1507, TFDEN-012-007, 001

NACHi

FD CONTROLLER
INSTRUCTION MANUAL
Robot language

7th edition

- Before attempting to operate the robot, please read through this operating manual carefully, and comply with all the safety-related items and instructions in the text.
- The installation, operation and maintenance of this robot should be undertaken only by those individuals who have attended one of our robot course.
- When using this robot, observe the low related with industrial robot and with safety issues in each country.
- This operating manual must be given without fail to the individual who will be actually operating the robot.
- Please direct any queries about parts of this operating manual which may not be completely clear or any inquiries concerning the after-sale service of this robot to any of the service centers listed on the back cover.

NACHI-FUJIKOSHI CORP.

Table of Contents

Chapter 1 Outline

Chapter 2 Syntax

Chapter 3 Program editing

Chapter 4 Compiling programs

Chapter 5 Command

Chapter 6 How to Use the Robot Language

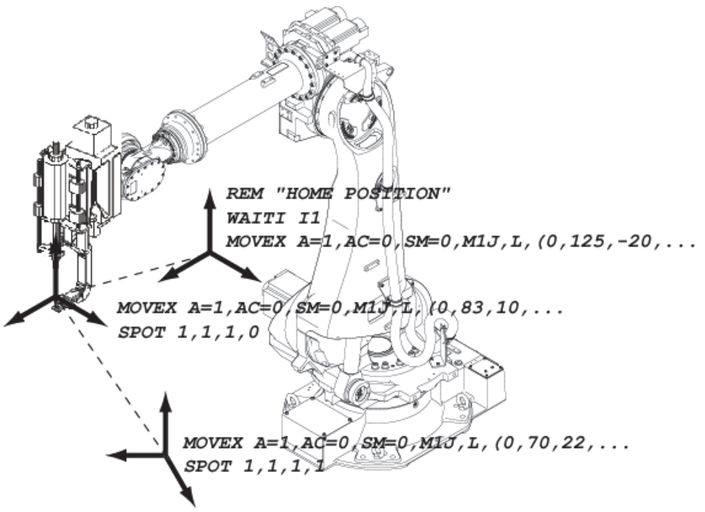
Chapter 1 Outline

This chapter describes the outline of the robot language.

1.1 What is "Robot language" ?

1.1.1 Outline

"Robot language" is a programming language for this robot controller. This function can make (teach) a work-program of a robot by combining move commands (`MOVEX`) and application commands like input signal waiting command (`WAITI`) etc. Because it is also possible to use various functions and calculation formula, this function is suitable for making a work-program that requires complicated flow controls or complicated coordinate calculations that are difficult to achieve when using normal teaching operation via teach pendant keys. Furthermore, it is also possible to create a customized monitor window by using the original commands for the robot language function.



```
REM "HOME POSITION"
WAITI I1
MOVEX A=1,AC=0,SM=0,M1J,L,(0,125,-20,...
```

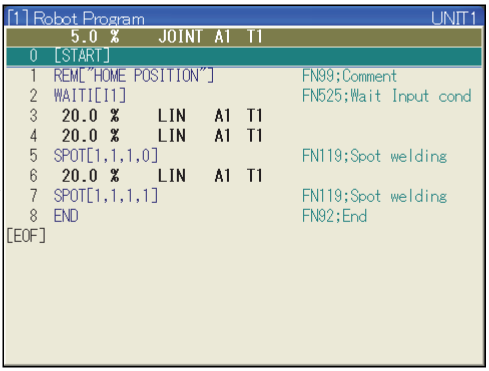
```
MOVEX A=1,AC=0,SM=0,M1J,L,(0,83,10,...
SPOT 1,1,1,0
```

```
MOVEX A=1,AC=0,SM=0,M1J,L,(0,70,22,...
SPOT 1,1,1,1
```

An image of robot motion

```
REM "HOME POSITION"
WAITI I1
MOVEX A=1,AC=0,SM=0,M1J,L,(0,125,-20,…
MOVEX A=1,AC=0,SM=0,M1J,L,(0,83,10,…
SPOT 1,1,1,0
MOVEX A=1,AC=0,SM=0,M1J,L,(0,70,22,…
SPOT 1,1,1,1
END
```

Robot language program (source program)



```
[1] Robot Program                                UNIT1
      5.0 %     JOINT  A1  T1
  0  [START]
  1  REM["HOME POSITION"]          FN99;Comment
  2  WAITI[I1]                     FN525;Wait Input cond
  3   20.0 %     LIN   A1  T1
  4   20.0 %     LIN   A1  T1
  5  SPOT[1,1,1,0]                 FN119;Spot welding
  6   20.0 %     LIN   A1  T1
  7  SPOT[1,1,1,1]                 FN119;Spot welding
  8  END                           FN92;End
[EOF]
```

Work program (executable format)

Fig. 1.1.1 An image of a work program generated using robot language

Concerning the syntax of robot language, refer to the chapter 2. And, while you are not accustomed to the robot language, we recommend you to try to make several simple work-programs with normal teaching operation by referring to the instruction manual "BASIC OPERATIONS MANUAL" and then try to "reverse compile" those programs to get robot language format source programs. This trial will help you to understand the relationship between the robot language and the work-program. Concerning how to make "reverse-compile" operation, refer to the chapter 4.

INFO.

In this instruction manual, the details of the application commands are not described. For details, please refer to the online help displayed on the teach pendant screen or the instruction manual "COMMAND REFERENCE".

INFO.

Concerning the operations about the customized monitor window, refer to the instruction manual "User Task."

1.1.2 Characteristics and precautions

- The source program format of robot language is plain text format. The source program can be made using a text editor of your PC or "ASCII File Edit" screen of this robot controller.

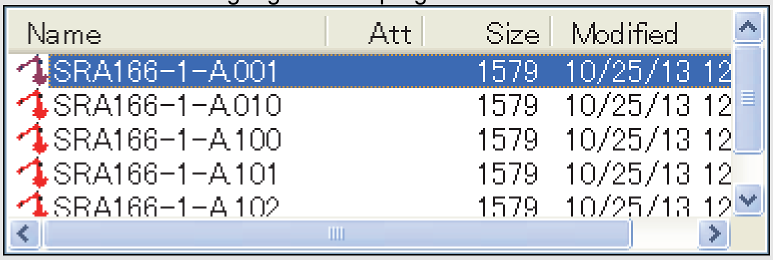
- For example, if the robot type if NB4-02, the file name of the robot language program (source program) and the work-program (executable format) are like the following examples. If the filename does not follow this style, the file is not recognized by this controller.

Robot language program (source program)	`SRA166-1-A.001` `NB4-02-A.001`
Work-program (executable format)	`SRA166-1.001` `NB4-02.001`

To confirm the file name that must be used for the source program file, open the "PROGRAM" folder in the <Service Utilities> - [8 File manager] - [2 Directory] screen. If there are work-programs that were already made, their file names will be displayed.

To make a filename of a robot language source program, attach "-A" to the filename.

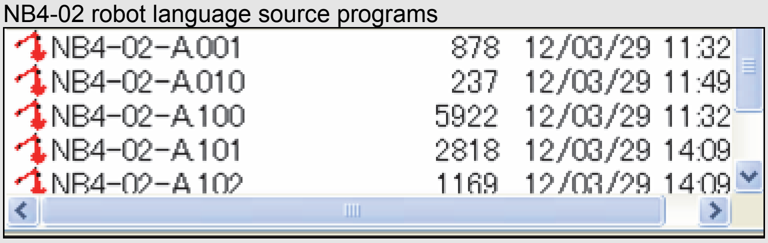
SRA166-01 robot language source programs



Name	Att	Size	Modified
SRA166-1-A001		1579	10/25/13 12
SRA166-1-A010		1579	10/25/13 12
SRA166-1-A100		1579	10/25/13 12
SRA166-1-A101		1579	10/25/13 12
SRA166-1-A102		1579	10/25/13 12

(In this case, "SRA166-1-**A**.001" is the source program name for "SRA166-1.001")

NB4-02 robot language source programs



NB4-02-A001	878	12/03/29 11:32
NB4-02-A010	237	12/03/29 11:49
NB4-02-A100	5922	12/03/29 11:32
NB4-02-A101	2818	12/03/29 14:09
NB4-02-A102	1169	12/03/29 14:09

(In this case, "NB4-02-**A**.001" is the source program name for "NB4-02.001")

- Robot language is based on the SLIM (**S**tandard **L**anguage for **I**ndustrial **M**anipulator) language which complies with the JIS B9439 standard.

- The robot language source program must be converted (compiled) to an executable format in advance to use. The robot language format and the executable format can be converted to each other. (These operations are referred to "Compile" and "Reverse-compile"). And, when making a work-program with normal teaching operation using the teach pendant, the file is recorded to the internal memory automatically in a style of executable format from the beginning. So it is not necessary to compile the file to use.

- The work-programs made from robot language and the work-programs made from normal teaching operation can mingle with each other. It is also possible to use a robot in a way in which only complicated calculation and condition jump/call process etc. are controlled with robot language logic and the other robot program created with normal teaching operation are called from those robot language programs.

- This robot language function supports 2 types of move command teaching method. The first one is a method in which the position data is directly described in the move command and the second one is a method in which a pose variable is called using the parameter in the move command. The pose variables can be made via position record operation using the teach pendant or robot language's "Pose operation" etc. It is also possible to manage the plural pose variables (made via position record operation) altogether in a form of "Pose file". For details of "Pose file", refer to the chapter 3.

How to describe the pose constant in a program ☞ "2.4.3 Pose constants"

Example 1 : MOVEX-X

```
MOVEX A=1,AC=0,SM=0,M1X,L,(1800,0,2000,0,-90,-180),R=5.0,H=1,MS
```

Example 2 : MOVEX-J

```
MOVEX A=1,AC=0,SM=0,M1J,L,(0,90,0,0,0,0),R=5.0,H=1,MS
```

How to designate the position using pose variable ☞ "2.5.9 Pose variable","3.3 Creating pose files"

```
MOVEX A=1,AC=0,SM=0,M1X,P,P1,R=5.0,H=1,MS
```

- After converting to an executable format, it is possible to modify the position data of a move command using the normal position modification operation with the teach pendant even if the move command is made from robot language. However, in this case, please be sure that in spite of the format of the original move command, the robot position data is forcibly converted to encoder value format (this is the same format of which a move command that is recorded with normal position record operation).

- Even in case of a work program that is generated from robot language, it is possible to make step addition or step deletion in the same way with normal work-program. But, it is recommended to modify the original source program (not the executable format work-program) and make compile operation again. And, to make small modifications to the source program file, "ASCII File Edit" function of this robot controller is convenient. For details, refer to the chapter 3.

1.2 Teaching (programming) and playback

Outline of the operation

1 Create a source program file using a text file editor on your PC or "ASCII File Edit" function of this controller.
☞"3.1 Editing using a personal computer "

2 For example, if the robot type is "SRA166-01" or "NB4-02", save the source program to the USB memory using a file name like the following;

`SRA166-1-A.001` (The file name of the source program)
`NB4-02-A.001` (The file name of the source program)

- "`SRA166-1`" and "`NB4-02`" shows the robot type.
- "`-A`" shows that the file is robot language source program.
- The file name extension "`001`" shows the program number. (MAX is 9999)

☞See the item of file name in "1.1.2Characteristics and precautions "
☞See "3.1.2　Loading a robot language source program into this controller "
☞See the Chapter 6 of "BASIC OPERATIONS MANUAL"

3 Copy the source program file from the USB memory to the internal memory of this controller.

This operation can be done in the <Service Utilities> - [7 File Manager] - [1 File Copy] screen.
☞See the Chapter 6 of "BASIC OPERATIONS MANUAL"

4 Convert the source program to an "executable format" by executing "Compile" operation.

This operation can be done in the <Service Utilities> - [9 Program Conversion] - [8 Language] screen.
☞See "4.1 Compiling" of this instruction manual.

When the compile operation is normally finished, an executable program file will be generated.
In this example, the file name would be the following;

`SRA166-1.001` (The filename of a executable format)
`NB4-02.001` (The filename of a executable format)

5 For the generated executable format program, CHECK GO/BACK or playback operation etc. can be applied in the same procedures with the normal work-program.

☞See the Chapter 4 and 5 of "BASIC OPERATIONS MANUAL"

6 If there exists problems in the robot motion, revise the source program file and compile it again. For small modifications, "ASCII File Edit" function is convenient.

☞See "3.2 Editing using the teach pendant"

INFO.

When using integer variables etc., it is possible to check their values with "Monitor" function. This "Monitor" function is included in ＜Service Utilities＞ menu.

WARNING

Because the robot language can create the coordinate freely using calculation formula etc., please pay special attention when moving the robot. If the robot makes unexpected motion, death or serious injury may result.

Chapter 2 Syntax

This chapter describes the syntax of the robot language when the robot language is used to prepare programs.

2.1 An example of robot language program

A robot language source program

```
100 'SHIFT
110 USE 100
120 FOR V1%=1 TO 7
130 R1=(10,1,0,0,0,0)
140 P100=P[V1%]+R1
150 MOVEX A=1,M1X,P,P100,R=5.0,H=1,MS
160 NEXT
170 END
```

LINE 100 : Comment statement

Starting a line with a single quotation mark (') turns the statement on the line into a comment statement, thus making it possible to facilitate reading of the program. "REM" can be used instead of single quotation mark.

☞ "2.8.1 Comment statement"

LINE 110 : USE command

This statement selects the Pose file No. 100.
From this line downward, "Pose variable" will be retrieved from the Pose file No. 100.

☞ "2.5.9 Pose variable"

LINE 120 : FOR command

A flow control statement includes a jump between lines (GOTO), a conditional branch (IF), and others. "V1%" is a variable to which an integer can be assigned. In this line, this is used as a loop counter.

☞ "2.8.4 Flow control statement"

LINE 130 : Assignment statement

In an assignment statement, values are directly assigned to the shift variable "R1".

☞ "2.5.10 Shift variable", "2.8.3 Substitution statement"

LINE 140 : Assignment statement

The operation of the pose variable "P[V1%]" and the shift variable "R1" is carried out to assign the result to the pose variable "P100".

☞ "2.5.9 Pose variable", "2.5.10 Shift variable"

LINE 150 : MOVEX command

A move command (MOVEX command) is executed using pose variable P100, thus making the robot move.

☞ "2.5.9 Pose variable"

LINE 160 : NEXT command

This flow control statement pairs up with the FOR statement. In this case a jump to LINE 120 will be made.

☞ "2.8.4 Flow control statement"

LINE 170 : END command

The program is ended. The END command is always necessary at the bottom line of the program.

(*) The line numbers can be omitted.

☞ "2.2 Line numbers"

Fig. 2.1.1 Robot language syntax (Example)

2.2 Line numbers

A line is the smallest unit that makes up a robot language program. A robot language is executed on a line by line basis.
A total of **254 characters** may be written on one line, and up to **9999 lines** can be written in a program. A line consists of a comment statement, label, assignment statement, flow control statement or command statement. (☞2.8Statements) Including two or more statements in a line (e.g. "`FOR V1%=0 TO 100, NEXT`") is not permitted.

POINT

- Two or more of statements cannot be written on one line.
- It is not permitted to enter a line feed code anywhere in a statement.
- The maximum number of characters allowed in a statement is 254. A compiling error results when a line with 255 or more characters is written.

The line numbers indicate the execution positions in the program. Numbers from 1 to 9999 can be specified. Line numbers can also be used as the jump destinations of line jump instructions, etc. They are allocated in numerical order so that the lowest numbers come at the start of a program and progressively increase as the program advances.

POINT

Line numbers can be omitted. In this case, use labels for the jump destinations of line jump instructions, etc.
When a line number is omitted, any line number must not be written anywhere in the program. Programs will not run properly if some lines are numbered while others are not.

POINT

When reverse-compiled, the line number will be changed.

2.3 Characters

The table below lists the characters which can be used by the robot language.

Table 2.1 Characters which can be used by the robot language

Item	Characters which can be used
Alpha numerics	ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 01234567890
Symbols	! ” ’ # $ % & ( ) * + - . , / : ; = < > ? @ ` [ ] ¥ ^ { }~ | _
Blanks	(Half-size space) (tab)

POINT

No differentiation is made between upper-case and lower-case letters of the alphabet. Use half-size characters for all text except for comments and character string variables.

2.4 Constants

The following types of constants are available.
The constants shown below will be retained as the power failure retention data, even if the main power supply is turned OFF.

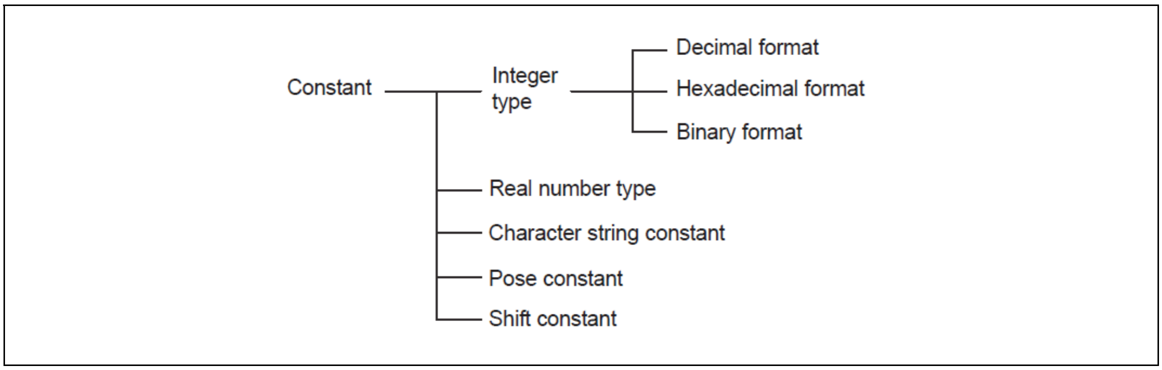


Fig. 2.4.1 Types of constants

2.4.1 Numeric constants

Numeric constants are numeric data used for executing operations.
Whether they are integer or real number constants is automatically determined internally.

<Examples>

・ `-10`	Integers
・ `1.23—E12`	Real numbers
・ `&H1F3`	Hexadecimal format
・ `&B10101`	Binary format
・ `90DEG`	Angle value (real number) expressed in degrees

Integers range from –2147483648 to +2147483648.
Real numbers range from –1.0+E38 to +1.0+E38.

INFO.

Refer to the “Appendix” about the hexadecimal number/ binary number system.

2.4.2 Character string constants

Alphanumeric characters and symbols enclosed in double quotation marks (") are referred to as a character string.

<Examples>

・ `PRINT #0, "ABCDE"`

2.4.3 Pose constants

"Pose constants" is a constant that represents the robot position and angle. This constant is used combining MOVEX command. There are 3 types of Pose constants.

MOVEX-X	This represents the TCP position and posture using (X,Y,Z,r,p,y)
MOVEX-J	This represents the robot position using a set of angles or positions of each axis.
MOVEX-E	This represents the robot position using a set of encoder values of each axis.

MOVEX-X

A MOVEX-X style pose constant represents the TCP position and the direction using (X, Y, Z, roll, pitch, yaw). The mechanism selection is "M*X" ("*" is the mechanism number).

The reference point is the origin of the mechanism coordinate system of the robot.

(Example) MOVEX A=1,**M1X,P,(2000,0,1330, 0, 0, 0)**,R= 10.0,H=1,MS

Table 2.2 The parameters for MOVEX-X

Parameter	MOVEX-X	
X	X coordinate of the TCP [mm]	The coordinates of the Tool Center Point (Based on the machine coordinate system)
Y	Y coordinate of the TCP [mm]	
Z	Z coordinate of the TCP [mm]	
r (Roll)	"Roll" angle of the tool [deg] (Around Z axis)	The direction of the tool coordinate system (Based on the machine coordinate system)
p (Pitch)	"Pitch" angle of the tool [deg] (Around Y axis)	
y (Yaw)	"Yaw" angle of the tool [deg] (Around X axis)	

Designate the coordinates of the TCP in (X, Y, Z) based on the machine coordinate system.

And, designate the angle of the tool in (Roll, Pitch, Yaw) based on the machine coordinate system's direction. Please be sure that the tool direction depends on the order of the rotation. (The rotation should be done in an order of Roll (Z), Pitch (Y), and Yaw (X))

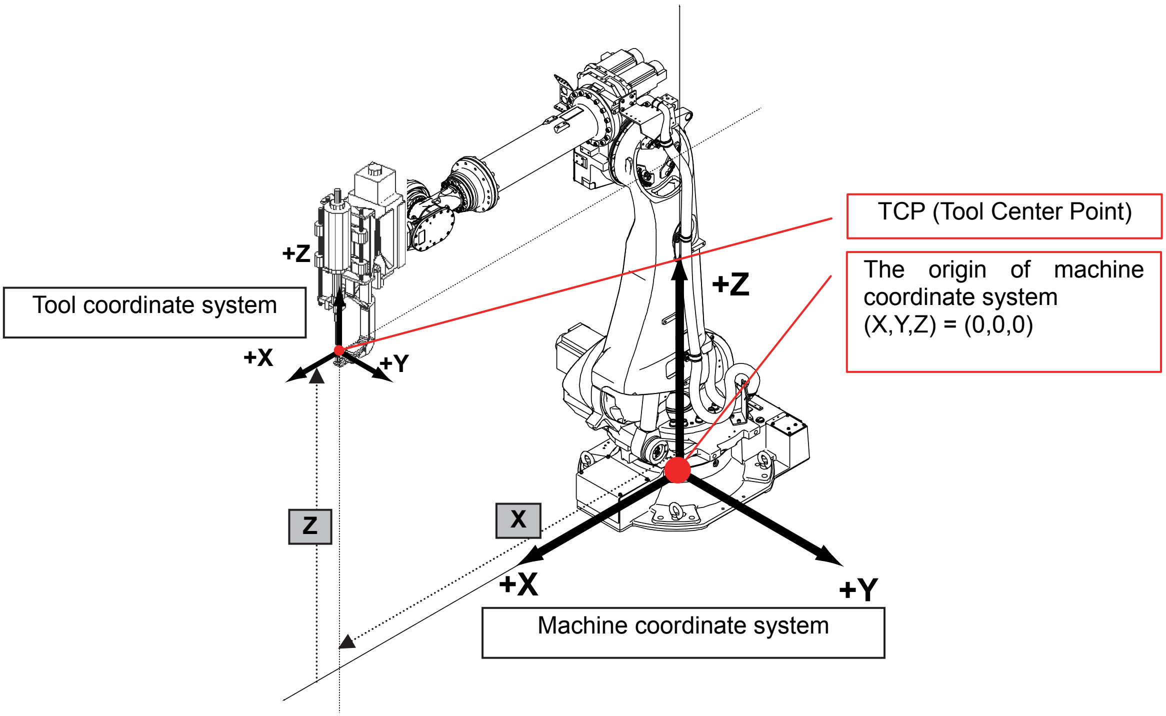

TCP (Tool Center Point)
The origin of machine coordinate system (X,Y,Z) = (0,0,0)
Tool coordinate system
+Z
+X
+Y
+Z
Z
X
+X
+Y
Machine coordinate system

Fig. 2.4.2 A machine coordinate system and a tool coordinate system

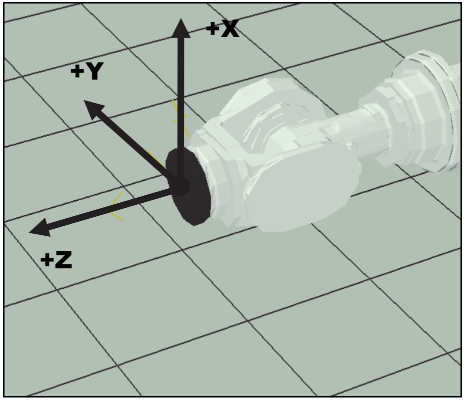
CAUTION

- Because the TCP position and the tool angle are written in the machine coordinate system, in a move command using this representation, the position where the actual TCP reaches will not change even if the tool constant is changed. However, please do not forget that the angle of each axis will change at the step. So please pay special attention for e.g. interference etc.
- When not using CONF setting, the robot may make unexpected posture (configuration). For details of the CONF setting, please refer to the online help of the MOVEX command or the Chapter 5. And, it is recommended to use MOVEX-J for the move command step that will be executed at first time. (Because MOVEX-J does not have redundancy of the robot posture to be generated and can make a robot unique posture)

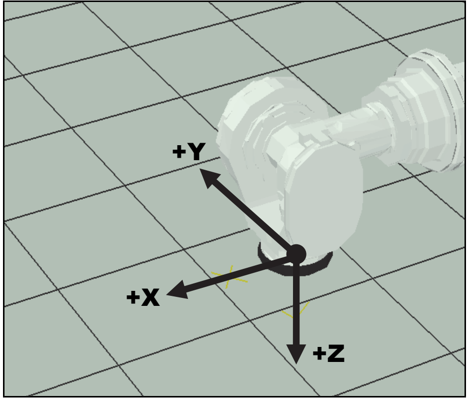
(NOTE) Examples of (X, Y, Z, roll, pitch, yaw)

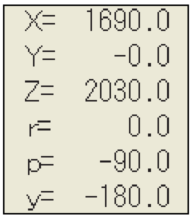
When rotating the machine coordinate system (=Robot coordinate system) in the order of roll(Z), pitch(Y), yaw(X), it becomes the direction of the tool coordinate system.

