_ms: Unit [ms].
Return value	null

WaitDI: Waiting for digital input from the control box

Table 3-2 Detailed Parameters of WaitDI

Attribute	Explanation
Prototype	WaitDI (id, status,maxtime, opt)
Description	Waiting for digital input from the control box
Parameter	• id: Control box DI port number, 0-7 control box DI0-DI7, 8-15 control box CI0-CI7; • status:0-False， 1-True; • maxtime:maximum waiting time, unit [ms]; • opt: Policy after timeout, 0-program stops and prompts timeout, 1-ignore timeout prompt to continue program execution, 2-wait indefinitely.
Return value	null

WaitToolDI: Waiting for tool numerical input

Table 3-3 Detailed Parameters of WaitToolDI

Attribute	Explanation
Prototype	WaitToolDI (id, status,maxtime, opt)
Description	Waiting for digital input from the control box
Parameter	• id: Tool DI port number, 0 - End-DI0, 1 - End-DI1; • status:0-False，1-True; • maxtime:maximum waiting time, unit [ms]; • opt: Policy after timeout, 0-program stops and prompts timeout, 1-ignore timeout prompt to continue program execution, 2-wait indefinitely.
Return value	null

WaitMultiDI: Waiting for multiple digital inputs from the control box

Table 3-4 Detailed Parameters of WaitMultiDI

Attribute	Explanation
Prototype	WaitMultiDI (mode, id, status,maxtime, opt)
Description	Waiting for multiple digital inputs from the control box
Parameter	• mode: [0] - Multi channel AND, [1] - Multi channel OR; • id: io number, bit0~bit7 corresponds to DI0~DI7, bit8~bit15 corresponds to CI0~CI7; • status: bit0~bit7 corresponds to DI0~DI7 status, bit8~bit15 corresponds to CI0~CI7 status: 0-False, 1-True; • maxtime:maximum waiting time, unit [ms]; • opt: Policy after timeout, 0-program stops and prompts timeout, 1-ignore timeout prompt to continue program execution, 2-wait indefinitely.
Return value	null

WaitAI: Waiting for analog input from the control box

Table 3-5 Detailed Parameters of WaitAI

Attribute	Explanation
Prototype	WaitAI (id, sign, value, maxtime, opt)
Description	Waiting for analog input from the control box
Parameter	• id: io number, range [0~1]; • sign: 0- greater than, 1- less than • value: Input the percentage of current or voltage value, with a range of [0~100] corresponding to current value [0~20mA] or voltage [0~10V]; • maxtime:maximum waiting time, unit [ms];

Table 3-5 (Continued)

Attribute	Explanation
Parameter	• opt: Policy after timeout, 0-program stops and prompts timeout, 1-ignore timeout prompt to continue program execution, 2-wait indefinitely.
Return value	null

WaitToolAI: Waiting for tool analog input

Table 3-6 Detailed Parameters of WaitToolAI

Attribute	Explanation
Prototype	WaitToolAI (id, sign, value, maxtime, opt)
Description	Waiting for tool analog input
Parameter	• id: io number, 0 End AI0; • sign: 0 greater than, 1 less than; • value: Input the percentage of current or voltage value, with a range of [0~100] corresponding to current value [0~20mA] or voltage [0~10V]; • maxtime: maximum waiting time, unit [ms]; • opt: Policy after timeout, 0-program stops and prompts timeout, 1-ignore timeout prompt to continue program execution, 2-wait indefinitely.
Return value	null

Code 3-1 Waiting Instruction Example

```
1.  --Waiting
2.  WaitMs (1000) -- Wait for a specified time of 1000ms
3.
4.  --Waiting for digital input from the control box
5.  WaitDI (1,1,0,1) -- Port number: Ctrl-DI1, status: true (on),maximum waiting time: 1000ms, policy after waiting timeout: ignore timeout prompt and continue program execution
6.
7.  --Waiting for tool digital input
8.  WaitToolDI (1,1,0,1) -- Port number: End DI0, status: true (open),maximum waiting time: 1000ms, policy after waiting timeout: ignore timeout prompt and continue program execution
9.
10. --Waiting for multiple digital inputs from the control box
11. WaitMultiDI (0,3,11000,0) -- Multi channel AND, IO port numbers: DI0 and DI1, DI0 on, DI1 off,maximum waiting time: 1000ms, policy after timeout: program stops and prompts timeout.
12. WaitMultiDI (1,3,3,1,0) -- Multiple OR, IO port numbers: DI0 and DI1, DI0 open, DI1 open,maximum waiting time: 1000ms, policy after timeout: program stops and prompts timeout.
13.
14. --Waiting for analog input from the control box
15. WaitAI (0,0,201000,0) -- IO port: control box AI0, condition:<, value: 20, maximum waiting time: 1000ms, policy after timeout: program stops and prompts timeout.
```

Code 3-1 (continued)

```
16.
17.  --Waiting for tool analog input
18.  WaitToolAI (0,0,201000,0) -- IO port: Control box End-AI0, condition:<, value: 20,maximum waiting time: 1000ms, policy after timeout: program stops and prompts timeout.
```

3.1.3 Pause

Pause: Pause

Table 3-7 Detailed Parameters of Pause

Attribute	Explanation
Prototype	Pause (num)
Description	Call subroutines
Parameter	• Num: custom numerical value
Return value	null

FR Lua has defined the following pause methods

Code 3-2 Pause Example

```
1.  Pause (0) -- No function
2.  Pause (2) -- Cylinder not in place
3.  Pause (3) -- The screw is not in place
4.  Pause (4) -- Floating lock handling
5.  Pause (5) -- Sliding tooth treatment
```

3.1.4 Subroutines

NewDofile: subroutine call

Table 3-8 Detailed Parameters of NewDofile

Attribute	Explanation
Prototype	NewDofile (name_path, layer, id)
Description	Call subroutines
Parameter	• Name_cath: The file path containing the file subroutine, "/fruser/####.lua"; • Layer: the layer number that calls the subroutine; • id: ID number.
Return value	null

DofileEnd: Subroutine call ends

Table 3-9 Detailed Parameters of DofileEnd

Attribute	Explanation
Prototype	DofileEnd ()
Description	Subroutine call ends
Parameter	null
Return value	null

Code 3-3 Example of calling and closing subroutines

```
1.  --Call the dofile1.lua subroutine
2.  NewDofile("/fruser/dofile1.lua",1,2);
3.
4.  DofileEnd();-- Subroutine call ends
```

3.1.5 Variables

The basic content of variables is detailed in section 2.1.3. FR Lua also defines query variable types and system variable queries and assignments.

RegisterVar: Variable type query

Table 3-10 Register Var Detailed Parameters

Attribute	Explanation
Prototype	RegisterVar (type, var)
Description	Variable type query
Parameter	null
Return value	null

GetSysVarvalue: Retrieve system variables

Table 3-11 Detailed Parameters of Gets ysVarvalue

Attribute	Explanation
Prototype	GetSysVarvalue (s_var)
Description	Retrieve system variables
Parameter	• s_var: System variable name.
Return value	Var_ralue: System variable value

SetSysVarvalueSet system variables

Table 3-12 Detailed Parameters of tSysVarvalue

Attribute	Explanation
Prototype	SetSysVarvalue (s_var, value)
Description	Set system variables
Parameter	• s_var: system variable name; • value: The value of the input variable.
Return value	null

Code 3-4 Example of Operations Related to FR Lua Variables and System Variable values

```
1. local frvalue1 = 0.0
2. Register Var ("number", "frvalue1") -- Query for numeric variables
3. local frString = “X:3.4, Y:0.0”
4. Register Var ("string", "frString") -- Character variable query
5.
6. TEST_1=Get SysVarvalue (s_var_3) -- Get the system variable value and assign it to TEST_1
7. Set System Variable value (s_var_3,1) -- Set System Variable value
```

3.2 Motion command

3.2.1 Point to point

PTP: point-to-point

Table 3-13 PTP Detailed Parameters

Attribute	Explanation
Prototype	PTP (point_name, ovl, blendT, offset_flag, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz)
Description	point-to-point motion
Parameter	• point_name: Name of the target teaching point • ovl: Debugging speed, range [0~100%]; • blend T: [-1] - Non smooth, [0~500] - Smooth time, unit: [ms]; • offset_flag: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - default offset in the tool coordinate system is 0; • offset_x~offset_rz: offset, unit [mm] [°];
Return value	null

MoveJ: Joint Space Motion

Table 3-14 Detailed Parameters of MoveJ

Attribute	Explanation
Prototype	MoveJ (j1, j2, j3, j4, j5, j6, x, y, z, rx, ry, rz, tool, user, speed, acc, ovl, ep1, ep2, ep3, ep4, blendT, offset, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz)
Description	Joint Space Motion
Parameter	• j1~j6: Target joint position, unit [°]; • x. y, z, rx, ry, rz: Cartesian pose of the target, unit [mm] [°]; • tool: tool number; • user: workpiece number; • speed: speed, range [0~100%]; • acc: Acceleration, range [0~100%], temporarily not open; • ovl: Debugging speed, range [0~100%]; • ep1~ep4: External axis 1 position~External axis 4 position; • blend T: [-1] - Non smooth, [0~500] - Smooth time, unit: [ms]; • offset: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - offset in the tool coordinate system; • offset_x~offset_rz: offset, unit [mm] [°].
Return value	null

Code 3-5 Using point-to-point instructions for motion example

```
1.  --Using MoveJ for exercise
2.  x,y,z,rx,ry,rz=GetForwardKin(149.135,-79.058,-78.558,-145.409,-94.182,88.654)
3.  MoveJ(149.135,-79.058,-78.558,-145.409,-94.182,88.654,
    x,y,z,rx,ry,rz,1,0,100,180,100,0.000,0.000,0.000,0.000,0,0,0,0,0,0,0,0)
4.  --Using PTP for movement
5.  PTP (DW01100, -1,0) -- Target Point name: DW01, Velocity Percentage: 100, Blocked: Yes
    (-1- Stop), offset: 0- No
6.  PTP(DW01,100,10,0)
7.  --Target Point name: DW01, Speed Percentage: 100, Blockage: No (10- Smooth Transition
    Time of 10ms), offset: 0- No
8.  PTP(DW01,100,10,1,0,0,0,0,0,0)
9.  --Target Point name: DW01, Velocity Percentage: 100, Blockage: No (10ms), offset: Yes (1-
    Workpiece/Base Coordinate System Offset), Pose offset: [0.0, 0.0, 0.0, 0.0]
10. PTP(DW01,100,10,2,0,0,0,0,0,0)
11. --Target Point name: DW01, Velocity Percentage: 100, Blockage: No (10ms), offset: Yes (2-
    Tool Coordinate System Offset), Pose offset: [0.0, 0.0, 0.0, 0.0, 0.0]
```

3.2.2 Straight Line

Lin: Linear motion

Table 3-15 Lin Detailed Parameters

Attribute	Explanation
Prototype	Lin (point_name, ovl, blendR, search, offset_flag, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz)
Description	Linear Lin motion
Parameter	• point_name: Target point name; • ovl: Debugging speed, default from 0 to 100 is 100.0; • blendR: [-1.0] - Motion in place (blocking), [0~1000] - Smooth radius (non blocking), unit [mm]; • search: [0] - No (welding wire) positioning, [1] (welding wire) positioning; • offset: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - offset in the tool coordinate system; • offset_x~offset_rz: offset, unit [mm] [°].
Return value	null

MoveL: Cartesian Space Linear Motion

Table 3-16 Detailed Parameters of MoveL

Attribute	Explanation
Prototype	MoveL (j1, j2, j3, j4, j5, j6, x, y, z, rx, ry, rz, tool, user, speed, acc, ovl, ep1,ep2, ep3, ep4, blendR, search, offset, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz)
Description	Cartesian space linear motion
Parameter	• j1~j6: Target joint position, unit [°]; • x, y, z, rx, ry, rz: Cartesian pose of the target, unit [mm] [°]; • tool: tool number; • user: workpiece number; • speed: speed, range [0~100%]; • acc: Acceleration, range [0~100%], temporarily not open; • ovl: Debugging speed, range [0~100%]; • ep1~ep4: External axis 1 position~External axis 4 position; • blendR: [-1] - Non smooth, [0~1000] - Smooth radius, unit [mm]; • search: [0] - Non welding wire positioning, [1] - welding wire positioning; • offset: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - offset in the tool coordinate system; • offset_x~offset_rz: offset, unit [mm] [°].
Return value	null

Code 3-6 Motion Example Using Linear Instructions

```
1.  --Using MoveL for Linear Motion
    2.  j1,j2,j3,j4,j5,j6=GetInverseKin(0,-315.039,327.526,786.334,0.052,-32.916,-32.464,-1)
    3.  MoveL(j1, j2,j3,j4,j5,j6,-315.039,327.526,786.334,0.052,-32.916,-32.464, 1,0, 100, 180, 100, -1, 0.000,0.000,0.000,0.000,0,0,0,0,0,0,0,0)
    4.  --Basic Lin Linear Motion
    5.  Lin (DW01100, -1,0,0) -- Target Point name: DW01, Velocity Percentage: 100, Blockage: Yes (-1), Positioning: No, offset: No
    6.  Lin (DW01100,10,0,0) -- Target Point Information: DW01, Velocity Percentage: 100, Blockage: No (10mm), Positioning: No, offset: No
```

3.2.3 Arc

ARC: Arc Motion

Table 3-17 ARC Detailed Parameters

Attribute	Explanation
Prototype	ARC (point_p_name, poffset, offset_px, offset_py, offset_pz, offset_prx, offset_pry, offset_prz, point_t_name, toffset, offset_tx, offset_ty, offset_tz, offset_trx, offset_try, offset_trz, ovl, blend)
Description	ARC arc motion
Parameter	• point_p_name: the name of the midpoint of the arc; • poffset: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - offset in the tool coordinate system; • offset_px~offset_prz: offset, unit [mm] [°]; • point_t_name: the name of the endpoint of the arc; • Toffst: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - offset in the tool coordinate system; • offset_tx~offset_trz: Offset amount, unit [mm] [°]. • ovl: Debugging speed, range [0~100%]; • blendR: [-1] - Non smooth, [0~1000] - Smooth radius, unit [mm].
Return value	null

MoveC: Cartesian space circular motion

Table 3-18 Detailed Parameters of MoveC

Attribute	Explanation
Prototype	MoveC (pj1, pj2, pj3, pj4, pj5, pj6, px, py, pz, prx, pry, prz, ptool, puser, pspeed, pacc, pep1, pep2, pep3, pep4, poffset, offset_px, offset_py, offset_pz, offset_prx, offset_pry, offset_prz, tj1, tj2, tj3, tj4, tj5, tj6, tx, ty, tz, trx, try, trz, ttool, tuser, tspeed, tacc, tep1, tep2, tep3, tep4, toffset, offset_tx, offset_ty, offset_tz, offset_trx, offset_try, offset_trz, ovl, blendR)

Table 3-18 (Continued)

Attribute	Explanation
Prototype	MoveC (pj1, pj2, pj3, pj4, pj5, pj6, px, py, pz, prx, pry, prz, ptool, puser, pspeed, pacc, pep1, pep2, pep3, pep4, poffset, offset_px, offset_py, offset_pz, offset_prx, offset_pry, offset_prz, tj1, tj2, tj3, tj4, tj5, tj6, tx, ty, tz, trx, try, trz, ttool, tuser, tspeed, tacc, tep1, tep2, tep3, tep4, toffset, offset_tx, offset_ty, offset_tz, offset_trx, offset_try, offset_trz, ovl, blendR)
Description	Cartesian space circular motion
Parameter	• pj1~pj6: Joint positions of path points, unit [°]; • px, py, pz, prx, pry, prz: Cartesian pose of path points, unit [mm] [°]; • ptool: tool number; • pusher: workpiece number; • pspeed: speed, range [0~100%]; • pacc: Acceleration, range [0~100%], temporarily not open; • pep1~pep4: External axis 1 position~External axis 4 position; • poffset: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - offset in the tool coordinate system; • offset_px~offset_prz: offset, unit [mm] [°]; • tj1~tj6: Joint position of target point, unit [°]; • tx, ty, tz, trx, try, trz: Cartesian pose of the target point, unit [mm] [°]; • ttool: tool number; • tuser: workpiece number; • tspeed: speed, range [0~100%]; • tacc: Acceleration, range [0~100%], temporarily not open; • tep1~tep4: External axis 1 position~External axis 4 position; • Toffst: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - offset in the tool coordinate system; • offset_tx~offset_trz: Offset amount, unit [mm] [°]. • ovl: Debugging speed, range [0~100%]; • blendR: [-1] - Non smooth, [0~1000] - Smooth radius, unit [mm].
Return value	null

Code 3-7 Using Arc Instructions for Motion Example

```
1. --Using MoveC for circular motion
2. pj1,pj2,pj3,pj4,pj5,pj6=GetInverseKin(0,388.104,-462.265,-5.226,177.576,-1.292,143.417,-1)
3. tj1,tj2,tj3,tj4,tj5,tj6=GetInverseKin(0, 271.474,-476.328,3.739,179.502,-2.433,134.753,-1)
4. MoveC(pj1, pj2,pj3,pj4,pj5,pj6,388.104,-462.265,-5.226,177.576,-1.292,143.417,1,0,100,180,0.000,0.000,0.000,0.000,0,0,0,0,0,0,0,
   tj1,tj2,tj3,tj4,tj5,tj6,271.474,-476.328,3.739,179.502, -2.433,134.753,1,0,100,180,0.000,0.000,0.000,0.000,0,0,0,0,0,0,0,100,-1)
5.
```

Code 3-7 (Continued)

```
1.  --Using ARC for basic circular motion
2.  PTP (DW01100, -1,0) -- PTP mode, moving to the starting point position
3.  -- (DW01100, -1,0,0) -- Move in a straight line to the starting point position
4.  ARC(DW02,0,0,0,0,0,0,0, DW03,0,0,0,0,0,0,0,100,-1)
5.  --The midpoint of DW02 arc motion, 0- not offset; DW03: End point coordinates of arc, 0-
    no offset, 100- percentage of motion speed, -1- stop at the end point
6.
7.  --Arc motion based on base coordinate offset using ARC
8.  PTP (DW01, 100, -1,0) --PTP mode, moving to the starting point position
9.  ARC(DW02, 1, 1, 2, 3, 4, 5, 6,  DW03, 1, 11, 12, 13, 14, 15, 16, 100, -1)
10. --The midpoint of DW02 arc motion, 1-base coordinate offset; 1, 2, 3, 4, 5, 6-offset
    Cartesian coordinates, DW03: arc endpoint coordinates, 1-base coordinate offset, 11, 12,
    13, 14, 15, 16 offset Cartesian coordinates, 100 motion velocity percentage, -1- offset at
    endpoint
11.
12. --Arc motion based on tool coordinate offset using ARC
13. PTP (DW01100, -1,0)--PTP mode, moving to the starting point position
14. ARC(DW02,2,1,2,3,4,5,6, DW03,2,11,12,13,14,15,16,100,-1)
15. --The midpoint of DW02 arc motion, 2-tool mark offset; 1, 2, 3, 4, 5, 6-offset Cartesian
    coordinates, DW03: arc endpoint coordinates, 2-tool coordinate offset, 11, 12, 13, 14, 15,
    16 offset Cartesian coordinates, 100 motion velocity percentage, -1- offset at endpoint
16.
17. --Using ARC to enable smooth circular motion
18. PTP (DW01100, -1,0) --point-to-point mode, moving to the starting point position
19. ARC(DW02,0,0,0,0,0,0,0, DW03,0,0,0,0,0,0,0,100,30)
20. --The midpoint of DW02 arc motion, 0- not offset; DW03: End point coordinates of arc, 0-
    no offset, 100- percentage of motion speed, 30- smooth 30mm
```

3.2.4 Complete Circle

Circle: Complete circular motion (Cartesian space)

Table 3-19 Detailed Parameters of Circle

Attribute	Explanation
Prototype	Circle (pj1, pj2, pj3, pj4, pj5, pj6, px, py, pz, prx, pry, prz, ptool, puser, pspeed, pacc, pep1, pep2, pep3, pep4, tj1, tj2, tj3, tj4, tj5, tj6, tx, ty, tz, trx, try, trz, ttool, tuser, tspeed, tacc, tep1, tep2, tep3, tep4, ovl, offset, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz)
Description	Complete circular motion (Cartesian space)
Parameter	• pj1~pj6: Joint positions of path points, unit [°]; • px, py, pz, prx, pry, prz: Cartesian pose of path points, unit [mm] [°];

Table 3-19 (Continued)

Attribute	Explanation
Parameter	• ptool: tool number; • pusher: workpiece number; • pspeed: speed, range [0~100%]; • pacc: Acceleration, range [0~100%], temporarily not open; • pep1~pep4: External axis 1 position~External axis 4 position; • tj1~tj6: Joint position of target point, unit [°]; • tx, ty, tz, trx, try, trz: Cartesian pose of the target point, unit [mm] [°]; • ttool: tool number; • tuser: workpiece number; • tspeed: speed, range [0~100%]; • tacc: Acceleration, range [0~100%], temporarily not open; • tep1~tep4: External axis 1 position~External axis 4 position; • ovl: Debugging speed, range [0~100%]; • offset: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - offset in the tool coordinate system; • offset_x~offset_rz: offset, unit [mm] [°].
Return value	null

Circle: Full circle motion

Table 3-20 Detailed Parameters of the New Circle

Attribute	Explanation
Prototype	Circle (pos_p_name, pos_t_name, ovl, offset_flag, offset, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz)
Description	Circular motion
Parameter	• pos_p_name: Name of the midpoint 1 of the entire circle; • pos_t_name: Name of the midpoint 2 of the entire circle; • ovl: Debugging speed, range [0~100%]; • offset: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - offset in the tool coordinate system; • offset_x~offset_rz: offset, unit [mm] [°].
Return value	null

Code 3-8 utilizes the circular instruction for motion

```
1. --Complete circular motion (Cartesian space)
2. pj1,pj2,pj3,pj4,pj5,pj6=GetInverseKin(0,388.104,-462.265,-5.226,177.576,-
   1.292,143.417,-1)
3. tj1,tj2,tj3,tj4,tj5,tj6=GetInverseKin(0, 271.474,-476.328,3.739,179.502,-2.433,134.753,-1)
```