Example1

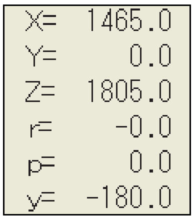


Example2





```
X=  1690.0
Y=    -0.0
Z=  2030.0
r=     0.0
p=   -90.0
y=  -180.0
```



```
X=  1465.0
Y=     0.0
Z=  1805.0
r=    -0.0
p=     0.0
y=  -180.0
```

- When the application command FN142 “GETP[V1!]” is executed, the present coordinates (X, Y, Z, roll, pitch, yaw) will be stored to the continuous 6 real number variables of (V1!, V2!, V3!, V4!, V5!, V6!).

(V1!=X, V2!=Y, V3!=Z, V4!=roll, V5!=pitch, V6!=yaw)

- When executing a shift operation for coordinates of (X,Y,Z,roll,pitch,yaw) using a shift register (ΔX,ΔY,ΔZ,θX,θY,θZ), the calculated (shifted) coordinates will be (X+ΔX, Y+ΔY, Z+ΔZ, roll+**θZ**, pitch+**θY**, yaw+**θX**). Please pay attention to the order of θX, θY, and θZ.

MOVEX-J

This represents the robot position using a set of angles [deg] or positions [mm] of each axis. The number of the parameters differs from the mechanism type. (The maximum number of the axes for 1 mechanism is 6)

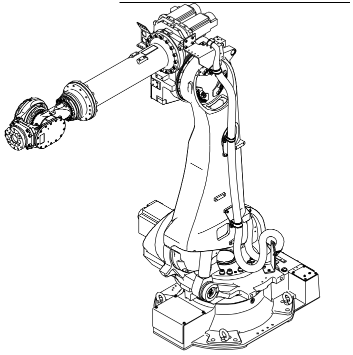
Table 2.3 The parameters for MOVEX-J

Parameter	MOVEX-J
J1	Angle [deg] or position [mm] for the 1st axis
J2	Angle [deg] or position [mm] for the 2nd axis
J3	Angle [deg] or position [mm] for the 3rd axis
J4	Angle [deg] or position [mm] for the 4th axis
J5	Angle [deg] or position [mm] for the 5th axis
J6	Angle [deg] or position [mm] for the 6th axis

In case of a normal 6-axes robot, write 6 values.

An example of 6-axes robot (SRA166)

```
MOVEX A=1,M1J,P,(0,90,0,0,0,0),R= 5.0,H=1,MS
```



If the robot has additional axes like servo gun or slider (traverse axis) etc. refer to the example shown as below.

An example of 6-axes robot that has slider (traverse axis)

```
MOVEX A=1,M1J,P,(0,90,0,0,0,0),R= 5.0,H=1,MS,M2J,P,(100),R= 10.0,H=1
```

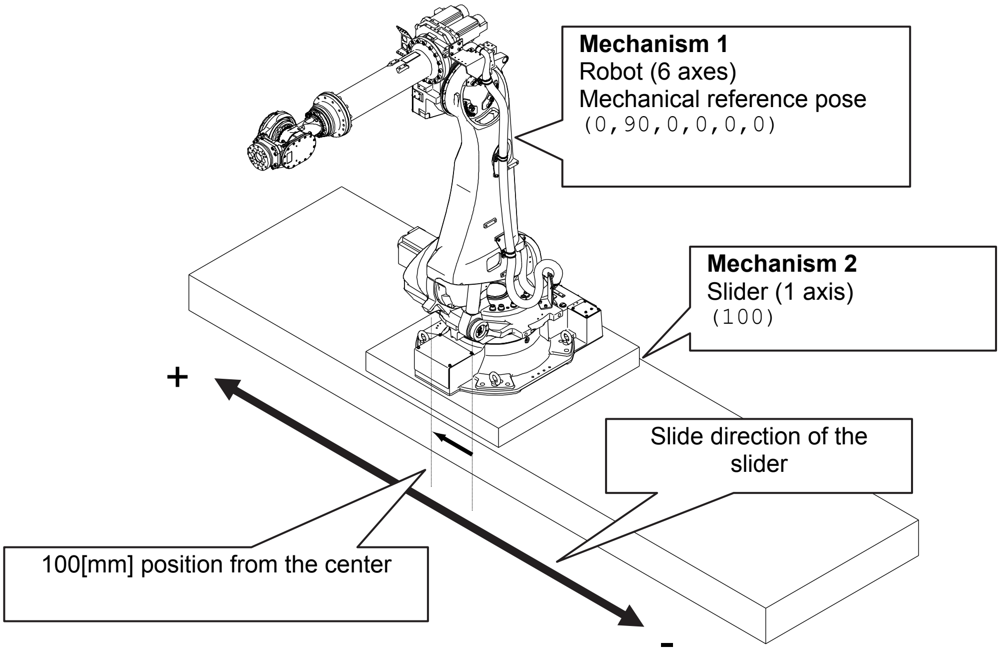

Mechanism 1
Robot (6 axes)
Mechanical reference pose
(0,90,0,0,0,0)
Mechanism 2
Slider (1 axis)
(100)
+
Slide direction of the slider
100[mm] position from the center
-

Fig. 2.4.3 A robot with a slider

In this example, the mechanism 1 makes its mechanical reference pose and the mechanism 2 makes 100 [mm] position.

MOVEX-E

This represents the robot position using a set of encoder values of each axis. When there are 2 or more mechanisms in the UNIT, please refer to the example of MOVEX-J.

Example: `MOVEX A=1,M1E,P,(&H80000,&H80000,&H80000,&H80000,&H80000,&H80000),R= 10.0,H=1,MS`

In case of 2 or more mechanisms, refer to the example of MOVEX-J.

Table 2.4 The parameters for MOVEX-E

Parameter	MOVEX-E
E1	Encoder value for the 1st axis
E2	Encoder value for the 2nd axis
E3	Encoder value for the 3rd axis
E4	Encoder value for the 4th axis
E5	Encoder value for the 5th axis
E6	Encoder value for the 6th axis

POINT The movement command generated from this format (MOVEX-E) will be the same with a movement command that is recorded by the [REC] key on a teach pendant.

2.4.4 Shift constants

The shift constants are used for adding the translational amounts for the robot poses. They are described using the (X, Y, Z, r, p, y) format.

Table 2.5 Shift constant

Parameter	Shift amount
X	Shift in X-axis direction [mm]
Y	Shift in Y-axis direction [mm]
Z	Shift in Z-axis direction [mm]
r (Roll)	Rotational shift around Z axis [deg]
p (Pitch)	Rotational shift around Y axis [deg]
y (Yaw)	Rotational shift around X axis [deg]

2.4.5 MOVEX-X with User coordinate system (position and direction)

When a "U" is attached to the MOVEX-X pose constant description like the following example, the target teach point can be set based on an User coordinate system.

Example: `MOVEX A=1,M1X,P,(100, 0, 200, 0, 0, 180)U,R= 10.0,H=1,MS`

Table 2.6 MOVEX-X with User coordinate system

Parameter	Description
X	X coordinate of the TCP [mm] (based on the User coordinate system)
Y	Y coordinate of the TCP [mm] (based on the User coordinate system)
Z	Z coordinate of the TCP [mm] (based on the User coordinate system)
r (Roll)	"Roll" angle of the tool [deg] (Around Z axis of the User coordinate system)
p (Pitch)	"Pitch" angle of the tool [deg] (Around Y axis of the User coordinate system)
y (Yaw)	"Yaw" angle of the tool [deg] (Around X axis of the User coordinate system)

POINT Before using User coordinate system, please select the user coordinate system to be used by executing FN113 CHGCOORD in advance.

2.5 Variables

The structures for holding the values used in programming are called variables. Variables are used as references for operations and other such uses. Their names are reserved in advance, and users cannot assign the names of their choice to them.
The following variables are available.

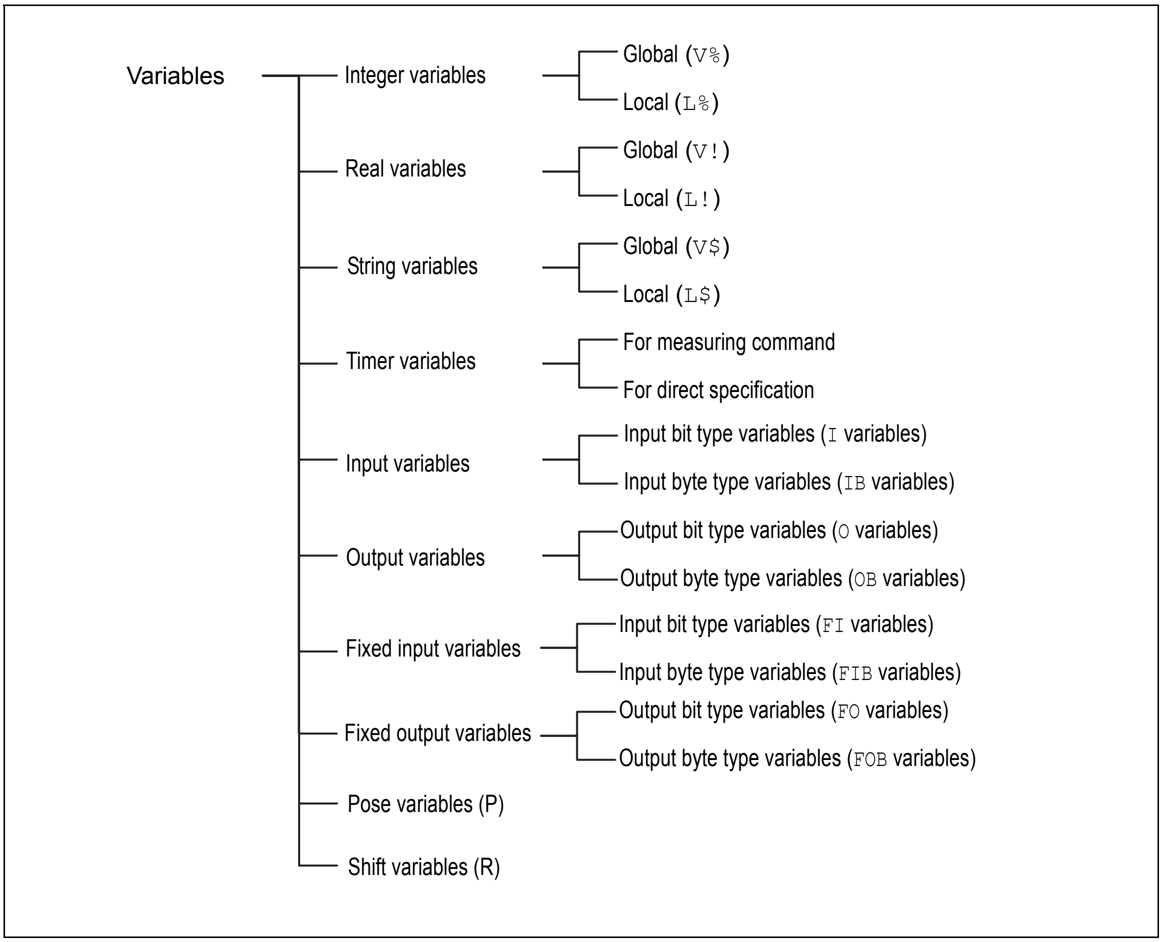

Variables
Integer variables
Global (V%)
Local (L%)
Real variables
Global (V!)
Local (L!)
String variables
Global (V$)
Local (L$)
Timer variables
For measuring command
For direct specification
Input variables
Input bit type variables (I variables)
Input byte type variables (IB variables)
Output variables
Output bit type variables (O variables)
Output byte type variables (OB variables)
Fixed input variables
Input bit type variables (FI variables)
Input byte type variables (FIB variables)
Fixed output variables
Output bit type variables (FO variables)
Output byte type variables (FOB variables)
Pose variables (P)
Shift variables (R)

Fig. 2.5.1 Types of variables

Table 2.7 Global variables and local variables

Global variables	These can be referenced from any unit or user task. They start from V.
Local variable	These are allocated to individual units and user tasks, and they cannot be referenced from other units. They start from L.

2.5.1 Integer variables

The "Integer variables" are used to handle values with no decimal point included.

Table 2.8 Integer variables

Format	`Vn%, V%[n]` n=1 to 250, 301 to 500 (Variables can be also used.) `Ln%, L%[n]` n=1 to 200, 301 to 500(Variables can be also used.) `V1%` and `V%[1]` denote the same variable area.
Range	`−2147483648 to +2147483647`

Sample	`V1%=V2%+1`
	`L%[1]=10`
	`CALLN 10, V1%, 10`
	`JMPN 10, V1%, 20`

Storage	All global integer variables are retained as power failure retention data even if the main power supply is turned OFF. In contrast, all local integer variables are not retained.

POINT

• The global integer variables 201 to 250 are used to execute passing of PLC's internal integer variables and integer values. Writing values in global variables will write their corresponding values in the PLC's internal integer variables. Furthermore, reading the said global variables will read the value of the PLC's internal integer variables. Table 2.9 below shows correspondence of global integer variables to PLC's internal integer variables.

• Writing values in the global integer variables 201 to 250 aforementioned will cause delay until the changes affect the PLC's internal integer variables. In contrast, refer to "Precautions for using global integer variables V201% to V250%".

Table 2.9 Correspondence of global integer variables to PLC's internal integer variables

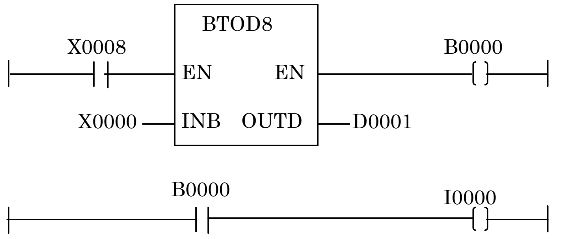
Global integer variable number	PLC's internal integer variable number
`V201% or V%[201]`	`D0001`
`V202% or V%[202]`	`D0002`
`V203% or V%[203]`	`D0003`
. . .	. . .
`V248% or V%[248]`	`D0048`
`V249% or V%[249]`	`D0049`
`V250% or V%[250]`	`D0050`

Precautions for using global integer variables V201% to V250%

The global integer variables `V201%` to `V250%` are used to read/write integer variables out of the PLC's internal variables when reading/writing values. Therefore, the global integer variables are not available to advance execution. Furthermore, delay occurs after values are set to the global integer variables `V201%` to `V250%` until the set values affect the PLC's internal integer variables. In contrast, if changes are made to the PLC's internal integer variables, delay will occur before the changes affect the global integer variables `V201%` to `V250%`. Furthermore, the delay duration varies with the robot configuration, usage, and others.
To prevent the influence of the delay aforementioned, be sure to provide interlock between the PLC and the robot language according to the procedure shown below.
For details of PLC's internal variables, refer to the instruction manual "Software PLC".

The following section shows the interlocking procedure for passing the PLC's internal variables with the use of the global integer variables `V201%` to `V250%`. Furthermore, note that the interlocking procedure varies with reading or writing of the internal variables.

<<For reading of PLC's internal variables>>



```
WAIT (I1)
V1%=V201%
END
```

Fig. 2.5.2 Example of use of global integer variables 201 to 250

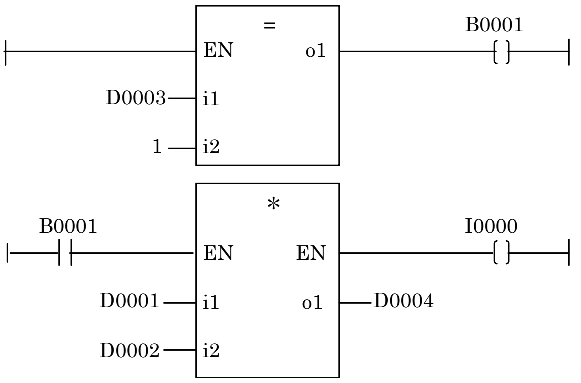
The above figure shows the case where eight bits of this controller's standard inputs `X0000` to `X0007` are assigned to the global integer variable V1% as integer values.

In the above figure, external signals are input in the order presented below.

(1) Input integer values set with this controller to `X0000` to `X0007`.
(2) Turn ON `X0008` indicating that the said inputs have been determined.

In the above figure, since the external signals are input in the order of `X0000` to `X0007` and then `X0008`, `D0001` has been determined at the time when `I1` (`I0000` for PLC) turns ON. Consequently, interlock between the PLC and the robot language is provided through turning ON the `I1`.

<<For writing in PLC's internal variables>>



```
V201%=V1%
V202%=V2%
V203%=1
*WAIT_PLC
WAITI(I1)
V203%=0
END
```

Fig. 2.5.3 Example of use of global real number variables 201 to 250

The above figure shows the case where the global integer variables `V1%` and `V2%` are stored in the PLC's internal integer variables `D0001` and `D0002` respectively, and then the product of those variables are stored in the PLC's internal integer variable `D0004`.

In the above figure, since the order in which values are set to `V201%`, `V202%` and `V203%` will never be reversed, set "1" to V203% (D0003 for PLC). The contents of `D0001` and `D0002` have been determined at the time when the `I1` (`I0000` for PLC) turns ON. Consequently, interlock between the PLC and the robot language is providing through setting "1" to `V203%` and turning ON the `I1`.

2.5.2 Real number variables

The "Real number variables" are used to handle values with decimal point included.

Table 2.10 Real number variables

Format	`Vn!, V![n]` n=1 to 250, 301 to 500 (Variables can be also used.) `Ln!, L![n]` n=1 to 200, 301 to 500 (Variables can be also used.) `V1!` and `V![1]` denote the same variable area.
Range	−1.0E38 to +1.0E38

Sample	`V1!=V2!*103.45`
	`L![1]=1.567E-2`
	`GETANGLE V1!`

Storage	All global integer variables are retained as power failure retention data even if the main power supply is turned OFF. In contrast, all local integer variables are not retained.

If the real number variables are integers, the calculation results may become integers. For robot language calculations, follow the types of variables on the right-hand side not those of variables on the left-hand side. As a specific example, refer to the calculation results from
"Table 2.11 Examples of calculations with mixed variables of real numbers and integers (1) " and
"Table 2.12 Examples of calculations with mixed variables of real numbers and integers (2)".

Table 2.11 Examples of calculations with mixed variables of real numbers and integers (1)

Format	V42! result and display	Expected value	Remark
V42!=152/10	15	15.2	−
V42!=152.0/10	15	15.2	"152.0" is handled as an integer.
V42!=152/10.0	15	15.2	"10.0" is handled as an integer.
V42!=152.0/10.0	15	15.2	The numerator and denominator are both handled as an integer.
V42!=152/10+ V41!	25	25.2	V41!=10 or V41!=10.0
V42!=152/10+ V41!	25.1	25.3	V41!=10.1
V42!=152/10+10	25	25.2	−
V42!=152/10+10.0	25	25.2	−
V42!=152/10+10.1	25.1	25.3	−

The table above lists examples showing that the divisions produce results in integers.
These calculations do not produce results as expected.

Table 2.12 Examples of calculations with mixed variables of real numbers and integers (2)

Format	V42! result and display	Expected value	Remarks
V42!=V41!/10	15.2	15.2	V41!=152
V42!=V41!/10	15.2	15.2	V41!=152.0
V42!=V41!/10	15.21	15.21	V41!=152.1
V42!=152*0.1	15.2	15.2	Example of an alternate solution to V42! 152/10

The table above lists examples showing that the divisions produce results in real numbers.
These calculations produce results as expected.

POINT

• The global real number variables 201 to 250 are used to execute passing of PLC's internal integer variables and real number values. Writing values in global variables aforementioned will write their corresponding values in the PLC's internal real number variables. Furthermore, reading the said global variables will read the value of the PLC's internal real number variables. The table below shows correspondence of global real number variables to PLC's internal real number variables.

• Writing values in the global real number variables 201 to 250 aforementioned will cause delay until the changes affect the PLC's internal real number variables.

IMPORTANT

For precautions for using global real number variables V201! to V250!, refer to [Precautions for using global integer variables V201% to V250%].

Table 2.13 Correspondence of global real number variables to PLC's internal real number variables

Global real number variable number	PLC's internal real number variable number
V201! or V![201]	R0001
V202! or V![202]	R0002
V203! or V![203]	R0003
. . .	. . .
V248! or V![248]	R0048
V249! or V![249]	R0049
V250! or V![250]	R0050

2.5.3 Character string variables

"Character string variables" handle character strings. ASCII and Shift JIS are handled. The 2-byte codes specified by shift JIS can also be handled also.

Table 2.14 Character string variables

Format	Vn$, V$[n] n=1 to 50 (Variables can be also used.) Ln$, L$[n] n=1 to 50 (Variables can be also used.) V1$ and V$[1] denote the same variable area.
Range	0 to 199 characters (199 bytes)

Sample	V1$="Execution state"
	L$[1]=V1$+"Under suspension"
	LETVS V1$, "1A"

Storage	The first ten global character string variables are retained as power failure retention data even if the main power supply is turned OFF. In contrast, all local character string variables are not retained.

2.5.4 Timer variables

Timer variables are not supported at the present time.

2.5.5 Input signal variables

These handle the input ports in bit or group (1 group = 10 bits) units.

Table 2.15 Input signal variables

	When the ports are handled in bits	When the ports are handled in group

Range	`In, I[n]`
	`n=1 to 2048, 5101 to 5196`
	(variables can be used)
		`IBn, IB[n]`
		`n=1 to 205`
		(variables can be used)

Range	`0,1`	`0 to 1023`
		`(In case of IB[205], 0 to 255)`

Sample	`WAIT I[1],2,100` The robot will wait until the `I1` signal turns ON. If the signal turns ON, the next command will be executed. If the signal does not turn ON within 2 seconds, the robot will jump to the step 100. Waiting time : 2 sec Shelter step : 100	`WAITAD IB[1],255,2,100` The robot will wait until the binary value of the Group 1 signals turn to 255. If the condition is satisfied, the next command will be executed. If the condition is not satisfied within 2 seconds, the robot will jump to the step 100. Waiting time : 2 sec Shelter step : 100

The input signal variables are referenced only: they cannot be written.
The numbers of the groups in the table correspond to "`n`" of `IB[n]`.

Table 2.16 Group numbers of input signals

Group	Input signal	Group	Input signal	Group	Input signal
1	1 to 10	11	101 to 110	21	201 to 210
2	11 to 20	12	111 to 120	…	…
3	21 to 30	13	121 to 130	30	291 to 300
4	31 to 40	14	131 to 140	…	…
5	41 to 50	15	141 to 150	50	491 to 500
6	51 to 60	16	151 to 160	…	…
7	61 to 70	17	161 to 170	100	991 to 1000
8	71 to 80	18	171 to 180	…	…
9	81 to 90	19	181 to 190	…	…
10	91 to 100	20	191 to 200	204	2031 to 2040
				205	2041 to 2048

(Example 1)

```
IF I1=1 THEN *FINISH ELSE *REPEAT
```

A step jump to the label "`*FINISH`" will be executed if the input signal `I1` is `1` (=ON).

(Example 2)

```
SWITCH IB[1]
CASE 1
GOTO *LABEL1
BREAK
CASE 2
GOTO *LABEL2
BREAK
CASE
GOTO *LABEL3
BREAK
ENDS
```

The jump destination is selected by referring to the input signals. If only `I1` is ON, the destination is "`*LABEL1`". If only `I2` is ON, the destination is "`*LABEL2`". And in other cases, the destination is "`*LABEL3`".

2.5.6 Output signal variables

These variables handle the output ports in bit or group (1 group = 10 bits) units.

The output signals cannot be operated by assigning them directly to the output signal variables. These variables are used as parameters of the `SET`, `OUT`, `SETM` etc. and other function commands.

Table 2.17 Output signal variables

	When the ports are handled in bits	When the ports are handled in 10 bits
Range	`On, O[n]` `n=1 to 2048` (variables can be used)	`OBn, OB[n]` `n=1 to 205` (variables can be used)

Range	`0,1`	`0 to 1023`
		`(In case of OB[205], 0 to 255)`

Examples	`SET O1` or `SET O[1]`
	The output signal `O1` turns ON.
	`RESET O2` or `RESET O[2]`
	The output signal `O2` turns OFF.
		`OUT OB1, 1023`
		-> The all of `O1`～`O10` will be turned ON. (excluding the assigned signals)
		`OUT OB205, 0`
		-> The all of `O2041`～`O2048` will be turned OFF. (excluding the assigned signals)

. The number of the group in the table corresponds to "`n`" of `OB[n]`.

Table 2.18 Group numbers of output signals

Group	Output signal	Group	Output signal	Group	Output signal
1	1 to 10	11	101 to 110	21	201 to 210
2	11 to 20	12	111 to 120	…	…
3	21 to 30	13	121 to 130	30	291 to 300
4	31 to 40	14	131 to 140	…	…
5	41 to 50	15	141 to 150	50	491 to 500
6	51 to 60	16	151 to 160	…	…
7	61 to 70	17	161 to 170	100	991 to 1000
8	71 to 80	18	171 to 180	…	…
9	81 to 90	19	181 to 190	…	…
10	91 to 100	20	191 to 200	204	2031 to 2040
				205	2041 to 2048

(Example)

```
IF O1=1 THEN *FINISH ELSE *REPEAT
```

A step jump to the label "`*FINISH`" will be executed if the output signal `O1` is `1` (=ON).