Code 3-8 (continued)

```
4.  Circle(pj1, pj2,pj3,pj4,pj5,pj6,388.104,-462.265,-5.226,177.576,-1.292, 143.417, 1, 0, 100,
    180, 0.000, 0.000, 0.000, 0.000, tj1, tj2, tj3, tj4, tj5, tj6, 271.474, -476.328, 3.739, 179.502,-
    2.433,134.753,1,0,100,180,0.000,0.000,0.000,0.000,100,0,0,0,0,0,0,0)
5.
6.  --Circle movement
7.  PTP (DW01100, -1,0) PTP motion to the starting point position
8.  --Lin (DW01100, -1,0,0) -- Straight line motion to starting point position
9.
10. Circle(DW02,DW03,100,0)
11. --The midpoint of DW02's circular motion (path point 1); DW03: End point coordinates of
    arc (path point 2), 100-- percentage of motion speed, 0- no offset
12.
13. Circle(DW02, DW03,100,1,0,0,10,0,0,0)
14. --The midpoint of DW02's circular motion; DW03: End point coordinates of arc, 100-
    percentage of motion speed, 1- based on base coordinate offset, 0,0,10,0,0,0 joint offset
    angle
15.
16. Circle(DW02, DW03,100,2, 0,0,10,0,0,0)
17. --The midpoint of DW02's circular motion; DW03: End point coordinates of arc, 100-
    percentage of motion speed, 2- based on tool coordinate offset, 0,0,10,0,0,0 joint offset angle
```

3.2.5 Spiral

Spiral: Spiral motion

Table 3-21 Detailed Parameters of Spiral

Attribute	Explanation
Prototype	Spiral (pos_1_name, pos_2_name, pos_3_name, ovl, offset_flag, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz, circle_num, circle_angle_Co_rx, circle_angle_Co_ry, circle_angle_Co_rz, rad_add, rotaxis_add)
Description	Spiral motion
Parameter	• pos_1_name: the name of the midpoint 1 of the spiral line; • pos_2_name: the name of the midpoint 2 of the spiral line; • pos_3_name: the name of the midpoint 3 of the spiral line; • ovl: Debugging speed, range [0~100%], default 100.0; • offset_flag: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - default offset in the tool coordinate system is 0; • offset_x~offset_rz: offset, unit [mm] [°]; • circle_num: number of spiral turns;

Table 3-21 (Continued)

Attribute	Explanation
Parameter	• circle_ongle Co_rx~circle_ongle Co_rx: attitude angle correction, unit [°] • radadded: radius increment, unit [mm]; • rotaxias_add: incremental axis direction, unit [mm].
Return value	null

Code 3-9 Spiral Instruction Motion Example

```
1. --Basic Spiral Spiral Motion
2. Spiral (DW01, DW02, DW03, 100,0,0,0,0,0,0,0,5,0,0,0,10,10)
3. --DW01- Name of midpoint 1 of spiral line, DW02- Name of midpoint 2 of spiral line, DW03- Name of midpoint 3 of spiral line, 100- Debugging speed, 0- No offset, 5- Number of spiral turns, (0,0,0) attitude angle correction, 10- Radius increment, 10- Axis direction increment;
4. --Spiral Spiral Motion with Base Coordinate Offset
5. Spiral (DW01, DW02, DW03,100,1,1,0,0,10, 0,0,0,0,0,0,10,10)
6. --DW01- Spiral midpoint 1 name, DW02- Spiral midpoint 2 name, DW03- Spiral midpoint 3 name, 100- Debugging speed, 1- Base coordinate offset, (0,0,10, 0,0,0) - Offset Parameter, 5- Spiral turns, (0,0,0) attitude angle correction, 10- Radius increment, 10- Axis direction increment;
7. --Spiral Spiral Motion with Tool Coordinate Offset
8. Spiral(DW01, DW02,DW03,100,2,1, 0,0,10, 0,0,0,0,0,0,10,10)
9. --DW01- Spiral midpoint 1 name, DW02- Spiral midpoint 2 name, DW03- Spiral midpoint 3 name, 100- Debugging speed, 0- Tool coordinate offset, (0,0,10, 0,0,0) - Offset Parameter, 5- Spiral turns, (0,0,0) attitude angle correction, 10- Radius increment, 10- Axis direction increment.
```

3.2.6 New Spiral

NewSpiral: New Spiral Motion

Table 3-22 Detailed Parameters of NewSpiral

Attribute	Explanation
Prototype	NewSpiral (desc_pos_name, ovl, offset_flag = 2, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz, circle_num, circle_angle, rad_init, rad_add , rot_direction)
Description	NewSpiral new spiral motion
Parameter	• desc_pos_name: Name of the starting point of the new spiral motion; • ovl: Debugging speed, default from 0 to 100 is 100.0; • offset_flag: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - default offset in the tool coordinate system 2 (fixed Parameter); • offset_x~offset_rz: offset, unit [mm] [°];

Table 3-22 (Continued)

Attribute	Explanation
Parameter	• circle_num: number of spiral turns; • circle_angle Spiral inclination angle, unit [°]; • rad_init: Initial radius of spiral, unit [mm]; • radadded: radius increment, unit [mm]; • rotaxias_add: incremental axis direction, unit [mm]; • rot_direction: Rotation direction, 0- clockwise, 1- counterclockwise.
Return value	null

Code 3-10 New Spiral Instruction Motion Example

```
1. --Clockwise N-Spinal spiral motion
2. PTP (DW01100,0,2,50,0,0, -30,0,0) -- Using PTP to move to the starting point of the spiral line (fixed motion mode)
3. --DW01 Spiral Starting Point 1 Name, 100 Debugging Speed, 2 Tool Coordinate Offset, (50, 0, 0, -30, 0, 0) - Offset Parameters (x, y, z, rx, ry, rz)
4. NewSpiral(DW01,100,2,50,0,0,30,0,0,5,30,50,10,15,0)
5. --DW01 Spiral Starting Point 1 Name, 100 Debugging Speed, 2 Tool Coordinate Offset, (50, 0, 0, -30, 0, 0) - Offset Parameters (x, y, z, rx, ry, rz), 5 Spiral Circles, 30 Spiral Tilt Angle, 50 Initial Radius, 10 Radius Increment, 15 Axis Direction Increment, 0 Clockwise

6. --Counterclockwise N-Spiral spiral motion
7. PTP (DW01100,0,2,50,0,0, -30,0,0) -- Use PTP to move to the starting point of the spiral line
8. NewSpiral(DW01,100,2,50,0,0,30,0,0,5,30,50,10,15,1)
9. --DW01 Spiral starting point 1 name, 100 Debugging speed, 2 Tool coordinate offset, (50, 0, 0, -30, 0, 0) - Offset Parameters (x, y, z, rx, ry, rz), 5 Spiral turns, 30 Spiral inclination angle, 50 Initial radius, 10 Radius increment, 15 Axis direction increment, 1 Counter clockwise
```

3.2.7 Horizontal Spiral

The H-Spiral horizontal spiral motion is completed by the combination of Horizon Spiral Motion Start and Horizon Spiral Motion End.

Horizon Spiral Motion Start: Horizontal spiral motion begins

Table 3-23 Detailed Parameters of Horizon Spatial Motion Start

Attribute	Explanation
Prototype	HorizonSpiralMotionStart (rad， vel, rot_direction, circle_angle)
Description	Horizontal spiral motion begins

Table 3-23 (Continued)

Attribute	Explanation
Parameter	• rad: Spiral radius, unit [mm]; • vel: rotational speed, unit [rev/s]; • rot_direction: Rotation direction, 0- clockwise, 1- counterclockwise; • circle_angle Spiral inclination angle, unit [°]
Return value	null

Horizon Spiral Motion End: Ending Horizontal Spiral Motion

Table 3-24 Detailed Parameters of Horizon Spatial MotionEnd

Attribute	Explanation
Prototype	HorizonSpiralMotionEnd ()
Description	Horizontal spiral motion ends
Parameter	null
Return value	null

Code 3-11 H-Spiral Horizontal Spiral Motion Example

```
1.  --Clockwise H-Spiral horizontal spiral
2.  HorizonSpiralMotionStart(30,2,0,20)
3.  --Horizontal spiral, 30- rotation radius, 2- selected speed, 0- clockwise rotation, 20- rotation
    tilt angle
4.  Lin(DW01,100,-1,0,0)
5.  Horizon Spiral MotionEnd() -- End of Horizontal Spiral
6.
7.  --Counterclockwise H-Spiral horizontal spiral
8.  HorizonSpiralMotionStart(30,2,1,20)
9.  --Horizontal spiral, 30- rotation radius, 2- selected speed, 1- counterclockwise rotation, 20-
    rotation tilt angle
10. Lin(DW01,100,-1,0,0)
11. Horizon Spiral MotionEnd() -- End of Horizontal Spiral
```

3.2.8 Spline

The spline instruction is divided into three parts: spline group start, spline segment, and spline group end. The spline group start is the starting symbol of spline motion, and the current node graph of the spline segment includes SPL, SLIN, and SCIRC. The spline group end is the ending symbol of spline motion.

SplineStart: Spline motion begins

Table 3-25 Detailed Parameters of SplineStart

Attribute	Explanation
Prototype	SplineStart ()
Description	Spline group starts
Parameter	null
Return value	null

SPL method one: SPTP type spline segments

Table 3-26 Detailed Parameters of SPTP

Attribute	Explanation
Prototype	SPTP(point_name, ovl)
Description	SPTP spline segment
Parameter	• point_name: Target point name; • ovl: Debugging speed, range [0~100%].
Return value	null

SPL method 2: SplinePTP type spline segments

Table 3-27 Detailed Parameters of SplinePTP

Attribute	Explanation
Prototype	SplinePTP (j1, j2, j3, j4, j5, j6, x, y, z, rx, ry, rz, tool, user, speed, acc, ovl)
Description	SplinePP spline motion
Parameter	• j1~j6: Target joint position, unit [°]; • x, y, z, rx, ry, rz: Cartesian pose of the target, unit [mm] [°]; • tool: tool number; • user: workpiece number; • speed: speed, range [0~100%]; • acc: Acceleration, range [0~100%], temporarily not open; • ovl: Debugging speed, range [0~100%].
Return value	null

SLIN method 1: Spline segments of SLIN type

Table 3-28 SLIN Detailed Parameters

Attribute	Explanation
Prototype	SLIN (point_name, ovl)
Description	SLIN spline segment
Parameter	• point_name: Target point name; • ovl: Debugging speed, range [0~100%].
Return value	null

SLIN method 2: SplineLINE type spline segments

Table 3-29 Detailed Parameters of SplineLINE

Attribute	Explanation
Prototype	SplineLINE (j1, j2, j3, j4, j5, j6, x, y, z, rx, ry, rz, tool, user, speed, acc, ovl)
Description	SplineLINE spline segment
Parameter	• j1~j6: Target joint position, unit [°]; • x, y, z, rx, ry, rz: Cartesian pose of the target, unit [mm] [°]; • tool: tool number; • user: workpiece number; • speed: speed, range [0~100%]; • acc: Acceleration, range [0~100%], temporarily not open; • ovl: Debugging speed, range [0~100%].
Return value	null

SCIRC Method 1: Spline Segment of SCIRC Type

Table 3-30 Detailed Parameters of SCIRC

Attribute	Explanation
Prototype	SCIRC(pos_p_name, pos_t_name, ovl)
Description	SCIRC spline segment
Parameter	• pos_p_name: the name of the midpoint of the arc; • pos_t_name: name of the endpoint of the arc; • ovl: Debugging speed, range [0~100%].
Return value	null

SCIRC Method 2: SplineCIRC type spline segments

Table 3-31 Detailed Parameters of SplineCIRC

Attribute	Explanation
Prototype	SplineCIRC (pj1, pj2, pj3, pj4, pj5, pj6, px, py, pz, prx, pry, prz, ptool, puser, pspeed, pacc, tj1, tj2, tj3, tj4, tj5, tj6, tx, ty, tz, trx, try, trz, ttool, tuser, tspeed, tacc,ovl)
Description	SplineCIRC spline segment
Parameter	• pj1~pj6: Joint position of the midpoint of the arc, unit [°]; • px, py, pz, prx, pry, prz: Cartesian pose of the midpoint of the arc, unit [mm] [°]; • ptool: tool number for the midpoint of the arc; • pusher: workpiece number at the midpoint of the arc; • pspeed: velocity at the midpoint of the arc, range [0~100%]; • pacc: Acceleration at the midpoint of the arc, range [0~100%], temporarily closed; • tj1~tj6: Joint position at the end of the arc, unit [°]; • tx, ty, tz, trx, try, trz: Cartesian pose of the endpoint of the arc, unit [mm] [°]; • ttool: tool number for the endpoint of the arc; • tuser: Arc endpoint workpiece number; • tspeed: End velocity of arc, range [0~100%]; • tacc: End acceleration of arc, range [0~100%], temporarily closed; • ovl: Debugging speed, range [0~100%].
Return value	null

SplineEnd: End of spline group

Table 3-32 Detailed Parameters of SplineEnd

Attribute	Explanation
Prototype	SplineEnd ()
Description	SplineEnd spline group ends
Parameter	null
Return value	null

Code 3-12 Example of Same Way Movement

```
1.  --Spline motion of SPTP spline segments
2.  SplineStart() -- spline motion begins
3.  SPTP (DW01100) -- DW01- Point Name, 100- Debugging Speed
4.  SPTP (DW02100) -- DW02- Point Name, 100- Debugging Speed
```

Code 3-12 (continued)

```
5.  SPTP (DW03100) -- DW03- Point Name, 100- Debugging Speed
6.  SPTP (DW04100) -- DW04- Point Name, 100- Debugging Speed
7.  SplineEnd() -- End of spline motion
8.
9.  --Spline motion of SLIN spline segments
10. SplineStart() --spline motion begins
11. SLIN (DW01100) -- DW01- Point Name, 100- Debugging Speed
12. SLIN (DW02100) -- DW02- Point Name, 100- Debugging Speed
13. SLIN (DW03100) -- DW03- Point Name, 100- Debugging Speed
14. SLIN (DW04100) -- DW04- Point Name, 100- Debugging Speed
15. SplineEnd() -- End of spline motion

16. --Spline motion of SCIRC spline segments
17. SplineStart() -- spline motion begins
18. SCIRC (DW01, DW02100) -- DW01- midpoint name of arc, endpoint name of DW02 arc,
    100- debugging speed 100%
19. SCIRC (DW03, DW04100) -- DW03- midpoint name of arc, endpoint name of DW04 arc,
    100- debugging speed 100%
20. SCIRC (DW05, DW06100) -- DW05- Center Point Name of Arc, End Point Name of DW06
    Arc, 100- Debugging Speed 100%
21. SCIRC (DW07, DW08100) -- DW07- midpoint name of arc, endpoint name of DW08 arc,
    100- debugging speed 100%
22. SplineEnd() -- End of spline motion
```

Code 3-13 Example of Spline Motion in Method 2

```
1.  --Spline motion of SplinePP spline segments
2.  SplineStart() -- spline motion begins
3.  SplinePTP(-88.938,-67.089,-119.074,-57.750,78.739,-53.107,-154.495,-456.371,271.098,-
    172.005,-27.192,-130.384,1,0,100,180,100)
4.  SplinePTP(-50.137,-67.089,-119.074,-57.750,78.739,-53.108,165.568,-452.472,271.098, -
    172.005, -27.192,-91.582,1,0,100,180,100)
5.  SplinePTP(-116.604,-103.398,-106.020,-60.282,89.088,-26.541,-340.231,-440.449,
    59.996,179.861,-0.950,179.936,1,0,100,180,100)
6.  SplinePTP(-117.355,-89.202,-120.591,-59.927,89.057,-27.266,-297.517,-
    341.920,69.240,179.817,-0.966,179.910,1,0,100,180,100)
7.  SplineEnd() -- End of spline motion

8.  --Spline motion of SplineLINE spline segments
9.  SplineStart() -- spline motion begins
10. SplineLINE(-88.938,-67.089,-119.074,-57.750,78.739,-53.107,-154.495,-456.371,
    271.098, -172.005,-27.192,-130.384,1,0,100,180,100)
```

Code 3-13 (continued)

```
11. SplineLINE(-50.137,-67.089,-119.074,-57.750,78.739,-53.108,165.568,-452.472,
    271.098,-172.005,-27.192,-91.582,1,0,100,180,100)
12. SplineLINE(-116.604,-103.398,-106.020,-60.282,89.088,-26.541,-340.231,-440.449,
    59.996,179.861,-0.950,179.936,1,0,100,180,100)
13. SplineLINE(-117.355,-89.202,-120.591,-59.927,89.057,-27.266,-297.517,-341.920,
    69.240,179.817,-0.966,179.910,1,0,100,180,100)
14. SplineEnd() -- End of spline motion

15. --Spline Motion of SplineCIRC Spline Segment
16. SplineStart() - spline motion begins
17. SplineCIRC(-88.938, -67.089, -119.074, -57.750, 78.739, -53.107, -154.495, -456.371,
    271.098, -172.005, -27.192, -130.384, 1, 0, 100, 180, -50.137, -67.089, -119.074, -57.750,
    78.739, -53.108, 165.568, -452.472, 271.098, -172.005, -27.192, -91.582, 1,0,100,180,100)
18.
19. SplineCIRC(-116.604, -103.398, -106.020, -60.282, 89.088, -26.541, -340.231, -440.449,
    59.996, 179.861, -0.950, 179.936, 1, 0, 100, 180, -117.355, -89.202, -120.591, -59.927,
    89.057, -27.266,-297.517,-341.920,69.240,179.817,-0.966,179.910,1,0,100,180,100)
20.
21. SplineCIRC(-110.420, -104.178, -103.638, -59.952, 89.153, -26.480, -297.923, -494.668,
    71.977, -178.379,-1.753,-173.981,1,0,100,180,-115.854,-85.308,-123.979,-59.371,89.058,-
    27.513,-279.135,-330.367,72.864,-179.245,-1.455,-178.362,1,0,100,180,100)
22.
23. SplineCIRC(-108.752,-104.491,-103.204,-59.601,89.035,-26.465,-285.668,-507.818,
    73.101, -178.008,-2.068,-172.346,1,0,100,180,-123.459,-95.123,-113.554,-62.039,87.801,
    -26.914,-361.158,-339.783,72.733,178.366,-1.636,173.492,1,0,100,180,100)
24. SplineEnd() -- End of spline motion
```

3.2.9 New spline

NewSplineStart: New spline multi-point trajectory start

Table 3-33 rameters of NewSplineStart

Attribute	Explanation
Prototype	NewSplineStart (Con_mode, Gac_time)
Description	New spline multi-point trajectory starting
Parameter	• Con_mode: Control mode, 0-Arc transition point, 1-Given transition point; • Gac_time: Global average connection time, greater than 10.
Return value	null

NewSP: Method 1: New spline multi-point trajectory segment

Table 3-34 Detailed Parameters of NewSP

Attribute	Explanation
Prototype	NewSP (point_name, ovl, blendR, islast_point)
Description	New spline multi-point trajectory segment
Parameter	• point_name: Point name; • ovl: Debugging speed, range [0~100%]; • blendR: Smooth radius [0~1000], unit [mm]; • islast_point: Is it the last point? 0- No, 1- Yes.
Return value	null

NewSplinePoint: Method 2: New Spline Multi point Trajectory Segment

Table 3-35 Detailed Parameters of NewSplinePoint

Attribute	Explanation
Prototype	NewSplinePoint(j1, j2, j3, j4, j5, j6, x, y, z, rx, ry, rz, tool, user, speed, acc, ovl , blendR)
Description	New spline multi-point trajectory segment
Parameter	j1~j6: Target joint position, unit [°]; x, y, z, rx, ry, rz: Cartesian pose of the target, unit [mm] [°]; tool: tool number; user: workpiece number; speed: speed, range [0~100%]; acc: Acceleration, range [0~100%], temporarily not open; ovl: Debugging speed, range [0~100%]; blendR: [-1] - Non smooth, [0~1000] - Smooth radius, unit [mm].
Return value	null

NewSplineEnd: End of spline group

Table 3-36 Detailed Parameters of NewSplineEnd

Attribute	Explanation
Prototype	NewSplineEnd ()
Description	End of new spline group
Parameter	null
Return value	null

Code 3-14 Method 1: New Spline Instruction Motion Example

```
1.  --New Spline Motion of N-Spline Arc Transition Point Control mode
2.  NewSplineStart (0,10) -- spline motion starts, 0-arc transition point control mode, 10- global average connection time 10ms
3.  NewSP (DW01100,10,0) -- DW01- Point name, 100- Debugging speed, 10- Smooth transition radius of 10mm, 0- Not the last point
4.  NewSP (DW02100,10,0) -- DW02- Point name, 100- Debugging speed, 10- Smooth transition radius of 10mm, 0- Not the last point
5.  NewSP (DW03100,10,0) -- DW03- Point name, 100- Debugging speed, 10- Smooth transition radius of 10mm, 0- Not the last point
6.  NewSP (DW04100,10,1) -- DW04- Point name, 100- Debugging speed, 10- Smooth transition radius of 10mm, 1- is the last point
7.  NewSplineEnd() - End of new spline motion

8.  -New spline motion with given path point control mode in N-Spline
9.  NewSplineStart (1,10) -- spline motion starts, 1- given path point control mode, 10- global average connection time 10ms
10. NewSP (DW01100,10,0) -- DW01- Point name, 100- Debugging speed, 10- Smooth transition radius of 10mm, 0- Not the last point
11. NewSP (DW02100,10,0) -- DW02- Point name, 100- Debugging speed, 10- Smooth transition radius of 10mm, 0- Not the last point
12. NewSP (DW03100,10,0) -- DW03- Point name, 100- Debugging speed, 10- Smooth transition radius of 10mm, 0- Not the last point
13. NewSP (DW04100,10,1) -- DW04- Point name, 100- Debugging speed, 10- Smooth transition radius of 10mm, 1- is the last point
14. NewSplineEnd() -- End of new spline motion
```

Code 3-15 Method 2 New Spline Instruction Motion Example

```
1.  --New Spline Motion of N-Spline Arc Transition Point Control mode
2.  NewSplineStart (0,10) -- spline motion starts, 0-arc transition point control mode, 10- global average connection time 10ms
3.
4.  NewSplinePoint(-88.938,-67.089,-119.074,-57.750,78.739,-53.107,-154.495,-456.371,271.098,-172.005,-27.192,-130.384,1,0,100,180,100,10,0)
5.  NewSplinePoint(-50.137,-67.089,-119.074,-57.750,78.739,-53.108,165.568,-452.472,271.098,-172.005,-27.192,-91.582,1,0,100,180,100,10,0)
6.  NewSplinePoint(-116.604,-103.398,-106.020,-60.282,89.088,-26.541,-340.231,-440.449,59.996,179.861,-0.950,179.936,1,0,100,180,100,10,0)
7.  NewSplinePoint(-117.355,-89.202,-120.591,-59.927,89.057,-27.266,-297.517,-341.920,69.240,179.817,-0.966,179.910,1,0,100,180,100,10,1)
```