2.5.7 Fixed input signal variables

These variables handle the fixed input ports in bit or group (1 group = 1 bytes = 8 bits) units.

Table 2.19 Fixed input signal variables

	When the ports are handled in bits	When the ports are handled in 8 bits
Range	`FIn, FI[n]` `n=1～48` (variables can be used)	`FIBn, FIB[n]` `n=1～6` (variables can be used)
Range	`0,1`	`0～255`
Examples	`V1% = FI1`	`V1% = FIB1`

The fixed input signal variables are read only. Writing operation is impossible.

Table 2.20 Fixed input signals

No.	Signal name	Description
1	Motors-ON	Lit with Motor ON of the operating box. (operation panel)
2	G-STOP	Lit off when G-STOP input is OFF. For details of G-STOP signal, refer to the section 3.7 in “SETUP MANUAL”.
3	Start 1	Lit when the start 1 button of the operation box (panel) is ON.
4	Start 2	Lit when the start 2 button of the starting box 2 is ON.
5	Start 3	Lit when the start 3 button of the starting box 4 is ON.
6	Start 4	Lit when the start 4 button of the starting box 5 is ON.
7	Stop	Normally, this light is ON. It lights off when any stop buttons of the teach pendant, the operation box or the start box are pressed ON.
8	Playback mode	Lit when the selector switch of operation box is in Playback mode with the external mode select input OFF. It lights off when the selector switch of operation box is in Teach mode.
9	Mat switch	Lit when the enable switch and MAT switches (TBEX1: 13-14, 15-16) are ON.
10	-	–
11	High-speed Teach	Lit when High-speed Teach is ON.
12	P1 Normal	Lit when DC 24 V is supplied properly.
13	Ext Emergency stop	Lit off with the “emergency stop” off or when the external emergency stop input (TBEX1: 1-2, 3-4) is OFF (on inputting the emergency stop) is OFF. Lit by releasing the emergency stop input.
14	Emergency stop	Lit off with the “TP emergency stop” off or when the emergency stop button of either the operation box or the start box is depressed (on inputting the emergency stop). Lit by releasing the entire emergency stop input.
15	Safety plug	Lit when all the following conditions are met. ･”Playback mode” lights. ･Safety plug input (TBEX1_5-6,7-8) is ON.
16	Confirm motors-ON	Lit with the motor power relays (CRON1 & CRON2) are ON
17	TP Emergency stop	Lit by releasing the emergency stop input. Lit off with the TP emergency stop button depressed (on inputting the emergency stop).
18	Teach mode	Lit when “TEACH mode” is selected by the selector switch of operation box. Light off during “Auto mode”,
19	-	–
20	TP enable SW	Lit when [ENABLE SWITCH] (Deadman switch) is ON.
21	-	–
22	CRON	Lit with the motor power relays (CRON1 and CRON2) are ON.
23	Servo-ON	Lit when the servo ON relay (CRES1) is ON.
24	Servo enable	Lit when Amplifier normal signal is ON.

No.	Signal name	Description
25	−	−
26	−	−
27	−	−
28	Magnet-ON	Lit when the magnet switches (MS1 and MS2) are ON.
29	−	−
30	Weld detection	Normally, it lights on with the motor power OFF. Lit off when contact stick occurs in the magnet switch or the relay with the motor power OFF.
31	Inconsistency	Normally, it lights off. Lit when the inconsistency of a redundant safety circuit is detected.
32	−	-
33	Inconsist (GSTOP)	Normally, it lights. Lit off when the inconsistency of a G-STOP redundant safety circuit is detected.
34	Inconsist(mode)	Normally, it lights. Lit off when the inconsistency of a mode input redundant safety circuit is detected.
35	Inconsist(MAT-SW)	Normally, it lights. Lit off when the inconsistency of a MAT-SW redundant safety circuit is detected.
36	Inconsist(HI-SP)	Normally, it lights. Lit off when the inconsistency of a high-speed teaching redundant safety circuit is detected.
37	Inconsist(Ext ES)	Normally, it lights. Lit off when the inconsistency of an extension emergency redundant safety circuit is detected.
38	Inconsist(E.S.)	Normally, it lights. Lit off when the inconsistency of an emergency redundant safety circuit is detected.
39	Inconsist(S.plug)	Normally, it lights. Lit off when the inconsistency of a safety plug redundant safety circuit is detected.
40	Inconsist(TP-ES)	Normally, it lights. Lit off when the inconsistency of a TP emergency redundant safety circuit is detected.
41	Inconsist(ENB-SW)	Normally, it lights. Lit off when the inconsistency of an enable switch redundant safety circuit is detected.
42	Inconsist(CRON)	Normally, it lights. Lit off when the inconsistency of a CRON redundant safety circuit is detected.
43	−	−
44	−	−
45	−	−
46	−	−
47	−	−
48	−	−

The number of the group in the table corresponds to `"n"` of `FIB[n]`.

Table 2.21 Group numbers of fixed input signals

Group No.	Fixed input signals
1	1～8
2	9～16
3	17～24
4	25～32
5	33～40
6	41～48

INFO.

The conditions of the fixed input signals can be checked in the following menu (monitor window).
<Service Utilities> - [3 Monitor1] [5 Fixed Inputs]

2.5.8 Fixed output signal variables

These variables handle the fixed output ports in bit or group (1 group = 1 bytes = 8 bits) units.

Table 2.22 Fixed output signal variables

	When the ports are handled in bits	When the ports are handled in 8 bits
Range	`FOn, FO[n]` `n=1～24` (variables can be used)	`FOBn, FOB[n]` `n=1～3` (variables can be used)
Range	`0,1`	`0～255`
Examples	`V1% = FO1`	`V1% = FOB1`

The fixed output signal variables are read only. Writing operation is impossible.

Table 2.23 Fixed output signals

No.	Signal name	Description
1	Motors-ON lamp	Light off : Motor power OFF, Light on : Motor power ON (Light on : The status that the motor power can be turned ON by turning enable switch ON.)
2	Motors-ON request	Lit when the Motors-ON button of the operation box 1 is ON.
3	Start lamp 1	With “Multi-station” start method Lit with starting the station 1. Flushes with the station 1 in a queue.
4	Start lamp 2	Lit with starting the station 2. Flushes with the station 2 in a queue.
5	Start lamp 3	Lit with starting the station 3. Flushes with the station 3 in a queue.
6	Start lamp 4	Lit with starting the station 4. Flushes with the station 4 in a queue.
7	Stop lamp	Lit during temporary stop. While the robot is running, this lamp turns to OFF. (While moving the robot with axis operation, this lamp keeps ON)
8	TP enable release	Lit when enable switch is released.
9	Motors-ON enable	Lit when the motor ON has been applied and allowed by software. Lit off if the servo error occurs, when the motor ON has not been allowed by software or CPU error occurs.
10	Magnet-ON enable	Lit with the servo ON allowed by software.
11	Internal/External	Lit when the start method is specified to “External”. Lit off when the start method is specified to “Controller” or “Multi-station”.
12	WPS E-STOP ctrl	Lit when outputting the software enable signals to the welding power supply. This is one of the required conditions to output the emergency stop signals to the welding power supply.
13	CPU failure	Always lighting off. CPU trouble is indicated only by the hardware output. Check the output status with TBEX2: CPU error output.
14	TP mode	Lit when TP selector switch is set to “TEACH”. Lit off when “AUTO” is selected.
15	Ext motors-ON	Lit when external motors-ON is input.
16	−	−
17-24	−	−

The number of the group in the table corresponds to `"n"` of `FOB[n]`.

Table 2.24 Group numbers of fixed output signals

Group No.	Fixed output signals
1	1～8
2	9～16
3	17～24

2.5.9 Pose variable

The "Pose variables" hold the robot poses.
Normally, to use the pose variables, it is necessary to load a pose file before using them. But it is also possible to generate pose variables in a robot language program like the following using `(X, Y, Z, roll, pitch, yaw)` format.

```
P1 = (1800, 0, 2000, 0, -90, -180)
```

If there are additional axes like e.g. servo gun etc, write the joint value at the end.

```
P1 = (1800, 0, 2000, 0, -90, -180, -20)
```

Table 2.25 Pose variable

Format	`Pn, P[n]` n=1 to 999 (variables can be used)
Examples	`P1=(100,0,100,0,0,90)`

(Example 1)
Mechanism 1 : A robot with 6-axes
Mechanism 2 : None

```
m
USE 1
P1 = (1800,0,2000,0,-90,-180)
MOVEX A=1,M1X,P,P1,R=10.0,H=1,MS
END
```

In this example, the number of the elements in the pose variable `P1` is 6.

(Example 2)
Mechanism 1 : A robot with 6-axes
Mechanism 2 : Servo gun (1 axis)

```
USE 1
P1 = (1800,0,2000,0,-90,-180,-20)
MOVEX A=1,M1X,P,P1,R=10.0,H=1,MS,M2X,P,P1,R= 10,H=1
END
```

In this example, because the total number of the axes is 7, the number of the parameters of pose variable is 7. Please be sure that the pose variable is used for both of mechanism 1 and 2. The mechanism 1 gets the first 6 values and the mechanism 2 gets the 7th data.

POINT

Pose variables are not kept when the main power is lost.
Even when number is set to pose variable, pose file is not renewed.
By executing function `FN74 POSESAVE`, pose variable can be copied to pose file.

POINT

- For the pose file creation, pose variable recording and modification, refer to "3.3 Creating pose files"

- To use a pose variable in a robot language program, please load the pose file using `USE` command in advance. If the file is not loaded, an error message will be displayed when using a pose variable because the pose variable cannot be accessed.

- The pose variables that are not saved into the pose file will be lost when turning OFF the controller power. Because those variables are not included in the data area for power failure detection function.

- Even if values are set to a pose variable in a robot language program, the pose file to which the pose variable belongs will not be modified automatically. To over-write the pose file, please use "`FN74 POSSAVE`".

POINT

When trying to move a robot using pose variables, please pay special attention. If the setting values (coordinates) are wrong, the robot may make unexpected and dangerous motion. Before moving a real robot, we strongly recommend you to check the motion of the program using an offline simulator "FD on Desk".

2.5.10 Shift variable

The "Shift variables" hold the shift amounts. This is also called as "Shift register".
This variable uses `MOVEX-X` representation `(X, Y, Z, roll, pitch, yaw).`
External axis (additional axis) is not supported.

Table 2.26 Shift variable

Format	`Rn, R[n]` n=1 to 9 (variables can be used)
Examples	`R1=(10,1,0,0,0,0)`

2.6 User variables

In ***"User variables" (Any variables)***, it is possible to put a name for a variable. And, it is also possible to make an array of the variable.

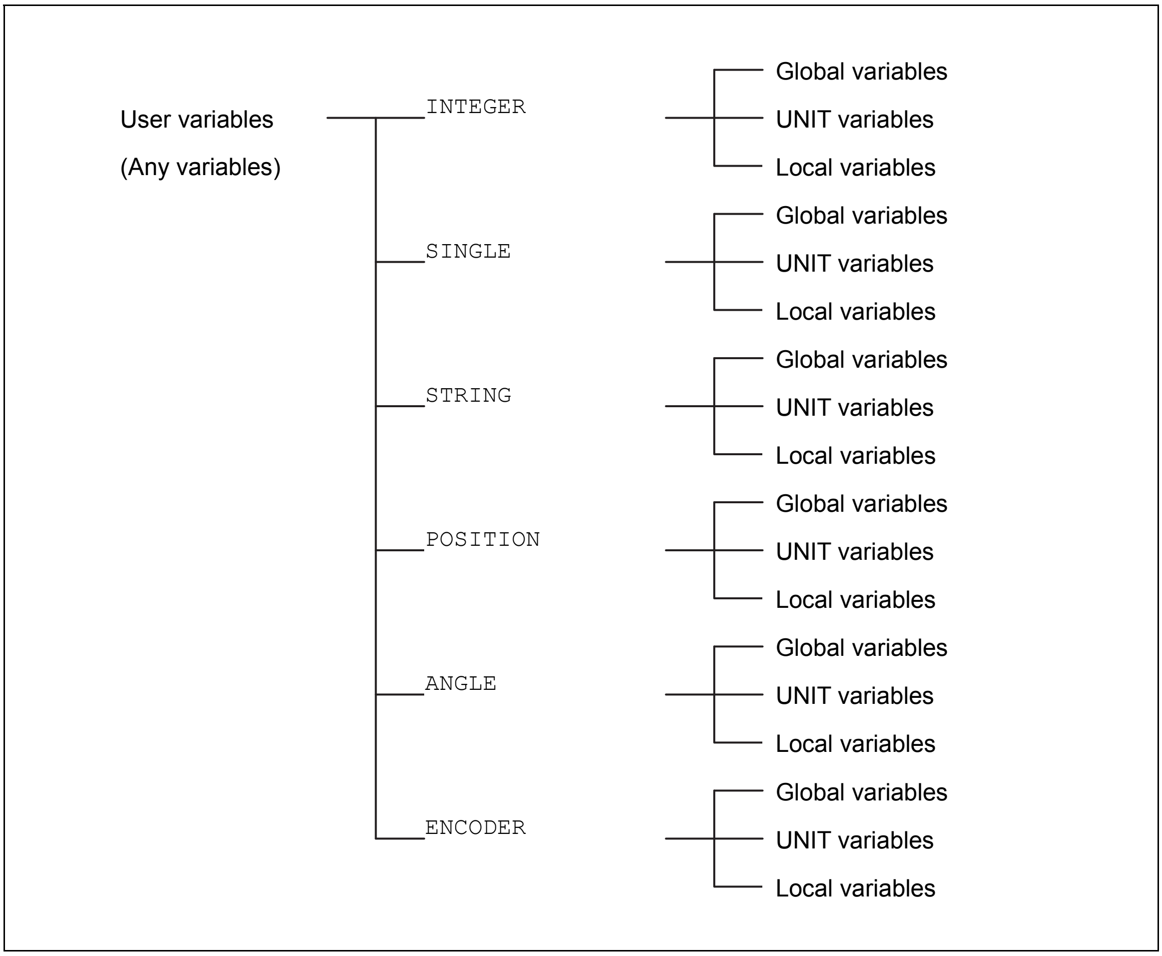


Fig. 2.6.1 User variables

Table 2.27 Global variables / UNIT variables / Local variables

Global variables	Global variables can be accessed from the all UNITs. Global variables can be defined in the "`GLOBAL`" section in the "`Public.inc`".
UNIT variables	UNIT variables can be read from the all UNITs. But the data save (write) is possible only from each UNIT itself. Global variables can be defined in the "UNIT* " section in the "`Public.inc`". `("*" is the UNIT number)`
Local variables	Local variables are variables that are defined in a program (including User procedure) and cannot be accessed from the other programs or other UNITs.

POINT

The respective contents of the variables are saved into the following files.

Global variables and UNIT variables : `PUBLIC.DAT`
Local variables : `LOCAL-**.DAT` ("**" is UNIT No.)

(NOTE) The "`LOCAL-**.DAT`" will be deleted after executing `END` command or selecting a work-program.

2.6.1 How to define user variables

For example, user variables can be defined like the followings;

```
DIM intData[10] AS INTEGER
DIM angHOME AS ANGLE
```

The details are shown as below.

Table 2.28 Definition of user variables

Format	`DIM` *`VariableName`* `([`*`Array`*`])` `AS` *`VariableType`* (NOTE) When not using arrays, the under line part can be omitted.
VariableName	Up to 20 letters can be used for the variable's name. ☞See"Table 2.29 Available letters for the variable's name" The first letter must be alphabet. The capital letters and the small letters are distinguished. The reserved name of the system or the same name with other variables cannot be used.
Array	Up to 3 dimensions is possible. (But, in case of `POSITION`, `ANGLE`, `ENCODER`, up to 2 dimensions.) The element of the arrays will start from "1". The number of the elements is up to 256. ☞See "Table 2.30 Declaration of array"
VariableType	`INTEGER` integer value (☞"2.6.2 INTEGER") `SINGLE` real value (☞"2.6.3 SINGLE") `STRING` character strings (☞"2.6.4 STRING") `POSITION` position value (☞"2.6.5 POSITION") `ANGLE` joint value (☞"2.6.6 ANGLE") `ENCODER` encoder value (☞"2.6.7 ENCODER")

Table 2.29 Available letters for the variable's name

Item	Available letters
Alphabets and numbers	ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 01234567890
Symbol	_

Table 2.30 Declaration of array

Array	Example
1 dimension	`DIM intData[6] AS INTEGER`
2 dimensions	`DIM intData[3, 6] AS INTEGER`
3 dimensions	`DIM intData[3, 3, 6] AS INTEGER`

POINT

When defining a user variable, it is recommended to use a name that can make it easy to understand the meaning and the type of the variable. For example, for a variable to keep the Home position as `ANGLE` type data, the name should be like `"angHOME"`.

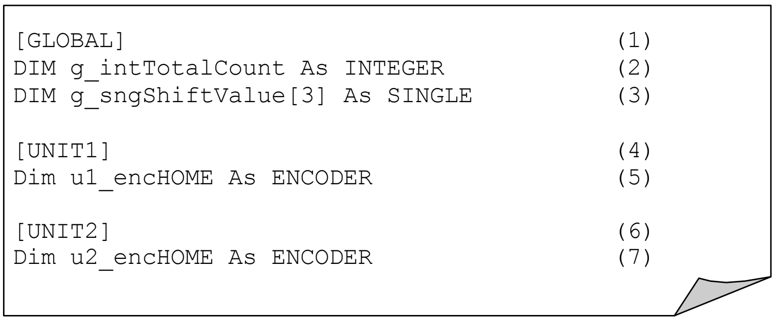
POINT

The following commands do not support the user variables.
Please use constants, integer variables(V%), real variables (V!) etc.

Command	Name
JUMP	FN20 Step jump
CALL	FN21 Step call
ON	FN603 ON GOTO jump
PRINT	FN606 PRINT
MOVE	Movement command

Definition of a user variable (Global variables / UNIT variables)

To use the global variables and the UNIT variables, please prepare a file whose name is **"Public.inc"** And place it to the folder of **"Work¥Program¥"** in the internal memory of the controller.



```
[GLOBAL]                                        (1)
DIM g_intTotalCount As INTEGER                  (2)
DIM g_sngShiftValue[3] As SINGLE                (3)

[UNIT1]                                         (4)
Dim u1_encHOME As ENCODER                       (5)

[UNIT2]                                         (6)
Dim u2_encHOME As ENCODER                       (7)
```

(1) Start the section for the global variables
(2) Declaration of a global variable
(3) Declaration of a global variable
(4) Start the section for the UNIT1 variables
(5) Declaration of a variable for UNIT1
(6) Start the section for the UNIT2 variables
(7) Declaration of a variable for UNIT2

Fig. 2.6.2 Definition file "Public.inc"

POINT

- Before starting the declaration of the user variables, describe the [GLOBAL] or [UNIT*]. The [GLOBAL] is a section start of global variables and the [UNIT*] is a section start of UNIT variables.

- Line feed code cannot be inserted in 1 line.

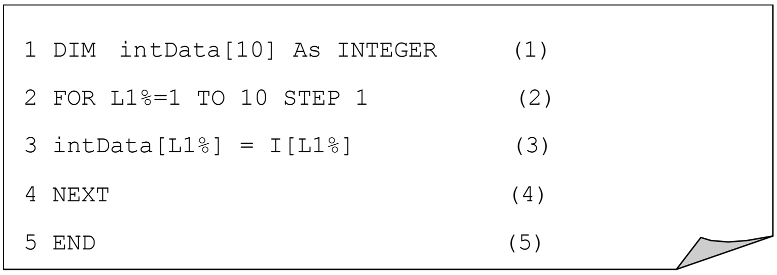
- When the user variable definition file is copied or edited in the ASCII file editor, turn OFF the controller power and then turn ON it again to initialize the data memory. When turning ON the power, the syntax of the ***"Public.inc"*** will be checked and then the memory for the variables will be allocated and the ***"PUBLIC.DAT"*** will be created. If there are syntax errors, error message will be displayed.

CAUTION

The data values of the defined user variables are stored in the data file of "PUBLIC.DAT". If the variable name is changed or the number of arrays increases, the value of the variable will be cleared to initial value (0).

Definition of a user variable (Local variables)

To use the local variables, please write a declaration in the robot language program like the following.



```
1 DIM  intData[10] As INTEGER      (1)
2 FOR L1%=1 TO 10 STEP 1            (2)
3 intData[L1%] = I[L1%]            (3)
4 NEXT                             (4)
5 END                              (5)
```

(1) Declaration of local variable	A user variable is created
(2) Loop orocess (start)	
(3) Set the input signal condition to the local variable	The data is set to the variable
(4) Loop process (end)	
(5) Program end	The user variables are deleted

Fig. 2.6.3 Local variables

POINT

- The same name cannot be definedin the identical program.
- The same name with the global variables or UNIT variables cannot be defined.
- The same name with the reserved words cannot be defined.
- To use the local user variables in a program, both of the definition and the initialization (data setting) must be done in advance. (Just after definition, the value is random value.)
- The suffix value of an array starts from "1". (not "0")
- The content of local user variables are stored in a data file ("LOCAL-**.DAT"). But this file will be deleted when executing END command or selecting (opening) a work program.

CAUTION

The data values of the (local) user variables are stored in the data file "LOCAL-**.DAT". When changing the variable's name or increasing the number of arrays, re-select the work program or re-execute the definition command.

2.6.2 INTEGER type variable

This type handles integers.

Table 2.31 INTEGER variable

Format	DIM *VariableName*([*Array*]) AS **INTEGER** (NOTE) When not using arrays, the under line part can be omitted.
Array	Up to 3 dimensions is possible.
Range	−2147483648～+2147483647

Sample	Get the signal condition from `I1` to `I10`.
	
	`DIM intData[10] AS INTEGER`
	`FOR L1%=1 TO 10 STEP 1`
	` intData[L1%] = I[L1%]`
	`NEXT`

2.6.3 SINGLE type variable

This type handles real numbers.

Table 2.32 SINGLE variable

Format	DIM *VariableName*([*Array*]) AS **SINGLE** (NOTE) When not using arrays, the under line part can be omitted.
Array	Up to 3 dimensions is possible.
Range	−1.0E38～+1.0E38

Sample	Get the TCP speed (command / current) of the mechanism 1.
	
	`DIM sngTCPSpeed[2] AS SINGLE`
	`sngTCPSpeed[1] = SYSTEM!(300)  'TCP speed (command)`
	`sngTCPSpeed[2] = SYSTEM!(800)  'TCP speed (current)`

2.6.4 STRING type variable

This type handles character string.

Only ASCII code and SHIFT JIS code (Japanese code) are supported. 2 bytes code of SHIFT JIS is also available.

Table 2.33　STRING variable

Format	DIM *VariableName*([*Array*]) AS **STRING** (NOTE) When not using arrays, the under line part can be omitted.
Array	Up to 3 dimensions is possible.
Range	From 0 to 199 characters (199 bytes)

Sample	Set the output message to the RS232C port.
	
	`DIM strOutMessage AS STRING`
	`strOutMessage = “SHIFT 1¥n¥r”`

2.6.5 POSITION type variable

This type handles position data in a format of `(X, Y, Z, ROLL, PITCH, YAW, CONF)`

Table 2.34 POSITION variable

Format	`DIM` *`VariableName`*`([`*`Array`*`])` `AS` **`POSITION`** (NOTE) When not using arrays, the under line part can be omitted.
Array	Up to 2 dimensions is possible.
Range	－1.0E38～＋1.0E38

Sample	Get the current position and move the robot via the input signals.
	`I1 is +X / I3 is +Y / I5 is +Z`
	`I2 is -X / I4 is -Y / I6 is -Z`
	
	`DIM posCurrent AS POSITION`
	`DIM sgnPosition[7] AS SINGLE                '(X,Y,Z,Roll,Pitch,Yaw,CONF)`
	`USE 1`
	`GETPOS 0,1,posCurrent,0,1`
	`OPEPOS 0,posCurrent,sgnPosition[1],0,0`
	
	`WHILE(I10)`
	` IF (I1)                                    '+X`
	`  sgnPosition[1] = sgnPosition[1] + 2.0`
	` ELSEIF (I2)                                '-X`
	`  sgnPosition[1] = sgnPosition[1] - 2.0`
	` ELSEIF (I3)                                '+Y`
	`  sgnPosition[2] = sgnPosition[2] + 2.0`
	` ELSEIF (I4)                                '-Y`
	`  sgnPosition[2] = sgnPosition[2] - 2.0`
	` ELSEIF (I5)                                '+Z`
	`  sgnPosition[3] = sgnPosition[3] + 2.0`
	` ELSEIF (I6)                                '-Z`
	`  sgnPosition[3] = sgnPosition[3] - 2.0`
	` ENDIF`
	` OPEPOS 0,posCurrent,sgnPosition[1],0,1`
	` POS2POSE 0,posCurrent,0,1,P1`
	` MOVEX A=1,AC=0,SM=0,M1J,P,P1,R= 10,H=1,MS   'MOVE to P1`
	`ENDW`
	
	`END`

POINT

- In case of a servo gun, slider, positioner etc. the data is regarded as a set of joint values.
- In case of rotational axis, the unit is [DEG]. In case of linear axis, the unit is [mm].
- To set a data to the `POSITION` type variable, please use exclusive function commands.
- For example, in a case of muti-mechanism specification like the following, one `POSITION` variable holds 9 elements.