Code 3-15 (continued)

```
8.  NewSplineEnd() -- End of new spline motion
9.
10. --New spline motion with given path point control mode in N-Spline
    NewSplineStart(1,10)
    --Spline motion begins, 1-given path point control mode, 10 global average connection time
    10ms

11. NewSplinePoint(-88.938,-67.089,-119.074,-57.750,78.739,-53.107,-154.495,-
    456.371,271.098,-172.005,-27.192,-130.384,1,0,100,180,100,0,0)
12. NewSplinePoint(-50.137,-67.089,-119.074,-57.750,78.739,-53.108,165.568,-
    452.472,271.098,-172.005,-27.192,-91.582,1,0,100,180,100,0,0)
13. NewSplinePoint(-116.604,-103.398,-106.020,-60.282,89.088,-26.541,-340.231,-
    440.449,59.996,179.861,-0.950,179.936,1,0,100,180,100,0,0)
14. NewSplinePoint(-117.355,-89.202,-120.591,-59.927,89.057,-27.266,-297.517,-
    341.920,69.240,179.817,-0.966,179.910,1,0,100,180,100,0,1)
15. NewSplineEnd() -- End of new spline motion
```

3.2.10 Swing

WeaveStart: Swing begins

Table 3-37 Parameters of WeaveStart

Attribute	Explanation
Prototype	WeaveStart(weaveNum)
Description	Initial Swing
Parameter	• weaveNum: Configuration number for swing welding Parameters.
Return value	null

WeaveEnd: End of swing

Table 3-38 Detailed Parameters of WeaveEnd

Attribute	Explanation
Prototype	WeaveEnd(weaveNum)
Description	Terminal swing
Parameter	• weaveNum: Configuration number for swing welding Parameters.
Return value	null

WeaveStartSim: Simulation swing begins

Table 3-39 Detailed Parameters of WeaveStartSim

Attribute	Explanation
Prototype	WeaveStartSim(weaveNum)
Description	Simulation swing begins
Parameter	• weaveNum: Configuration number for swing welding Parameters.
Return value	null

WeaveEndSim: Simulation swing ends

Table 3-40 Detailed Parameters of WeaveEndSim

Attribute	Explanation
Prototype	WeaveEndSim (weaveNum)
Description	Simulation swing ends
Parameter	• weaveNum: Configuration number for swing welding Parameters.
Return value	null

WeaveInspectStart: Start trajectory warning

Table 3-41 WeaveInspectStart Detailed Parameters

Attribute	Explanation
Prototype	WeaveInspectStart (weaveNum)
Description	Start trajectory warning
Parameter	• weaveNum: Configuration number for swing welding Parameters.
Return value	null

WeaveInspectEnd: Stop trajectory warning

Table 3-42 WeaveInspectEnd Detailed Parameters

Attribute	Explanation
Prototype	WeaveInspectEnd (weaveNum)
Description	Stop trajectory warning
Parameter	• weaveNum: Configuration number for swing welding Parameters.
Return value	null

Code 3-16 Swing Command Motion Example

```
1.  WeaveInspectStart(0);-- Start trajectory warning
1.  Lin(DW01,100,0,0,0)
2.  WeaveInspectEnd(0);-- Stop trajectory warning
3.
4.  WeaveStartSim (weaveNum) -- Simulate swing start
5.  Lin(DW01,100,0,0,0)
6.  WeaveEndSim (weaveNum) -- simulation swing ends
7.
8.  WeaveStart (weaveNum) -- Start swinging
9.  Lin(DW01,100,0,0,0)
10. WeaveEnd (weaveNum) -- End swing
```

3.2.11 Trajectory Reproduction

LoadTPD: Track Preloading

Table 3-43 Detailed Parameters of LoadTPD

Attribute	Explanation
Prototype	LoadTPD(name)
Description	Trajectory preloading
Parameter	• name: Track name.
Return value	null

MoveTPD: Trajectory Reproduction

Table 3-44 Detailed Parameters of MoveTPD

Attribute	Explanation
Prototype	MoveTPD(name, blend, ovl)
Description	Trajectory reproduction
Parameter	• name: Trajectory Name,/fruser/traj/trajHelix_aima_2.txt;; • blend: Smooth or not, 0-Not smooth, 1-Smooth; • ovl: Debugging speed, range [0~100].
Return value	null

Code 3-17 Trajectory Reproduction Example

```
1.  LoadTPD("20lin")
2.  MoveTPD("20lin",0,25)
```

3.2.12 point offset

The point offset command is an overall offset command. By inputting various offsets, the open and close commands are added to the program. The motion commands between the start and close will be offset based on the base coordinates (or workpiece coordinates).

PointsOffsetEnable: The overall offset of the point position begins

Table 3-45 PointsOffsetEnable detailed Parameters

Attribute	Explanation
Prototype	PointsOffsetEnable(flag, x,y,z,rx,ry,rz)
Description	The overall offset of the point position begins
Parameter	• flag: offset in the base coordinate or workpiece coordinate system, offset in the tool coordinate system; • x, y, z, rx, ry, rz: pose offset, unit [mm] [°].
Return value	null

PointsOffsetDisable: End of overall point offset

Table 3-46 PointsOffsetDisable Detailed Parameters

Attribute	Explanation
Prototype	PointsOffsetDisable()
Description	The overall offset of the point position has ended
Parameter	null
Return value	null

Code 3-18 point offset example

```
1. PointsOffsetEnable (0,0,0,10,0,0,0)
2. --Starting from the overall offset of the point position, 0-offset in the base coordinate or workpiece coordinate system, (0,0,10,0,0,0) - offset amount
3. PTP (DW01100, -1,0) PTP motion
4. PointsOffsetDisab() -- End of overall point offset
```

3.2.13 Servo

Servo control (Cartesian space motion) instructions, which can control robot

motion through absolute pose control or based on current pose offset.

ServoMoveStart: Servo motion begins

Table 3-47: Detailed Parameters of ServoMoveStart

Attribute	Explanation
Prototype	ServoMoveStart()
Description	Servo motion begins
Parameter	null
Return value	null

ServoMoveEnd: End of servo motion

Table 3-48: Detailed Parameters of ServoMoveEnd

Attribute	Explanation
Prototype	ServoMoveEnd()
Description	Servo motion ends
Parameter	null
Return value	null

ServoCart: Cartesian Space Servo mode Motion

Table 3-49: Detailed Parameters of ServoCart

Attribute	Explanation
Prototype	ServoCart (mode, x, y, z, Rx, Ry, Rz, pos_gainx, pos_gainy, pos_gainz, pos_gainrx, pos_gainry, pos_gainrz, acc, vel, cmdT, filterT, gain)
Description	Cartesian spatial servo mode motion
Parameter	• mode: [0] - Absolute motion (base coordinate system), [1] - Incremental motion (base coordinate system), [2] - Incremental motion (tool coordinate system); • x, y, z, Rx, Ry, Rz: Cartesian pose or pose increment, unit [mm]; • Pos_gainx, pos_gainy, pos_gainz, pos_gainrx, pos_gainry, pos_gainrz: pose incremental proportional coefficients, only effective under incremental motion, range [0~1]; • acc: Acceleration, range [0~100]; • vel: speed, range [0~100]; • CmdT: Instruction issuance cycle, unit s, recommended range [0.001~0.0016]; • FilterT: filtering time, filtering time, in seconds; • Gain: Proportional amplifier at the target position.
Return value	null

Code 3-19 Servo Example

```
1.  --Servo control
2.  mode = 2
3.  --[0] - Absolute motion (base coordinate system), [1] - Incremental motion (base coordinate
    system), [2] - Incremental motion (tool coordinate system)
4.  count = 0
5.  ServoMoveStart() - servo motion starts
6.  While (count<100) do
7.  ServoCart (mode, 0.0, 0.0, 0.5, 0.0, 0.0, 0.0, 40) -- Cartesian space servo mode motion
8.  count = count + 1
9.  WaitMs(10)
10. end
11. ServoMoveEnd() -- End of servo motion
```

3.2.14 Trajectory

The Trajtory instruction is a universal interface for cameras to directly provide trajectories, which can be imported into the system to enable robots to move according to the trajectory of the imported file when there are already discrete trajectory point files in a fixed format.

1. Trajectory file import function: Select the local computer file to import into the robot control system;

2. Trajectory Preloading: Select the imported trajectory file and load it through instructions;

3. Trajectory motion: The robot motion is issued through a combination of preloaded trajectory files and selected debugging speed commands;

4. print trajectory point numbers: print trajectory point numbers during the robot's trajectory to view the current progress of the movement.

LoadTrajectory: Trajectory Preloading

Table 3-50 Detailed Parameters of LoadTrajectory

Attribute	Explanation
Prototype	LoadTrajectory (name)
Description	Trajectory preloading
Parameter	• name: Trajectory name, such as:/fruser/traj/trajHelix_aim_1. txt.
Return value	null

Obtain the starting point Parameters, tool coordinate number, and workpiece coordinate number of the trajectory through Get Trajectory Start Pose, Get

ActualTCPNum, and Get ActualWObjNum, respectively.

Get trajectory starting pose: Get trajectory starting pose

Table 3-51 Detailed Parameters of Get TrajectoryStartPose

Attribute	Explanation
Prototype	GetTrajectoryStartPose (name)
Description	Obtain the starting pose of the trajectory
Parameter	• name: Trajectory name, such as:/fruser/traj/trajHelix_aim_1. txt.
Return value	desc_pose {x,y,z,rx,ry,rz}

FHIR ctualTCPNum: Get the current tool coordinate system number

Table 3-52: Detailed Parameters of VNet TCPNum

Attribute	Explanation
Prototype	GetActualTCPNum (flag)
Description	Obtain the current tool coordinate system number
Parameter	• flag: 0- blocking, 1- non blocking default 1.
Return value	Tool_id: Tool coordinate system number

FHIR ctualWObjNum: Get the current workpiece coordinate system number

Table 3-53 Detailed Parameters of vDctualWObjNum

Attribute	Explanation
Prototype	GetActualWObjNum (flag)
Description	Obtain the current tool coordinate system number
Parameter	• flag: 0- blocking, 1- non blocking default 1.
Return value	Wobj-id: workpiece coordinate system number

MoveCart: Cartesian Space Point to Point Motion

Table 3-54: Detailed Parameters of MoveCart

Attribute	Explanation
Prototype	MoveCart (desc_pos, ool, user, vel, acc, ovl, blendT, config)
Description	Cartesian space point-to-point motion

Table 3-54(Continued)

Attribute	Explanation
Parameter	• Desc_pos: Target Cartesian position; • tool: tool number, [0~14]; • user: workpiece number, [0~14]; vel: speed, range [0~100], default is 100; • Acc: Acceleration, range [0~100], temporarily not open, default is 100; • ovl: Debugging speed, range [0~100%]; • blend T: [-1.0] - Motion in place (blocking), [0~500] - Smooth time (non blocking), unit [ms] defaults to -1.0; • Config: Joint configuration, [-1] - solve based on the current joint position, [0~7] - solve based on the joint configuration, default is -1.
Return value	null

MoveTrajectory: Trajectory Reproduction

Table 3-55 Detailed Parameters of MoveTrajectory

Attribute	Explanation
Prototype	MoveTrajectory (name, ovl)
Description	Trajectory reproduction
Parameter	• name: Trajectory name, such as:/fruser/traj/trajHelix_aim_1. txt; • ovl: Debugging speed, range [0~100%].
Return value	null

Get Trajectory PointNum: Get trajectory point number

Table 3-56 Detailed Parameters of Get TrajectoryPointNum

Attribute	Explanation
Prototype	GetTrajectoryPointNum ()
Description	Obtain trajectory point number
Parameter	null
Return value	Num: Trajectory point number

Code 3-20 Trajectory Example

```
1.  --Trajectory
2.  LoadTrajectory ("/fruser/traj/trajHelix_ima_1. txt") -- Absolute path of preloaded trajectory file
```

Code 3-20 (continued)

```
3.  startPose = GetTrajectoryStartPose("/fruser/traj/trajHelix_aima_1.txt")
4.  --Obtain the starting pose of the trajectory
5.  Tool_num=VNet TCPNum() -- Get the current tool coordinate system number
6.  Wobj_num=vDctualWObjNum() -- Get the current workpiece coordinate system number
7.  MoveCart(startPose, tool_num,wobj_num,100,100,25,-1,-1)
8.  --Cartesian space point-to-point motion to the starting point of the trajectory, startPose - target
    Cartesian position, 100- velocity, 100- acceleration, -1- stop in place, -1- joint solved according
    to the configuration
9.  MoveTrajectory("/fruser/traj/trajHelix_aima_1.txt",25)
10. --Trajectory reproduction,/fruser/traj/trajHelix_aim_1.txt - Trajectory file name, 25- Debugging
    speed (debugging speed)
11. Num=Get Trajectory PointNum() -- Get trajectory point number
12. Register Var ("number", "num") -- print out the number information
```

3.2.15 Trajectory J

The Trajtory J instruction, like Trajtory, is a universal interface suitable for cameras to directly provide trajectories. It can be imported into the system when there are already discrete trajectory point files in a fixed format, allowing the robot to move according to the trajectory of the imported file.

LoadTrajectory J: Trajectory Preprocessing

Table 3-57 Detailed Parameters of LoadTrajectory J

Attribute	Explanation
Prototype	LoadTrajectoryJ(name, ovl, opt)
Description	Trajectory preprocessing
Parameter	• name: Track name, such as:/fruser/traj/trajHelix_aima_2.txt; • ovl: Debugging speed, range [0~100]; • opt: 0- Path point, 1- Control point.
Return value	null

MoveTrajectory J: Trajectory Reproduction

Table 3-58: Detailed Parameters of MoveTrajectory J

Attribute	Explanation
Prototype	MoveTrajectoryJ ( )
Description	Trajectory reproduction
Parameter	null
Return value	null

Code 3-21 Trajectory J Example

```
1.  LoadTrajectoryJ("/fruser/traj/trajHelix_aima_2.txt",30,0)
2.  --Trajectory J preprocessing,/fruser/traj/trajHelix_ima_2. txt - Trajectory file name, 30-Debugging speed, 1- Control points
3.  startPose = GetTrajectoryStartPose("/fruser/traj/trajHelix_aima_2.txt")
4.  --Obtain the starting pose of the trajectory
5.  Tool_num=VNet TCPNum() -- Get the current tool coordinate system number
6.  Wobj_num=vDctualWObjNum() -- Get the current workpiece coordinate system number
7.  MoveCart(startPose, tool_num,wobj_num,100,100,25,-1,-1)
8.  --Cartesian space point-to-point motion to the starting point of the trajectory, startPose - target Cartesian position, 100- velocity, 100- acceleration, -1- stop in place, -1- joint solved according to the configuration
9.  MoveTrajectoryJ()
10. Num=Get Trajectory PointNum() -- Get trajectory point number
11. Register Var ("number", "num") -- print out the number information
```

3.2.16 DMP

DMP/dmpMotion is a trajectory imitation learning method that requires prior planning of reference trajectories. The specific path of DMP is a new trajectory that imitates the reference trajectory from a new starting point.

DMP: Trajectory Imitation

Table 3-59 Detailed Parameters of DMP

Attribute	Explanation
Prototype	DMP (point_name, ovl)
Description	Trajectory imitation
Parameter	• point_name: Target Point Name • ovl: Debugging speed, range [0~100%].
Return value	null

DmpMotion: Trajectory imitation

Table 3-60 Detailed Parameters of dmpMotion

Attribute	Explanation
Prototype	dmpMotion (joint_pos, desc_pos , tool, user, vel, acc, ovl, exaxis_pos)
Description	Trajectory imitation

Table 3-60(Continued)

Attribute	Explanation
Parameter	• Joint_pos: Target joint position, unit [°]; • Desc_pos: Target Cartesian pose, unit [mm] [°]. The default initial value is [0.0, 0.0, 0.0, 0.0, 0.0, 0.0], and the default value is called the forward kinematics solution Return value; • tool: tool number, [0~14]; • user: workpiece number, [0~14]; • vel: Speed percentage, [0~100] defaults to 100.0; • acc: Acceleration percentage, [0~100], temporarily not open; • ovl: Debugging speed, range [0~100%]; • Exaxis_pos: The default positions for external axis 1 to external axis 4 are [0.0, 0.0, 0.0, 0.0].
Return value	null

Code 3-22 Trajectory Imitation Example

```
1. --DMP trajectory imitation
2. DMP (DW01100) -- Trajectory Imitation, DW01- Starting Point Name, 100- Debugging Speed
3. --DmpMotion trajectory imitation
4. dmpMotion({-88.938,-67.089,-119.074,-57.750,78.739,-53.107},{-154.495,-456.371, 271.098, -172.005,-27.192,-130.384},1,0,100,180,100, {0.000,0.000,0.000,0.000})
```

3.2.17 Workpiece Conversion

WPTrsf， Workpiece coordinate system conversion, this instruction is implemented by executing internal PTP and LIN instructions, and the point position in the workpiece coordinate system is automatically converted.

WorkPieceTrsfStart: Start of workpiece coordinate conversion

Table 3-61 Detailed Parameters of WorkPieceTrsfStart

Attribute	Explanation
Prototype	WorkPieceTrsfStart (id)
Description	Workpiece coordinate conversion begins
Parameter	• id: Target workpiece coordinate system number, such as 0-wobjcoord0, 1-wobjcoord1.
Return value	null

WorkPieceTrsfEnd: End of workpiece coordinate conversion

Table 3-62 Detailed Parameters of WorkPieceTrsfEnd

Attribute	Explanation
Prototype	WorkPieceTrsfEnd ( )
Description	Workpiece coordinate conversion begins
Parameter	null
Return value	null

Code 3-23 Example of Coordinate Conversion for Workpiece

```
1.  --Perform workpiece coordinate conversion
2.  WorkPieceTrsfStart (1) -- Start of workpiece coordinate conversion, 1 workpiece coordinate
    system number
3.  PTP(DW01,100,0,0)
4.  --DW01- Point name to be converted, 100- Debugging speed, 0- Blocking (stop), 0- No
    offset
5.  PTP(DW02,100,0,0)
6.  --DW02- Point name to be converted, 100- Debugging speed, 0- Blocking (stop), 0- No
    offset
7.  PTP(DW03,100,0,0)
8.  --DW03- Point name to be converted, 100- Debugging speed, 0- Blocking (stop), 0- No
    offset
9.  --End of workpiece conversion
10. WorkPieceTrsfEnd()
```

3.2.18 Tool Conversion

ToolTrsf is a tool coordinate system conversion instruction that automatically converts point positions in the tool coordinate system by executing internal PTP and LIN instructions.

SetToolList: Set tool coordinate system

Table 3-63 SetToolList Detailed Parameters

Attribute	Explanation
Prototype	SetToolList (name)
Description	Set tool coordinate system
Parameter	• name: Target tool coordinate system name, such as toolcoold0, toolcoold1.
Return value	null

ToolTrsfStart: Tool coordinate system conversion begins

Table 3-64 Detailed Parameters of ToolTrsfStart

Attribute	Explanation
Prototype	ToolTrsfStart (id)
Description	Tool coordinate system conversion begins
Parameter	• id: Target tool coordinate system number, such as 0-toolcoold0, 1-toolcoold1.
Return value	null

ToolTrsfEnd: Tool coordinate system conversion completed

Table 3-65 Detailed Parameters of ToolTrsfEnd

Attribute	Explanation
Prototype	ToolTrsfEnd ()
Description	Tool coordinate system conversion completed
Parameter	null
Return value	null

Code 3-24 Tool Coordinate Conversion Example

```
1. --Tool coordinate conversion
2. SetToolList (toolcoord0) -- Set tool coordinate conversion, toolcoord0- Target tool coordinate name
3. ToolTrsfStart (0) -- Tool coordinate conversion begins, 0-Tool coordinate system number
4. PTP (DW01100,0,0) - DW01- Point name to be converted, 100- Debugging speed, 0- Blocking (stop), 0- No offset
5. PTP (DW02100,0,0) -- DW02- Point name to be converted, 100- Debugging speed, 0- Blocking (stop), 0- No offset
6. PTP (DW03100,0,0) -- DW03- Point name to be converted, 100- Debugging speed, 0- Blocking (stop), 0- No offset
7. ToolTrsfEnd() -- Tool coordinate conversion completed
```

3.3 Control instruction

3.3.1 Digital IO

The digital 'IO' instruction is divided into two parts: setting IO (SetDO/SPLCSetDO) and getting IO (dDI/SPLCDetBI).

SetDO: Set the digital quantity blocking output of the control box

Table 3-66 SetDO Detailed Parameters

Attribute	Explanation
Prototype	SetDO (id, status, smooth, thread)
Description	Set control box digital quantity blocking output
Parameter	• id: io number, 0~7: Control box DO0~DO7, 8~15: Control box CO0~CO7; • status:0-Flase，1-True; • smooth:0-Break，1-Serious; • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	null

SPLCsetDO: Set control box digital quantity non blocking output

Table 3-67 Detailed Parameters of SPLCsetDO

Attribute	Explanation
Prototype	SPLCSetDO (id, status)
Description	Set control box digital quantity non blocking output
Parameter	• id: io number, 0~7: Control box DO0~DO7, 8~15: Control box CO0~CO7; • status: 0-False, 1-True.
Return value	null

SetToolDO: Set tool digital quantity to block output

Table 3-68 SetToolDO Detailed Parameters

Attribute	Explanation
Prototype	SetToolDO (id, status, smooth, thread)
Description	Set tool digital quantity blocking output
Parameter	• id: io number, 0 End-DO0, 1 End-DO1; • status:0-False，1-True; • smooth:0-break，1-Serious; • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	null

SPLSetToolDO: Set tool digital quantity non blocking output

Table 3-69 Detailed Parameters of SPLSetToolDO

Attribute	Explanation
Prototype	SPLSetToolDO (id, status, smooth, thread)

Table 3-69(Continued)

Attribute	Explanation
Description	Set tool digital quantity non blocking output
Parameter	• id: io number, 0-End-DO0, 1-End-DO1; • status: 0-False, 1-True.
Return value	null

DDI: Block the acquisition of control box digital input

Table 3-70 Detailed Parameters of dDI

Attribute	Explanation
Prototype	ret = GetDI(id， thread)
Description	Block the acquisition of control box digital input
Parameter	• id: io number, 0~7: control box DI0~DI7, 8~15: control box CI0~CI7; • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	• ret: 0-Invalid, 1-Valid

SPLCSDBI: Non blocking access to IO

Table 3-71: Detailed Parameters of SPLCedEI

Attribute	Explanation
Prototype	SPLCGetDI (id, status, stime)
Description	Non blocking acquisition of control box digital input
Parameter	• id: io number, 0~7: control box DI0~DI7, 8~15: control box CI0~CI7; • status:0-False， 1-True; • Stime: waiting time unit [ms].
Return value	• ret: 0-Invalid, 1-Valid

Get Tool DI: Block the tool from obtaining numerical input

Table 3-72 Detailed Parameters of Get Tool DI

Attribute	Explanation
Prototype	GetToolDI (id, thread)
Description	Block the acquisition of control box digital input
Parameter	• id: io number, 0-EndDI0, 1-EndDI1; • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	• ret: 0-Invalid, 1-Valid.

SPLCSDBI: Non blocking access to IO

Table 3-73: Detailed Parameters of SPLCedEI

Attribute	Explanation
Prototype	SPLCGetToolDI (id, status, stime)
Description	Non blocking acquisition of control box digital input
Parameter	• id: io number, 0-EndDI0, 1-EndDI1; • status:0-False，1-True; • Stime: waiting time unit [ms].
Return value	• ret: 0-Invalid, 1-Valid

Code 3-25 Example of Digital IO

```
1.  --Set digital IO
2.  SetDO (0,1,0,1) -- Set control box digital quantity blocking output
3.  SPLCsetDO (1,1) -- Set control box digital quantity non blocking output
4.  SetToolDO (1,0,1,1) -- Set tool digital quantity to block output
5.  SPLCSetToolDO (1,0) -- Set tool digital quantity non blocking output
6.  --PTP (DW01100,0,0) PTP motion mode
7.  -Get digital IO
8.  Ret1=dDI (0,1) -- Block the acquisition of control box digital input
9.  Ret2=SPLCdEI (1,01000) -- Non blocking acquisition of control box digital input
10. Ret3=Get Tool DI (1,0) -- Block the tool from obtaining numerical input
11. Ret4=SPLCDetToolDI (1,0100) -- Non blocking tool for obtaining numerical input
```

3.3.2 Analog IO

In this instruction, it is divided into two parts: setting analog output (SetAO/PLCSetAO) and obtaining analog input (vDI/SPLCDeAI).

SetAO: Set control box analog blocking output

Table 3-74 SetAO Detailed Parameters

Attribute	Explanation
Prototype	SetAO (id, value, thread)
Description	Set control box analog blocking output
Parameter	• id: io number, 0-AI0, 1-AI1; • value: percentage of current or voltage value, range [0~100%] corresponding to current value [0~20mA] or voltage [0~10V]; • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	null

SPLCSetAO: Set control box analog non blocking output

Table 3-75 Detailed Parameters of SPLCSetAO

Attribute	Explanation
Prototype	SPLCSetAO (id, value)
Description	Set control box analog non blocking output
Parameter	• id: io number, 0-AI0, 1-AI1; • value: Percentage of current or voltage value, range [0~100%] corresponds to current value [0~20mA] or voltage [0~10V].
Return value	null

SetToolAO: Set tool analog output

Table 3-76 SetToolAO Detailed Parameters

Attribute	Explanation
Prototype	SetToolAO (id, value, thread)
Description	Set control box analog blocking output
Parameter	• id: io number, 0-End-AO0; • value: percentage of current or voltage value, range [0~100%] corresponding to current value [0~20mA] or voltage [0~10V]; • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	null

SPLCSetToolAO: Set control box analog non blocking output

Table 3-77 Detailed Parameters of SPLCSetToolAO

Attribute	Explanation
Prototype	SPLCSetToolAO (id, value)
Description	Set control box analog non blocking output
Parameter	• id: io number, 0-End-AO0; • value: Percentage of current or voltage value, range [0~100%] corresponds to current value [0~20mA] or voltage [0~10V].
Return value	null

FHIR I: Obtain analog input from the control box

Table 3-78: Detailed Parameters of FHIR I

Attribute	Explanation
Prototype	GetAI(id, thread)
Description	Block the acquisition of control box analog input
Parameter	• id: io number, 0 AI0, 1 AI1; • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	value: Input current or voltage value percentage, range [0~100] corresponds to current value [0~20mA] or voltage [0~10V]

SPLCVEI: Non blocking acquisition of control box analog input

Table 3-79 Detailed Parameters of SPLCVEAI

Attribute	Explanation
Prototype	SPLCGetAI (id, condition, value, stime)
Description	Non blocking acquisition of control box analog input
Parameter	• id: io number, 0 AI0, 1 AI1; • value: numerical value, 1%~100%; • condition:0 - >， 1 - <; • Stime:maximum time, unit [ms];
Return value	status: return status, 1-successful, 0-failed.

Get Tool AI: Block tool to obtain analog input

Table 3-80 Detailed Parameters of Get Tool AI

Attribute	Explanation
Prototype	GetToolAI (id, thread)
Description	Block the acquisition of control box analog input
Parameter	• id: io number, 0 AI0, 1 AI1; • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	value: Input current or voltage value percentage, range [0~100] corresponds to current value [0~20mA] or voltage [0~10V]

SPLCDetToolAI: Non blocking acquisition of control box analog input

Table 3-81: Detailed Parameters of SPLCDetToolAI

Attribute	Explanation
Prototype	SPLCGetToolAI (id, condition, value, stime)
Description	Non blocking acquisition of control box analog input
Parameter	• id: io number, 0 AI0, 1 AI1; • value: numerical value, 1%~100%; • condition:0->， 1-<; • Stime:maximum time, unit [ms];
Return value	status: return status, 1-successful, 0-failed.

Code 3-26 Simulated IO Example

```
1.  - Set analog quantity
2.  SetAO (0,10,0) - Set control box analog blocking output
3.  SPLCSetAO (1,10) - Set control box analog non blocking output
4.  SetToolAO (0,10,0) - Set tool analog blocking output
5.  SPLCSetToolAO (0,10) - Set tool analog non blocking output
6.
  -Obtain analog quantity
7.  value1=FHIR I (0,0) - Block the acquisition of control box analog input
8.  value2=SPLCVEI (1,0,301000) - Non blocking acquisition of control box analog input
9.  value3=Get Tool AI (0,1) - Block the analog input of the acquisition tool
10. value3=SPLCDetToolAI (0,0,10,50) - Non blocking acquisition tool analog input
```

3.3.3 Virtual IO

Virtual IO is a virtual IO control instruction that sets or retrieves simulated external DI and AI states.

SetVirtualDI: Set up simulated external DI

Table 3-82 SetVirtualDI Detailed Parameters

Attribute	Explanation
Prototype	SetVirtualDI (id, status)
Description	Set up simulated external DI
Parameter	• id: io number, 0~7: control box DI0~DI7, 8~15: control box CI0~CI7; • status: 0-Flash, 1-True.
Return value	null

SetVirtualToolDI: Set simulated external tool DI

Table 3-83 SetVirtualToolDI Detailed Parameters

Attribute	Explanation
Prototype	SetVirtualToolDI (id, status)
Description	Set up simulation external tool DI
Parameter	• id: io number, 0 - End-DI0, 1 - End-DI1; • status: 0-Flash, 1-True.
Return value	null

GetVirtualDI: Get simulated external DI

Table 3-84 Detailed Parameters of GetVirtualDI

Attribute	Explanation
Prototype	GetVirtualDI (id)
Description	Obtain simulated external DI
Parameter	• id: io number, 0~7: control box DI0~DI7, 8~15: control box CI0~CI7.
Return value	ret: 0-Invalid, 1-Valid.

GetVirtualToolDI: Get simulated external tool DI

Table 3-85 Detailed Parameters of GetVirtualToolDI

Attribute	Explanation
Prototype	GetVirtualToolDI (id)
Description	Obtain simulated external tool DI
Parameter	• id: io number, 0 - End-DI0, 1 - End-DI1.
Return value	ret: 0-Invalid, 1-Valid.

SetVirtualAI: Set up simulated external AI

Table 3-86 Detailed Parameters of GetVirtualAI

Attribute	Explanation
Prototype	SetVirtualAI (id, value)
Description	Set up simulated external AI
Parameter	• id: io number, 0 AI0, 1 AI1; • value: Corresponding current value [0~20mA] or voltage value [0~10V].
Return value	null

SetVirtualToolAI: Set up simulated external AI

Table 3-87 Detailed Parameters of GetVirtualToolAI

Attribute	Explanation
Prototype	SetVirtualToolAI (id, value)
Description	Set up simulated external AI
Parameter	• id: io number, 0 End AI0; • value: Corresponding current value [0~20mA] or voltage value [0~10V].
Return value	null

GetVirtualAI， Obtain simulated external AI

Table 3-88 Detailed Parameters of GetVirtualAI

Attribute	Explanation
Prototype	GetVirtualAI (id)
Description	Obtain simulated external AI
Parameter	• id: io number, 0 AI0, 1 AI1.
Return value	value: Input current or voltage value percentage, range [0~100] corresponds to current value [0~20mA] or voltage [0~10V]

GetVirtualToolAI， Obtain simulated external tool AI

Table 3-89 Detailed Parameters of GetVirtualToolAI

Attribute	Explanation
Prototype	value = GetVirtualToolAI (id)
Description	Obtain simulated external tool AI
Parameter	• id: io number, 0-End-AI0.
Return value	value: Input current or voltage value percentage, range [0~100] corresponds to current value [0~20mA] or voltage [0~10V]

Code 3-27 Virtual IO Example

```
1.  --Simulate external DI settings and retrieval
2.  SetVirtualDI (0,1) -- Set simulated external DI, 0-port number DI0, 1-True
3.  SetVirtualAI (0,5) -- Set simulated external AI, 0-port number AI0,5- numerical value 5ma
4.  Ret1=GetVirtualDI (1) -- Get simulated external DI, 1-port number DI1
5.  value1=GetVirtualAI (1) -- Get simulated external AI, 1-port number AI1
6.
```

Code 3-27(Continued)

```
7.  --Simulate external tool DI settings and retrieval
8.  SetVirtualToolDI (1,0) - Set simulation external tool DI
9.  SetVirtualToolAI (0,12) - Set up simulated external tool AI
10. Ret2=GetVirtualToolDI (1) - Get simulated external tool DI
11. value2=GetVirtualToolAI (0) - Get simulated external tool AI
```

3.3.4 Sports DO

The relevant instructions for motion DO are divided into continuous output mode and single output mode to achieve the function of continuously outputting DO signals according to the set interval during linear motion.

MoveDOStart: Parallel setting of control box DO status starts during movement

Table 3-90 Detailed Parameters of MoveDOStart

Attribute	Explanation
Prototype	MoveDOStart (doNum， distance，dutyCycle)
Description	Parallel setting of control box DO status during exercise begins
Parameter	• doNum: Control box DO number, 0~7: Control box DO0~DO7, 8~15: Control box CO0~CO7; • distance: interval distance, range: 0~500, unit [mm, default 10]; • dutyCycle: Output pulse duty cycle unit [%], 0~99, default 50%.
Return value	null

MoveDOStop: Parallel setting of control box DO status to stop during movement

Table 3-91 Detailed Parameters of MoveDOStop

Attribute	Explanation
Prototype	MoveDOStop ()
Description	Parallel setting of control box DO status to stop during exercise
Parameter	null
Return value	null

MoveToolDOStart: Parallel setting of tool DO status during motion begins

Table 3-92 Detailed Parameters of MoveToolDOStart

Attribute	Explanation
Prototype	MoveToolDOStart (doNum， distance，dutyCycle)
Description	Parallel setting tool DO status starts during exercise
Parameter	• doNum: Tool DO number, 0 End-DO0, 1 End-DO1; • distance: interval distance, range: 0~500, unit [mm, default 10]; • dutyCycle: Output pulse duty cycle unit [%], 0~99, default 50%.
Return value	null

MoveToolDOStop: Set tool DO status to stop in parallel during motion

Table 3-93 Detailed Parameters of MoveToolDOStop

Attribute	Explanation
Prototype	MoveToolDOStop ()
Description	Stop the DO state of the parallel setting tool during exercise
Parameter	null
Return value	null

Code 3-28 Motion DO Example

```
1.  --Control box
2.  MoveDOStart (1,10,50)
3.  --Set motion DO continuous output, 1-port number DO1, 10 time interval 10mm, 50 output
    pulse duty cycle 50%
4.  Lin (DW01100, -1,0,0) - Linear Motion
5.  MoveDOStop() - Stop motion DO input
6.
7.  --Tools
8.  MoveToolDOStart(0,10,50)
9.  Lin (DW01100, -1,0,0) -- Linear Motion
10. MoveToolDOStop() -- Stop motion DO input
```

3.3.5 Exercise AO

MoveAO， When used in conjunction with motion commands, it can achieve proportional output of AO signals based on real-time TCP speed during the motion process.

MoveAOStart: Control box motion AO starts

Table 3-94 Detailed Parameters of MoveAOStart

Attribute	Explanation
Prototype	MoveAOStart (AONum, maxTCPSpeed, maxAOPercent, zeroZoneCmp)
Description	Control box motion AO starts
Parameter	• AONum: Control box AO number, 0-AO0,1-AO1; • maxTCpspeed: maximum TCP speed value [1-5000mm/s], default 1000; • maxAOPercent: The AO percentage corresponding to themaximum TCP speed value, with a default of 100%; • ZeroZoneCmp: Dead zone compensation value AO percentage, shaping, default is 20%, range [0-100].
Return value	null

MoveAOStop: Control box motion AO ends

Table 3-95 Detailed Parameters of MoveAOStop1

Attribute	Explanation
Prototype	MoveAOStop()
Description	Control box motion AO ends
Parameter	null
Return value	null

MoveToolAOStart: Tool motion AO starts

Table 3-96 Detailed Parameters of MoveAOStart

Attribute	Explanation
Prototype	MoveToolAOStart (AONum, maxTCPSpeed, maxAOPercent, zeroZoneCmp)
Description	Tool movement AO begins
Parameter	• AONum: Control box AO number, 0-AO0,1-AO1; • maxTCpspeed: maximum TCP speed value [1-5000mm/s], default 1000; • maxAOPercent: The AO percentage corresponding to themaximum TCP speed value, with a default of 100%; • ZeroZoneCmp: Dead zone compensation value AO percentage, shaping, default is 20%, range [0-100].
Return value	null

MoveToolAOStop: Tool motion AO ends

Table 3-97 Detailed Parameters of MoveAOStop

Attribute	Explanation
Prototype	MoveToolAOStop ()
Description	Tool motion AO ends
Parameter	null
Return value	null

Code 3-29 Motion AO Example

```
1. --Control box
2. MoveAOStart (11000100,20) -- Set motion AO output, 1-port number AO11000-maximum TCP speed, 100-maximum TCP speed percentage, 20- dead zone compensation value AO percentage
3. Lin (DW01100,0,0,0) -- Linear Motion
4. MoveAOStop() -- Stop motion AO output
5. -- Tools
6. MoveToolAOStart (0,1000,100,20)
7. Lin(DW01,100,0,0,0)
8. MoveToolAOStop ()
```

3.3.6 Expanding IO

Aux IO is a command function for external IO expansion control between robots and PLCs, which requires the robot to establish UDP communication with the PLC. On the basis of the original 16 input/output channels, 128 input/output channels can be expanded.

ExtDevSetUDPComParam: Configure UDP communication data

Table 3-98 Detailed Parameters of ExtDevSetUDPComParam

Attribute	Explanation
Prototype	ExtDevSetUDPComParam (ip, port, period)
Description	UDP Extended Axis Communication Parameter Configuration
Parameter	• IP: PLC IP address; • Port: Port number; • Period: Communication cycle (ms).
Return value	null

ExtDevLoadUDPDriver loads UDP communication

Table 3-99 Detailed Parameters of ExtDevLoadUDPDriver

Attribute	Explanation
Prototype	ExtDevLoadUDPDriver()
Description	Load UDP communication
Parameter	null
Return value	null

Code 3-30 Motion AO Example

```
1. --UDP Extended Axis Communication Parameter Configuration and Loading
2. ExtDevSetUDPComParam("192.168.58.88",2021,2)
3. --UDP communication configuration, "192.168.58.88" - IP address, 2021- port number, 2-
   communication cycle
4. ExtDevLoadUDPDriver() -- Load UDP driver to enable communication.
5. WaitMs (500) - Wait for 500 milliseconds to ensure that the UDP driver has been loaded
   correctly.
```

SetAuxDO: Set Extended DO

Table 3-100 SetAuxDO Detailed Parameters

Attribute	Explanation
Prototype	SetAuxDO (DONum, status, smooth, thread)
Description	Set extended DO
Parameter	• DOUm: DO number, range [0~127]; • status:0-Flase，1-True; • smooth:0-Break，1-Serious; • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	null

VDuxDI: Get Extended DI

Table 3-101 Detailed Parameters of vDuxDI

Attribute	Explanation
Prototype	GetAuxDI (DINum)
Description	Get extended DI value
Parameter	• DINum: DI number, range [0~127].
Return value	IsOpen: 0-off; 1- Open

SetAuxAO: Set Extended AO

Table 3-102 SetAuxAO Detailed Parameters

Attribute	Explanation
Prototype	SetAuxAO (AONum, value, thread)
Description	Set up extended AO
Parameter	• id: AO number, range [0~3]; • value: percentage of current or voltage value, range [0~100%] corresponding to current value [0~20mA] or voltage [0~10V]; • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	null

GetAuxAI: Get Extended AI value

Table 3-103 Detailed Parameters of GetAuxAI

Attribute	Explanation
Prototype	GetAuxAI (AINum, thread)
Description	Obtain extended AI values
Parameter	• AINum: AuxAI number, range [0~3]; • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	value: Input current or voltage value percentage, range [0~100] corresponds to current value [0~20mA] or voltage [0~10V]

WaitAuxDI: Waiting for extended DI input

Table 3-104 Detailed Parameters of WaitAuxDI

Attribute	Explanation
Prototype	WaitAuxDI(DINum, bOpen,time, timeout)
Description	Waiting for extended DI input
Parameter	• DINum: DI number; • bBOpen: Switch True on, False off; • time: maximum waiting time (ms); • timeout: waiting for timeout processing 0- stop error, 1- continue waiting, 2- keep waiting.
Return value	null

WaitAuxAI: Waiting for extended AI input

Table 3-105 Detailed Parameters of WaitAuxAI

Attribute	Explanation
Prototype	WaitAuxAI (AINum, sign, value, time, timeout)
Description	Waiting for extended AI input
Parameter	• AINum: AI number; • sign: 0- greater than; 1- Less than; • value: AI value; • time: maximum waiting time (ms); • timeout: waiting for timeout processing 0- stop error, 1- continue waiting, 2- keep waiting.
Return value	null

Code 3-31 Example of Extended IO Instruction Set

```
1.  --Set extended DO
2.  SetAuxDO(0,1,1,0)
3.  --Set up extended AO
4.  SetAuxAO(0,10,0)
5.  --Waiting for extended DI input
6.  WaitAuxDI(0,0,1000,0)
7.  --Waiting for extended AI input
8.  WaitAuxAI(0,0,50,1000,0)
9.  --Get extended DI value
10. Ret = GetAuxDI(0,0)
11. --Obtain extended AI values
12. value = GetAuxAI(0,0)
```

3.3.7 Coordinate System

The coordinate system instruction is divided into two parts: "Set tool coordinate system" and "Set workpiece coordinate system".

SetToolList: Set tool coordinate series table

Table 3-106 SetToolList Detailed Parameters

Attribute	Explanation
Prototype	SetToolList(name)
Description	Set tool coordinate series table
Parameter	• name: The name of the tool coordinate system, such as toolcoord0.
Return value	null

SetWObjList: Set the workpiece coordinate series table

Table 3-107 SetWObjList Detailed Parameters

Attribute	Explanation
Prototype	SetWObjList (name)
Description	Set tool coordinate series table
Parameter	• name: The coordinate system name of the workpiece, such as wobjcoord0.
Return value	null

Code 3-32 Coordinate System Example

```
1. SetWObjList (wobjcoord0) -- Set workpiece coordinates
2. SetToolList (toolcoord0) -- Set tool coordinates
```

3.3.8 mode Switching

mode can be used to switch the robot mode. This command can switch the robot to manual mode, usually added at the end of a program, so that the user can automatically switch the robot to manual mode and drag it after the program runs.

mode: Switch from robot mode to manual mode

Table 3-108 Detailed mode Parameters

Attribute	Explanation
Prototype	mode(state)
Description	Control the robot to switch to manual mode
Parameter	• State: 0- Robot mode, default 1- Manual mode.
Return value	null

Code 3-33 Coordinate System Example

```
1. Lin(DW01,100,-1,0,0)
2. Lin(DW02,100,-1,0,0)
3. Lin(DW03,100,-1,0,0)
4. mode(1)
```

3.3.9 Collision Level

By setting collision levels, the collision levels of each axis can be adjusted in real-time during program execution, making deployment of application scenarios more

flexible. In custom percentage mode, 1~100% corresponds to 0~100N.

SetAnticollision: Collision Level Setting

Table 3-109 SetAnticollision Detailed Parameters

Attribute	Explanation
Prototype	SetAnticollision (mode, level, config)
Description	Set collision level
Parameter	• mode: 0- standard level, 1- custom percentage; • Level={j1, j2, j3, j4, j5, j6}: collision threshold, a total of 11 levels, 1 is level 1, 2 is level 1, and 2100 is the collision off level; • Config: 0- Do not update configuration file, 1- Update configuration file, default is 0.
Return value	null

Code 3-34 Coordinate System Example

```
1. level={4,4,4,4,4,5}
2. SetAnticollision (0, level, 0) -- Set collision level, 0-Standard mode, level - Collision level of each joint, 0-Do not update configuration file
1. level1={40,40,40,40,40,50}
2. SetAnticollision (1, level 1,0) -- Set collision level, 1- Custom percentage mode, level - Collision threshold for each joint, 0- Do not update configuration file
```

3.3.10 Collision Detection

Custom collision-detection threshold function activated—configure thresholds for both joint-side and TCP-side collision detection.

CustomCollisionDetectionStart： Collision threshold function enabled

Table 3-110 CustomCollisionDetectionStart Detailed Parameters

Attribute	Explanation
Prototype	CustomCollisionDetectionStart(flag,jointDetectionThreshold,tcpDetectionThreshold, block)
Description	Custom collision detection threshold function activated
Parameter	• flag: 1 – joint detection only; 2 – TCP detection only; 3 – both joint and TCP detection • jointDetectionThreshold，j1-j6 • tcpDetectionThreshold: TCP collision detection threshold,x-y-z-a-b-c • block: 0 – non-blocking; 1 – blocking
Return	null

CustomCollisionDetectionEnd：Collision threshold function disabled

Table 3-111 CustomCollisionDetectionEnd Detailed Parameters

Attribute	Explanation
Prototype	CustomCollisionDetectionEnd ()
Description	Custom collision-detection threshold function disabled
Parameter	null
Return	null

Code 3-35 Collision Threshold Function Example

```
1. CustomCollisionDetectionStart(1,{100,100,100,100,100,100},{300,300,300,300,300,300},0)
2. CustomCollisionDetectionEnd()
```

3.3.11 Acceleration

The Acc command is used to enable the independent setting of robot acceleration. By adjusting the motion command and adjusting the speed, the acceleration and deceleration time can be increased or decreased, and the robot's action rhythm time can be adjusted.