M1: 6-axes robot
M2: 1 axis Servo Gun
M3: 1 axis Slider

```
(M1:X, M1:Y, M1:Z, M1:ROLL, M1:PITCH, M1:YAW, M1:CONF, M2:J1, M3:T1)
```

2.6.6 ANGLE type variable

This type handles position data in a format of `(J1, J2, J3, J4, J5, J6).`

Table 2.35 ANGLE variable

Format	`DIM` *`Variable's name`* `([`*`Array`*`])` `AS` **`ANGLE`** (NOTE) When not using arrays, the under line part can be omitted.
Array	Up to 2 dimensions is possible.
Range	`－1.0E38～＋1.0E38`

Sample	Get the current position and move the robot via the input signals.
	
	`DIM angCurrent AS ANGLE`
	`DIM sgnPosition[6] AS SINGLE`
	
	`GETANG 0,1, angCurrent`
	`OPEANG 0,posCurrent,sgnPosition[1],0,0`
	
	`WHILE(I10)`
	`  IF (I1)                                           'J1`
	`   sgnPosition[1] = sgnPosition[1] + 1.0`
	`  ELSEIF (I2)                                       'J2`
	`   sgnPosition[2] = sgnPosition[2] + 1.0`
	`  ELSEIF (I3)                                       'J3`
	`   sgnPosition[3] = sgnPosition[3] + 1.0`
	`  ELSEIF (I4)                                       'J4`
	`   sgnPosition[4] = sgnPosition[4] + 1.0`
	`  ELSEIF (I5)                                       'J5`
	`   sgnPosition[5] = sgnPosition[5] + 1.0`
	`  ELSEIF (I6)                                       'J6`
	`   sgnPosition[6] = sgnPosition[6] + 1.0`
	`  ENDIF`
	`  OPEPOSE 0,P1,sgnPosition[1],0,1`
	`  MOVEX A=1,AC=0,M1X,P,P1,R= 10.0,H=1,MS            'MOVE to P1`
	`ENDW`
	
	`END`

POINT

- In case of rotational axis, the unit is [DEG]. In case of linear axis, the unit is [mm].
- To set a data to the `ANGLE` type variable, please use exclusive function commands.

2.6.7 ENCODER type variable

This type handles encoder value data in a format of `(J1, J2, J3, J4, J5, J6).`

Table 2.36 ENCODER variable

<table>
<tr><td>Format</td><td>DIM <i>VariableName</i>([<i>Array</i>]) AS <b>ENCODER</b>
(NOTE) When not using arrays, the under line part can be omitted.</td></tr>
<tr><td>Array</td><td>Up to 2 dimensions is possible.</td></tr>
<tr><td>Range</td><td>From 0x000100 to 0x0ffff00</td></tr>
<tr><td>Sample</td><td>Get the current position and move the robot via the input signals.

```
DIM encCurrent AS ENCODER
DIM intPosition[6] AS INTEGER

GETENC 0,1, encCurrent
OPEENC 0,encCurrent,intPosition[1],0,0

WHILE(I10)
 IF (I1)                                           'J1
  intPosition[1] = intPosition[1] + 32
 ELSEIF (I2)                                       'J2
  intPosition[2] = intPosition[2] + 32
 ELSEIF (I3)                                       'J3
  intPosition[3] = intPosition[3] + 32
 ELSEIF (I4)                                       'J4
  intPosition[4] = intPosition[4] + 32
 ELSEIF (I5)                                       'J5
  intPosition[5] = intPosition[5] + 32
 ELSEIF (I6)                                       'J6
  intPosition[6] = intPosition[6] + 32
 ENDIF
 OPEENC 0,encCurrent,intPosition[1],0,1
 ENC2POSE 0,encCurrent,P1
 MOVEX A=1,AC=0,M1X,P,P1,R= 10.0,H=1,MS            'MOVE to P1
ENDW

END
```

</td></tr>
</table>

POINT

- To set a data to the ENCODER type variable, please use exclusive function commands.

2.6.8 Conversion commands

The user variables can be converted to other different types with the following commands. (e.g. from POSITION to ANGLE, to ENCODER, to pose variable etc.) To move a robot, the data must be converted to a *"pose variable"* at last time and the pose variable must be handed to MOVEX command.

Table 2.37 Conversion commands

Command	FN No.	From	To	Remarks
POS2POSE	FN809	POSITION	Pose variable	
ANG2POSE	FN810	ANGLE	Pose variable	
ENC2POSE	FN811	ENCODER	Pose variable	
POSE2POS	FN812	Pose variable	POSITION	
ANG2POS	FN813	ANGLE	POSITION	
ENC2POS	FN814	ENCODER	POSITION	
POSE2ANG	FN815	Pose variable	ANGLE	
POS2ANG	FN816	POSITION	ANGLE	
ENC2ANG	FN817	ENCODER	ANGLE	
POSE2ENC	FN818	Pose variable	ENCODER	
POS2ENC	FN819	POSITION	ENCODER	
ANG2ENC	FN820	ANGLE	ENCODER	
CVTCOORDPOS	FN821	POSITION	POSITION	Coordinate system conversion

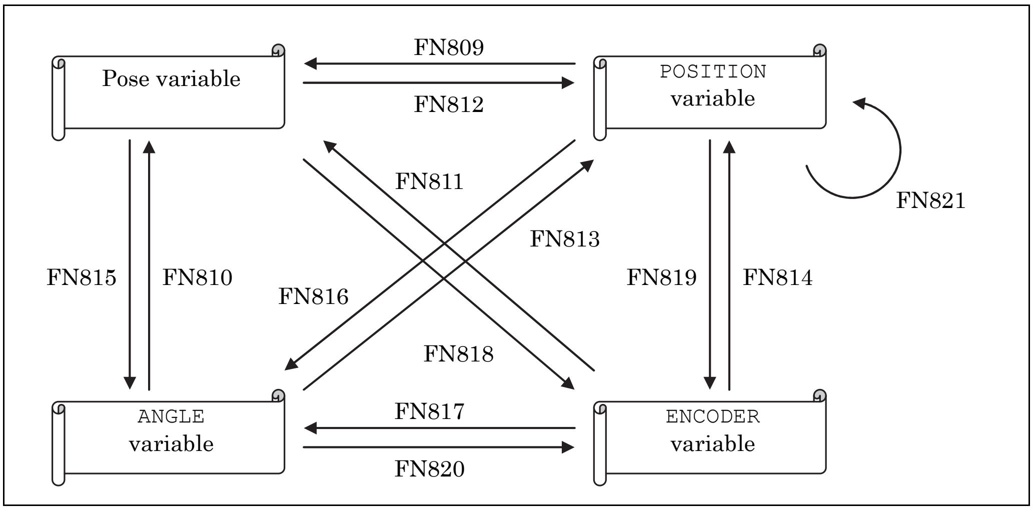


Fig. 2.6.4 Conversion

POINT

For details, refer to the online help of the respective commands.

(Example) Conversion from the POSITION variable

In this example, POSITION variable (X,Y,Z,Roll,Pitch,Yaw,CONF), is converted to the pose variable, the ANGLE variable, and the ENCODER variable.

Table 2.38 POS2POSE

Function	Convert POSITION variable to pose variable
Format	POS2POSE *MechanismNo, PosVariableName, CoordinateSystem, CoordSystemNo, PoseVariableName*
Parameters	MechanismNo Mechanism No. (0-9) (0:stands for the all mechanisms in the UNIT) PosVariableName The name of the POSITION variable CoordinateSystem The coordinate system of the POSITION variable 0 : Machine coordinate system 1 : Tool coordinate system 2 : User coordinate system 3 : World coordinate system 4 : Work-piece coordinate system CoordSystemNo The Tool No. or the User coordinate system No. (NOTE) In case of the other coordinate systems, this parameter is always 1. PoseVariableName The pose variable name to be written (e.g. "P1")

Table 2.39 POS2ANG

Function	Convert POSITION variable to ANGLE variable
Format	POSANG *MechanismNo, PosVariableName, CoordinateSystem, CoordSystemNo, AngVariableName*
Parameters	MechanismNo Mechanism No. (0-9) (0:stands for the all mechanisms in the UNIT) PosVariableName The name of the POSITION variable CoordinateSystem The coordinate system of the POSITION variable 0 : Machine coordinate system 1 : Tool coordinate system 2 : User coordinate system 3 : World coordinate system 4 : Work-piece coordinate system CoordSystemNo The Tool No. or the User coordinate system No. (NOTE) In case of the other coordinate systems, this parameter is always 1. AngVariableName The ANGLE variable name to be written

Table 2.40 POS2ENC

Function	Convert POSITION variable to ENCODER variable
Format	POS2ENC *MechanismNo, PosVariableName, CoordinateSystem, CoordSystemNo, EncVariableName*
Parameters	MechanismNo Mechanism No. (0-9) (0:stands for the all mechanisms in the UNIT) PosVariableName The name of the POSITION variable CoordinateSystem The coordinate system of the POSITION variable 0 : Machine coordinate system 1 : Tool coordinate system 2 : User coordinate system 3 : World coordinate system 4 : Work-piece coordinate system CoordSystemNo The Tool No. or the User coordinate system No. (NOTE) In case of the other coordinate systems, this parameter is always 1. EncVariableName The ENCODER variable name to be written

Sample

System configuration	The UNIT has only 1 mechanism (M1 : 6-axes robot)

Sample

(Source code)

```
USE 1
DIM posCurrent AS POSITION
DIM angCurrent AS ANGLE
DIM encCurrent AS ENCODER
DIM sngPosition[7] AS SINGLE

GETPOS 1, 1, posCurrent, 0, 1                '(1)
OPEPOS 1, posCurrent, sngPosition[1], 0, 0   '(2)
sngPosition[1] = sngPosition[1] + 10.0       '(3)
OPEPOS 1, posCurrent, sngPosition[1], 0, 1   '(4)
POS2POSE 1, posCurrent, 0, 1, P1             '(5)
POS2ANG  1, posCurrent, 0, 1, angCurrent     '(6)
POS2ENC  1, posCurrent, 0, 1, encCurrent     '(7)

MOVEX A=1,M1X,P,P1,R=10,H=1,MS
END
```

(Explanation)

(1) Get the mechanism 1 current position to the `POSITION` variable.
(2) Extract the `POSITION` variable to the `SINGLE` array
(3) Add 10 [mm] for the 1st element (X coordinate) of the `SINGLE` aray
(4) Write the `SINGLE` array back to the `POSITION` variable.
(5) Convert the `POSITION` variable to the pose variable
(6) Convert the `POSITION` variable to the `ANGLE` variable
(7) Convert the `POSITION` variable to the `ENCODER` variable

In case of a robot of multi-mechanism specification, to extract the `POSITION` variable to the real variables or the SINGLE variables, the number of the elements must be like the following. Please be sure that 7 elements are required always for every mechanism in spite of the mechanism type. The elements of "-" are overwritten with 0.0.

(Example)
M1: 6-axes robot
M2: 1 axis Servo Gun
M3: 1 axis Slider
Number of the elements must be : 7x3 = 21
-> `sngPosition[21]`

No.	Value	No.	Value	No.	Value
1	M1:X	8	M2:J1	15	M3:T1
2	M1:Y	9	-	16	-
3	M1:Z	10	-	17	-
4	M1:ROLL	11	-	18	-
5	M1:PITCH	12	-	19	-
6	M1:YAW	13	-	20	-
7	M1:CONF	14	-	21	-

2.6.9 Position get commands (GET)

And, it is possible to get the position of the robot, to execute extracttion & substitution between the user variables and the real/integer variables using function commands. In this section, position get commands are described.

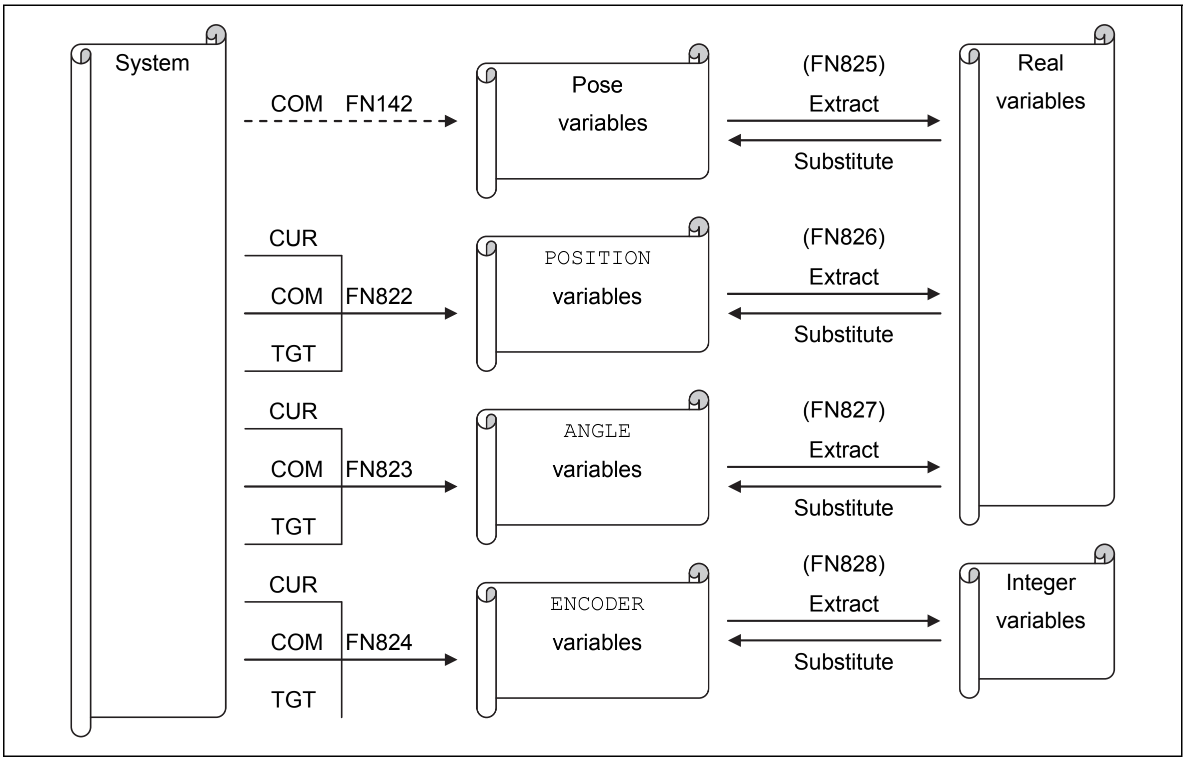


Fig. 2.6.5 Get / Extraction / Substitution

By using the followings commands, it is possible to get the position as the POSITION / ANGLE / ENCODER format.

Table 2.41 FN822 GETPOS

Function	Get the position to the **POSITION** variable	
Format	GETPOS *MechanismNo, PosType,* ***PosVariableName****,* *CoordinateSystem, CoordSystemNo*	
Parameters	MechanismNo	Mechanism No. (0-9) (0:stands for the all mechanisms in the UNIT)
	PosType	Position type(0-2) 0: Command position 1: Current position 2: Object position (target position)
	PosVariableName	The name of the POSITION variable
	CoordinateSystem	The coordinate system of the POSITION variable 0 : Machine coordinate system 1 : Tool coordinate system 2 : User coordinate system 3 : World coordinate system 4 : Work-piece coordinate system
	CoordSystemNo	The Tool No. or the User coordinate system No. (NOTE) In case of the other coordinate systems, this parameter is always 1.

Table 2.42 FN823 GETANG

Function	Get the position to the **ANGLE** variable	
Format	GETANG *MechanismNo, PosType,* ***AngVariableName***	
Parameters	MechanismNo	Mechanism No. (0-9) (0:stands for the all mechanisms in the UNIT)
	PosType	Position type(0-2) 0: Command position 1: Current position 2: Object position (target position)
	AngVariableName	The name of the ANGLE variable to be used

Table 2.43 FN824 GETENC

Function	Get the position to the **ENCODER** variable	
Format	GETENC *MechanismNo, PosType,* ***EncVariableName***	
Parameters	MechanismNo	Mechanism No. (0-9) (0:stands for the all mechanisms in the UNIT)
	PosType	Position type(0-2) 0: Command position 1: Current position 2: Object position (target position)
	EncVariableName	The name of the ENCODER variable

2.6.10 Extract / substitution commands (OPE)

Table 2.44 `FN825 OPEPOSE`

Function	Extract a pose variable to real variables (`V!` or `L!)` or `SINGLE` variables Or Substitute real variables (V! or L!) or SINGLE variables to a pose variable
Format	`OPEPOSE` *`MechanismNo, PoseVariableNo, RealVariable, TargetData, OperationType`*
Parameters	`MechanismNo` Mechanism No. (0-9) (0:stands for the all mechanisms in the UNIT) `PoseVariableNo` Pose variable No. `RealVariable` Real variable - Global real variable `V!` - Local real variable `L!` - `SINGLE` type variable (NOTE) Depending on the "`TargetData`", plural elements may be required. In case of `SINGLE` variable, use an array. `TargetData` Select the operation target. 0 : All 1 : X 2 : Y 3 : Z 4 : R 5 : P 6 : Y `OperationType` Select the Extract / Substitute. 0: Extract 1: Substitute

Sample	`OPEPOSE 1, P1, V1!, 1, 0`
	Extract the X coordinate value of the pose variable 1 (`P1`) to `V1!.`
	
	`OPEPOSE 1, P1, V1!, 1, 1`
	Substitute `V1!` to the X coordinate value of the pose variable 1 (P1)

(NOTE) Required number of the elements
M1: 6-axes robot
M2: 1 axis Servo Gun
M3: 1 axis Slider

In a case of multi-mechanism specification like this, the pose variable contains 8 elements like the following.
`(M1X, M1Y, M1Z, M1ROLL, M1PITCH, M1YAW, M2J1, M3T1)`

For example, when extracting the all elements in this pose variable to global real variables starting from V1!, the data will be made like the following. In short, 6x3=18 elements are required. In case of `SINGLE` variable, please pay attention to the number of elements of the array. And the elements of "-" will be overwritten with 0.0.

```
OPEPOSE 0, P1, V1!, 0, 0
```

`V!`	Value	`V!`	Value	`V!`	Value
1	`M1X`	7	`M2J1`	13	`M3T1`
2	`M1Y`	8	-	14	-
3	`M1Z`	9	-	15	-
4	`M1ROLL`	10	-	16	-
5	`M1PITCH`	11	-	17	-
6	`M1YAW`	12	-	18	-

Table 2.45 FN826 OPEPOS

Function	Extract a POSITION variable to real variables (V! or L!) or SINGLE variables Or Substitute real variables (V! or L!) or SINGLE variables to a POSITION variable
Format	OPEPOS *MechanismNo, PosVariableName, RealVariable, TargetData, OperationType*
Parameters	MechanismNo: Mechanism No. (0-9) (0:stands for the all mechanisms in the UNIT) PosVariableName: The name of the POSITION variable RealVariable: Real variable - Global real variable V! - Local real variable L! - SINGLE type variable (NOTE) Depending on the "TargetData", plural elements may be required. In case of SINGLE variable, use an array. TargetData: Select the operation target. 0 : All 1 : X 2 : Y 3 : Z 4 : ROLL 5 : PITCH 6 : YAW 7 : CONF (NOTE) 1-axis mechanism can use only "0". OperationType: Select the Extract / Substitute. 0: Extract 1: Substitute
Sample	OPEPOS 1, posPosition, V1!, 1, 0 Extract the X coordinate value of the POSITION variable to V1!. OPEPOS 1, posPosition, V1!, 1, 1 Substitute V1! to the X coordinate value of the POSITION variable.

INFO.

(NOTE) Required number of the elements
M1: 6-axes robot
M2: 1 axis Servo Gun
M3: 1 axis Slider

In a case of muti-mechanism specification like this, the POSITION variable contains 9 elements like the following.
(M1X, M1Y, M1Z, M1ROLL, M1PITCH, M1YAW, M1CONF, M2J1, M3T1)

For example, when extracting the all elements in this POSITION variable to global real variables starting from V1!, the data will be made like the following. In short, 7x3=21 elements are required. In case of SINGLE variable, please pay attention to the number of elements of the array. And the elements of "-" will be overwritten with 0.0.

```
OPEPOS 0, posPosition, V1!, 0, 0
```

V!	Value	V!	Value	V!	Value
1	M1X	8	M2J1	15	M3T1
2	M1Y	9	-	16	-
3	M1Z	10	-	17	-
4	M1ROLL	11	-	18	-
5	M1PITCH	12	-	19	-
6	M1YAW	13	-	20	-
7	M1CONF	14	-	21	-

Table 2.46 `FN827 OPEANG`

Function	Extract an `ANGLE` variable to real variables (`V!` or `L!)` or `SINGLE` variables Or Substitute real variables (`V!` or `L!`) or `SINGLE` variables to an `ANGLE` variable
Format	`OPEANG` *`MechanismNo, AngVariableName, RealVariable, TargetData, OperationType`*
Parameters	`MechanismNo` Mechanism No. (0-9) (0:stands for the all mechanisms in the UNIT) `AngVariableName` The name of the `ANGLE` variable `RealVariable` Real variable - Global real variable `V!` - Local real variable `L!` - `SINGLE` type variable (NOTE) Depending on the "`TargetData`", plural elements may be required. In case of `SINGLE` variable, use an array. `TargetData` Select the operation target. 0 : All 1 : J1 2 : J2 3 : J3 4 : J4 5 : J5 6 : J6 (NOTE) 1-axis mechanism can use only "0". `OperationType` Select the Extract / Substitute. 0: Extract 1: Substitute

Sample	`OPEANG 1, angAngle, V1!, 1, 0`
	Extract the J1 value of the `ANGLE` variable to `V1!`.
	
	`OPEANG 1, angAngle, V1!, 1, 1`
	Substitute `V1!` to the J1 value of the `ANGLE` variable.

(NOTE) Required number of the elements
M1: 6-axes robot
M2: 1 axis Servo Gun
M3: 1 axis Slider

In a case of muti-mechanism specification like this, the `ANGLE` variable contains 8 elements like the following.
`(M1J1, M1J2, M1J3, M1J4, M1J5, M1J6, M2J1, M3T1)`

For example, when extracting the all elements in this `ANGLE` variable to global real variables starting from V1!, the data will be made like the following. In short, 6x3=18 elements are required. In case of `SINGLE` variable, please pay attention to the number of elements of the array. And the elements of "-" will be overwritten with 0.0.

```
OPEANG 0, angAngle, V1!, 0, 0
```

`V!`	Value	`V!`	Value	`V!`	Value
1	`M1J1`	7	`M2J1`	13	`M3T1`
2	`M1J2`	8	-	14	-
3	`M1J3`	9	-	15	-
4	`M1J4`	10	-	16	-
5	`M1J5`	11	-	17	-
6	`M1J6`	12	-	18	-

Table 2.47 FN828 OPEENC

Function	Extract an ENCODER variable to integer variables (V% or L%) or INTEGER variables Or Substitute integer variables (V% or L%) or INTEGER variables to an ENCODER variable	
Format	OPEENC *MechanismNo, EncVariableName, IntVariable, TargetData, OperationType*	
Parameters	MechanismNo	Mechanism No. (0-9) (0:stands for the all mechanisms in the UNIT)
	EncVariableName	The name of the ENCODER variable
	IntVariable	Integer variables. - Global integer variable V% - Local integer variable L% - INTEGER type variable (NOTE) Depending on the "TargetData", plural elements may be required. In case of INTEGER variable, use an array.
	TargetData	Select the operation target. 0 : All 1 : J1 2 : J2 3 : J3 4 : J4 5 : J5 6 : J6 (NOTE) 1-axis mechanism can use only "0".
	OperationType	Select the Extract / Substitute. 0: Extract 1: Substitute
Sample	OPEENC 1, encEncoder, V1%, 1, 0 Extract the J1 value of the ENCODER variable to V1%. OPEENC 1, encEncoder, V1%, 1, 1 Substitute V1% to the J1 value of the ENCODER variable.	

(NOTE) Required number of the elements
M1: 6-axes robot
M2: 1 axis Servo Gun
M3: 1 axis Slider

In a case of muti-mechanism specification like this, the ENCODER variable contains 8 elements like the following.
(M1J1, M1J2, M1J3, M1J4, M1J5, M1J6, M2J1, M3T1)

For example, when extracting the all elements in this ENCODER variable to global integer variables starting from V1%, the data will be made like the following. In short, 6x3=18 elements are required. In case of INTEGER variable, please pay attention to the number of elements of the array. And the elements of "-" will be overwritten with 0.

```
OPEENC 0, encEncoder, V1%, 0, 0
```

V%	Value	V%	Value	V%	Value
1	M1J1	7	M2J1	13	M3T1
2	M1J2	8	-	14	-
3	M1J3	9	-	15	-
4	M1J4	10	-	16	-
5	M1J5	11	-	17	-
6	M1J6	12	-	18	-

(NOTE) Hexa-decimal value and Decimal value
"80000" of hexa-decimal notation equals "524288" of decimal notation.