SetOaccScale， Set robot acceleration

Table 3-112 SetOaccScale Detailed Parameters

Attribute	Explanation
Prototype	SetOaccScale(acc)
Description	Set robot acceleration
Parameter	• acc: Percentage of robot acceleration.
Return value	null

Code 3-36 Acceleration Example

```
1. SetOaccScale (20) --20- Set acceleration percentage
```

3.4 Peripheral instruction

3.4.1 Gripper

ActGripper， Gripper activation/reset

Table 3-113 Detailed Parameters of ActGripper

Attribute	Explanation
Prototype	ActGripper(index,action)
Description	Activate Gripper
Parameter	• Index: Gripper number; • Action: 0- Reset, 1- Activate.
Return value	null

MoveGripper Gripper Motion Control Parameters

Table 3-114 Detailed Parameters of MoveGripper

Attribute	Explanation
Prototype	MoveGripper (index, pos, vel, force, max_time, block)
Description	Set gripper motion control Parameters
Parameter	• Index: Gripper number, range [1~8]; • POS: Position percentage, range [0~100]; • vel: speed percentage, range [0~100]; • force: percentage of torque, range [0~100]; • max_time: maximum waiting time, range [0~30000], unit: ms; • block: whether it is blocked, 0-blocking, 1-non blocking.
Return value	null

Code 3-37 Gripper Example

```
1. ActGripper (1,0) - Gripper Reset, 1-Gripper Number, 0-Reset
2. WaitMs (1000) - Wait for 1000ms to ensure successful jaw reset
3. ActGripper (1,1) - Gripper Activation, 1-Gripper Number, 1-gripper Activation
4. WaitMs (10) - Wait for 1000ms to ensure successful jaw reset
5. MoveGripper(1,62,27,51,3000,1)
6. -Control gripper movement, 1-gripper number, 62 gripper position, 27 gripper opening and
   closing speed, 51 gripper opening and closing torque, 3000 grippermaximum waiting time,
   0-blockage
```

3.4.2 Spray gun

The spray gun command can control actions such as "start spraying", "stop spraying", "start cleaning", and "stop light spraying" of the spray gun.

SprayStart: Spraying begins

Table 3-115 Detailed Parameters of SprayStart

Attribute	Explanation
Prototype	SprayStart ()
Description	Spraying begins
Parameter	null
Return value	null

SprayStop: Stop spraying

Table 3-116 Detailed Parameters of SprayStop

Attribute	Explanation
Prototype	SprayStop ()
Description	Stop spraying
Parameter	null
Return value	null

PowerCleanStart: Start cleaning the gun

Table 3-117 Detailed Parameters of PowerCleanStart

Attribute	Explanation
Prototype	PowerCleanStart ()
Description	Start cleaning the gun
Parameter	null
Return value	null

PowerCleanStop: Gun cleaning stops

Table 3-118 Detailed Parameters of PowerCleanStop

Attribute	Explanation
Prototype	PowerCleanStop ()
Description	Stop clearing the gun
Parameter	null
Return value	null

Code 3-38 Spray Gun Example

```
1. Lin (SprayStart, 100, -1,0,0) -- Start spraying point and move to spraying starting point
2. SprayStart() - Start spraying
3. Lin (Sprayline, 100, -1,0,0) -- Spray path
4. Lin (template3100, -1,0,0) -- Stop spraying point
5. SprayStop () - Stop spraying
6. Lin (template 4100, -1,0,0) -- gun cleaning point, move to gun cleaning point, wait for gun
   cleaning processing
7. PowerCleanStart() -- Start cleaning the gun
8. WaitMs (5000) -- gun cleaning time 5000ms
9. PowerCleanStop() -- Stop gun cleaning
```

3.4.3 Expansion axis

The expansion axis is divided into two modes: Controller PLC (UDP) and Controller Servo Driver (485).

EXT_AXIS_PTP: UDP mode Extended Axis Motion

Table 3-119 Detailed Parameters of EXT_AXIS_PTP

Attribute	Explanation
Prototype	EXT_AXIS_PTP (mode, name, Vel)
Description	UDP mode Extended Axis Motion
Parameter	• mode: Motion mode, 0-asynchronous, 1-synchronous; • name: Point name; • vel: Debugging speed.
Return value	null

ExtAxisMoveJ: UDP mode Extended Axis Motion

Table 3-120 Detailed Parameters of ExtAxisMoveJ

Attribute	Explanation
Prototype	ExtAxisMoveJ (mode, E1，E2，E3，E4, Vel)
Description	UDP mode Extended Axis Motion
Parameter	• mode: Motion mode, 0-asynchronous, 1-synchronous;; • E1, E2, E3, E4: External axis positions • vel: Debugging speed.
Return value	null

ExtAxisSetHoming: UDP Extended Axis returns to Zero

Table 3-121 Detailed Parameters of ExtAxisSetHoming

Attribute	Explanation
Prototype	ExtAxisSetHoming(axisID， mode，searchVel ， latchVel)
Description	UDP extension axis returns to zero
Parameter	• Axisid: axis number [1-4]; • mode: return to zero mode: 0. return to zero at the current position, 1. return to zero at the negative limit, 2. return to zero at the positive limit; • SearchVel zero search speed (mm/s); • latchVel: Zero positioning speed (mm/s).
Return value	null

ExtAxisSeroOn: UDP Extended Axis Enable

Table 3-122 Detailed Parameters of ExtAxisSeroOn

Attribute	Explanation
Prototype	ExtAxisServoOn(axisID，status)
Description	UDP Extension Axis Enable
Parameter	• Axisid: axis number [1-4]; • status: 0- Enable; 1- Enable.
Return value	null

Code 3-39 UDP Extension Axis Example

```
1.  --UDP Extension Axis Example
2.  ExtDevSetUDPComParam ("192.168.58.88", 2021,10) -- Configure UDP communication Parameters
3.  ExtDevLoadUDPDriver() -- Load UDP driver to enable communication
4.  WaitMs (500) -- Wait for 500 milliseconds to ensure that the UDP driver has been loaded correctly
5.  ExtAxisSeroOn (1,0) -- disable, disable axis 1
6.  ExtAxisSeroOn (1,1) -- Enable, enable axis 1
7.  ExtAxisSetHoming (1,0,40,45) -- Zeroing, 1-Extended axis number, 0-Current position zeroing, 40 Zeroing speed, 45 Zeroing clamp speed
8.  WaitMs (1000) - Wait for 1000 milliseconds
9.  EXT_EAXIS_PTP (0, DW01100) -- Motion command, 0-Asynchronous motion, DW01-Point name, 100- Debugging speed
10. WaitMs (1000) -- Wait for 1000 milliseconds
```

Controller servo driver (485) mode is used to configure the Parameters of the extended axis.

AuxServosetStatusid: Set the 485 extension axis data axis number in the status feedback

Table 3-123 AuxServosetStatusID Detailed Parameters

Attribute	Explanation
Prototype	AuxServosetStatusID(servoid)
Description	Set the 485 extension axis data axis number in the status feedback
Parameter	• servoid: servo drive ID, range [1-15], corresponding to slave ID.
Return value	null

AuxServoEnable， Is the 485 extension axis enabled

Table 3-124 Detailed Parameters of AuxServoEnable

Attribute	Explanation
Prototype	AuxServoEnable(servoid, status)
Description	Enable/disable 485 extension axis
Parameter	• servoid: servo drive ID, range [1-15], corresponding to slave ID; • status: Enable status, 0-disable, 1-enable.
Return value	null

AuxServoSetControlmode， Set the mode of 485 extended axis control

Table 3-125 Detailed Parameters of AuxServoSetControlmode

Attribute	Explanation
Prototype	AuxServoSetControlmode(servoid, mode)
Description	Set 485 extension axis control mode
Parameter	• servoid: servo drive ID, range [1-15], corresponding to slave ID; • mode: Control mode, 0-Position mode, 1-Speed mode.
Return value	null

AuxServoHoming: Set 485 extension axis return to zero mode

Table 3-126 Detailed Parameters of AuxServoHoming

Attribute	Explanation
Prototype	AuxServoHoming(servoid, mode, searchVel, latchVel)
Description	Set 485 extension axis to zero
Parameter	• servoid: servo drive ID, range [1-15], corresponding to slave ID; • mode: return to zero mode, 1- return to zero at the current position; 2-Negative limit returns to zero; 3-Positive limit return to zero; • searchVel: Zero return speed, mm/s or ° /s; • latchVel: clamp speed, mm/s or ° /s;
Return value	null

AuxServoSetTarget speed: Set 485 extension axis target speed in speed mode

Table 3-127 Detailed Parameters of AuxServoSetTarget Speed

Attribute	Explanation
Prototype	AuxServoSetTargetSpeed(servoid, speed)
Description	Set 485 Extended Axis Target Speed (Speed mode)
Parameter	• servoid: servo drive ID, range [1-15], corresponding to slave ID; • speed: Target speed, mm/s or ° /s.
Return value	null

Code 3-40 Controller+Servo Drive Axis Example

```
1.  --Controller+servo drive (position mode)
2.  AuxServoSetStatusID (1) -- Set the 485 extension axis data axis number in the status feedback
3.  AuxServoEnable (1,0) -- Set 485 extension axis enable, 1-servo drive ID, 0-disable
4.  WaitMs (500) - Wait for 500 milliseconds
5.  AuxServoEnable (1,1) -- Set 485 extension axis enable, 1-servo driver ID, 1-enable
6.  WaitMs (500) -- Wait for 500 milliseconds
7.  AuxServoHoming (1,1,10,10) -- Set 485 extension axis zeroing mode, 1-servo driver ID, 1-current position zeroing, 10 zeroing speed, 10 clamp speed
8.  WaitMs (500) -- Wait for 500 milliseconds
9.  AuxServoSetTarget Pos (1300,30) -- Set 485 Extended Axis Target Position (Position mode), 1- Servo Driver ID, 300- Target Position, 30- Target Speed
10. WaitMs (500) -- Wait for 500 milliseconds
11.
```

Code 3-40（continued）

```
12. -Controller+servo drive (speed mode)
13. AuxServoSetStatusID (1) -- Set the 485 extension axis data axis number in the status feedback
14. AuxServoEnable (1,0) -- Set 485 extension axis enable, 1-servo drive ID, 0-disable
15. WaitMs (500) - Wait for 500 milliseconds
16. AuxServoSetControlmode (1,1) -- Set 485 Extended Axis Control mode, 1-Servo Driver ID, 1-Speed mode
17. WaitMs (500) -- Wait for 500 milliseconds
18. AuxServoEnable (1,1) -- Set 485 extension axis enable, 1-servo driver ID, 1-enable
19. WaitMs (500) -- Wait for 500 milliseconds
20. AuxServoHoming (1,1,10,10) -- Set 485 extension axis zeroing mode, 1-servo driver ID, 1-current position zeroing, 10 zeroing speed, 10 clamp speed
21. WaitMs (500) -- Wait for 500 milliseconds
22. AuxServoSetTarget Speed (1, 30) -- Set 485 Extended Axis Target Position (Speed mode), 1- Servo Driver ID, 30- Target Speed
23. WaitMs (500) -- Wait for 500 milliseconds
```

3.4.4 Conveyor Belt

ConveyorIODetect: IO real-time detection

Table 3-128 Detailed Parameters of ConveyorIODetect

Attribute	Explanation
Prototype	ConveyorIODetect(max_t)
Description	Real time IO detection of conveyor belt workpieces
Parameter	• max_t: maximum detection time, in milliseconds.
Return value	null

ConveyorGet RackData: Real time location detection

Table 3-129 Detailed Parameters of ConveyorGet RackData

Attribute	Explanation
Prototype	ConveyorGetTrackData(mode)
Description	Real time location detection to obtain the current status of the location
Parameter	• mode: 1- Tracking and grasping 2- Tracking motion 3- TPD tracking.
Return value	null

ConveyorTrackStart: Enable belt tracking

Table 3-130 Detailed Parameters of ConveyorTrackStart

Attribute	Explanation
Prototype	ConveyorTrackStart(status)
Description	Drive belt tracking begins
Parameter	• status: Status, 1- Start, 0- Stop.
Return value	null

ConveyorTrackEnd: Stop belt tracking

Table 3-131 Detailed Parameters of ConveyorTrackEnd

Attribute	Explanation
Prototype	ConveyorTrackEnd()
Description	Drive belt tracking stops
Parameter	null
Return value	null

Code 3-41 Spray Gun Example

```
1.  PTP (conversterstart, 30, -1,0) -- robot grasping starting point
2.  While (1) do - loop capture
3.      ConveyorIODetect (10000) -- IO real-time object detection
4.      ConveyorGet RackData (1) -- Object Position Acquisition
5.      ConveyorTrackStart (1) -- Conveyor belt tracking begins
6.      Lin (cvrCatchPoint, 10, -1,0,0) -- The robot reaches the grasping point
7.      MoveGripper (1255255,010000) -- Gripping objects with grippers
8.      Lin (cvrRaisePoint, 10, -1,0,0) -- Robot lifting
9.      ConveyorTrackEnd() -- Conveyor belt tracking ends
10.     PTP (conveyor, 30, -1,0) -- robot arrives at waiting point
11.     PTP (converents, 30, -1,0) -- robot reaches placement point
12.     MoveGripper (1,0255010000) -- Gripper release
13.     PTP (conversterstart, 50, -1,0) -- The robot returns to the starting point of grasping
    again and waits for the next grasping
14. end -- End
```

3.4.5 Grinding equipment

PolishingUnloadComDriver: Unload the polishing head communication driver

Table 3-132 Detailed Parameters of PolishingUnloadComDriver

Attribute	Explanation
Prototype	PolishingUnloadComDriver ()
Description	Unloading of communication driver for polishing head
Parameter	null
Return value	null

PolishingLoadComDriver: Load the polishing head communication driver

Table 3-133 Detailed Parameters of PolishingLoadComDriver

Attribute	Explanation
Prototype	PolishingLoadComDriver ()
Description	Polishing head communication driver loading
Parameter	null
Return value	null

PolishingDeviceEnable: Device Enable Settings

Table 3-134 Detailed Parameters of PolishingDeviceEnable

Attribute	Explanation
Prototype	PolishingDeviceEnable (status)
Description	Grinding head equipment enable
Parameter	• status: 0- Enable below, 1- Enable above.
Return value	null

PolishingClearError: Error Clearing

Table 3-135 Detailed Parameters of PolishingClearError

Attribute	Explanation
Prototype	PolishingClearError ()
Description	Clear the error message of the polishing head equipment
Parameter	null
Return value	null

Polishing TorqueSensorReset: Clear the polishing head force sensor to zero

Table 3-136 PolishingTorqueSensorReset Detailed Parameters

Attribute	Explanation
Prototype	PolishingTorqueSensorReset ()
Description	The polishing head force sensor is reset to zero.
Parameter	null
Return value	null

Polishing SetTarget Velocity: Grinding Head Speed Setting

Table 3-137 Detailed Parameters of PolishingSetTarget Velocity

Attribute	Explanation
Prototype	PolishingSetTargetVelocity (rot)
Description	Grinding head speed setting
Parameter	• rot: rotational speed, unit [r/min].
Return value	null

PolishingSetTarget Torque: Setting Force

Table 3-138 Detailed Parameters of PolishingSetTarget Torque

Attribute	Explanation
Prototype	PolishingSetTargetTorque (setN)
Description	Setting the polishing head with setting power
Parameter	• SetN: Set force, unit [N].
Return value	null

Polishing SetTarget Position: Set the extension distance of the polishing head

Table 3-139 Detailed Parameters of PolishingSetTarget Position

Attribute	Explanation
Prototype	PolishingSetTargetPosition (distance)
Description	Set the extension distance of the polishing head
Parameter	• distance: Extended distance, measured in millimeters.
Return value	null

Polishing Set Operation mode: Set the polishing head control mode

Table 3-140 Detailed Parameters of PolishingSetOperamode

Attribute	Explanation
Prototype	PolishingSetTargetPosition (mode)
Description	Grinding head mode setting
Parameter	• mode: 1- return to zero mode, 2- Position mode, 3- Torque mode.
Return value	null

Polishing SetTarget Touchforce: Contact Force Settings

Table 3-141 Detailed Parameters of PolishingSetTarget TouchForce

Attribute	Explanation
Prototype	PolishingSetTargetTouchForce (conN)
Description	Contact force setting
Parameter	• conN: Contact force, unit [N].
Return value	null

PolishingSetTarget Touchtime: Set the force transition time setting

Table 3-142 Detailed Parameters of PolishingSetTarget TouchTime

Attribute	Explanation
Prototype	PolishingSetTargetTouchForceTime (settime)
Description	Set the transition time for force setting
Parameter	• settime: Time, unit [ms].
Return value	null

PolishingSetWorkPieceWeight: workpiece weight setting

Table 3-143 Detailed Parameters of PolishingSetWorkPieceWeight

Attribute	Explanation
Prototype	PolishingSetWorkPieceWeight (weight)
Description	Workpiece weight setting
Parameter	• weight: Weight, unit [N].
Return value	null

Code 3-42 Grinding Equipment Example

```
1.  PolishingLoadComDriver -- Load the polishing head communication driver
2.  PolishingDeviceEnable (1) -- Enable on device
3.  Polishing ClearError (1) -- Clear polishing head device error messages
4.  PolishingTorqueSensorReset() -- Force sensor reset to zero
5.  Polishing SetTarget Velocity (500) -- Set the polishing head speed
6.  Polishing Set Target Torque (10) -- Set the polishing head setting force
7.  Polishing SetTarget Position (100) -- Set the extension distance of the polishing head
8.  Polishing SetOperation mode (3) -- Set the polishing head control mode
9.  Polishing SetTarget TouchForce (5) -- Set the contact force of the polishing head
10. Polishing SetTarget TouchTime (500) -- Set the transition time for the contact force of the
    polishing head
11. PolishingSetWorkPieceWeight (20) -- Set workpiece weight
12. PolishingUnloadComDriver -- Unload Grinding Head Communication Driver
```

3.5 Welding instruction

3.5.1 Welding

WeldingDicturrent: Set welding current

Table 3-144: Detailed Parameters of WeldingAKS Current

Attribute	Explanation
Prototype	WeldingSetCurrent(ioType, current,blend,AOIndex)
Description	Set welding current
Parameter	• ioType: Type 0- Controller IO; 1. Digital communication protocol; • Current: welding current value (A); • blend: smooth, 0-not smooth, 1-smooth; • AOIndex: Analog output port (0-1) of welding current control box. When the mode is digital communication protocol, blend is 0 and AOIndex is 0.
Return value	null

WeldingSetvoltage: Set the welding voltage

Table 3-145 Detailed Parameters of WeldingSetVoltage

Attribute	Explanation
Prototype	WeldingSetVoltage(ioType, voltage, blend ,AOIndex)
Description	Set welding voltage
Parameter	• ioType: Type 0- Controller IO; 1-Expand IO; • voltage: Welding voltage value (V); • blend: smooth, 0-not smooth, 1-smooth; • AOIndex: Welding current control AO port (0-1). When the mode is digital communication protocol, blend is 0. When AOIndex is 0 protocol, blend is 0 and AOIndex is 0.
Return value	null

ARCStart: Start Arc

Table 3-146 ARCStart Detailed Parameters

Attribute	Explanation
Prototype	ARCStart (ioType, arcNum, timeout)
Description	Arc Initiation
Parameter	• ioType: Type 0- Controller IO; 1-Expand IO; • arcNum: Welding machine configuration file number; • timeout:maximum waiting time.
Return value	null

ARCEnd: End Arc

Table 3-147 ARCEnd Detailed Parameters

Attribute	Explanation
Prototype	ARCEnd(ioType, arcNum, timeout)
Description	End Arc
Parameter	• ioType: 0- Controller IO; 1-Expand IO; • arcNum: Welding machine configuration file number; • timeout:maximum waiting time.
Return value	null

SetAspirated: Air supply

Table 3-148 SetAspirated Detailed Parameters

Attribute	Explanation
Prototype	SetAspirated(ioType, airControl)
Description	Air supply
Parameter	• ioType: 0- Controller IO; 1-Expand IO; • airControl: Air supply control 0- Stop air supply; 1. Air supply.
Return value	null

SetEversewireFeed: Reverse wire feeding

Table 3-149 SetVerseWireFeed Detailed Parameters

Attribute	Explanation
Prototype	SetReverseWireFeed(ioType, wireFeed)
Description	Reverse wire feeding
Parameter	• ioType: 0- Controller IO; 1-Expand IO; • wireFeed: Wire feeding control 0- Stop wire feeding; 1. Wire feeding.
Return value	null

SetForwardwireFeed: Forward Wire Feed

Table 3-150 SetForwardWireFeed Detailed Parameters

Attribute	Explanation
Prototype	SetForwardWireFeed(ioType, wireFeed)
Description	Forward wire feeding

Table 3-150(Continued)

Attribute	Explanation
Parameter	• ioType: 0- Controller IO; 1-Expand IO • wireFeed: Wire feeding control 0- Stop wire feeding; 1. Wire feeding
Return value	null

WeldingSetVoltageGradualChangeStart: Set the welding voltage gradient to start.

Table 3-151 WeldingSetVoltageGradualChangeStart Detailed Parameters

Attribute	Explanation
Prototype	WeldingSetVoltageGradualChangeStart (ioType, voltageStart, voltageEnd, aoIndex, blend)
Description	Set the welding voltage gradient to start.
Parameter	• ioType: Type: 0 - Controller IO; 1 - Expansion IO; • voltageStart: Starting voltage, unit: V; • voltageEnd: Ending voltage, unit: V; • aoIndex: Control box AO port number (0-1); • blend: 0 - Not smooth, 1 - Smooth
Return value	null

WeldingSetVoltageGradualChangeEnd: Set the end of the welding voltage gradient

Table 3-152 WeldingSetVoltageGradualChangeEnd Detailed Parameters

Attribute	Explanation
Prototype	WeldingSetVoltageGradualChangeEnd ( )
Description	Set the end of the welding voltage gradient
Parameter	null
Return value	null

WeldingSetCurrentGradualChangeStart: Set the start of the welding current gradient

Table 3-153 WeldingSetCurrentGradualChangeStart Detailed Parameters

Attribute	Explanation
Prototype	WeldingSetCurrentGradualChangeStart (ioType, currentStart, currentEnd, aoIndex, blend)
Description	Set the start of the welding current gradient.
Parameter	• ioType: Type: 0 - Controller IO; 1 - Expansion IO; • currentStart: Starting current, unit: A; • currentEnd: Ending current, unit: A; • aoIndex: Control box AO port number (0-1); • blend: 0 - Not smooth, 1 - Smooth
Return value	null

WeldingSetCurrentGradualChangeEnd: Set the end of the welding current gradient.

Table 3-154 WeldingSetCurrentGradualChangeEnd Detailed Parameters

Attribute	Explanation
Prototype	WeldingSetCurrentGradualChangeEnd ( )
Description	Set the end of the welding current gradient.
Parameter	null
Return value	null

WeaveChangeStart: Start of weaving gradient.

Table 3-155 WeaveChangeStart Detailed Parameters

Attribute	Explanation
Prototype	WeaveChangeStart (weaveChangeFlag, weaveChangeNum, velStart, velEnd)
Description	Start of weaving gradient
Parameter	• weaveChangeFlag: 1 - Change weaving parameter, 2 - Change weaving parameter + welding speed; • weaveChangeNum: Weaving gradient number, range 0-7; • velStart: Weaving start speed, unit: cpm; • velEnd: Weaving end speed, unit: cpm;
Return value	null

WeaveChangeEnd: End of weaving gradient.