Sample program

This is a sample program in which the robot moves via jog operation using the external input signals.

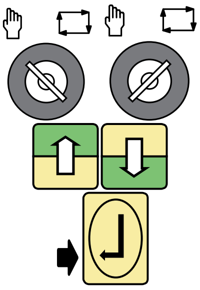
```
USE 500
DIM sgnPosition[7] As SINGLE                          (1) Define an user variable
DIM sgnShift As SINGLE                                (2) Define an user variable
DIM posCurrent As POSITION                            (3) Define an user variable
GETPOS 0,0,posCurrent,0,1                             (4) Get the current position
OPEPOS 0,posCurrent,sgnPosition[1],0,0                (5) Extract the current position
sgnShift = 20                                         (6)Se the shift amount
IF (I1)
sgnPosition[1] = sgnPosition[1]+sgnShift              (7) X+
ELSEIF (i2)
sgnPosition[1] = sgnPosition[1]-sgnShift              (8) X-
ELSEIF (i3)
sgnPosition[2] = sgnPosition[2]+sgnShift              (9) Y+
ELSEIF (i4)
sgnPosition[2] = sgnPosition[2]-sgnShift              (10) Y-
ELSEIF (i5)
sgnPosition[3] = sgnPosition[3]+sgnShift              (11) Z+
ELSEIF (i6)
sgnPosition[3] = sgnPosition[3]-sgnShift              (12) Z-
ENDIF
OPEPOSE 0,P1,sgnPosition[1],0,1                       (13) Set theP1
MOVEX A=1,AC=0,SM=0,M1X,P,P1,R= 5.0,H=1,MS, CONF=0000 (14) MOVEX
END
```

(1) Define an user variable to plus / minus the axis position
(2) An user variable for the jog motion amount
(3) An user variable for the current position
(4) Get the current position
(5) Extract the current position to the user variables
(6) Set the jog motion amount
(7) ～ (12) Plus/Minus the jog amount reading the input signals
(13) Substitute the position data to the pose variable
(14) Execute a MOVEX command

Fig. 2.6.6 Sample program

2.6.11 User variable monitor

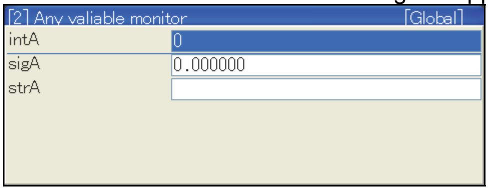
By using "Any variable monitor"(User variable monitor), the defined user variables can be displayed. Although this monitor window is refreshed every 0.5 seconds, variables are not displayed depending on the executing timing of the work-program.



1 **Select TEACH mode or PLAYBACK mode.**

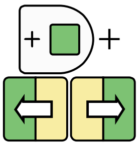
2 **Open <Service Utilities> - [3 Monitor1] [17 Any variable monitor].**

>> A monitor window like the following will appear.



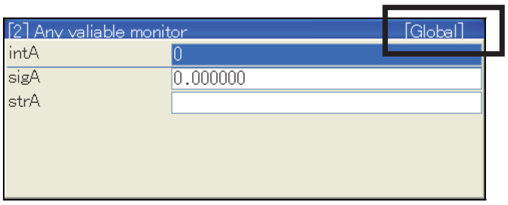
[2] Any valiable monitor [Global]

intA	0
sigA	0.000000
strA	



3 **Select the variable type by pressing [ENABLE] + [Left / right] keys.**

The title bar shows the current variable type.



[2] Any valiable monitor [Global]

intA	0
sigA	0.000000
strA	

The following data can be displayed in this monitor.

Display	Variable type	Remarks
[Global]	Global variables	
[Unit(*U1)]	Unit variable of the current unit. "*U1" is the current unit No.	
[Program(#0001)]	Local variable of the program. "#0001" is the program that is being executed.	
[Unit(U1)] : [Unit(U9)]	Unit variables	Only for UNITs that are registered in the controller
[UserTask(1)] : [UserTask (4)]	Unit variable of the UserTask	This is displayed only while UserTask is being executed
[UserTask1 Program(#0001)] : [UserTask4 Program(#0001)]	Local variable of the UserTask program. "#0001" is the program that is being executed.	This is displayed only while UserTask is being executed

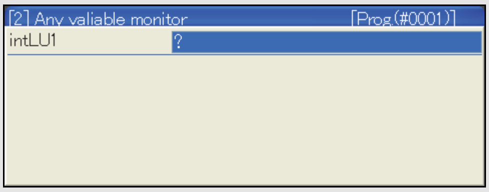
4 The cursor can be moved by [up]/[Down] keys.

5 By [ENABLE] + [up]/[Down] keys., the cursor can be set to every variable.

POINT

The local variables will be displayed when the DIM command is executed and then they will disappear when executing END command or when program selection operation is executed.

When user variables are defined, "?" is displayed at first. And this will be displayed until some values are set.



[2] Any valiable monitor [Prog.(#0001)]

intLU1 ?

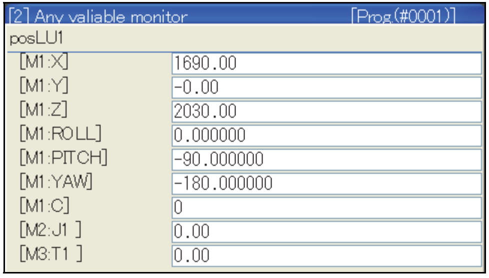
Displaying a POSITION type user variable

In case of a POSITION type variable, the name of the variable is displayed first and the elements follow the line.

Respective data is displayed like [M1:X] (1 is the mechanism No. and X is the data name). In case of a manipulator type mechanism, the data is displayed in the format of (X, Y, Z, ROLL, PITCH, YAW, CONF). In case of other mechanism types, the joint values are displayed one by one.

(Example)

```
DIM posLU1 AS POSITION
```



[2] Any valiable monitor [Prog(#0001)]

posLU1

[M1:X]	1690.00
[M1:Y]	-0.00
[M1:Z]	2030.00
[M1:ROLL]	0.000000
[M1:PITCH]	-90.000000
[M1:YAW]	-180.000000
[M1:C]	0
[M2:J1]	0.00
[M3:T1]	0.00

Mechanism type	Data name	Value	Remarks
Manipulator	X	X coordinate of the TCP	
	Y	Y coordinate of the TCP	
	Z	Z coordinate of the TCP	
	ROLL	Tool posture (ROLL)	
	PITCH	Tool posture (PITCH)	
	YAW	Tool posture (YAW)	
	C (CONF)	The configuration of the robot posture	
Others	Axis name	Joint value. [DEG] is used for rotational axis. [mm] is used for linear axis.	

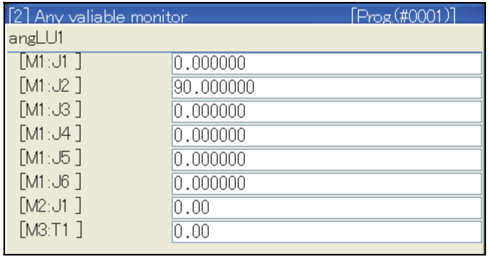
Displaying an ANGLE type user variable

In case of an ANGLE type variable, the name of the variable is displayed first and the elements follow the line.

Respective data is displayed like [M1:J1] ("M1" shows the mechanism No. and "J1" shows the axis name). In case of rotational axis, the unit is [DEG]. In case of linear axis, the unit is [mm].

(Example)

```
DIM angLU1 AS ANGLE
```



[2] Any valiable monitor [Prog(#0001)]

angLU1

[M1:J1]	0.000000
[M1:J2]	90.000000
[M1:J3]	0.000000
[M1:J4]	0.000000
[M1:J5]	0.000000
[M1:J6]	0.000000
[M2:J1]	0.00
[M3:T1]	0.00

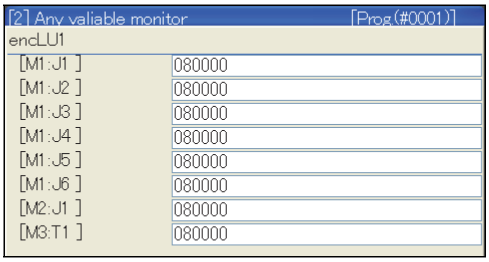
Displaying an ENCODER type user variable

In case of an ENCODER type variable, the name of the variable is displayed first and the elements follow the line.

Respective data is displayed like [M1:J1] ("M1" shows the mechanism No. and "J1" shows the axis name). The data is displayed as hexa-decimal value.

(Example)

```
DIM encLU1 AS ENCODER
```



[2] Any valiable monitor [Prog(#0001)]

encLU1

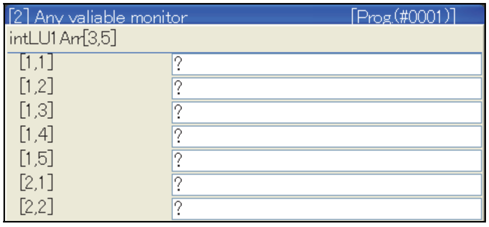
[M1:J1]	080000
[M1:J2]	080000
[M1:J3]	080000
[M1:J4]	080000
[M1:J5]	080000
[M1:J6]	080000
[M2:J1]	080000
[M3:T1]	080000

Displaying an user variable defined with array

In case of a user variable that was defined with array, the name of the variable is displayed first and the elements follow the line.

(Example)

```
DIM intLUArr[3,5] AS INTEGER
```



[2] Any valiable monitor [Prog(#0001)]

intLU1Arr[3,5]

[1,1]	?
[1,2]	?
[1,3]	?
[1,4]	?
[1,5]	?
[2,1]	?
[2,2]	?

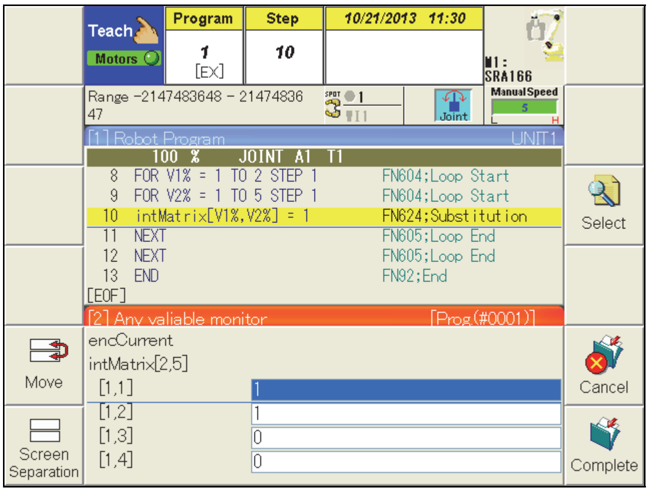
How to jump to the designated array number

Using this selection function, it is possible to jump to the designated array number.

POINT

This selection function is not available while running a program.

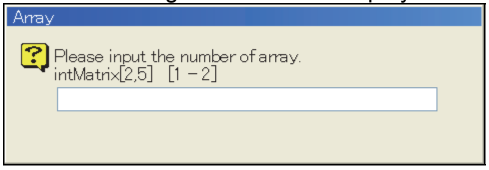
1 Press [EDIT] key while the User variable monitor is active.
>>The monitor turns to edit mode.



```
Program 1 [EX]   Step 10   10/21/2013 11:30   M1:SRA166
Range -2147483648 - 21474836
47
[1] Robot Program                                   UNIT1
      100 %    JOINT  A1  T1
  8  FOR V1% = 1 TO 2 STEP 1         FN604;Loop Start
  9  FOR V2% = 1 TO 5 STEP 1         FN604;Loop Start
 10  intMatrix[V1%,V2%] = 1          FN624;Substitution
 11  NEXT                            FN605;Loop End
 12  NEXT                            FN605;Loop End
 13  END                             FN92;End
[EOF]
[2] Any valiable monitor                      [Prog(#0001)]
encCurrent
intMatrix[2,5]
  [1,1]        1
  [1,2]        1
  [1,3]        0
  [1,4]        0
```

2 Set the curor to the desired variable and press [Select].
>> The following screen will be displayed.

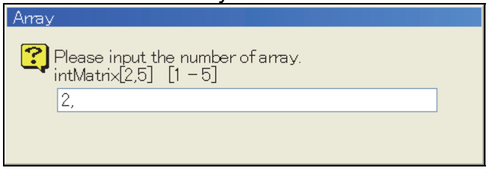
Select



Array

Please input the number of array.
intMatrix[2,5] [1 - 2]

3 Enter the number of the element to be displayed and press [Enter].
>>In case of an array of 2 dimensions or more, the following numbers are required.



Array

Please input the number of array.
intMatrix[2,5] [1 - 5]
2,

4 Enter the all numbers.
>>The cursor jumps to the element.

Edit the value of the user variable

By opening an editor mode, it is possible to edit the value of the user variable.

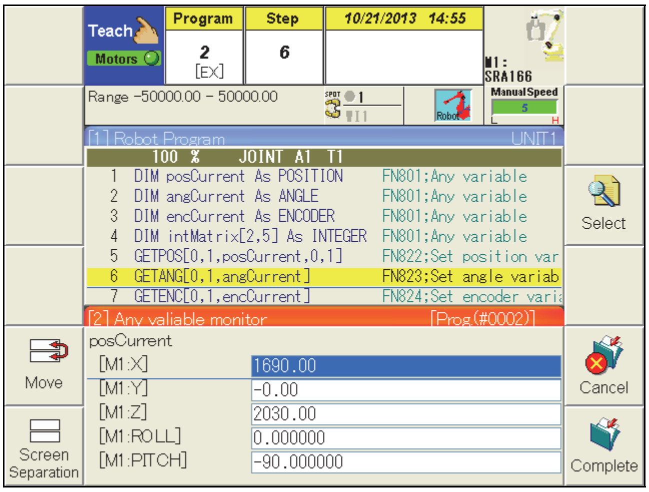
POINT

While running a work-program, it is not possible to edit the user variables.

POINT

The local variables that the values have not been set after the definition cannot be edited.

1 **Press [EDIT] key while the User variable monitor is active.**
>>The monitor turns to edit mode.



Program 2 [EX] | Step 6 | 10/21/2013 14:55 | M1:SRA166

Range -50000.00 - 50000.00

[1] Robot Program UNIT1

```
     100 %      JOINT  A1  T1
  1  DIM posCurrent As POSITION      FN801;Any variable
  2  DIM angCurrent As ANGLE         FN801;Any variable
  3  DIM encCurrent As ENCODER       FN801;Any variable
  4  DIM intMatrix[2,5] As INTEGER   FN801;Any variable
  5  GETPOS[0,1,posCurrent,0,1]      FN822;Set position var
  6  GETANG[0,1,angCurrent]          FN823;Set angle variab
  7  GETENC[0,1,encCurrent]          FN824;Set encoder vari
```

[2] Any valiable monitor [Prog.(#0002)]

```
posCurrent
  [M1:X]        1690.00
  [M1:Y]        -0.00
  [M1:Z]        2030.00
  [M1:ROLL]     0.000000
  [M1:PITCH]    -90.000000
```

2 **Set the cursor to the desired variable and press [Enter].**

3 **In case of `STRING` type variables, the value can be edited by [Enable] + [Edit].**

4 **In case of `ENCODER` type variables, enter the value using hexadecimal value. ([A] – [F] can be inputted by [Enable] + [1] – [6].)**

5 **After completing the input, press <Complete>.**
>> The value of the user variable is modified.
>> To cancel the edit, press <Cancel> f-key or [R] key.

Example of user variable monitor

In this example, the following 3 function commands are executed.

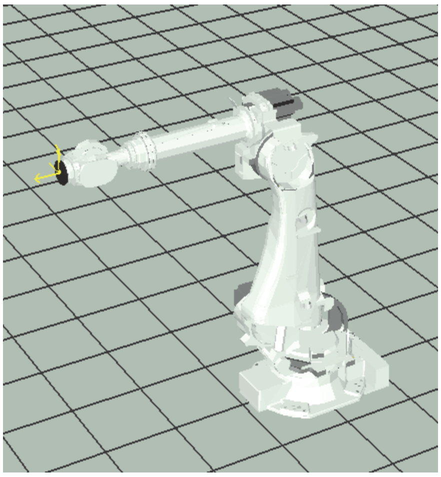
`FN822 GETPOS` (The position is stored to `POSITION` type user variable)
`FN823 GETANG` (The position is stored to `ANGLE` type user variable)
`FN824 GETENC` (The position is stored to `ENCODER` type user variable)

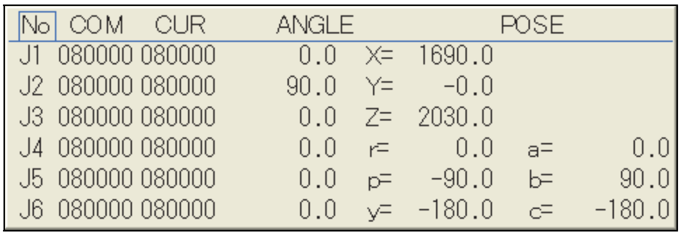
PROGRAM

```
DIM posData AS POSITION
DIM angData AS ANGLE
DIM encData AS ENCODER
MOVEX A=1,AC=0,SM=0,M1J,P,(0,90,0,0,0,0),R= 10,H=1,MS
GETPOS 0,1,posData,0,1    'The current position is stored to posData(1)
GETANG 0,1,angData        'The current position is stored to angData (2)
GETENC 0,1,encData        'The current position is stored to encData (3)
END
```

The robot posture just after executing the MOVEX command

SRA166-01

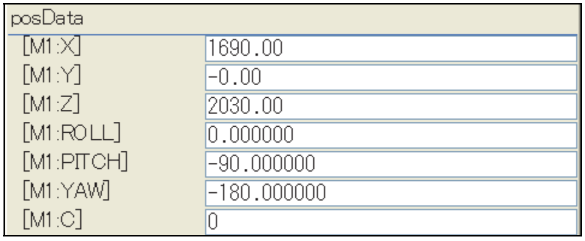




No	COM	CUR	ANGLE	POSE			
J1	080000	080000	0.0	X=	1690.0		
J2	080000	080000	90.0	Y=	-0.0		
J3	080000	080000	0.0	Z=	2030.0		
J4	080000	080000	0.0	r=	0.0	a=	0.0
J5	080000	080000	0.0	p=	-90.0	b=	90.0
J6	080000	080000	0.0	y=	-180.0	c=	-180.0

`posData` when executing the `GETPOS`

Because the Mechanism 1 (M1) is a manipulator, the data is stored in the order of `(X,Y,Z,ROLL,PITCH,YAW,CONF)`.

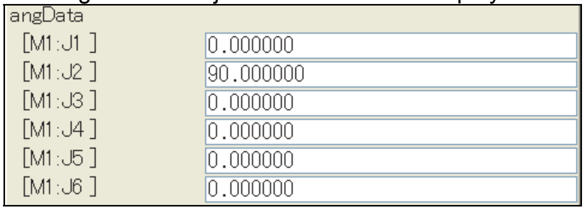


posData

[M1:X]	1690.00
[M1:Y]	-0.00
[M1:Z]	2030.00
[M1:ROLL]	0.000000
[M1:PITCH]	-90.000000
[M1:YAW]	-180.000000
[M1:C]	0

`angData` when executing the `GETANG`

The angles of each joint are stored and displayed. The unit is [DEG].

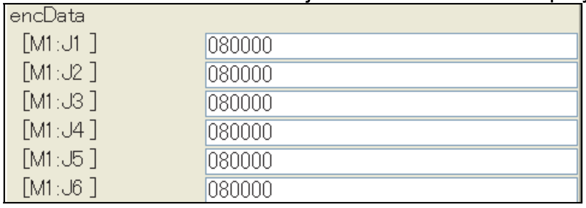


angData

[M1:J1]	0.000000
[M1:J2]	90.000000
[M1:J3]	0.000000
[M1:J4]	0.000000
[M1:J5]	0.000000
[M1:J6]	0.000000

`encData` when executing the `GETENC`

The encoder values of each joint are stored and displayed (The value is hexadecimal)



encData

[M1:J1]	080000
[M1:J2]	080000
[M1:J3]	080000
[M1:J4]	080000
[M1:J5]	080000
[M1:J6]	080000

POINT

If there are plural mechanisms, the data of the mechanism that is not designated in the 1st parameter is not modified. (The initial value is 0)

2.7 Expressions and operations

An "expression" refers to general numerical formulas and functions which connect variables in operation expressions or simply to characters and numerical values or to variables only.

<Examples>

- `"ABC"`
- `3＊5/4+10`
- `V1%+V2%*V1!`
- `SIN(V3!)`
- `1023`
- `V$[4]`

Operations use operators and functions to operate the expressions, and they are classified into the six types below.

- Arithmetic operations
- Relational operations
- Logical operations
- Character string operations
- Pose operations
- Functions

2.7.1 Arithmetic operations

The following operators can be used as arithmetic operators.

Table 2.48 Arithmetic operators

Operator	Description of operation	Example
^	Exponential (power) operations	10^2
-	Minus sign	-100
*	Multiplication	3*5
/	Division	3/2
¥	Division (quotient)	5¥2
MOD	Division (remainder)	5 MOD 2
+	Addition	10+12
-	Subtraction	103-99

2.7.2 Relational operations

Relational operations are used when comparing two numerical values. The results are obtained as true (1) or false (0) and used, for instance, to change the execution sequence of conditional decision statements and other programs.

Table 2.49 Relational operator

Operator	Description of operation	Example
=	Is equal to	V1%=V2%
<>	Is not equal to	V1%<>V2%
<	Less than	V1%<V2%
>	Greater than	V1%>V2%
<=	Less than or equal to	V1%<=V2%
>=	Greater than or equal to	V1%>=V2%

2.7.3 Logical operations

Logical operations connect multiple numbers of conditions, and they are used for bit operations.

Table 2.50 Logical operators

Operator	Description of operation	Example
NOT	Negation	NOT V1%
AND	Logical product	V1% AND V2%
OR	Logical sum	V1% OR V2%
XOR	Exclusive OR	V1% XOR V2%

2.7.4 Character string operations

Character string operations are restricted to connecting character strings using the "+" operator and comparing character strings using relational operators. Character strings are connected by inserting the "+" operator between two character strings.

For example:

```
V1$=″ABC″
V2$=V1$+″123″
```

In this case, "ABC123" character string is assigned to V2$. In addition, character strings can be compared using the "=" and "<>" relational operators. If the two character strings are equal, an equality sign is used (an equality is established). On the other hand, if they are not equal, an inequality sign is used (an inequality is established).

2.7.5 Pose operations

Pose operations are limited solely to the addition of pose constants (variables) and shift constants (variables). And calculation result is not stored to the recorded pose variables.

<Examples>

- P1=P1+R1
- P1=P1+(100,0,0,0,0,0)
- P1=(0,2000,100,0,0,0)+R1

POINT

It is not enabled to make pose constants + shift constants.

※ It is available to adjust the timing of actually storing the robot posture (pose) in the pose variables using the <Constant Setting> menu. When the setting condition of “18 Prefetch of pose substitution” in <Constant Setting> - [5 Operation Constants] - [1 Operation condition] is “Disabled”, the postural data are not stored until the robot reaches the positional posture taught by the MOVE command right before the pose variable substitution command. On the other hand, the postural data are immediately stored before the robot reaching the specified positional posture when the setting condition is “Enabled”.

2.7.6 General functions

Functions perform specific operations for specific arguments, and they return the results of those operations.

Types and descriptions of the functions

Variables can be used for `f/p/s` and or so.
Each input range is as followed.