Table 3-156 WeaveChangeEnd Detailed Parameters

Attribute	Explanation
Prototype	WeaveChangeEnd( )
Description	End of weaving gradient.
Parameter	null
Return value	null

WeldingSetCurrertRelation: Set the relationship between welding current and output analog quantity.

Table 3-157 WeldingSetCurrertRelation Detailed Parameters

Attribute	Explanation
Prototype	WeldingSetCurrertRelation (currentMin, currentMax, outputVoltageMin, outputVoltageMax, AOIndex)
Description	Set the relationship between welding current and output analog quantity.
Parameter	• currentMin: Welding current - analog quantity linear relationship: current value on the left side (A); • currentMax: Welding current - analog quantity linear relationship: current value on the right side (A); • outputVoltageMin: Welding current - analog quantity linear relationship: current value at the right point (A); • outputVoltageMax: Right-point current value (A) of the linear relationship between welding current and analog quantity.; • AOIndex: Welding current analog output port;
Return value	null

WeldingSetVoltageRelation: Set the relationship between welding voltage and output analog quantity.

Table 3-158 WeldingSetVoltageRelation Detailed Parameters

Attribute	Explanation
Prototype	WeldingSetVoltageRelation (currentMin, currentMax, outputVoltageMin, outputVoltageMax, AOIndex)
Description	Set the relationship between welding voltage and output analog quantity.
Parameter	• weldVoltageMin: Welding voltage - analog output linear relationship: welding voltage value at the left point (V); • weldVoltageMax: Welding voltage - analog output linear relationship: welding voltage value at the right point (V); • outputVoltageMin: Welding voltage - analog output linear relationship: analog output voltage value at the left point (V); • outputVoltageMax: Welding voltage - analog output linear relationship: analog output voltage value at the right point (V); • AOIndex: Welding voltage analog output port;
Return value	null

WeldingGetCurrertRelation: Get the relationship between welding current and output analog quantity

Table 3-159 WeldingGetCurrertRelation Detailed Parameters

Attribute	Explanation
Prototype	WeldingGetCurrertRelation ()
Description	Get the relationship between welding current and output analog quantity.
Parameter	null
Return value	• currentMin: Welding current - analog quantity linear relationship: current value at the left point (A); • currentMax: Welding current - analog quantity linear relationship: current value at the right point (A); • outputVoltageMin: Welding current - analog quantity linear relationship: current value at the right point (A).; • AOIndex: Welding voltage analog output port;

WeldingGetVoltageRelation: Retrieve the relationship between welding voltage and output analog quantity

Table 3-160 WeldingGetVoltageRelation Detailed Parameters

Attribute	Explanation
Prototype	WeldingGetVoltageRelation ()
Description	Retrieve the relationship between welding voltage and output analog quantity
Parameter	null
Return value	• weldVoltageMin: The welding voltage value at the left point of the linear relationship between welding voltage and analog output voltage (A); • weldVoltageMax: The welding voltage value at the right point of the linear relationship between welding voltage and analog output voltage (A); • outputVoltageMin: The analog output voltage value at the left point of the linear relationship between welding voltage and analog output voltage (V); • outputVoltageMax: The analog output voltage value at the right point of the linear relationship between welding voltage and analog output voltage (V); • AOIndex: Analog output port for welding voltage;

WeldingSetProcessParam: Set welding process parameter

Table 3-161 WeldingSetProcessParam Detailed Parameters

Attribute	Explanation
Prototype	WeldingSetProcessParam (id, startCurrent, startVolage, startTime, weldCurrent, weldVoltage, endCurrent, endVoltage, endTime)
Description	Set welding process parameter
Parameter	• id: Welding process number; • startCurrent: Starting current (A); • startVoltage: Starting voltage (V); • startTime: Starting time (ms); • weldCurrent: Welding current (A); • weldVoltage: Welding voltage (V); • endCurrent: Ending current (A); • endVoltage: Ending voltage (V); • endTime: Ending time (ms);
Return value	null

WeldingGetProcessParam: Retrieve welding process parameter

Table 3-162 WeldingGetProcessParam Detailed Parameters

Attribute	Explanation
Prototype	WeldingSetProcessParam ( )
Description	Retrieve welding process parameter
Parameter	null
Return value	• id: Welding process number; • startCurrent: Starting current (A); • startVoltage: Starting voltage (V); • startTime: Starting time (ms); • weldCurrent: Welding current (A); • weldVoltage: Welding voltage (V); • endCurrent: Ending current (A); • endVoltage: Ending voltage (V); • endTime: Ending time (ms);

Code 3-43 Welding Example

```
1.  --Controller IO soldering
2.  WellIOType=0 -- Set mode controller IO
3.  -Set current and voltage
4.  WeldingDictCurrent (weldIOType, 2,1,0) -- Current setting, 2-Welding voltage 2A, 1-Welding current control AO port 1,0- Non smooth
5.  WeldSetVoltage (weldIOType, 2,1,0) -- Voltage setting, 2-Welding voltage 2A, 1-Welding current control AO port 1,0- Non smooth
6.  -- Move to the starting point of welding
7.  PTP(mulitilinesafe,10,-1,0)
8.  PTP(mulitilineorigin1,10,-1,0)
9.  --Start an arc
10. ARCStart (weldIOType, 01000) -- arc start, weldIOType - controller IO mode, 0-welding process number 01000-maximum waiting time 1000ms
11. Lin(DW01,100,-1,0,0);
12. ARCEnd (weldIOType, 01000) -- arc extinguishing, weldIOType - controller IO mode, 0-welding process number 01000-maximum waiting time 1000ms
13. --Air supply
14. SetAspirated (wellIOType, 1) -- Air supply, wellIOType - Controller IO mode, 1-On
15. Lin(DW01,100,-1,0,0);
16. SetAspirated (wellIOType, 0) -- Stop gas, wellIOType - Controller IO mode, 0-Stop
17. WaitMs (1000) - Wait for 1000 milliseconds
18. --Forward wire feeding
19. SetForwardWireFeed (wellIOType, 1) -- Forward wire feeding, wellIOType - Controller IO mode, 1- Enable
20. Lin(DW01,100,-1,0,0);
21.
```

Code 3-43(Continued)

```
22. SetForwardWireFeed (wellIOType, 0) -- Forward wire feeding, wellIOType - Controller IO mode, 0-Stop
23. WaitMs (1000) - Wait for 1000 milliseconds
24. --Reverse wire feeding
25. SetEverseWireFeed (wellIOType, 1) -- Reverse wire feeding, wellIOType - Controller IO mode, 1- Enable
26. Lin(DW01,100,-1,0,0);
27. SetEverseWireFeed (wellIOType, 0) --Reverse wire feeding, wellIOType - Controller IO mode, 0-Stop
28. WaitMs (1000) -- Wait for 1000 milliseconds
```

Code 3-44 Welding gradient example

```
1.  PTP(p1,100,-1,0) --安全点 Safe point
2.  WeldingSetVoltage(0,20,1,0) -- Set voltage
3.  WeldingSetCurrent(0,210,0,0) -- Set current
4.  WeldingSetVoltageGradualChangeStart(0,20,25,1,0)--Set the start of welding voltage gradient
5.  WeldingSetCurrentGradualChangeStart(0,210,265,0,0)--Set the start of welding current gradient
6.  ArcWeldTraceControl(1,0,1,0.06,5,5,300,1,0.06,5,5,300,0,0,4,1,10,0,0) --Arc tracking control
7.  PTP(p2,30,-1,0) -- Welding start point
8.  ARCStart(0,0,10000) -- Arc initiation command
9.  WeaveStart(0) -- Weaving start
10. WeaveChangeStart(2,1,12,15) -- Weaving gradient start
11. Lin(p3,1,-1,0,0) -- Move to welding end point
12. WeaveChangeEnd() -- Weaving gradient end
13. WeaveEnd(0) -- Weaving end
14. ARCEnd(0,0,10000) -- Arc termination
15. ArcWeldTraceControl(0,0,1,0.06,5,5,300,1,0.06,5,5,300,0,0,4,1,10,0,0)
16. WeldingSetCurrentGradualChangeEnd() -- Set the end of welding current gradient
17. WeldingSetVoltageGradualChangeEnd() -- Set the end of welding voltage gradient
18. PTP(p1,30,-1,0) -- Safe point
19.
```

3.5.2 Arc Tracking

ArcWeldTraceControl: Arc Tracking Control

Table 3-163 Detailed Parameters of ArcWeldTraceControl

Attribute	Explanation
Prototype	ArcWeldTraceControl(flag, delaytime, isLeftRight, klr, tStartLr, stepmaxLr, summaxLr, isUpLow, kud, tStartUd, stepmaxUd, summaxUd, axisSelect, referenceType, referSampleStartUd, referSampleCountUd, referenceCurrent)
Description	Arc tracking control
Parameter	• flag: switch, 0-off; 1- Open; • delaytime: Lag time, in milliseconds; • isLeftRight: Left and right deviation compensation 0-off, 1-on; • klr: left and right adjustment coefficient (sensitivity); • tStartLr: Start compensating for time cyc on both sides; • stepmaxLr: maximum compensation amount in millimeters for each left and right operation; • summaxLr: maximum compensation amount on both sides in millimeters; • IsUpLow: Up and down deviation compensation 0-off, 1-on; • kud: Up and down adjustment coefficient (sensitivity); • tStartUd: Start compensating time cyc from top to bottom; • stepmaxUd:maximum compensation amount in mm for each up and down step; • summaxUd: themaximum compensation amount for the upper and lower totals; •axisSlect: selection of upper and lower coordinate systems, 0-swing; 1. Tools; 2-Base; • referenceType: Upper and lower reference current setting method, 0-feedback; 1- Constant; • referSampleStartUd: Start counting of upper and lower reference current sampling (feedback), cyc; • referSampleCountUd: Up and down reference current sampling cycle count (feedback), cyc; • referenceCurrent: Upper and lower reference currents in mA.
Return value	null

ArcWeldTraceReplayStart: Arc tracking with multi-layer and multi-channel compensation enabled

Table 3-164 Detailed Parameters of ArcWeldTraceReplayStart

Attribute	Explanation
Prototype	ArcWeldTraceReplayStart ( )
Description	Arc tracking with multi-layer and multi-channel compensation activated
Parameter	null
Return value	null

ArcWeldTraceReplayEnd:

Table 3-165 Detailed Parameters of ArcWeldTraceReplayEnd

Attribute	Explanation
Prototype	ArcWeldTraceReplayEnd ( )
Description	Arc tracking with multi-layer and multi-channel compensation shutdown
Parameter	null
Return value	null

MultiplayerOffsetTrsfToBase: Offset coordinate variation - multi-layer and multi pass welding

Table 3-166 Detailed Parameters of MultiplayerOffsetTrsfToBase

Attribute	Explanation
Prototype	MultilayerOffsetTrsfToBase (pointO.x, pointO.y, pointO.z, pointX.x, pointX.y, pointX.z, pointZ.x, pointZ.y, pointZ.z, dx, dy, dry)
Description	Offset coordinate change - multi-layer and multi pass welding
Parameter	• pointO. x, pointO. y, pointO. z: Cartesian pose of reference point O; • pointX. x, pointX. y, pointX. z: Cartesian pose of the reference point offset in the X direction; • pointZ. x, pointZ. y, pointZ. z: Cartesian pose of the reference point Z offset direction; • dx: x-direction offset, unit [mm]; • dy: x-direction offset, unit [mm]; • dry: offset around the y-axis, unit [°].
Return value	offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz: offset amount

Code 3-45 Example of Arc Tracking

```
1.  --Move to the starting point of welding
2.  PTP(mulitilinesafe,10,-1,0)
3.  PTP(mulitilineorigin1,10,-1,0)
4.
5.  --Welding (first position)
6.  ARCStart(1,0,3000)
7.  WeaveStart(0)
8.  ArcWeldTraceControl(1,0,1,0.06,5,5,50,1,0.06,5,5,55,0,0,4,1,10)
9.  Lin(mulitilineorigin2,1,-1,0,0)
10. ArcWeldTraceControl(0,0,1,0.06,5,5,50,1,0.06,5,5,55,0,0,4,1,10)
```

Code 3-7 44(continued）

```
11. WeaveEnd(0)
12. ARCEnd(1,0,3000)
13. PTP(mulitilinesafe,10,-1,0)
14. Pause (0) -- No function
15. --Welding (second position)
16. Offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz=MultiplayerOffsetTrsfToBase (mulitilineorigin1, mulitilineX1, mulitilineZ1, 10,0,0) -- offset coordinate change - multi-layer and multi pass welding
17. PTP(mulitilineorigin1,10,-1,1, offset_x,offset_y,offset_z,offset_rx,offset_ry,offset_rz)
18. ARCStart(1,0,3000)
19.
20. Offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz=MultiplayerOffsetTrsfToBase (mulitilineorigin2, mulitilineX2, mulitilineZ2, 10,0,0) - offset coordinate change - multi-layer and multi pass welding
21. ArcWeldTraceReplayStart() -- Arc tracking with multi-layer and multi-channel compensation enabled
22. Lin(mulitilineorigin2,2,-1,0,1, offset_x,offset_y,offset_z,offset_rx,offset_ry,offset_rz)
23. ArcWeldTraceReplayEnd() -- Arc tracking with multi-layer and multi-channel compensation closed
24. ARCEnd(1,0,3000)
25. PTP(mulitilinesafe,10,-1,0)
26. Pause (0) -- No function
27.
28. --Welding (third position)
29. Offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz=MultiplayerOffsetTrsfToBase (mulitilineorigin1, mulitilineX1, mulitilineZ1,0,10,0) - offset coordinate change - multi-layer and multi pass welding
30. PTP(mulitilineorigin1,10,-1,1, offset_x,offset_y,offset_z,offset_rx,offset_ry,offset_rz)
31. ARCStart(1,0,3000)
32. Offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz=MultiplayerOffsetTrsfToBase (mulitilineorigin2, mulitilineX2, mulitilineZ2,0,10,0) - offset coordinate change - multi-layer and multi pass welding
33. ArcWeldTraceReplayStart() -- Arc tracking with multi-layer and multi-channel compensation enabled
34. Lin(mulitilineorigin2,2,-1,0,1, offset_x,offset_y,offset_z,offset_rx,offset_ry,offset_rz)
35. ArcWeldTraceReplayEnd() -- Arc tracking with multi-layer and multi-channel compensation closed
36. ARCEnd(1,0,3000)
37. PTP(mulitilinesafe,10,-1,0)
```

3.5.3 Laser Tracking

Laser tracking requires sensor loading, sensor activation, laser tracking, data recording, sensor point movement, and positioning commands to be completed together.

LoadPosSensorDriver: Sensor loading

Table 3-167: Detailed Parameters of LoadPosSensorDriver

Attribute	Explanation
Prototype	LoadPosSensorDriver (choiceid)
Description	Sensor function selection loading
Parameter	• choiceid: Function Number, 101- Ruiniu RRT-SV2-BP, 102- Chuangxiang CXZK-RBTA4L, 103- Full Vision FV-160G4-WD-PP-RL, 104- Tongzhou Laser Sensor, 105- Aotai Laser Sensor.
Return value	null

UnloadPosSensorDriver: Sensor Unloading

Table 3-168 Detailed Parameters of UnloadPosSensorDriver

Attribute	Explanation
Prototype	UnloadPosSensorDriver (choiceid)
Description	Sensor function selection uninstallation
Parameter	• choiceid: Function Number, 101- Ruiniu RRT-SV2-BP, 102- Chuangxiang CXZK-RBTA4L, 103- Full Vision FV-160G4-WD-PP-RL, 104- Tongzhou Laser Sensor, 105- Aotai Laser Sensor.
Return value	null

LTLaserOn: Turn on the sensor

Table 3-169 Detailed Parameters of LTLaserOn

Attribute	Explanation
Prototype	LTLaserOn (Taskid)
Description	Open the sensor
Parameter	• Taskid: Select the weld type (Ruiniu RRT-SV2-BP, Chuangxiang CXZK-RBTA4L), choose the task number (Full View FV-160G4-WD-PP-RL, Aotai Laser Sensor), and select the solution (Tongzhou Laser Sensor).
Return value	null

LTLaserOff: Turn off the sensor

Table 3-170 Detailed Parameters of LTLaserOff

Attribute	Explanation
Prototype	LTLaserOff ()
Description	Turn off the sensor
Parameter	null
Return value	null

LTTrackOn: Start Tracking

Table 3-171 Detailed Parameters of LTTrackOn

Attribute	Explanation
Prototype	LTTrackOn (toolid)
Description	Start tracking
Parameter	• toolid: Coordinate system name.
Return value	null

LTTrackOff: Turn off tracking

Table 3-172 LTTrackOff Detailed Parameters

Attribute	Explanation
Prototype	LTTrackOff ( )
Description	Close Tracking
Parameter	null
Return value	null

LaserSensorRecord: Data Recording

Table 3-173: Detailed Parameters of LaserSensorRecord

Attribute	Explanation
Prototype	LaserSensorRecord (features，time, speed)
Description	data record
Parameter	• features: Function selection, 0-stop recording, 1-real-time tracking, 2-start recording, 3-trajectory reproduction (when selecting trajectory reproduction, laser tracking reproduction can be selected);

Table 3-173(Continued)

Attribute	Explanation
	• time: waiting time; • speed: Running speed.
Return value	null

MoveLTR: Laser Tracking Reproduction

Table 3-174 Detailed Parameters of MoveLTR

Attribute	Explanation
Prototype	MoveLTR ( )
Description	Laser tracking reproduction (this command can only be used after selecting the trajectory reproduction for data recording)
Parameter	null
Return value	null

LTSearchStart: Start location search

Table 3-175 LTSearchStart Detailed Parameters

Attribute	Explanation
Prototype	LTSearchStart (refdirection, refdpion, ovl, length, max_time, toolid)
Description	Start searching for location
Parameter	• refdirection: direction, 0-+x, 1-x, 2-+y, 3-y, 4+z, 5-z, 6-specified direction (custom reference point direction); • refdpiont: Direction point. When the direction is 6, the direction point needs to be specified, while others default to {0, 0, 0, 0, 0, 0,}; • ovl: Speed percentage, unit [%]; • rength: length, unit [mm]; • max_time: maximum positioning time, unit [ms]; • roolid: Coordinate system name.
Return value	null

LTSearchStop: Stop locating

Table 3-176 LTSearchStop Detailed Parameters

Attribute	Explanation
Prototype	LTSearchStop ( )
Description	Stop locating
Parameter	null
Return value	null

Code 3-46 Laser Tracking Example

```
1.  LoadPosSensorDriver (101) -- Load Sensor Driver
2.  LTLaserOn (1) -- Turn on the sensor
3.  LTTrackOn (1) -- Start Tracking
4.  LaserSensorRecord (2, 5, 30) -- Record data
5.  LTSearchStart (0, 0, 50, 100, 5000, 1) -- Find Position
6.  MoveLTR() -- Laser Tracking Reproduction
7.  LTSearchStop() -- Stop finding
8.  LTTrackOff () -- Turn off tracking
9.  LTLaserOff() -- Turn off sensor
10. --Uninstall sensor driver
11. UnloadPosSensorDriver(101)
```

3.5.4 Laser Recording

The laser recording instruction realizes the function of extracting the starting and ending points of laser tracking recording, allowing the robot to automatically move to the starting position. It is suitable for situations where the robot starts moving from the outside of the workpiece and performs laser tracking recording. At the same time, the upper computer can obtain information about the starting and ending points in the recorded data for subsequent movements.

MoveToLaserRecordStart: Move to the starting point of the weld seam

Table 3-177 Detailed Parameters of MoveToLaserRecordStart

Attribute	Explanation
Prototype	MoveToLaserRecordStart ( )
Description	Move to the starting point of the weld seam
Parameter	null
Return value	null

MoveToLaserRecordEnd: Move to the end point of the weld seam

Table 3-178 Detailed Parameters of MoveToLaserRecordEnd

Attribute	Explanation
Prototype	MoveToLaserRecordEnd ( )
Description	Move to the starting point of the weld seam
Parameter	null
Return value	null

Code 3-47 Laser Recording Example

```
1.  Lin (recordStartPt, 100, -1,0,0) - Move to the starting position of the weld seam
2.  LaserSensorRecord (2,10,30) - Record the starting point of the weld seam
3.  Lin (recordEndPt, 100, -1,0,0) - Move to the end position of the weld seam
4.  LaserSensorRecord (0,10,30) - Record the end point of the weld seam
5.  MoveToLaserRecordStart (1,30) - Move to the welding start point
6.  ARCStart (0,01000) - Start Arc
7.  LaserSensorRecord (3,10,30) - Weld seam trajectory reproduction
8.  MoveLTR() - Linear movement of weld seam
9.  ARCEnd (0,01000) - Arc off
10. MoveToLaserRecordEnd (1,30) - Move to the welding end point
```

3.5.5 Wire positioning

The welding wire positioning instruction is generally applied in welding scenarios, requiring a combination of welding machine and robot IO and motion instructions.