- In case of integer variables ; -2147483648 to +2147483647
- In case of real variables ; -1.0E38 to +1.0E38
- In case of characters ; 0 to 199 characters (199 bytes)

Table 2.51 Functions

Name of function	Description of function	Attribute
DATE$	Converts the current date into a character string, and returns it.	Character string
TIME$	Converts the current time into a character string, and returns it.	Character string
TIMER	Returns the time (in milliseconds) which has elapsed since the power was turned on.	Real number value
SQR(f)	Obtains the square root.	Real number value
SIN(f)	Obtains sin(f) (where angle f is a radian).	Real number value
COS(f)	Obtains cos(f) (where angle f is a radian).	Real number value
TAN(f)	Obtains tan(f) (where angle f is a radian).	Real number value
ATN(f)	Obtains atan(f).	Real number value
ATN2(f1, f2)	Obtains atan(f1/f2).	Real number value
ABS(f)	Obtains the absolute value.	Real number value
DEGRAD(f)	Converts a degree value into a radian value.	Real number value
RADDEG(f)	Converts a radian value into a degree value.	Real number value
MIN(f1, f2)	Obtains whichever is lower, f1 or f2.	Real number value
MAX(f1, f2)	Obtains whichever is higher, f1 or f2.	Real number value
ORD(s)	Returns the first character code of a character string.	Integer value
CHR$(f)	Returns the character string of length 1 in which value f is the character code.	Character string
VAL(s)	Converts the numerical value expressed by a character string into a value.	Real number value
STR$(f)	Converts a numerical value into a character string.	Character string
BIN$(I)	Converts a numerical value into a character string expressed in binary notation.	Character string
HEX$(i)	Converts a numerical value into a character string expressed in hexadecimal notation.	Character string
MIRROR$(s)	Returns the character string in which the character string s list has been reversed.	Character string
LEFT$(s, f)	Cuts out the character string of length f from the left of character string s.	Character string
RIGHT$(s, f)	Cuts out the character string of length f from the right of character string s.	Character string
MID$(s,f1, f2)	Cuts out the character string of length f2 starting with the f1-th character from the left of character string s.	Character string
STRPOS(s1, s2)	Returns the position that first coincides with character string s2 in character string s1 (199 if the position is not found; 0 if there is no s1 or s2 data).	Integer value
LEN(s)	Returns the length of a character string.	Integer value

Types and descriptions of input functions

Table 2.52 Input functions

Name of function	Description of function	Attribute
`INP(i)`	Used to return the value of input signal specified with the argument " i " through setting it "0" (OFF) or "1" (ON). ("i" can be specified with 1 to 2048 and 5101 to 5196.)	Integer value
`INPB(i)`	Used to get input signals specified with the argument "i" in bytes and then covert the signals into decimal integer values to return them. ("i" can be specified with 1 to 256.) ☞ Refer to "Table 2.53 Argument "i" of input function INPB".	Integer value
`GETSIG(i1,i2)`	Used to get input/output signals specified with the argument "i1" in bits. ("i1" can be specified with 1 to 2048 and 5101 to 5196. For output signals, however, 1 to 2048 are only available.) When i2=0, input signals will be got. When i2=1, output signals will be got.	Integer value
`GETSIGB(i1,i2)`	Used to get input/output signals specified with the argument "i1" in bytes. ("i1" can be specified with 0 to 259.) When i2=0, input signals will be got. When i2=1, output signals will be got. ☞ Refer to "Table 2.54 Argument "i1" of input function GETSIGB"	Integer value
`GETFIX(i1,i2)`	The fixed I/O signals designated by i1 can be acquired bit by bit. IF i2= 0, fixed input signal is acquired. (i1 is from 1 to 32). IF i2= 1, fixed output signal is acquired. (i1 is from 1 to 16).	Integer value
`GETFIXB(i1,i2)`	The fixed I/O signals designated by i1 can be acquired byte by byte. IF i2= 0, fixed input signal is acquired. (i1 is from 1 to 4). IF i2= 1, fixed output signal is acquired. (i1 is from 1 to 2).	Integer value

Table 2.53 Argument "i" of input function INPB

i	Input number
1	0001 to 0008
2	0009 to 0016
3	0017 to 0024
4	0025 to 0032
5	0033 to 0040
6	0041 to 0048
7	0049 to 0056
8	0057 to 0064
9	0065 to 0072
10	0073 to 0080
11	0081 to 0088
12	0089 to 0096
13	0097 to 0104
14	0105 to 0112
15	0113 to 0120
	*** to ***
	*** to ***
242	1929 to 1936
243	1937 to 1944
244	1945 to 1952
245	1953 to 1960
246	1961 to 1968
247	1969 to 1976
248	1977 to 1984
249	1985 to 1992
250	1993 to 2000
251	2001 to 2008
252	2009 to 2016
253	2017 to 2024
254	2025 to 2032
255	2033 to 2040
256	2041 to 2048

Input numbers 1 to 2048 are grouped by every eight inputs.

Table 2.54 Argument "i1" of input function GETSIGB

i1	Input/output number
0	Board internal I/O
1	Board internal I/O
2	Board internal I/O
3	Board internal I/O
4	0001 to 0008
5	0009 to 0016
6	0017 to 0024
7	0025 to 0032
8	0033 to 0040
9	0041 to 0048
10	0049 to 0056
11	0057 to 0064
12	0065 to 0072
13	0073 to 0080
14	0081 to 0088
	*** to ***
	*** to ***
245	1929 to 1936
246	1937 to 1944
247	1945 to 1952
248	1953 to 1960
249	1961 to 1968
250	1969 to 1976
251	1977 to 1984
252	1985 to 1992
253	1993 to 2000
254	2001 to 2008
255	2009 to 2016
256	2017 to 2024
257	2025 to 2032
258	2033 to 2040
259	2041 to 2048

The board internal I/O and input/output numbers 1 to 2048 are grouped by every eight inputs/outputs.

The same numbers are assigned to input and output signals.

2.7.7 System functions

"System functions" are provided in advance for reading the information inside this controller such as the positions of the robot axes. The following types are available.

Table 2.55 System functions

Name of function	Description of function	Attribute
`ERRMES$(i)`	Returns an error message of the control unit that corresponds to error number `i`.	Character string
`SYSTEM%(i)`	Returns the integer information held by the control unit that corresponds to `i`. ☞ Refer to "Table 2.56 Return value of system function `SYSTEM%`"	Integer value
`SYSTEM!(i)`	Returns the real number information held by the control unit that corresponds to `i`. ☞ Refer to "Table 2.57 Return value of system function SYSTEM!"	Real number value
`SYSTEM$(i)`	Returns the character string information held by the control unit that corresponds to `i`. ☞ Refer to "Table 2.58 Return value of system function SYSTEM$"	Character string

Table 2.56 Return value of system function `SYSTEM%`

Argument	Return value
0	User task status Bit 0: User task program now starting Bit 1: 2nd or subsequent cycle from startup Bit 2: User screen now open Bit 3: User screen now temporarily closed Bit 4: 2nd or subsequent cycle after user screen opened Bit 5: User screen active status (key information is coming) Bit 6 to 31: Not used
1	Number of units
2	Mode (0 = teach, 1 = playback, 2 = high-speed teach)
4	Playback mode (0 = 1-step, 1 = 1-cycle, 2 = continuous)
100	Number of user task program now being executed
101 to 109	Numbers of task programs now being executed in each unit 1 to 9
110(120)	Number of user task 1 playback line
111 to 119	Numbers of task programs now being executed in each unit 1 to 9
131 to 139	Numbers of unit axes for each unit 1 to 9
141 to 149	Numbers of unit errors for each unit 1 to 9
151 to 159	Shift status for each unit 1 to 9 (0; not in shift, 1; now shifting)
161 to 169	In-position status for each unit 1 to 9 (0; not in-position,1; in-position)
200 to 205	Current encoder values of axes J1 to J6 of mechanism 1
210 to 215	Current encoder values of axes J1 to J6 of mechanism 2
220 to 225	Current encoder values of axes J1 to J6 of mechanism 3
230 to 235	Current encoder values of axes J1 to J6 of mechanism 4
240 to 245	Current encoder values of axes J1 to J6 of mechanism 5
250 to 255	Current encoder values of axes J1 to J6 of mechanism 6
260 to 265	Current encoder values of axes J1 to J6 of mechanism 7
270 to 275	Current encoder values of axes J1 to J6 of mechanism 8
280 to 285	Current encoder values of axes J1 to J6 of mechanism 9
1000 to 1003	Playback program number of each user task 1 to 4
1010 to 1013	Playback line number of each user task 1 to 4

2001～2006	Returns the seam welding gun number that is being used in the designated seam welding area (from SEAMST to SEAMEND) of the specified welder.
2011～2016	Returns the mechanism number of the electrode wheel on the moving side that is set in the seam welding constants of [13 Spot Welding Application][11 Seam welding] for the designated welder.
2021～2026	Returns the mechanism number of the electrode wheel on the fixed side that is set in the seam welding constants of [13 Spot Welding Application][11 Seam welding] for the designated welder.

Table 2.57 Return value of system function `SYSTEM!`

Argument	Return value	Units
100 to 105	Current joint angles of axes J1 to J6 of mechanism 1	rad
110 to 115	Current joint angles of axes J1 to J6 of mechanism 2	rad
120 to 125	Current joint angles of axes J1 to J6 of mechanism 3	rad
130 to 135	Current joint angles of axes J1 to J6 of mechanism 4	rad
140 to 145	Current joint angles of axes J1 to J6 of mechanism 5	rad
150 to 152	Current fingertip positions (X, Y, Z coordinates) of mechanism 1	mm
153 to 155	Current fingertip positions (X, Y, Z coordinates) of mechanism 2	mm
156 to 158	Current fingertip positions (X, Y, Z coordinates) of mechanism 3	mm
159 to 161	Current fingertip positions (X, Y, Z coordinates) of mechanism 4	mm
162 to 167	Current fingertip positions (X, Y, Z coordinates) of mechanism 5	mm
200 to 205	Current joint angles of axes J1 to J6 of mechanism 6	rad
210 to 215	Current joint angles of axes J1 to J6 of mechanism 7	rad
220 to 225	Current joint angles of axes J1 to J6 of mechanism 8	rad
230 to 235	Current joint angles of axes J1 to J6 of mechanism 9	rad
250 to 252	Current fingertip positions (X, Y, Z coordinates) of mechanism 6	mm
253 to 255	Current fingertip positions (X, Y, Z coordinates) of mechanism 7	mm
256 to 258	Current fingertip positions (X, Y, Z coordinates) of mechanism 8	mm
259 to 261	Current fingertip positions (X, Y, Z coordinates) of mechanism 9	mm
270	Analog voltage 1	V
271	Analog voltage 2	V
272	Analog voltage 3	V
273	Analog voltage 4	V
274	Analog voltage 5	V
275	Analog voltage 6	V
276	Analog voltage 7	V
277	Analog voltage 8	V
300	TCP command speed of the mechanism 1	mm/sec
301	TCP command speed of the mechanism 2	mm/sec
302	TCP command speed of the mechanism 3	mm/sec
303	TCP command speed of the mechanism 4	mm/sec
304	TCP command speed of the mechanism 5	mm/sec
305	TCP command speed of the mechanism 6	mm/sec
306	TCP command speed of the mechanism 7	mm/sec
307	TCP command speed of the mechanism 8	mm/sec
308	TCP command speed of the mechanism 9	mm/sec
310～318	Commanded TCP position of the mechanism 1 (X,Y,Z,R,P,Y,A,B,C)	mm・deg
319～327	Commanded TCP position of the mechanism 2 (X,Y,Z,R,P,Y,A,B,C)	mm・deg
328～336	Commanded TCP position of the mechanism 3 (X,Y,Z,R,P,Y,A,B,C)	mm・deg
337～345	Commanded TCP position of the mechanism 4 (X,Y,Z,R,P,Y,A,B,C)	mm・deg
346～354	Commanded TCP position of the mechanism 5 (X,Y,Z,R,P,Y,A,B,C)	mm・deg
355～363	Commanded TCP position of the mechanism 6 (X,Y,Z,R,P,Y,A,B,C)	mm・deg
364～372	Commanded TCP position of the mechanism 7 (X,Y,Z,R,P,Y,A,B,C)	mm・deg

373～381	Commanded TCP position of the mechanism 8 (X,Y,Z,R,P,Y,A,B,C)	mm・deg
382～390	Commanded TCP position of the mechanism 9 (X,Y,Z,R,P,Y,A,B,C)	mm・deg
800	The current speed of TCP of the mechanism 1	mm/sec
801	The current speed of TCP of the mechanism 2	mm/sec
802	The current speed of TCP of the mechanism 3	mm/sec
803	The current speed of TCP of the mechanism 4	mm/sec
804	The current speed of TCP of the mechanism 5	mm/sec
805	The current speed of TCP of the mechanism 6	mm/sec
806	The current speed of TCP of the mechanism 7	mm/sec
807	The current speed of TCP of the mechanism 8	mm/sec
808	The current speed of TCP of the mechanism 9	mm/sec
810～818	Wrist tip position of the mechanism 1 (X, Y, Z, R, P, Y, A, B, C)	mm・deg
819～827	Wrist tip position of the mechanism 2 (X, Y, Z, R, P, Y, A, B, C)	mm・deg
828～836	Wrist tip position of the mechanism 3 (X, Y, Z, R, P, Y, A, B, C)	mm・deg
837～845	Wrist tip position of the mechanism 4 (X, Y, Z, R, P, Y, A, B, C)	mm・deg
846～854	Wrist tip position of the mechanism 5 (X, Y, Z, R, P, Y, A, B, C)	mm・deg
855～863	Wrist tip position of the mechanism 6 (X, Y, Z, R, P, Y, A, B, C)	mm・deg
864～872	Wrist tip position of the mechanism 7 (X, Y, Z, R, P, Y, A, B, C)	mm・deg
873～881	Wrist tip position of the mechanism 8 (X, Y, Z, R, P, Y, A, B, C)	mm・deg
882～890	Wrist tip position of the mechanism 9 (X, Y, Z, R, P, Y, A, B, C)	mm・deg
1001～1006	The servo-gun commanded pressure of the designated welding machine.	Kgf
1011～1016	The servo-gun actual pressure of the designated welding machine.	Kgf
2001～2006	Returns the rotation speed of the moving side electrode wheel of the designated welder.	Rotation speed
2011～2016	Returns the rotation speed of the fixed side electrode wheel of the designated welder.	Rotation speed
2021～2026	Returns the seam welding length (from the seam welding start to the seam welding end) in [mm] for the designated welder. This will be cleared when starting a seam welding.	Mm
2031～2036	Returns the seam welding time (from the seam welding start to the seam welding end) in [sec] for the designated welder. This will be cleared when starting a seam welding.	Sec
2041～2046	Returns the "Elect.distance" (from the seam welding start to the seam welding end) in [sec] for the designated welder. This will be cleared when starting a seam welding.	Mm
2051～2056	Returns the "Elect.time" (from the seam welding start to the seam welding end) in [sec] for the designated welder. This will be cleared when starting a seam welding.	Sec

In the case of a servo gun, slider or other rectilinear axis, the current joint angle is expressed in [mm] rather than [rad].

Even in the inch mode, the return values for lengths are expressed in [mm].

Table 2.58 Return value of system function SYSTEM$

Argument	Explanation
0	Software version
101	Unit name of unit 1
102	Unit name of unit 2
103	Unit name of unit 3
104	Unit name of unit 4
105	Unit name of unit 5
106	Unit name of unit 6
107	Unit name of unit 7
108	Unit name of unit 8
109	Unit name of unit 9
300 to 2347	Names of input signals
2348 to 4098	Names of output signals
5001 to 5100	Palletize name

2.7.8 Priority of operators

When a multiple number of operators are present in an expression, the sequence in which they are applied is determined by the sequence of priority which has been set for each operator. The sequence of priority is set as shown below.

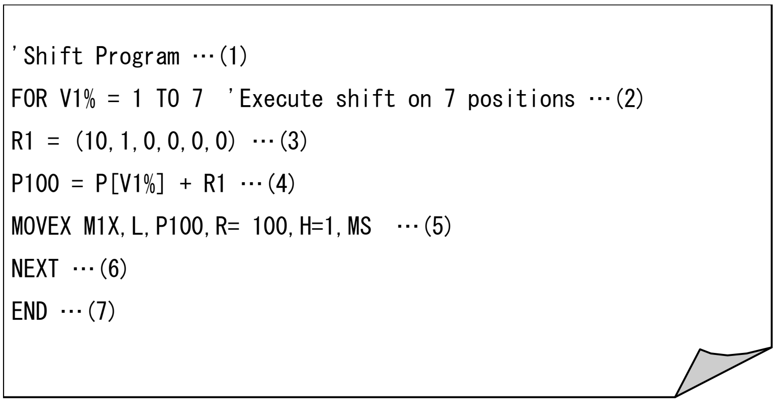
Table 2.59 Priority of operators

Operator	Priority
+,- (signs)	Higher
NOT	▲
^	
/,*,¥,MOD	
+(addition), -(subtraction)	
=,<>,>,<,>=,<=	
AND	
OR	▼
XOR	Lower

2.8 Statements

Statements must be allocated on a one statement per line basis in source programs. In this manual, one line of a program and one statement are considered to be the same unless otherwise specified.

Robot Language Program

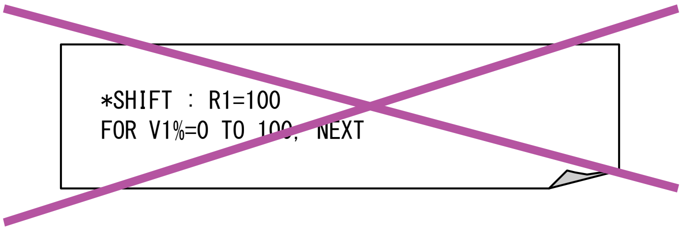


```
'Shift Program …(1)
FOR V1% = 1 TO 7  'Execute shift on 7 positions …(2)
R1 = (10,1,0,0,0,0) …(3)
P100 = P[V1%] + R1 …(4)
MOVEX M1X,L,P100,R= 100,H=1,MS  …(5)
NEXT …(6)
END …(7)
```

(1) Comment statement
(2) Flow control statement + Comment statement
(3) Substitution statement
(4) Substitution statement
(5) Command statement
(6) Flow control statement
(7) Flow control statement

Fig. 2.8.1 Statements

Robot Language Program



```
*SHIFT : R1=100
FOR V1%=0 TO 100, NEXT
```

Two or more statements cannot be described on a single line.

Fig. 2.8.2 Description of only one statement on a single line

POINT

- Two or more statements cannot be described on a single line.
- Insertion of any line feed code in a statement is prohibited.
- A maximum of 254 characters can be used in a statement. Writing 255 or more characters on a single line will result in a compiling error.

2.8.1 Comment statement

If a statement starts with "'" (a single quotation mark) or the description "REM," the line concerned is a comment statement. Furthermore, comment statement can be described after the command statement and flow control statement with a single quotation mark.
Comment statements are written to make it easier to understand the program. Therefore, the contents of comment statements are not executed. Comments may be up to 199 characters long.

<Examples>

```
・ ’SPOT1 Program
・ REM "ARC ON"
・ SET O1  ‘Task complete
```

POINT

Maximum line number of the program with comment statement is fewer than that of the program without comment statement, because comment statement needs memory to store them. Maximum line number varies depending on the length of comment statement.

2.8.2 Labels

Any statement starting with "*" (an asterisk) and followed by letters of the alphabet is identified as a label. GOTO, IF and other statements are provided as instructions that control the program flow, and label names are specified as their branch destinations.
Labels are written using up to 16 characters starting with a letter of the alphabet. The character sthat may be used are alphanumerics, "." (period) and "_" (underbar). The total maximum number of characters which can be written in labels in a program is 1024.

<Examples>

```
・ *HANDLING1
・ *ARC_WELDING
```

INFO.

For example, the labels can be used as a jump destination for "FN602 IF".

```
IF I1=1 THEN *HANDLING1 ELSE *ARC_WELDING
```

2.8.3 Substitution statement

Values can be substituted for the respective variables in a program whether the variables are integer variables, real number variables, character string variables, shift variables or pose variables.
Any statement starting with the name of a variable and followed by "=" is identified as a substitution statement.

<Examples>

```
・ V1%=10
・ V3$=V1$
・ R1=(100,0,0,0,0,0)
・ P1=(100,1000,1000,0,90,90)
```

2.8.4 Flow control statement

The instructions used to change the program flow such as the repeat execution, branch, and subroutine call instructions are called flow control statements. Each type of flow control statement is described below.

Table 2.60 Flow control statement

Command	Outline
GOTO line number/label	This transfers control unconditionally to the line indicated by the line number or label. Example) GOTO 100 GOTO *LOOP_END
GOSUB line number/label	This transfers control to the subroutine indicated by the line number or label name. Example) GOSUB 120 GOSUB *FUNC1
RETURN	This returns control from the subroutine to the call source (line following the line on which the subroutine was called by GOSUB).

IF conditional expression
THEN line number/label
ELSE line number/label
	This transfers control to the line number or label written after THEN if the conditions are satisfied. It transfers control to the line number or label written after ELSE if the conditions are not satisfied. If ELSE is not written and the conditions are not satisfied, it transfers control to the next line. A line number or label must be written after THEN or ELSE without fail. Execution statements cannot be written.
	Example) IF V1%=1 THEN 100 ELSE 200
	IF V1%>100 AND V1%<200 THEN *START

FOR variable name = initial value TO end value STEP increment NEXT	This repeatedly executes the instruction written in the loop between FOR and NEXT. The variable name must be an integer variable (V%, L%) or real number variable (V!, L!). The variable must be specified directly (as V1% not as V%[1]). The initial value is the value which is set into the variable in the initial loop. When the instruction reaches NEXT, the increment is added to the variable and compared with the end value. If the instruction has exceeded the end value, control is transferred to the line following NEXT. In all other cases, it transfers to the instruction following FOR. "+1" is used as the increment when STEP has been omitted. It is possible to write another loop between FOR and NEXT inside a loop between FOR and NEXT. This is referred to as nesting. In nesting, different variable names must be used without fail. The maximum number of nesting level is 4. The following precautions must be heeded in order for FOR statements to be used. (1) It is not possible to make changes to the variables used by FOR statements inside the same loop between FOR and NEXT. (2) It is not possible to write instructions for jumping out of a FOR-NEXT loop inside the loop (such as GOTO and RETURN). (3) If increment is 0, this loop never ends because variable never changes. (4) If sigh of increments and initial/end number does not match, loop is never started and program is aborted. (5) If reverse conversion is executed, increment is surely recorded. (6) If increment is not 1, loop will end when variable becomes larger than the end value. (7) If variable is integer and increment is decimal (smaller than 1), this loop never ends because decimal is ignored. (8) If increment includes decimal value, loop may act incorrectly because of miscalculation. Beware that above situations are never detected as compile error. Example) FOR V1%=0 TO 100 FOR V2%=0 TO 10 V3%=V1%*10+V2% ~~V1%=2~~ ····Prohibited ~~RETURN~~ ····Prohibited NEXT NEXT

Command	Outline

WHILE
conditional expression
ENDW
	While condition expression is satisfied, commands between WHILE and ENDW are repeated. One WHILE/ENDW loop can contain another WHILE/ENDW loop. Maximum number of nesting level is 4.
	
	Example)
	`V1% = 0`
	`WHILE V1% < 10`
	`  . . .`
	`  . . .`
	`V1% = V1% + 1`
	`ENDW`

IF conditional expression
ELSEIF
conditional expression
ELSE
ENDIF
	When IF conditional expression is satisfied, commands between IF and ELSEIF is executed.
	When IF conditional expression is not satisfied and ELSEIF conditional expression is satisfied, commands between ELSEIF and ELSE is executed.
	When IF conditional expression is not satisfied and ELSEIF conditional expression is not satisfied, commands between ELSE and ENDIF is executed.
	Following statement is also available,
	
	Example 1)
	IF conditional expression
	
	ENDIF
	
	Example 2)
	IF conditional expression
	
	ELSEIF conditional expression
	
	ENDIF
	
	Example 3)
	IF conditional expression
	
	ELSE
	
	ENDIF
	
	Example 4)
	IF conditional expression
	
	ELSEIF conditional expression
	
	ELSEIF conditional expression
	
	ELSE
	
	ENDIF
	
	One IF – ENDIF can contain another IF – ENDIF Maximum number of nesting level is 4.

Command	Outline

SWITCH operation
CASE integer constant
BREAK
ENDS
	Execute only one command between CASE and ENDS corresponding to the value of integer constant as the result of SWITCH operation.
	
	Example)
	`SWITCH V1%`
	`CASE 1`
	`・・・`
	`BREAK`
	`CASE 2`
	`CASE 3`
	`・・・`
	`BREAK`
	`CASE`
	`・・・`
	`BREAK`
	`ENDS`
	
	When V1%=1, commands between CASE 1 and first BREAK and executed.
	When V1%=2, commands between CASE 2 and second BREAK are executed.
	When V1%=3, commands between CASE 3 and second BREAK are executed.
	If V1% is not 1 neither 2 and 3, commands between CASE and last BREAK s executed.
	
	NOTE)
	CASE statement described in advance is estimated in advance. In case of the following Example 1, the commands between first CASE and first BREAK are executed irrelevant to the value of V1%.
	Example 1)
	`SWITCH V1%`
	`CASE`
	`・・・`
	`BREAK`
	`CASE 1`
	`・・・`
	`BREAK`
	`CASE 2`
	`・・・`
	`BREAK`
	`ENDS`
	
	In case of the following Example 2, when V1%=1, commands between first CASE 1 to first BREAK is executed.
	Example 2)
	`SWITCH V1%`
	`CASE 1`
	`・・・`
	`BREAK`
	`CASE 1`
	`・・・`
	`BREAK`
	`CASE 2`
	`・・・`
	`BREAK`
	`CASE`
	`・・・`
	`BREAK`
	`ENDS`

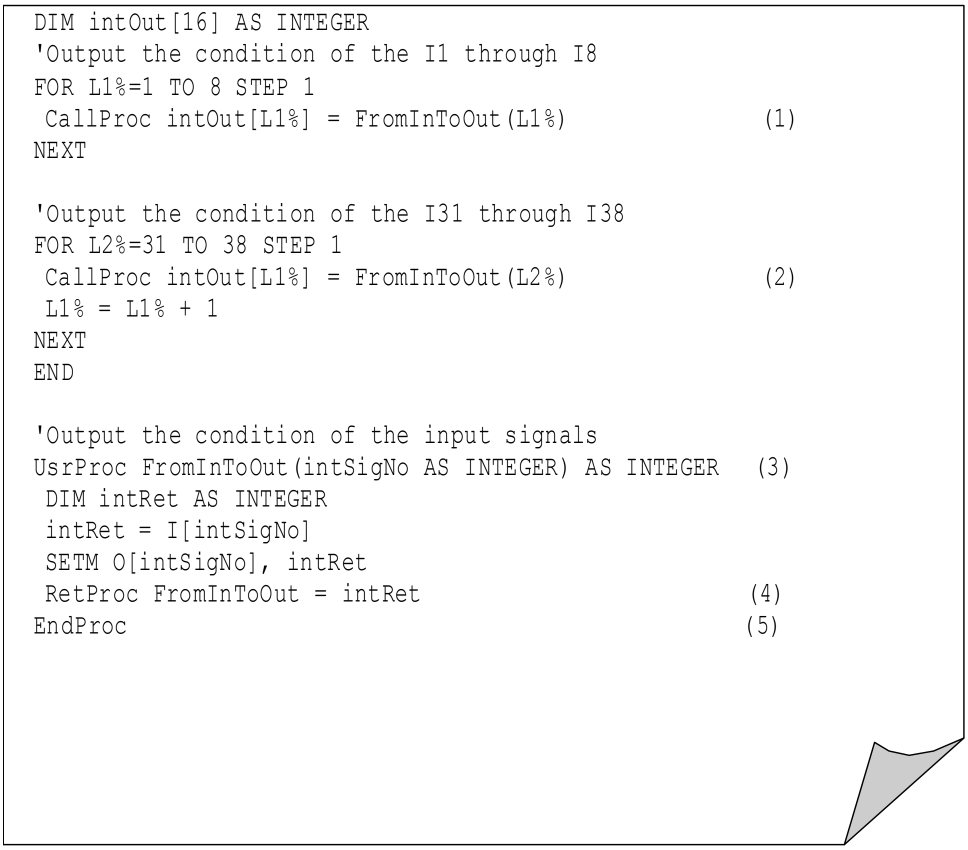
Command	Outline
ON integer variable GOTO line number/label, line number/label...	This transfers control to the line number or label written at the same order as that of the value indicated by the conditional expression. The line numbers and labels are assigned in ascending order by number such as 1, 2, 3 ... from the left. Example) `ON V1% GOTO 100, 200, 300` This example shows the case where control will be transferred to the 100th line when V1%=1 and to the 200th line when V1%=2. A maximum of 10 line numbers or labels can be described
STOP	This stops the execution of the program.
END	The program is ended. When a playback program is executed in the continuous playback mode, it is re-executed from the start of the program. **At least one END instruction must be written per program.**

2.8.5 Command statement

The command statements, such as the MOVE-X commands for moving the robot and the SET commands for turning ON the output signals, constitute the heart of a program. The robot language programs consist almost entirely of command statements.
These are very important statements, and many instructions are found in them.

2.9 User procedure

It is possible to create a ***user procedure*** in a program and use it as a sub routine with parameters. By using this function, it becomes possible to improve the readability of the program by executing all the same processes as an identical function (procedure) with parameters and it becomes possible to maintain the program.



```
DIM intOut[16] AS INTEGER
'Output the condition of the I1 through I8
FOR L1%=1 TO 8 STEP 1
 CallProc intOut[L1%] = FromInToOut(L1%)              (1)
NEXT

'Output the condition of the I31 through I38
FOR L2%=31 TO 38 STEP 1
 CallProc intOut[L1%] = FromInToOut(L2%)              (2)
 L1% = L1% + 1
NEXT
END

'Output the condition of the input signals
UsrProc FromInToOut(intSigNo AS INTEGER) AS INTEGER   (3)
 DIM intRet AS INTEGER
 intRet = I[intSigNo]
 SETM O[intSigNo], intRet
 RetProc FromInToOut = intRet                        (4)
EndProc                                              (5)
```

(1)(2) Call the user procedure
The user procedure is called.

(3) Definition of the user procedure
Parameters and return value can be designated.

(4) The return value of the user procedure
Set the return value.

(5) Finish the procedure
Finish the procedure and return to the call point.

Fig. 2.9.1 An example of user procedure

To create a user procedure, please follow these steps.

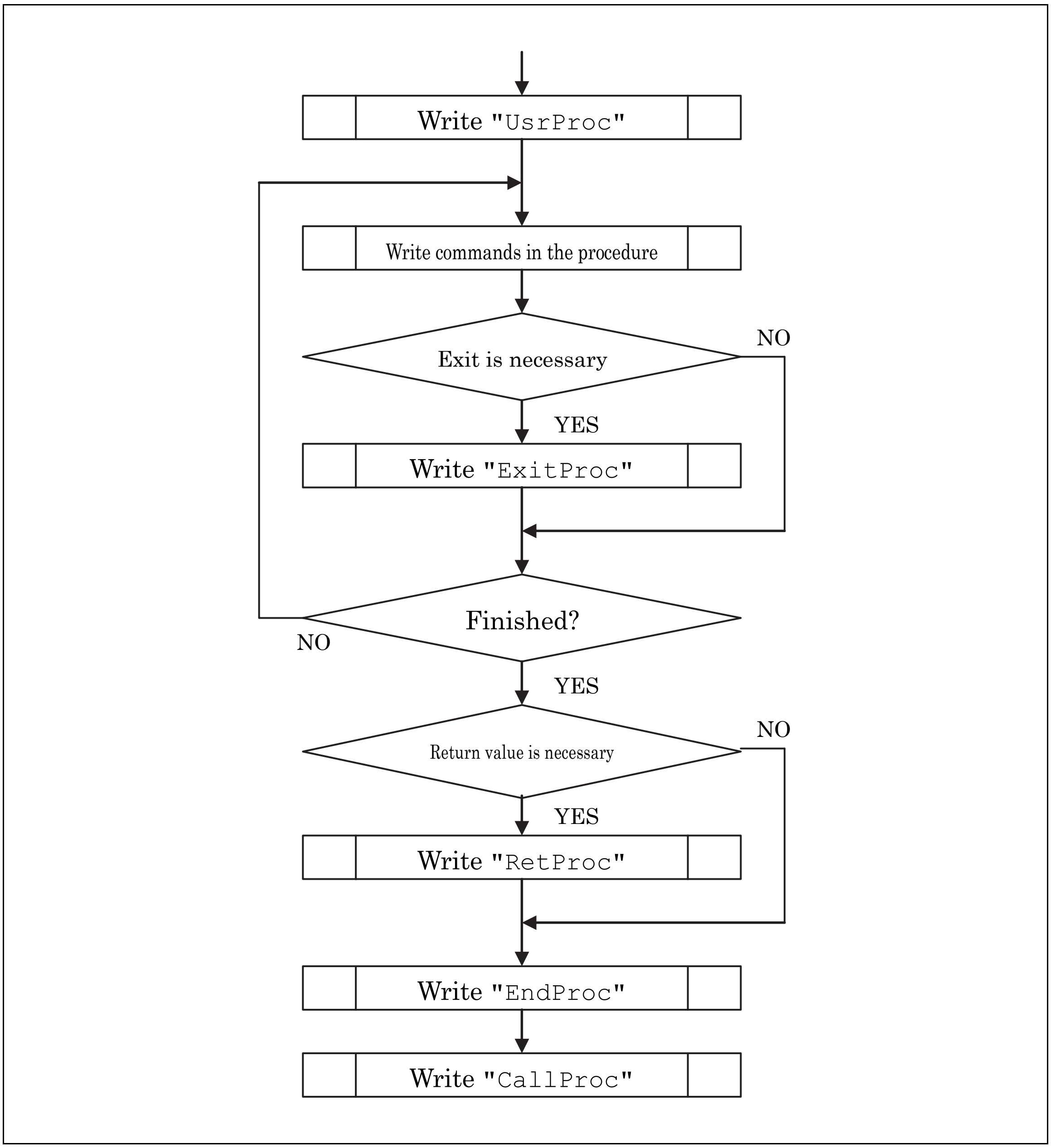


Fig. 2.9.2 Creating steps of user procedure

2.9.1 FN802 User Procedure

This function command creates a User procedure.

Table 2.61 FN802 User procedure

Format	`UsrProc` *`ProcedureName(Parameters) ReturnValue`*
Procedure name	Designate the procedure name up to 32 letters. ☞ See "Table 2.29 Available letters for the variable's name" This first letter must be alphabet. The capictal letter and the small letters are distinguished. The same variable names, the same procedure names, and the reserved names cannot be used as the procedure name.
Parameters	Up to 10 parameters can be designated at maximum. The parameters must be separated with ",". If the parameters are not necessary, omit them. **[Sample]** `intSigNo AS INTEGER` ("`DIM`" is not necessary.) The parameter is designated by a user variable. It is also possible to use an array. The value of the original variable is not changed. (The copy of the variable is used in the procedure)
Return value	It is possible to return a value. If not necessary, skip this. An array cannot be used as the return value. **[Sample]** `AS INTEGER` ("`DIM`" and the variable's name are not necessary) Write the data type of the user variable only.

Sample	**[No parameters, no return value]**
	`UsrProc SignalCheck()`
	
	[Only parameters]
	`UsrProc SignalCheck(intInOut AS INTEGER)`
	
	[Only return value]
	`UsrProc SignalCheck() AS INTEGER`
	
	[With parameters and return value]
	`UsrProc SignalCheck(intInOut AS INTEGER, intSigNo AS INTEGER) AS INTEGER`

Function	FN802 User procedure

2.9.2 FN803 Exit User procedure

Exit from the User Procedure in half way and return to the call point.

Table 2.62 FN803 Exit User procedure

Format	`ExitProc`
Function	FN803 Exit User procedure

POINT

If a return value is necessary, please set the value before exitting the user procedure.

2.9.3 FN804 End User procedure

End the User procedure and return to the call point.

Table 2.63 FN804 End User procedure

Format	`EndProc`
Return value	Before executing this command, it is possible to set the return value. If not necessary, skip this. ☞"2.9.4 FN805 Return User procedure"
Function	FN804 End User procedure

POINT

If a return value is necessary, please set the value before exitting the user procedure.

2.9.4 FN805 Return User procedure

Set the return value of the user procedure.

Table 2.64 FN805 Return User procedure

Format	`RetProc` *`ProcedureName = ReturnValue`*
Procedure name	Set the procedure name that was used by the `FN802 UsrProc`.
Return value	Designate the user variable that has the same type in the FN802 UsrProc

Sample	`UserProc SignalCheck(intSigNo AS INTEGER) AS INTEGER`
	` DIM intStatus;`
	` intStatus = I[intSigNo]`
	` RetProc SignalCheck = intStatus`
	`EndProc`

Function	FN805 Return User procedure

POINT

If a return value is necessary, please set the value before exitting the user procedure.

POINT

Constant value cannot be used as the return value.

2.9.5 FN806 Call User procedure

This function calls a user procedure.

Table 2.65 FN806 Call User procedure

Format	`CallProc` *`Variable`* `=` *`ProcedureName(parameter1, parameter2, ... )`*
Substitution (Variable)	If the User procedure has a return value, please set the variable for the substitution. If the user procedure does not have a return value, this description can be skipped. "=" is also unnecessary.
Procedure name	Set the user procedure name to be called.
Parameters	If the user procedure has parameters, designate the variables as the parameters.

Sample	**[No parameters, no return value]**
	`CallProc SignalCheck()`
	
	[Only parameters]
	`CallProc SignalCheck(V1%)`
	
	[Only return value]
	`CallProc V10% = SignalCheck()`
	
	[With parameters and return value]
	`CallProc V10% = SignalCheck(V1%,V2%)`

Function	FN806 Call User procedure

POINT

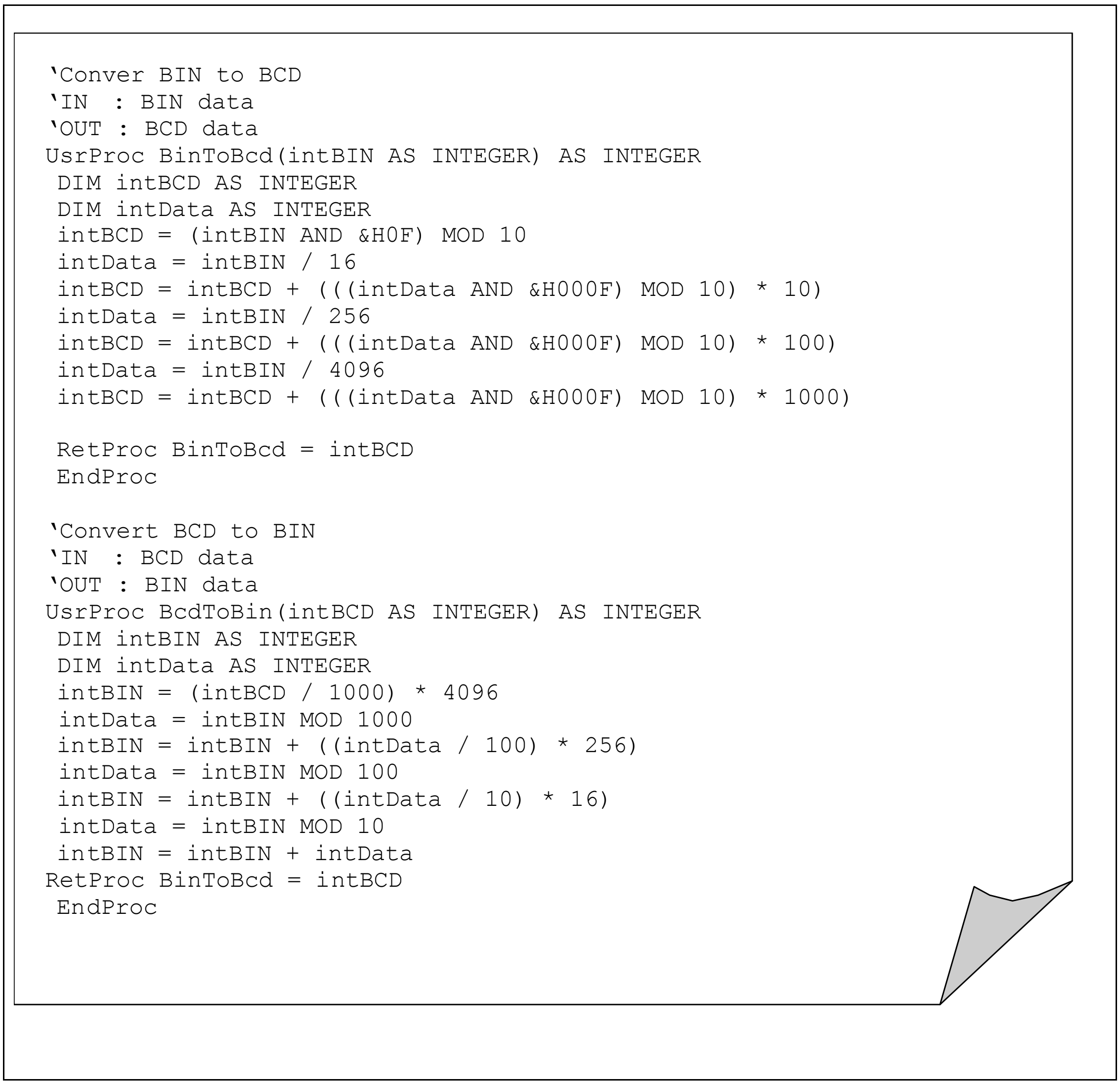
By following the declaration of the user procedure, each parameter must be set in proper order.

POINT

Constant values cannot be used as the parameters.

Sample program

This is an example of user procedure that coverts the input signal condition from BIN format to BCD format.



```
‘Conver BIN to BCD
‘IN  : BIN data
‘OUT : BCD data
UsrProc BinToBcd(intBIN AS INTEGER) AS INTEGER
 DIM intBCD AS INTEGER
 DIM intData AS INTEGER
 intBCD = (intBIN AND &H0F) MOD 10
 intData = intBIN / 16
 intBCD = intBCD + (((intData AND &H000F) MOD 10) * 10)
 intData = intBIN / 256
 intBCD = intBCD + (((intData AND &H000F) MOD 10) * 100)
 intData = intBIN / 4096
 intBCD = intBCD + (((intData AND &H000F) MOD 10) * 1000)

 RetProc BinToBcd = intBCD
 EndProc

‘Convert BCD to BIN
‘IN  : BCD data
‘OUT : BIN data
UsrProc BcdToBin(intBCD AS INTEGER) AS INTEGER
 DIM intBIN AS INTEGER
 DIM intData AS INTEGER
 intBIN = (intBCD / 1000) * 4096
 intData = intBIN MOD 1000
 intBIN = intBIN + ((intData / 100) * 256)
 intData = intBIN MOD 100
 intBIN = intBIN + ((intData / 10) * 16)
 intData = intBIN MOD 10
 intBIN = intBIN + intData
RetProc BinToBcd = intBCD
 EndProc
```

Fig. 2.9.3 Sample program

NOTE

Chapter 3 Program editing

This chapter describes how to edit robot language programs. There is a choice of two methods: one involves the use of a text editor which is available on the market and is run in a personal computer, and the other edits directly from the teach pendant. Also described is how to create pose files using the robot.

3.1 Editing using a personal computer

To create or edit a robot language source program, a text editor software in the market can be used on your PC.

3.1.1 Precautions for editing

Filename regulations	Refer to Chapter 1.
Editing method	Refer to the instructions accompanying the personal computer and text editor used.
Size of the files	Ensure that the size of the files is no more than 64KB (65534 bytes).
Precautions for statements, lines and characters	A total of 254 characters may be written on one line, and up to 999 lines can be written in a program. No differentiation is made between upper-case and lower-case letters of the alphabet. Use half-size characters for all text except for comments and character string variables. Also read carefully through the precautions set forth in "Chapter 2 Syntax" before proceeding.
Other	Create files using the shift JIS code for full-size characters and CR+LF for the line feed codes.

3.1.2 Loading a robot language source program into this controller

To copy a robot language source program file to the internal memory of this robot controller, please use (1) or (2). (It is recommended to use USB memory.)

(1) Copying the file using USB memory

Insert a USB memory and then open <Service Utilities> - [7 File Manager] - [1 File Copy] screen to copy the file from the USB memory to the internal memory. For details, refer to the following instruction manual.

"BASIC OPERATIONS MANUAL" Chapter 6

(2) Copying the file using Ethernet and FTP

After connecting this controller and your PC with a Ethernet cable, please transfer the file using a FTP client software on the PC. For details, refer to the following instruction manual.

"SETUP MANUAL" Chapter 8

INFO.

A FTP client software can be purchased in the market etc.

In either way, please do not forget to place the program file in the following folder. The program files not placed in this folder will be ignored by this robot controller.

```
¥Memory¥WORK¥PROGRAM¥
```

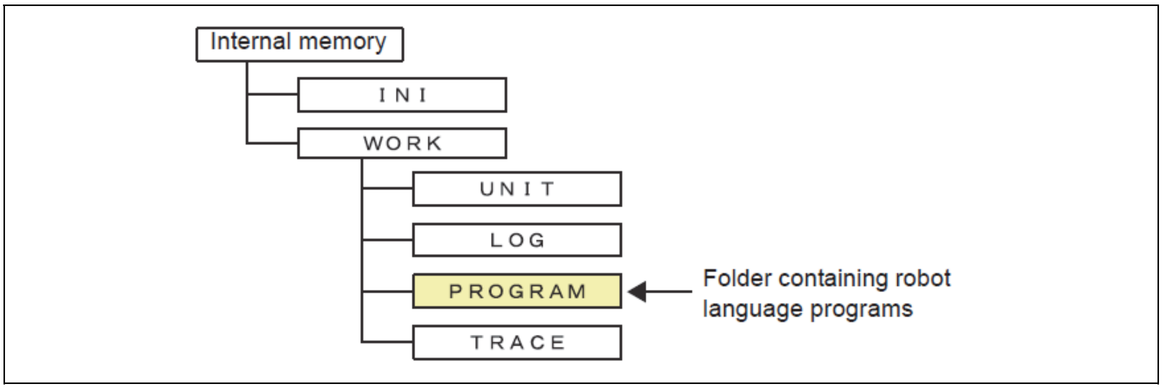


Fig 3.1 Folder containing robot language programs

3.2 Editing using the teach pendant

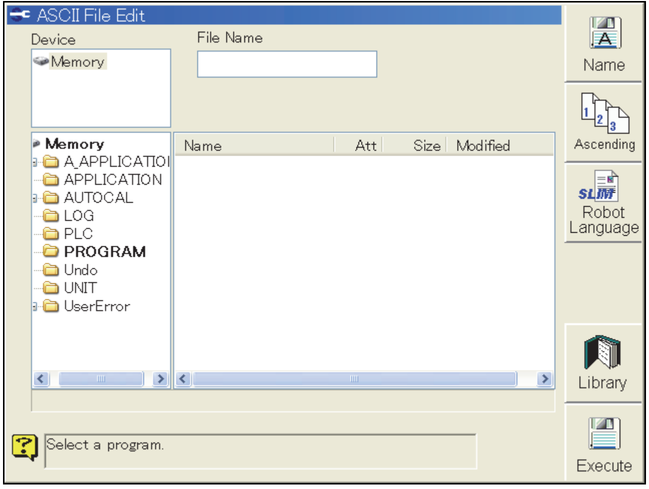
Use of the "ASCII file editing function" enables robot language files to be directly edited using the teach pendant of the AX control unit.
This is ideal for correcting mistakes made in writing the programs and making other simple modifications.

3.2.1 Basic file editing operations

Service Utilities

1 **Press [Service Utilities] f key.**
≫Service menu list will appear.

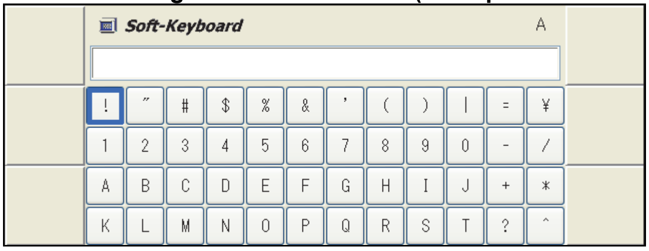
2 **Select [15 ASCII File Edit] and press [Enter].**
≫ASCII file editing screen shown below will appear.



Select a program.

Listed on this screen are only those files which can be edited using the ASCII file editing function (edit-enable files). The term "edit-enable files" refers to robot language files, that is to say, files which have "-A" added onto the end of their filenames to indicate that the files are text files and which have an extension in the form of a number (files such as "`NB4-02-`**`A`**`.010`"), and also files with the "TXT" extension.

3 **When creating a new file, align the cursor with [File Name], and input the name of the new file to be created.**
Open the soft keyboard by pressing [ENABLE] +[EDIT], and input the entire character string with no omissions. (Example: NB4-02-A.010)

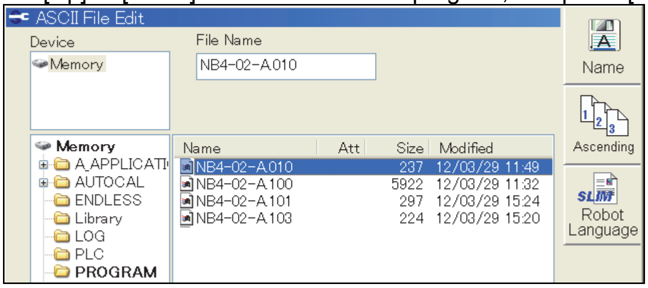


Complete

4 **Press f12 [Complete] key upon completion of the character input.**
>> Back to [ASCII File Edit] screen.

⇧ ⇩ ➡ ↵

5 **If a previously created robot language file is to be selected, select the file to be edited by following the steps below.**
If the robot language file is stored in the internal memory, select "Internal memory" for the device and "PROGRAM" for the folder. (In actual fact, this is the ¥WORK¥PROGRAM folder.)
>>A screen listing the edit-enable files such as the one shown below appears. Use the [Up] or [Down] cursor to select the program, and press [Enter].


ASCII File Edit

Device: Memory

File Name: NB4-02-A.010

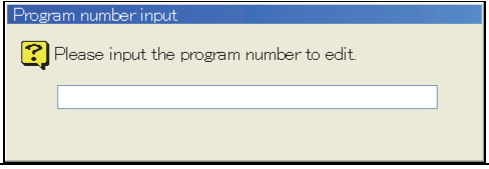
Memory
- A_APPLICATI
- AUTOCAL
- ENDLESS
- Library
- LOG
- PLC
- PROGRAM

Name	Att	Size	Modified
NB4-02-A.010		237	12/03/29 11:49
NB4-02-A.100		5922	12/03/29 11:32
NB4-02-A.101		297	12/03/29 15:24
NB4-02-A.103		224	12/03/29 15:20

If the robot language file is stored in a device other than the internal memory, select the device concerned and folder, and then select the file to be edited in the same way.

Robot Language

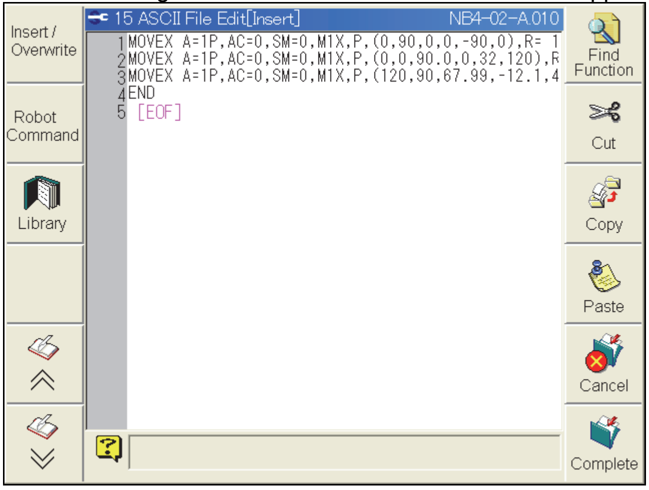
[When selecting a file among the robot language files]
Under the condition of "Memory" for "Device" and "PROGRAM" for "folder, it is possible to select a program for the current unit by inputting a program number.
>>A screen such as the one shown below appears. Input the program number and press [Enter].