WireSearchStart: Wire positioning begins

Table 3-179 WireSearchStart Detailed Parameters

Attribute	Explanation
Prototype	WireSearchStart (refPos, searchVel, searchDis, autoBackFlag, autoBackVel, autoBackDis, offectFlag)
Description	Wire positioning begins
Parameter	• pedlocation: whether the reference position has been updated, 0-no update, 1-update; • searchVel: Search speed%; • searchDis: Positioning distance mm; • autoBackflag: Automatic return flag, 0- Not automatic- Automatic; • autoBackvel: Automatic return speed%; • autoBackDis: automatically returns distance in mm; • offectflag: 1- Positioning with offset; 2. Find the teaching point location.
Return value	null

WireSearchEnd: End of wire positioning

Table 3-180 WireSearchEnd Detailed Parameters

Attribute	Explanation
Prototype	WireSearchEnd (refPos, searchVel, searchDis, autoBackFlag, autoBackVel, autoBackDis, offectFlag)
Description	Wire positioning completed
Parameter	• pedlocation: whether the reference position has been updated, 0-no update, 1-update; • searchVel: Search speed%; • searchDis: Positioning distance mm; • autoBackflag: Automatic return flag, 0- Not automatic- Automatic; • autoBackvel: Automatic return speed%; • autoBackDis: automatically returns distance in mm; • offectflag: 1- Positioning with offset; 2. Find the teaching point location.
Return value	null

GetWireSearchoffset: Calculate the offset of wire positioning

Table 3-181: Detailed Parameters of GetWireSearchOffset

Attribute	Explanation
Prototype	GetWireSearchOffset (seamType, method, varNameRef, varNameRes)
Description	Calculate the offset of welding wire positioning
Parameter	• seamType: Weld seam type; • method: Calculation method; • varNameRef: Benchmarks 1-6, "#" represents a non-point variable; • varNameRes: Contact points 1-6, where "#" represents a non-point variable.
Return value	null

WireSearchWait: Waiting for the completion of wire positioning

Table 3-182 Set PointToDatabase Detailed Parameters

Attribute	Explanation
Prototype	WireSearchWait(varname)
Description	Waiting for the completion of wire positioning
Parameter	• varname: Contact point names "RES0"~"RES99".
Return value	null

Set PointToDatabase: Writing wire positioning contact points into the database

Table 3-183 Set PointToDatabase Detailed Parameters

Attribute	Explanation
Prototype	SetPointToDatabase (varName, pos)
Description	Write the contact point of welding wire positioning into the database
Parameter	• varname: Contact point names "RES0"~"RES99"; • pos: Contact point data x, y, x, a, b, c.
Return value	null

Code 3-48 Welding Wire Positioning Example

```
1.  WireSearchStart (1,10300,10,0) -- Wire positioning begins
2.  Lin (2dx1,10,0,0,0,0) -- Starting point of positioning reference point
3.  Lin (2dx2,10,0,1,0) -- Positioning reference point direction point
4.  WireSearchWait ("REF0") -- Wait for wire positioning to be completed
5.  Lin (2dy1,10,0,0,0) -- Starting point of positioning reference point
6.  Lin (2dy2,10,0,1,0) -- Positioning reference point direction point
7.  WireSearchWait ("REF1") -- Wait for wire positioning to be completed
8.  WireSearchEnd (1,10300,10,0) -- End of wire positioning
9.  WireSearchStart(0,10,300,1,10,10,0)
10. Lin (2dx1,10,0,0,0,0) -- Positioning starting point
11. Lin (2dx2,10,0,1,0) -- Positioning direction point
12. WireSearchWait("RES0")
13. Lin (2dy1,10,0,0,0,0) -- Positioning starting point
14. Lin (2dy2,10,0,1,0) -- Positioning direction point
15. WireSearchWait("RES1")
16. WireSearchEnd(0,10,300,1,10,10,0)
17. F1, x1, y1, z1, a1, b1, c1=GetWireSearchOffset (0,1, "REF0", "REF1", "#", "#", "#",
    "RES0", "RES1", "#", "#", "#", "#") -- Calculate the positioning offset
18. RegisterVar("number","f1")
19. RegisterVar("number","x1")
20. RegisterVar("number","y1")
21. RegisterVar("number","z1")
22. RegisterVar("number","a1")
23. RegisterVar("number","b1")
24. RegisterVar("number","c1")
25. PointsOffsetEnable (f1, x1, y1, z1, a1, b1, c1) -- Motion offset
26. Lin(test1,10,0,0,0)
27. Lin(test2,10,0,0,0)
28. PointsOffsetDisable()
```

3.5.6 Attitude Adjustment

PostureAdjustOn: Enable posture adjustment

Table 3-184 PostureAdjusteOn Detailed Parameters

Attribute	Explanation
Prototype	PostureAdjustOn (plate_type, direction_type={PosA, PosB, PosC}, time, paDisatance_1, inflection_type, paDisatance_2, paDisatance_3 , paDisatance_4, paDisatance_5)
Description	Enable posture adjustment
Parameter	• plate_date: Plate type, 0-corrugated board, 1-corrugated board, 2-fence board, 4-corrugated shell steel • direction-type: direction of motion, from left to right (direction-type is PosA, PosB, PosC), from right to left (direction-type is PosA, PosC, PosB) • time: Attitude adjustment time, unit [ms]; • paDissentance_1: length of the first segment, unit [mm]; • inflection type: inflection point type, 0- from top to bottom, 1- from bottom to top; • paDissentance_2: Second segment length, unit [mm]; • paDisatance3: Third segment length, unit [mm]; • paDisatance4: Fourth segment length, unit [mm]; • paDissentance_5: Fifth segment length, unit [mm].
Return value	null

PostureAdjustOff: Turn off posture adjustment

Table 3-185 PostureAdjusteOff Detailed Parameters

Attribute	Explanation
Prototype	PostureAdjustOff ( )
Description	Close posture adjustment
Parameter	null
return value	null

Code 3-49 Attitude Adjustment Example

```
1.  --Enable posture adjustment
2.  PostureAdjustOn(0, PosA,PosB,PosC,1000,100,0,100,100,100,100)
3.
4.  PTP(DW01,100,10,0)
```

Code 3-48(Continued)

```
5.  --Close posture adjustment
6.  PostureAdjustOff()
```

3.6 Force Control Command

3.6.1 Force Control Set

FT_Guard: Collision Detection

Table 3-186 FT_Guard Detailed Parameters

Attribute	Explanation
Prototype	FT_Guard (flag, tool_id, select_Fx, select_Fy, select_Fz, select_Tx, select_TY, select_Tz, value_Fx，value_Fy，value_Fz，value_Tx，value_TY，value_Tz ，max_threshold_Fx,max_threshold_Fy,max_threshold_Fz,max_threshold_Tx, max_threshold_Ty,max_threshold_Tz, min_threshold_Fx, min_threshold_Fy, min_threshold_Fz, min_threshold_Tx, min_threshold_Ty, min_threshold_Tz)
Description	collision detection
Parameter	• flag: Torque activation flag, 0-disable collision protection, 1-enable collision protection; • Tool_i: Coordinate system name; • Select_fx~select_Tz: Select whether to detect collisions in six degrees of freedom, 0-no detection, 1-detection, select_Tx is set to not select; • value_Sx~value_Tz: The current values of the six degrees of freedom, with value_Tx set to 0; • max_threshord_FX~max_threshord_Tz: maximum threshold for six degrees of freedom, with max_threshord_Tx set to 0; • Min_threshold_fx~min_threshold_Tz: The minimum threshold for six degrees of freedom, with min_threshold_Tx set to 0.
Return value	null

FT_Control: Constant Force Control

Table 3-187 FT_Control Detailed Parameters

Attribute	Explanation
Prototype	FT_Control (flag, sensor_num, select, force_torque, gain, adj_sign, ILC_sign,max_dis,max_ang)
Description	Hengli Control

Table 3-172 (Continued)

Attribute	Explanation
Parameter	• flag: Constant force control on flag, 0-off, 1-on; • Sensor_num: force sensor number; • Select: Check if the six degrees of freedom detect fx, fy, fz, mx, my, mz, 0-inactive, 1-active; • Force_torque: detects force/torque, in N or Nm; • gain: f_p, f_i,f_d,m_p,m_i,m_d, Force PID Parameters, torque PID Parameters; • Add_sign: adaptive start stop state, 0-off, 1-on; • ILC_sign: ILC controls start stop status, 0-stop, 1-training, 2-practical operation; • MAX-DIS: maximum adjustment distance; • max_ang: maximum adjustment angle.
Return value	null

FT_Spiralsearch: Spiral Insertion

Table 3-188 FT_SpiralSearch Detailed Parameters

Attribute	Explanation
Prototype	FT_SpiralSearch(rcs, dr,ft ,max_t_ms,max_vel)
Description	Spiral insertion
Parameter	• rcs: Reference Coordinate System, 0-Tool Coordinate System, 1-Base Coordinate System • dr: feed rate per circle radius, unit mm default 0.7; • ft: Force/torque threshold, fx,fy,fz,tx,ty,tz， Range [0~100]; • max_t_ms:maximum exploration time, in milliseconds; • max_vel: maximum linear velocity, measured in millimeters per second.
Return value	null

FT_ComplianceStart: Smooth control enabled

Table 3-189 FT_ComplianeStart Detailed Parameters

Attribute	Explanation
Prototype	FT_ComplianceStart(p, force)
Description	Smooth control enabled
Parameter	• p: Position adjustment coefficient or compliance coefficient; • force: Soft opening force threshold, in units of N.
Return value	null

FT_ComplianeStop: Smooth Control Off

Table 3-190 FT_ComplianeStop Detailed Parameters

Attribute	Explanation
Prototype	FT_ComplianceStop ()
Description	Smooth control closed
Parameter	null
Return value	null

FT_SotInsertion: Rotating Insertion

Table 3-191 Detailed Parameters of FT_SotInsertion

Attribute	Explanation
Prototype	FT_RotInsertion(rcs, angVelRot, ft,max_angle, orn,max_angAcc, rotorn)
Description	Rotating insertion
Parameter	• rcs: Reference coordinate system, 0-tool coordinate system, 1-base coordinate system; • angVelRot rotational angular velocity, unit deg/s; • ft: Force or torque threshold (0~100), measured in N or Nm; • max angle of rotation, unit: deg; • orn: direction of force/torque, 1- along the z-axis direction, 2- around the z-axis direction; • max_angAcc: maximum rotational acceleration, unit deg/s ^ 2, not currently in use, default to 0; • rotorn: Rotation direction, 1- clockwise, 2- counterclockwise.
Return value	null

FT_LinInsertion: Linear Insertion

Table 3-192 Detailed Parameters of FT_LinInsertion

Attribute	Explanation
Prototype	FT_LinInsertion(rcs, ft, lin_v , lin_a , dismax, linorn)
Description	Straight line insertion
Parameter	• rcs: Reference coordinate system, 0-tool coordinate system, 1-base coordinate system; • ft: Force or torque threshold (0~100), measured in N or Nm; • lin-v: Linear velocity, unit mm/s, default 1; • lin_a: Linear acceleration, unit mm/s ^ 2, not using default 0 for now; • dismax:maximum insertion distance, in millimeters;

Table 3-191 (Continued)

Attribute	Explanation
	• linorn: Insertion direction: 0-negative direction, 1-positive direction.
Return value	null

FT-FindSurface: Surface Positioning

Table 3-193 Detailed Parameters of FT_SindSurface

Attribute	Explanation
Prototype	FT_FindSurface (rcs, dir, axis, lin_v, lin_a , dismax, ft)
Description	Surface positioning
Parameter	• rcs: Reference coordinate system, 0-tool coordinate system, 1-base coordinate system; • dir: direction of movement, 1-positive direction, 2-negative direction; • axis: moving axis, 1-x, 2-y, 3-z; • lin-v: Explore linear velocity, unit mm/s defaults to 3; • lin_a: Explore linear acceleration, unit mm/s ^ 2 defaults to 0; • dismax: Large exploration distance, in millimeters; • ft: Action termination force threshold, in units of N.
Return value	null

FT_CalCenterStart: Start calculating the position of the middle plane

Table 3-194 FT_CalCenterStart Detailed Parameters

Attribute	Explanation
Prototype	FT_CalCenterStart ()
Description	Starting from calculating the position of the middle plane
Parameter	null
Return value	null

FT_CalCenterEnd: End of calculating the position of the middle plane

Table 3-195 FT_CalCenterEnd Detailed Parameters

Attribute	Explanation
Prototype	FT_CalCenterEnd ()
Description	End of calculating the position of the middle plane
Parameter	null
Return value	null

FT_Click: Tap Force Detection

Table 3-196 FT_Click Detailed Parameters

Attribute	Explanation
Prototype	FT_Click (ft， lin_v，lin_a，dismax)
Description	Tap force detection
Parameter	• ft: Force or torque threshold (0~100), measured in N or Nm; • lin-v: Linear velocity, unit mm/s, default 1; • lin_a: Linear acceleration, unit mm/s ^ 2, not using default 0 for now; • dismax: maximum insertion distance, in millimeters.
Return value	null

Code 3-50 Examples of Force Control Set in Various modes

```
1.  --Examples of FT_Guard instruction in various modes
2.  FT_Guard (1,1,1,1,0,0,0,5,0,0,0,0,10,0,0,0,0,0,0,5,0,0,0,0,0,0,0) -- Force/moment collision detection enabled
3.  PTP (template1100, -1,0) -- Motion Instructions
4.  FT_Guard (0,1,1,1,0,0,0,5,0,0,0,0,10,0,0,0,0,0,0,5,0,0,0,0,0,0,0) -- Force/moment collision detection turned off
5.  --FT_Control, Constant Force Control
6.  FT_Control (1,11,0,1,0,0,10,5,0,0,0,0.001,0,0,0,0,0,0,0,0,0,0,0,0,5) -- Force/torque motion control enabled
7.  Lin (template3100, -1,0,0) -- Motion command
8.  FT_Control (0,11,1,0,1,0,0,10,5,0,0,0,0.001,0,0,0,0,0,0,0,0,0,0,0,10,5) -- Force/torque motion control turned off
9.  --FT_Spiral, spiral insertion
10. FT_Control (1,10,0,0,1,1,0,0,0,0,5,0,0,0.0005,0,0,0,0,0,0,0,10,0) -- Force/torque motion control enabled
11. FT_SpiralSearch (0,0.7,060000,5) -- # Spiral Insertion
12. FT_Control (0,10,0,0,1,1,0,0,0,0,5,0,0,0.0005,0,0,0,0,0,0,0,10,0) -- Force/torque motion control turned off
13. --FT-Rot, Rotate Insert
14. FT_Control (1,10,0,0,1,1,0,0,0,0,5,0,0,0.0005,0,0,0,0,0,0,0,10,0) - Force/torque motion control enabled
15. FT_SotInsertion (0,3,0,5,1,0,1) -- Rotating Insertion
16. FT_Control (0,10,0,0,1,1,0,0,0,0,5,0,0,0.0005,0,0,0,0,0,0,0,10,0) -- Force/torque motion control turned off
17. --FT_Lin, Linear Insertion
18. FT_Control (1,10,0,0,1,1,0,0,0,0,5,0,0,0.0005,0,0,0,0,0,0,0,10,0) -- Force/torque motion control enabled
19. FT_LinInsertion (0,50,1,0100,1) -- Linear Insertion
```

Code 3-49 (Continued)

```
20. FT_Control (0,10,0,0,1,1,0,0,0,0,5,0,0,0.0005,0,0,0,0,0,0,0,10,0) -- Force/torque motion control turned off
21. –FT_FindSurface, surface positioning
22. PTP (1,30, -1,0) - Initial Position
23. FT FindSurface (0,1,3,0100,5- Surface localization)
24. --FT_CalCenter, center positioning
25. PTP (1,30, -1,0) -- Initial Position
26. FT_CalCenterStart() -- # Surface localization Start
27. FT_Control (1,10,0,0,1,1,0,0,0,0,0, -10,0,0,0,0.00001,0,0,0,0,0,0,0,0100,0) -- Force/torque motion control enabled
28. FT_SindSurface (1,2,10,0200,5) -- Positioning Plane A
    FT_Control (0,10,0,0,1,1,0,0,0,0,0, -10,0,0,0,0.00001,0,0,0,0,0,0,0,0100,0) -- Force/torque motion control turned off
29. PTP (1,30, -1,0) - Initial Position
30. FT_Control (1,10,0,0,1,1,0,0,0,0,0, -10,0,0,0,0.00001,0,0,0,0,0,0,0,0100,0) - Force/torque motion control enabled
31. FT_FindSurface (1,1,2,20,0200,5) -- Positioning Plane B
32. FT_Control (0,10,0,0,1,1,0,0,0,0,10,0,0,0,0.00001,0,0,0,0,0,0,0,0100,0) - Force/torque motion control turned off
33. Pos={} -- Define array pos
34. Pos=FT_CalCenterEnd() -- Obtain the Cartesian pose of the positioning center
35. MoveCart (pos, vDctualTCPNum(), vDctualWObjNum(), 30,10100, -1,0) - moves to the center position of the positioning
```

3.6.2 Torque Recording

Torque recording command, realizing real-time torque recording and collision detection function.

TorqueRecordStart: Torque recording begins

Table 3-197 Detailed Parameters of TorqueRecordStart

Attribute	Explanation
Prototype	TorqueRecordStart (flag, negativevalues, positivevalues，collisionTime)
Description	Torque recording start/stop
Parameter	• flag: Smooth selection, 0-Not smooth, 1-Smooth; • negativevalues: Negative thresholds for each joint {j1, j2, j3, j4, j5, j6}; • positivevalues: Positive thresholds for each joint {j1, j2, j3, j4, j5, j6}; • collisiontime: The duration of collision detection for each joint {j1, j2, j3, j4,

Table 3-196 (Continued)

Attribute	Explanation
	j5, j6}.
Return value	null

TorqueRecordEnd: Torque recording stops

Table 3-198 TorqueRecordEnd Detailed Parameters

Attribute	Explanation
Prototype	TorqueRecordEnd ( )
Description	Torque recording stopped
Parameter	null
Return value	null

TorqueRecordReset: Torque Record Reset

Table 3-199 TorqueRecordReset Detailed Parameters

Attribute	Explanation
Prototype	TorqueRecordReset ( )
Description	Reset torque record
Parameter	null
Return value	null

Code 3-51 Example of torque recording

```
1. negativevalues = {-0.1, -0.1, -0.1, -0.1, -0.1, -0.1}
2. positivevalues = {0.1, 0.1, 0.1, 0.1, 0.1, 0.1}
3. collisionTime = {500, 500, 500, 500, 500, 500}
4. TorqueRecordStart(1, negativevalues,positivevalues,collisionTime)
5. TorqueRecordEnd()
6. WaitMs (1000) - Wait for 1000 milliseconds
7. TorqueRecordReset()
```

3.7 Communication instruction

3.7.1 Modbus

The Modbus instruction functionality supports bus operations based on the Modbus-TCP protocol. Users can employ the relevant commands to let the robot communicate with a Modbus-TCP (or Modbus-RTU) client or server—i.e., act as master or slave—and perform read/write operations on coils, discrete inputs, and registers.

3.7.1.1 Modbus-TCP

Related operation instructions for the main station:

Modbus MasterWriteDO: Write digital output (write coil)

Table 3-200: Detailed Parameters of Modbus MasterWriteDO

Attribute	Explanation
Prototype	ModbusMasterWriteDO (Modbus_name, Register_name, Register_num, {Register_value})
Description	Modbus TCP Write Digital Output
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: DO Name; • Register_num: Number of registers; • { Register_value }: Register value, the number of register values matches the number of registers {value_1, value_2,...}.
Return value	null

Modbus MasterReadDO: Read digital output (read coil)

Table 3-201: Detailed Parameters of Modbus MasterReadDO

Attribute	Explanation
Prototype	ModbusMasterReadDO (Modbus_name, Register_name, Register_num)
Description	Modbus TCP Write Digital Output
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: DO Name; • Register_num: Number of registers.
Return value	Reg_value1, Reg_value2,,...: int values, return the corresponding quantity of values based on the value of Regite_num

Modbus MasterReadDI: Read Digital Input (Read Discrete Input)

Table 3-202: Detailed Parameters of Modbus MasterReadDI

Attribute	Explanation
Prototype	ModbusMasterReadDI (Modbus_name, Register_name, Register_num)
Description	Modbus TCP Read Digital Input
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: DI Name; • Register_num: Number of registers;
Return value	Reg_value1, Reg_value2,...: int values, return the corresponding quantity of values based on the value of Regite_num

Code 3-52 Numerical Input/Output Example

```
1. ModbusMasterWriteDO(Modbus_0,Register_1,1,{1})
2. --Write digital output, Modbus 0- master station name, Register 1-DO name, 1- number of registers, {2}- Register value
3. DO_value = ModbusMasterReadDO(Modbus_0,Register_1,1)
4. --Read digital output, Modbus 0- master station name, Register 1-DO name, 1- register quantity
5. DI_value = ModbusMasterReadDI(Modbus_0,Register_2,1)
6. --Read digital output, Modbus 0- master station name, Register 1- DI name, 1- number of registers
```

Modbus MasterWriteAO: Write analog output (hold register)

Table 3-203: Detailed Parameters of Modbus MasterWriteAO

Attribute	Explanation
Prototype	ModbusMasterWriteAO (Modbus_name, Register_name, Register_num, {Register_value})
Description	Modbus TCP Write Analog Output
Parameter	• Modbus_name: The name of the main station for Modbus • Register_name: AO name; • Register_num: Number of registers; • { Register_value }: Register value, the number of register values matches the number of registers {value_1, value_2,...}.
Return value	null

Modbus MasterReadAO: Read analog output (read hold register)

Table 3-204 ModbusMasterReadAO Detailed Parameters

Attribute	Explanation
Prototype	ModbusMasterReadAO (Modbus_name, Register_name, Register_num)
Description	Modbus TCP read analog output
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: AO name; • Register_num: Number of registers.
Return value	Reg_value: Register value

Modbus MasterReadAI: Read Analog Input (Read Input Register)

Table 3-205: Detailed Parameters of Modbus MasterReadAI

Attribute	Explanation
Prototype	ModbusMasterReadAI (Modbus_name, Register_name, Register_num)
Description	Modbus TCP Read Input Register
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: AO name; • Register: Number of registers.
Return value	Reg_value1, Reg_value2,...: int values, return the corresponding quantity of values based on the value of Regite_num

Code 3-53 Analog Input/Output Example

```
1.  --Analog output settings
2.  ModbusMasterWriteAO(Modbus_0,Register_3,1,{2})
3.  --Write analog output, Modbus 0- master station name, Register 3-AO name, 1- number of
    registers, {2}- Register value
4.  --Analog output settings
5.  ModbusMasterWriteAO(Modbus_0,Register_3,1,{2})
6.  --Write analog output, Modbus 0- master station name, Register 3-AO name, 1- number of
    registers, {2}- Register value
7.  AO_value = ModbusMasterReadAO(Modbus_0,Register_3,1)
8.  --Read analog output, Modbus 0- master station name, Register 3-AO name, 1- register
    quantity
9.  AI_value = ModbusMasterReadAO(Modbus_0,Register_2,1)
10. --Read analog input, Modbus 0- master station name, Register 2-AI name, 1- register
    quantity
```

ModbusMasterWaitDI， Waiting for analog input settings (waiting for input register values)

Table 3-206: Detailed Parameters of Modbus MasterWaitDI

Attribute	Explanation
Prototype	ModbusMasterWaitDI (Modbus_name, Register _name, Waiting_state, Waiting_time)
Description	Modbus TCP waiting for analog input settings
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: DI Name; • Waiting_state: waiting state, 1-Ture, 2-Flase; • Waiting_time: timeout unit [ms].
Return value	null

Modbus MasterWaitAI: Waiting for Digital Input Settings (Waiting for Discrete Input values)

Table 3-207: Detailed Parameters of Modbus MasterWaitAI

Attribute	Explanation
Prototype	ModbusMasterWaitAI (Modbus_name, Register_name, Waiting_state, Register_value， Waiting_time)
Description	Modbus TCP waits for digital input settings
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: AI name; • Waiting_state: waiting state, 1-<, 2->; • Register_value: Register value; • Waiting_time: timeout [ms].
Return value	null

Code 3-54 Waiting for Digital/Analog Input/Output Example

```
1. ModbusMasterWaitDI(Modbus_0,Register_0,1,1000)
2. --Modbus 0- Master Station Name, Register 0-DI Name, 1-True, 1000- Timeout Time ms
3. ModbusMasterWaitAI(Modbus_0, Register_2,0,13,1000)
4. --Modbus 0- Master Station Name, Register 2-DA Name, 0->, 13- Register value, 1000- Timeout Time ms
```

Related instructions from the station

Modbus Slave WriteDO: Slave Digital Output Settings (Write Discrete Input)

Table 3-208: Detailed Parameters of Modbus SlaveDWriteDO

Attribute	Explanation
Prototype	ModbusSlaveWriteDO (Register_name, Register_num, {Register_value})
Description	Modbus TCP Slave Station Write Digital Output Settings
Parameter	• Register_name: DO Name; • Register_num: Number of registers; • {Register_value}: Register value, the number of register values matches the number of registers {value_1, value_2,...}.
Return value	null

Modbus Slave ReadDO: Read Digital Output (Read Discrete Input)

Table 3-209: Detailed Parameters of Modbus SlaveRadDO

Attribute	Explanation
Prototype	ModbusSlaveReadDO (Register_name, Register_num)
Description	Modbus TCP reads and writes digital outputs
Parameter	• Register_name: DO Name; • Register_num: Number of registers;
Return value	{Register_value}: Register value, the number of register values matches the number of registers {value_1, value_2,...}

Modbus Slave ReadDI: Read digital input (read coil)

Table 3-210: Detailed Parameters of Modbus SlaveReadDI

Attribute	Explanation
Prototype	ModbusMasterReadDI (Register_name, Register_num)
Description	Modbus TCP slave station reads digital input
Parameter	• Register_name: DI Name; • Register: Number of registers.
Return value	Reg_falue1, Reg_falue2,...: returns the corresponding quantity of values based on the value of Regite_num

Code 3-55 Slave Station Digital Input/Output Settings

```
1.  -Slave station digital output settings
2.  ModbusSlaveWriteDO(DO0,1,{2})
```

Code 3-54 (Continued)

```
3.  -Write digital output, DO0-DO number, 1-register quantity, {2}- Register value
4.  DO_value = ModbusSlaveReadDO(DO0,1)
5.  -Read digital output, DO0-DO number, 1-register quantity
6.
7.  -Digital input settings
8.  ModbusSlaveReadDI(DI1,3)
9.  -Read numerical input, DI1-DI name, 3-register quantity
```

Modbus SlaveWetDI: Waiting for digital input settings (waiting for coil values)

Table 3-211: Detailed Parameters of Modbus SlaveWetDI

Attribute	Explanation
Prototype	ModbusSlaveWaitDI (Register_name, Waiting_state, Waiting_time)
Description	Modbus TCP waits for digital input settings
Parameter	• Register_name: DI Name; • Waiting_state: waiting state, 1-Ture, 2-Flase; • Waiting_time: The unit of waiting time [ms].
Return value	null

ModbusSlaveWaitAI， Waiting for analog input settings (waiting to hold register values)

Table 3-212: Detailed Parameters of Modbus SlaveWaitAI

Attribute	Explanation
Prototype	ModbusSlaveWaitAI (Register_name, Waiting_state,Register_value ， Waiting_time)
Description	Modbus TCP slave station waiting for analog input settings
Parameter	• Register_name: AI name; • Waiting_state: waiting state, 1-<, 2->; • Register value: Register value; • Waiting_time: timeout [ms].
Return value	null

Code 3-56 slave station waiting for digital/analog input settings

```
1.  ModbusSlaveWaitDI(DI2,0,100)
2.  -Waiting for numerical input settings: DI2-DI name, 0-false, 100- waiting time ms
3.  ModbusSlaveWaitAI(AI1,0,12,133))
4.  -AI1-AI name, 0->, 12 register value, 133 timeout time ms
```

Modbus RegRead: Read Register Instruction

Table 3-213: Detailed Parameters of Modbus RegRead

Attribute	Explanation
Prototype	ModbusRegRead（fun_code， reg_add，reg_num，add, isthread）
Description	Read register instruction
Parameter	• fun_code: Function code, 1-0x01 coil, 2-0x02 discrete quantity, 3-0x03 hold register, 4-0x04 input register; • reg_add: Register address; • reg_num: Number of registers; • add: Address; • isthread: Whether to apply threads, 0- No, 1- Yes.
Return value	null

Modbus RegdData: Read register data

Table 3-214 Detailed Parameters of Modbus RegdData

Attribute	Explanation
Prototype	ModbusRegGetData（reg_num,isthread）
Description	Read register data
Parameter	• reg_num: Number of registers; • isthread: Whether to apply threads, 0- No, 1- Yes.
Return value	Reg_value: array variable

Modbus RegWrite: Write Register

Table 3-215 Detailed Parameters of Modbus RegWrite

Attribute	Explanation
Prototype	ModbusRegWrite（fun_code， reg_add，reg_num，reg_value，add, isthread）
Description	Write register
Parameter	• fun_code: function code, 5-0x05- single coil, 6-0x06- single register, 15-0x0f - multiple coils, 16-0x10- multiple registers; • reg_add: single coil, single register, multiple coils, multiple register addresses; • reg_num: Number of registers; • reg_ralue: byte array;

Table 3-214 (Continued)

Attribute	Explanation
	• add: Address; • isthread: Whether to apply threads, 0- No, 1- Yes.
Return value	null

Code 3-57 Modbus RTU Instruction Example

```
1.  addr = 0x1000
2.  va1 = {400, 600, 900, 700}
3.  ret = {}
4.  ModbusRegWrite(10, addr, 4,va1, 1, 0)
5.  --1-0x10- Multiple registers, addr - Register address, 4- Number of registers, va1- Byte
    array, 1- Address, 0- No threads applied
6.  WaitMs(10)
7.  ModbusRegRead(4, addr, 4, 1, 0)
8.  --1-0x04- Input register, addr - Register address, 4- Number of registers, 1- Address, 0- Do
    not apply thread
9.  WaitMs(10)
10. ret = ModbusRegGetData(4, 0)
11. --Read register data, 4- number of registers, 0- do not apply threads
12. WaitMs(10)
```

3.7.1.1 Modbus-RTU

Related operation instructions for the main station:

Modbus MasterWriteDO_RTU: Write digital output (write coil)

Table 3-216 Detailed Parameters of Modbus MasterWriteDO

Attribute	Explanation
Prototype	ModbusMasterWriteDO_RTU(Modbus_name, Register_name, Register_num, {Register_value})
Description	Modbus RTU Write Digital Output
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: DO Name; • Register_num: Number of registers; • { Register_value }: Register value, the number of register values matches the number of registers {value_1, value_2,...}.
Return value	null

ModbusMasterReadDO_RTU: Read digital output (read coil)

Table 3-217 Detailed Parameters of Modbus MasterReadDO_RTU

Attribute	Explanation
Prototype	ModbusMasterReadDO_RTU (Modbus_name,Register_name, Register_num)
Description	Modbus RTU Write Digital Output
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: DO Name; • Register_num: Number of registers.
Return value	Reg_value1, Reg_value2,,...: int values, return the corresponding quantity of values based on the value of Regite_num

Modbus MasterReadDI_RTU: Read Digital Input (Read Discrete Input)

Table 3-218: Detailed Parameters of Modbus MasterReadDI

Attribute	Explanation
Prototype	ModbusMasterReadDI_RTU(Modbus_name, Register_name, Register_num)
Description	Modbus RTU Read Digital Input
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: DI Name; • Register_num: Number of registers;
Return value	Reg_value1, Reg_value2,...: int values, return the corresponding quantity of values based on the value of Regite_num

Code 3-58 Numerical Input/Output Example

```
7.  ModbusMasterWriteDO_RTU (Modbus_0,Register_1,1,{1})
8.  --Write digital output, Modbus 0- master station name, Register 1-DO name, 1- number of registers, {2}- Register value
9.  DO_value = ModbusMasterReadDO_RTU (Modbus_0,Register_1,1)
10. --Read digital output, Modbus 0- master station name, Register 1-DO name, 1- register quantity
11. DI_value = ModbusMasterReadDI_RTU (Modbus_0,Register_2,1)
12. --Read digital output, Modbus 0- master station name, Register 1- DI name, 1- number of registers
```

Modbus MasterWriteAO_RTU: Write analog output (hold register)

Table 3-219: Detailed Parameters of Modbus MasterWriteAO_RTU

Attribute	Explanation
Prototype	ModbusMasterWriteAO_RTU(Modbus_name, Register_name, Register_num, {Register_value})
Description	Modbus RTU Write Analog Output
Parameter	• Modbus_name: The name of the main station for Modbus • Register_name: AO name; • Register_num: Number of registers; • { Register_value }: Register value, the number of register values matches the number of registers {value_1, value_2,...}.
Return value	null

ModbusMasterReadAO_RTU: Read analog output (read hold register)

Table 3-220 ModbusMasterReadAO Detailed Parameters

Attribute	Explanation
Prototype	ModbusMasterReadAO_RTU(Modbus_name, Register_name, Register_num)
Description	Modbus RTU read analog output
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: AO name; • Register_num: Number of registers.
Return value	Reg_value: Register value

Modbus MasterReadAI_RTU: Read Analog Input (Read Input Register)

Table 3-221: Detailed Parameters of Modbus MasterReadAI

Attribute	Explanation
Prototype	ModbusMasterReadAI_RTU(Modbus_name, Register_name, Register_num)
Description	Modbus RTU Read Input Register
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: AO name; • Register: Number of registers.
Return value	Reg_value1, Reg_value2,...: int values, return the corresponding quantity of values based on the value of Regite_num

Code 3-59 Analog Input/Output Example

```
11. --Analog output settings
12. ModbusMasterWriteAO_RTU (Modbus_0,Register_3,1,{2})
13. --Write analog output, Modbus 0- master station name, Register 3-AO name, 1- number of registers, {2}- Register value
14. --Analog output settings
15. ModbusMasterWriteAO_RTU (Modbus_0,Register_3,1,{2})
16. --Write analog output, Modbus 0- master station name, Register 3-AO name, 1- number of registers, {2}- Register value
17. AO_value = ModbusMasterReadAO_RTU (Modbus_0,Register_3,1)
18. --Read analog output, Modbus 0- master station name, Register 3-AO name, 1- register quantity
19. AI_value = ModbusMasterReadAO_RTU (Modbus_0,Register_2,1)
20. --Read analog input, Modbus 0- master station name, Register 2-AI name, 1- register quantity
```

ModbusMasterWaitDI_RTU， Waiting for analog input settings (waiting for input register values)

Table 3-222: Detailed Parameters of Modbus MasterWaitDI

Attribute	Explanation
Prototype	ModbusMasterWaitDI_RTU (Modbus_name, Register _name, Waiting_state, Waiting_time)
Description	Modbus RTU waiting for analog input settings
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: DI Name; • Waiting_state: waiting state, 1-Ture, 2-Flase; • Waiting_time: timeout unit [ms].
Return value	null

ModbusMasterWaitAI_RTU: Waiting for Digital Input Settings (Waiting for Discrete Input values)

Table 3-223: Detailed Parameters of Modbus MasterWaitAI_RTU

Attribute	Explanation
Prototype	ModbusMasterWaitAI_RTU (Modbus_name, Register_name, Waiting_state, Register_value， Waiting_time)
Description	Modbus RTU waits for digital input settings

Table 3-206 (Continued)

Attribute	Explanation
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: AI name; • Waiting_state: waiting state, 1-<, 2->; • Register_value: Register value; • Waiting_time: timeout [ms].
Return value	null

Code 3-60 Waiting for Digital/Analog Input/Output Example

```
5.  ModbusMasterWaitDI_RTU (Modbus_0,Register_0,1,1000)
6.  --Modbus 0- Master Station Name, Register 0-DI Name, 1-True, 1000- Timeout Time ms
7.  ModbusMasterWaitAI_RTU (Modbus_0, Register_2,0,13,1000)
8.  --Modbus 0- Master Station Name, Register 2-DA Name, 0->, 13- Register value, 1000- Timeout Time ms
```

Related instructions from the station

ModbusSlaveWriteDO_RTU: Slave Digital Output Settings (Write Discrete Input)

Table 3-224: Detailed Parameters of Modbus SlaveDWriteDO_RTU

Attribute	Explanation
Prototype	ModbusSlaveWriteDO_RTU (Register_name, Register_num, {Register_value})
Description	Modbus RTU Slave Station Write Digital Output Settings
Parameter	• Register_name: DO Name; • Register_num: Number of registers; • {Register_value}: Register value, the number of register values matches the number of registers {value_1, value_2,...}.
Return value	null

ModbusSlaveReadDO_RTU: Read Digital Output (Read Discrete Input)

Table 3-225: Detailed Parameters of Modbus SlaveRadDO_RTU

Attribute	Explanation
Prototype	ModbusSlaveReadDO_RTU (Register_name, Register_num)
Description	Modbus RTU reads and writes digital outputs
Parameter	• Register_name: DO Name; • Register_num: Number of registers;
Return value	{Register_value}: Register value, the number of register values matches the number of registers {value_1, value_2,...}

ModbusSlaveReadDI_RTU: Read digital input (read coil)

Table 3-226: Detailed Parameters of ModbusSlaveReadDI_RTU

Attribute	Explanation
Prototype	ModbusMasterReadDI_RTU (Register_name, Register_num)
Description	Modbus RTU slave station reads digital input
Parameter	• Register_name: DI Name; • Register: Number of registers.
Return value	Reg_falue1, Reg_falue2,...: returns the corresponding quantity of values based on the value of Regite_num

Code 3-61 Slave Station Digital Input/Output Settings

```
-Slave station digital output settings
ModbusSlaveWriteDO_RTU (DO0,1,{2})
-Write digital output, DO0-DO number, 1-register quantity, {2}- Register value
DO_value = ModbusSlaveReadDO_RTU (DO0,1)
-Read digital output, DO0-DO number, 1-register quantity

-Digital input settings
ModbusSlaveReadDI_RTU (DI1,3)
-Read numerical input, DI1-DI name, 3-register quantity
```

ModbusSlaveWaitDI_RTU: Waiting for digital input settings (waiting for coil values)

Table 3-227: Detailed Parameters of ModbusSlaveWaitDI_RTU

Attribute	Explanation
Prototype	ModbusSlaveWaitDI_RTU (Register_name, Waiting_state, Waiting_time)
Description	Modbus RTU waits for digital input settings
Parameter	• Register_name: DI Name; • Waiting_state: waiting state, 1-Ture, 2-Flase; • Waiting_time: The unit of waiting time [ms].
Return value	null

ModbusSlaveWaitAI_RTU， Waiting for analog input settings (waiting to hold register values)

Table 3-228: Detailed Parameters of ModbusSlaveWaitAI _RTU

Attribute	Explanation
Prototype	ModbusSlaveWaitAI_RTU (Register_name, Waiting_state,Register_value，Waiting_time)
Description	Modbus RTU slave station waiting for analog input settings
Parameter	• Register_name: AI name; • Waiting_state: waiting state, 1-<, 2->; • Register value: Register value; • Waiting_time: timeout [ms].
Return value	null

Code 3-62 slave station waiting for digital/analog input settings

```
5.  ModbusSlaveWaitDI(DI2,0,100)
6.  -Waiting for numerical input settings: DI2-DI name, 0-false, 100- waiting time ms
7.  ModbusSlaveWaitAI(AI1,0,12,133))
8.  -AI1-AI name, 0->, 12 register value, 133 timeout time ms
```

ModbusMasterReadReg_RTU: Read Register Instruction

Table 3-229: Detailed Parameters of ModbusMasterReadReg_RTU

Attribute	Explanation
Prototype	ModbusMasterReadReg_RTU（fun_code， reg_add，reg_num，add, isthread）
Description	Read register instruction

Table 3-212(Continued)

Attribute	Explanation
Parameter	• fun_code: Function code, 1-0x01 coil, 2-0x02 discrete quantity, 3-0x03 hold register, 4-0x04 input register; • reg_add: Register address; • reg_num: Number of registers; • add: Address; • isthread: Whether to apply threads, 0- No, 1- Yes.
Return value	null

ModbusMasterWriteReg_RTU: Write Register

Table 3-230 Detailed Parameters of ModbusMasterWrite_RTU

Attribute	Explanation
Prototype	ModbusMasterWrite_RTU（fun_code， reg_add，reg_num，reg_value，add, isthread）
Description	Write register
Parameter	• fun_code: function code, 5-0x05- single coil, 6-0x06- single register, 15-0x0f - multiple coils, 16-0x10- multiple registers; • reg_add: single coil, single register, multiple coils, multiple register addresses; • reg_num: Number of registers; • reg_ralue: byte array;
Return value	• add: Address; • isthread: Whether to apply threads, 0- No, 1- Yes. null

Code 3-63 Modbus_RTU Reg Instruction Example

```
13. addr = 0x1000
14. va1 = {400, 600, 900, 700}
15. ret = {}
16. ModbusMasterWriteReg_RTU (10, addr, 4,va1, 1, 0)
17. --1-0x10- Multiple registers, addr - Register address, 4- Number of registers, va1- Byte array, 1- Address, 0- No threads applied
18. WaitMs(10)
19. ModbusMasterReadReg_RTU (4, addr, 4, 1, 0)
20. --1-0x04- Input register, addr - Register address, 4- Number of registers, 1- Address, 0- Do not apply thread
21. WaitMs(10)
```

3.7.2 Xmlrpc

XMLRPC is a remote procedure call method that uses sockets to transfer data between programs using XML. Through this method, the robot controller can call functional functions (with Parameters) from remote programs/services and obtain the returned structural data.

XMLrpcClientCall: Data Remote Call

Table 3-231 Detailed Parameters of XMLrpcClientCall

Attribute	Explanation
Prototype	XmlrpcClientCall (url, func, type， func_ Para)
Description	Remote data call
Parameter	• url: Server URL; • func: Call the function; • type: The type of the input Parameter, a 1-double array, a 2-string array; • func_ Para: Call function Parameters.
Return value	null

Code 3-64 Xmplppc Instruction Example

```
1.  --Example of double array:
2.  xmlrpcclientcall(" http://192.168.58.20:50000/rpc2 ","example.array",1,{1.0,2.0,3.0})
3.  -- http://192.168.58.20:50000/rpc2 -Server URL, example. array - call function name, 1-
    type, {1.0,2.0,3.0}- Call function Parameters
4.  --Example of string array:
5.  xmlrpcclientcall(" http://192.168.58.20:50000/rpc2 ","example.array",2,{"hello","world"})
6.  -- http://192.168.58.20:50000/rpc2 -Server URL, example. array - call function name, 0-
    type, {1.0,2.0,3.0}-- Call function Parameters
```

3.8 Auxiliary instruction

3.8.1 Auxiliary Threads

FR Lua provides auxiliary thread functionality, where users can define an auxiliary thread to run simultaneously with the main thread. The auxiliary thread mainly interacts with external devices for data exchange.

NewAuxthread: Creating auxiliary threads

Table 3-232 Detailed Parameters of NewAuxThread

Attribute	Explanation
Prototype	NewAuxThread (func_name, func_ Para)
Description	Create auxiliary thread
Parameter	• func_name: Call function; • func_ Para: Call function Parameters.
Return value	null

Code 3-65 Auxiliary Thread Example

```
1.  --Definition of auxiliary thread function
2.  function auxThread_TCPCom(ip, port)
3.      local flag = 0
4.      Set SysNumber (1,0) -- System variable 1 is assigned a value of 0
5.      while 1 do
6.          if flag == 0 then
7.              Flag=SocketOpen (IP, port, "socket-0") -- Establish a connection with the
    server
8.          elseif flag == 1 then
9.              SocketSendString("hello world","socket_0",1)
10.         n. Svar=SocketReadAsciiFloat (1, "socket-0", 0) -- interacts with the server for
    data exchange
11.         if n == 1 then
12.             Set SysNumber (1, svar) - Assign svar to system variable 1
13.             end
14.         end
15.     end
16. end
17. -Create auxiliary thread
18. NewAuxThread(auxThread_TCPCom, {"127.0.0.1",8010})
19. WaitMs(100)
20. while 1 do
21.     v=Get SysNumber (1) - Get the value of system variable 1
22.     if v == 100 then
23.         PTP(P1,10,0,0)
24.     elseif v == 200 then
25.         PTP(P2,10,0,0)
26.     end
27. end
```

3.8.2 Call Function

FR Lua provides robot interface functions for customers to choose from and prompts them with the required Parameters for the function, making it convenient for customers to write script instructions

For example, the provided GetInverseKinRef and GetInverseKinHasSolution functions.

GetInverseKinRef: Inverse kinematics solution - specifying position reference

Table 3-233 GetInverseKinRef Detailed Parameters

Attribute	Explanation
Prototype	GetInverseKinRef (type, desc_pos, joint_pos_ref)
Description	Inverse kinematics, tool pose solving joint position, referencing specified joint position solving
Parameter	• type: 0- Absolute pose (base coordinate system), 1-Relative pose (base coordinate system), 2-Relative pose (tool coordinate system); • desc_pos: {x, y, z, rx, ry, rz} tool pose, unit [mm] [°]; • joint_pos_def: {j1, j2, j3, j4, j5, j6}, joint reference position, unit [°].
Return value	J1, j2, j3, j4, j5, j6: Joint position, unit [°]

GetInverseKinHasSolution， Inverse kinematics solution - Is there a solution

Table 3-234 Detailed Parameters of FHIR converseKinHasSolution

Attribute	Explanation
Prototype	GetInverseKinHasSolution (type, desc_pos, joint_pos_ref)
Description	Is there a solution for solving joint positions using inverse kinematics and tool pose
Parameter	• type: 0- Absolute pose (base coordinate system), 1-Relative pose (base coordinate system), 2-Relative pose (tool coordinate system); • desc_pos: {x, y, z, rx, ry, rz} tool pose, unit [mm] [°]; • joint_pos_def: {j1, j2, j3, j4, j5, j6], joint reference position, unit [°].
Return value	Result: 'True' - there is a solution, 'False' - there is no solution

Code 3-66 Call Function Example

```
1.  J1={95.442,-101.149,-98.699,-68.347,90.580,-47.174}
2.  P1={75.414,568.526,338.135,-178.348,-0.930,52.611]
3.  ret_1 = GetInverseKinRef(0,P1,J1)
```

Code 3-66 (Continued)

```
4.  --Inverse kinematics solution - specify reference position, 0-absolute pose (base coordinate system), P1 tool pose, J1 joint reference position
5.  ret_2 = GetInverseKinHasSolution(0,P1,J1)
6.  --Inverse kinematics solution - whether there is a solution, 0-absolute pose (base coordinate system), P1 tool pose, J1 joint reference position
```

3.8.3 Point Table

PointTableSwitch: Point Switching

Table 3-235 PointTableSwitch Detailed Parameters

Attribute	Explanation
Prototype	PointTableSwitch(point_table_name)
Description	Point table switching
Parameter	• Point_table_name: The name of the point table to be switched is pointTable1.db. When the point table is empty, that is, "", it means updating the Lua program to the initial program that has not applied the point table, in system mode.
Return value	null

Code 3-67 Example of Point Representation

```
1.  PointTableSwitch("point_table_a.db")
```