Program number input

Please input the program number to edit.

Execute

6 **Check the filename displayed. If it's the correct press the f12 [Execute] key.**
>>A text editing screen such as the one shown below appears.


15 ASCII File Edit[Insert] NB4-02-A.010

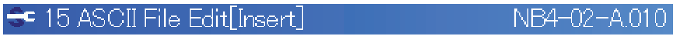
```
MOVEX A=1P,AC=0,SM=0,M1X,P,(0,90,0,0,-90,0),R= 1
MOVEX A=1P,AC=0,SM=0,M1X,P,(0,0,90.0,0,32,120),R
MOVEX A=1P,AC=0,SM=0,M1X,P,(120,90,67.99,-12.1,4
END
 [EOF]
```

In the example shown, a previously created program has been selected. When a new program number has been specified, nothing appears in center editing screen.

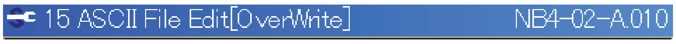
Insert / Overwrite

7 **Each time the f1 [Insert/Overwrite] key is pressed, the mode is switched between insert and overwrite.**
>>Whether the insert or overwrite mode has been established is displayed all the time on the taskbar.

"Insert"


15 ASCII File Edit[Insert] NB4-02-A.010

"Overwrite"

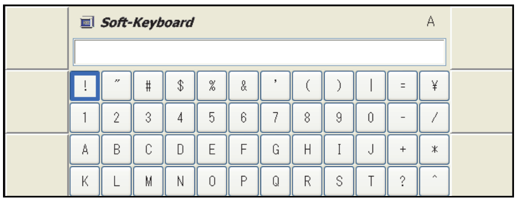

15 ASCII File Edit[OverWrite] NB4-02-A.010

+ ▢ + ⇧ ⇩

8 **Use the [Up], [Down], [Left] and [Right] cursor keys to move the cursor to the position where the editing is to be performed using the cursor keys. The page can be scrolled up or down using [ENABLE] + [Up] or [Down] cursor key.**

9 **To input numerical values, use the number keys [0] to [9].**

10 **To input characters such as letters using keys other than number keys, press the [ENABLE] + [EDIT] keys to display the soft keyboard, and then input them.**



If the f12 [Complete] key is pressed upon completion of the character input, the original screen is restored.

11 **When the [BS] key is pressed, the character to the left of the cursor position is deleted.**

12 **When the [DEL] key is pressed, the character to the right of the cursor position is deleted.**

13 **When the [Enter] key is pressed, a line feed (CR+LF) is input.**
>>In actual fact, there is no CR+LF display. The next line appears on the display, and the cursor moves to the next line.

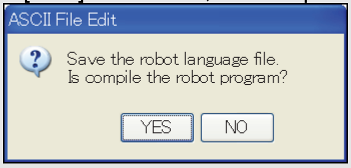
14 **When the [FN] key is pressed, a space can be input.**

15 **When the f5 or f6 key is pressed, the cursor can be made to jump to the start or end of each file.**

16 **Upon completion of the editing, press the f12 [Complete] key.**
>>What has been edited is now saved in the file. The data is saved without checking for syntax errors, etc. Since no backup files are left, care must be taken in executing these operations.
When the file saving process has completed, the [ASCII File Edit] screen will close.

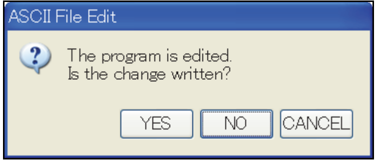
POINT

If a robot language source program file is created or modified in the "¥WORK¥PROGRAM" folder in the [Memory], a confirmation message for compile execution like the one shown as below is displayed when saving the file.
If [YES] is selected, the compilation process will start.


ASCII File Edit

Save the robot language file.
Is compile the robot program?

17 **If the editing is to be canceled and the file is not going to be saved, press the f11 [Cancel] or [RESET/R] key.**
>>A pop-up message requesting confirmation, such as the one shown below, now appears.


ASCII File Edit

The program is edited.
Is the change written?

If “NO” is selected, the display will return to the menu screen of each service without saving the files, destroying all the contents that were edited, and will end editing.
If “CANCEL” is selected, the pop-up message will disappear, returning to the [15 ASCII File Edit] screen, and editing can be continued.

POINT

After creating robot language program with [ASCII file edit], beware to proceed compiling to convert to the executable file with [Service utility]->[9 Program conversion]->[8 Language conversion].
Unless otherwise executing compiling, modified program is never affected to the executable program.

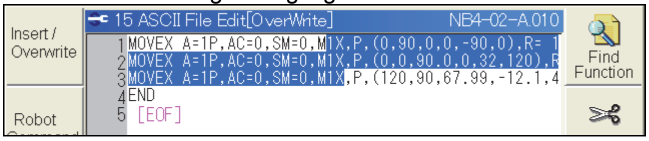
3.2.2 Useful editing functions

Delete multiple lines at one time

Cut

1 **Press the f8 [Cut] key, and select the lines to be deleted using the [Up] or [Down] cursor key.**

>>The selected range is highlighted.



```
MOVEX A=1P,AC=0,SM=0,M1X,P,(0,90,0,0,-90,0),R= 1
MOVEX A=1P,AC=0,SM=0,M1X,P,(0,0,90.0,0,32,120),R
MOVEX A=1P,AC=0,SM=0,M1X,P,(120,90,67.99,-12.1,4
END
 [EOF]
```

⇧ ⇩ ➡ ↵

2 **Select the range that is to be deleted by using the [Up] or [Down] cursor keys and press [Enter].**

>> By using the [Up] or [Down] cursor keys, a multiple number of lines can be selected as the deleting range. The selected range will be highlighted and when [Enter] is pressed, all of those highlighted lines will be deleted. After the lines have been deleted, the lines that were undeleted will replace the deleted lines, without altering its order, and be re-displayed.

To cancel this at any time, press the [RESET/R] key.

Move multiple lines at one time

Paste

1 **After selecting multiple lines in the same way as line deletion, put the cursor to the position right after the target place and press [Paste].**

≫The multiple lines deleted are now inserted immediately after the line with the cursor.

Copy multiple lines at one time

Copy

Paste

1 **Press f key [Copy] to select the copied range in the same way as line deletion**

2 **Put the cursor to the position right after the target place and press [Paste].**

≫The multiple lines deleted are now inserted immediately after the line with the cursor.

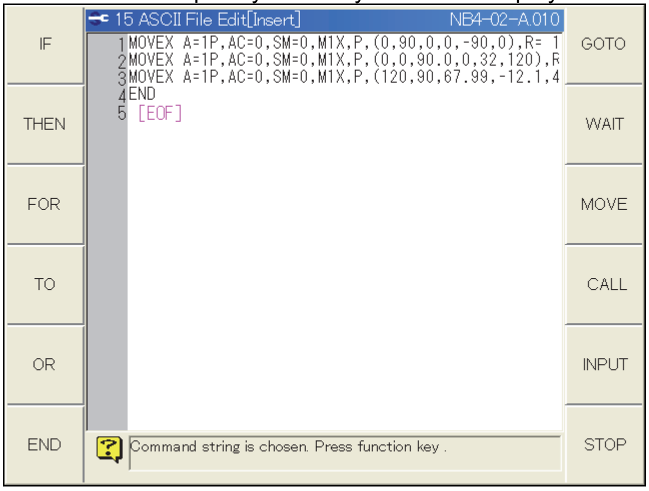
3.2.3 Inputting the basic instructions easily

If the AX control unit is not equipped with a keyboard, it will take some time to input commands. For this reason, provision has been made to enable just the frequently used commands such as `IF, THEN and ELSE` to be recorded easily.
A total of 24 key words can be recorded with a single action using the f keys.

1 Move the cursor to the location where the key word is to be input.

Robot Command

2 Press the f2 [Robot Command] key.
>>The first 12 frequently used key words are displayed.

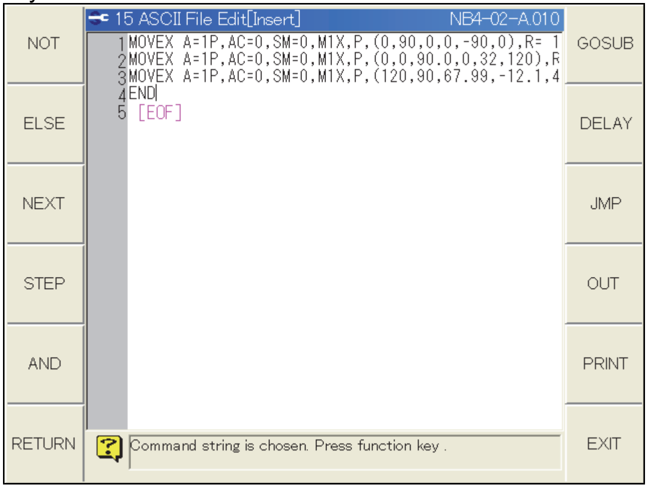


15 ASCII File Edit[Insert] NB4-02-A.010

```
MOVEX A=1P,AC=0,SM=0,M1X,P,(0,90,0,0,-90,0),R= 1
MOVEX A=1P,AC=0,SM=0,M1X,P,(0,0,90.0,0,32,120),R
MOVEX A=1P,AC=0,SM=0,M1X,P,(120,90,67.99,-12.1,4
END
 [EOF]
```

Command string is chosen. Press function key .

3 Press the [ENABLE] key.
>>The next 12 frequently used key words are displayed. In this way, the display of 24 key words can be selected.



15 ASCII File Edit[Insert] NB4-02-A.010

```
MOVEX A=1P,AC=0,SM=0,M1X,P,(0,90,0,0,-90,0),R= 1
MOVEX A=1P,AC=0,SM=0,M1X,P,(0,0,90.0,0,32,120),R
MOVEX A=1P,AC=0,SM=0,M1X,P,(120,90,67.99,-12.1,4
END
 [EOF]
```

Command string is chosen. Press function key .

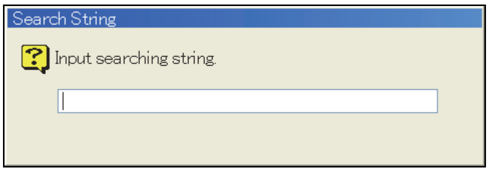
IF

4 Press the desired f key (such as the f1 [IF] key).
>>The f keys return to their original editing screen statuses, and the key word is inserted at the cursor position.
>>The original display screen is restored by the [RESET/R] key.

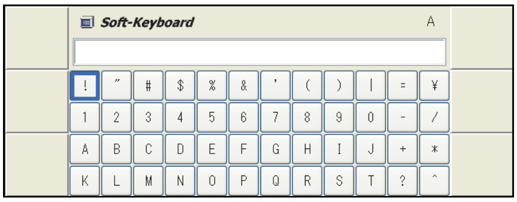
3.2.4 Searching character strings

Any character string can be searched inside the area of the file being edited.

1 **Press the f7 [Find Function] key.**
>>A dialog box, such as the one shown below, for inputting the character string to be searched now appears.



2 **Input the character string to be searched.**
If the character string is a numerical value, input it directly using the number keys. To input characters such as letters using keys other than number keys, press the [ENABLE] + [EDIT] keys to display the soft keyboard, and then input them.



If the f12 [Complete] key is pressed upon completion of the character input, the screen will go back to the dialog box for inputting the character string to be searched.

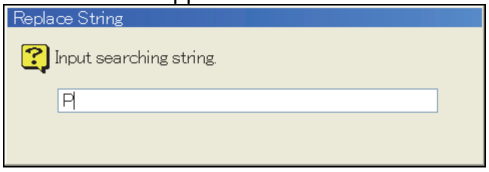
3 **Once the character string has been entered, press [Enter] key.**
>>The search is commenced. The character string is searched from the beginning of the program, the cursor is positioned at the location where it has been found, and the display is updated. For instance, if the "`MOVEX`" character string is searched, the cursor will move to "`M`" at the head of the first character string which has been found.

4 **To continue with the search, press the [ENABLE] + f7 <Find Function> keys.**
>>The cursor now moves to the next "`MOVEX`" character string which has been found. The display does not return from the end of the program to its beginning.

3.2.5 Replacing strings

Any character string can be replaced inside the area of the file being edited.

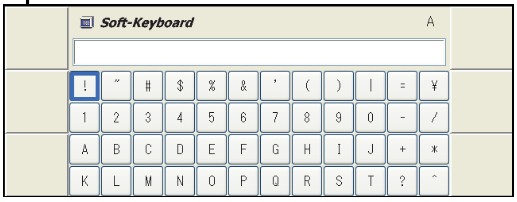
1 **Press the [ENABLE] + <Replace> key.**
>>A dialog box, such as the one shown below, for inputting the character string to be searched now appears.



Input searching string.

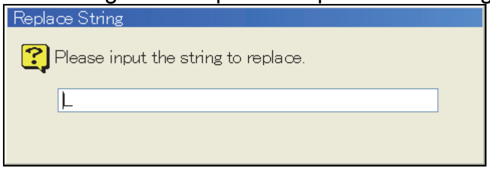
P

2 **Input the character string to be searched.**
If the character string is a numerical value, input it directly using the number keys. To input characters such as letters using keys other than number keys, press the [ENABLE] + [EDIT] keys to display the software keyboard, and then input them.



If the f12 <Complete> key is pressed upon completion of the character input, the screen will go back to the dialog box for inputting the character string to be searched.

3 **Once the character string has been entered, press [Enter] key.**
>>A dialog box to input the replacement strings will be displayed.

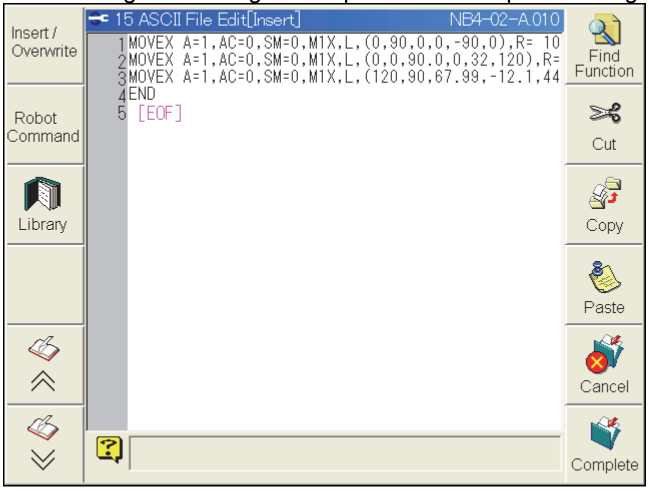


Please input the string to replace.

L

4 **Input the character string for replacement.**
If the character string is a numerical value, input it directly using the number keys. To input characters such as letters using keys other than number keys, press the [ENABLE] + [EDIT] keys to display the software keyboard, and then input them.
>>Input the string in the same way.

5 **Once the character string has been entered, press [Enter] key.**
>>The designated strings are replaced to the inputted strings.



```
MOVEX A=1,AC=0,SM=0,M1X,L,(0,90,0,0,-90,0),R= 10
MOVEX A=1,AC=0,SM=0,M1X,L,(0,0,90.0,0,32,120),R=
MOVEX A=1,AC=0,SM=0,M1X,L,(120,90,67.99,-12.1,44
END
 [EOF]
```

>>Press [Reset/R] key to cancel.

3.3 Creating pose files

3.3.1 Outline of pose files

A "Pose file" is a data file that consists of "Pose variables"(positions and postures).

In robot language, it is possible to write the robot positions or postures in values directly in a `MOVEX` command. However, it is difficult to move the robot accurately following the design because of the installation position error etc. Furthermore, when the position modifications becomes necessary because of the layout change, or, when the same teaching point is used in many steps in one program, the modification work for the all move commands in the original source program would take long time.

To solve this problem, in robot language, it is possible to handle the plural pose variables altogether in one data file. This is a "Pose file". The respective pose variables can be made or modified by moving the actual robot and recording the actual position. And, it is possible to move the robot by selecting the pose file number and calling the pose variable using the `MOVEX` command.

(Example)

```
REM "Selecting a pose file No.10"
USE 10
REM "Call the pose number 1,2, and 3 to move the robot"
MOVEX A=1,M1J,P,P1,R=10.0,H=1,MS
MOVEX A=1,M1J,P,P2,R=10.0,H=1,MS
MOVEX A=1,M1J,P,P3,R=10.0,H=1,MS
```

For example, when the robot type is `NB4-02`, the pose file is stored in the internal memory under filenames such as the following.

(☞See the item of file name in the section of "1.1.2 Characteristics and precautions")

"`NB4-02-P.nnn`" (nnn is the pose file number)

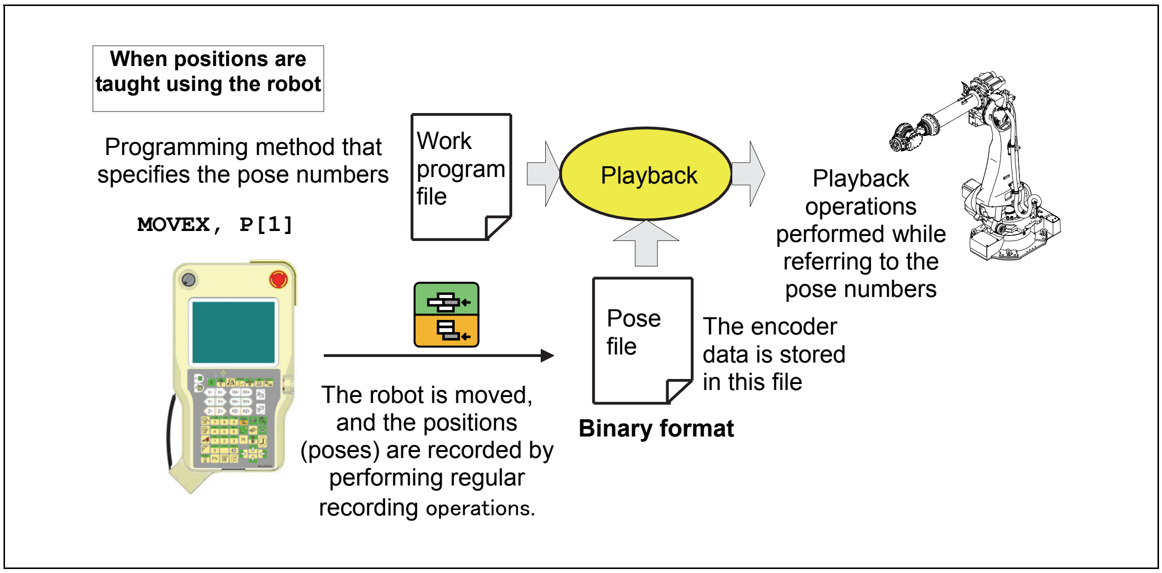

When positions are taught using the robot
Programming method that specifies the pose numbers
MOVEX, P[1]
Work program file
Playback
Playback operations performed while referring to the pose numbers
Pose file
The encoder data is stored in this file
Binary format
The robot is moved, and the positions (poses) are recorded by performing regular recording operations.

Fig 3.2 Robot language programs that use pose files

The pose files are not referenced at the compiling stage. They are referenced during playback operations. There is no need for there to be one pose file for each robot language program and vice versa. Multiple pose files can be prepared to support different types of work and then a playback operation can be performed selecting the desired pose file by the "`USE`" command.

3.3.2 Recording pose files

The operation method used for pose files is in no way different from the one used for regular teaching. All that needs to be done is to "make a pose recording declaration" before starting the recording operation. The programs which are actually created are the same as the execute form programs created by regular teaching.

During a recording operation which has been declared to be a pose recording, it is not possible to record any instructions other than those involved with poses (robot positions), that is to say, any application instructions.

Entering the pose recording mode

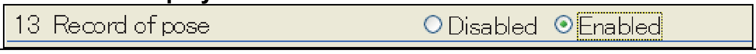
Teach

1 **Establish the teach mode.**

Teach/Play Condition

2 **Select [1 Teach/Playback Condition] from Service Utilities.**
Alternatively, press the f7 <Teach/Playback Conditions> key.

3 **Align the cursor with [13 Record of pose] on the list of teach/playback conditions displayed.**


13 Record of pose ○ Disabled ⊙ Enabled

+ + ⇦ ⇨

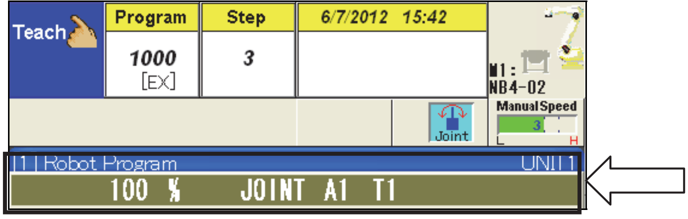
4 **Use the [ENABLE] + [Left] or [Right] cursor key to set pose recording to [Enable].**

Complete

5 **Press the f12 <Complete> key.**
>>In the course of the subsequent recording, only pose files will be recorded.
Regular task programs cannot be taught.
(This state is retained even when the main power of the controller is turned off.)

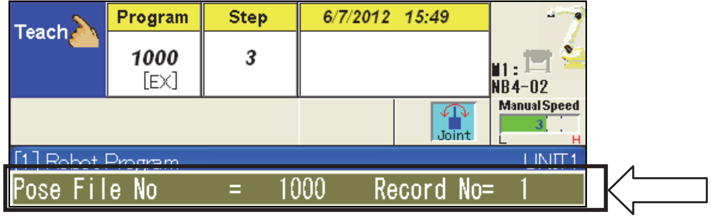
6 **When the display returns to the mode screen, the recording status display area (line below the blue title bar line) on the program monitor screen serves as the display field for the pose file number and recording number. Here, it is possible to determine whether pose recording is enabled or disabled.**

Record of pose <Disabled>



100 % JOINT A1 T1

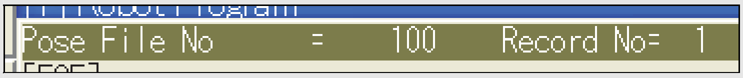
Record of pose <Enabled>



Pose File No = 1000 Record No= 1

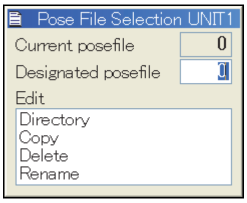
POINT

The "Program" and the "Step" displayed in the status window at the top of the screen (No.100 in the example shown in the figure) has nothing to do with the operation to select the pose files. The current pose file and the current pose number can be confirmed in the line shown as below.


Pose File No = 100 Record No= 1

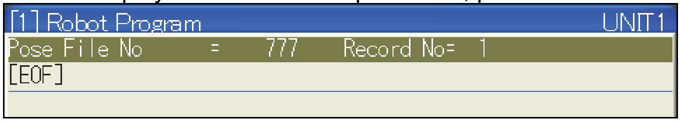
Recording a pose in the pose file (pose recording mode)

1 **Press [ENABLE] + [PROGRAM/STEP] key at the same time.**
≫ Following [pose fie selection] screen will appear.



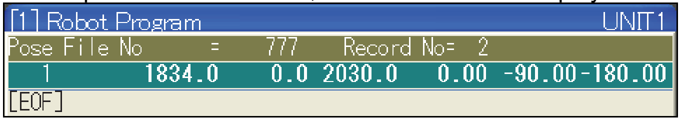
```
Pose File Selection UNIT1
Current posefile        0
Designated posefile     0
Edit
Directory
Copy
Delete
Rename
```

2 **Put the cursor to ”Designated pose file” and input the pose fie number and press [Enter].**
≫The number of the selected pose file appears in the pose file number and recording number display field. In the example shown, pose file No.777 has been selected.



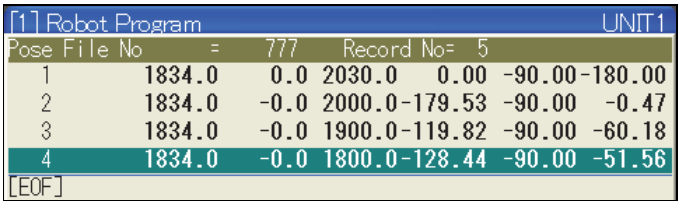
```
[1] Robot Program                                  UNIT1
Pose File No     =   777   Record No=  1
[EOF]
```

3 **Turn on the motor power, move the robot using the axis operation keys in the same way as for regular recording operations, and press the [REC] button.**
>>One pose will be recorded, and its data will be displayed as shown below.



```
[1] Robot Program                                  UNIT1
Pose File No     =   777   Record No=  2
   1         1834.0    0.0  2030.0    0.00  -90.00-180.00
[EOF]
```

4 **Every time the [RECORD] key is pressed, new poses will be recorded one after another.**



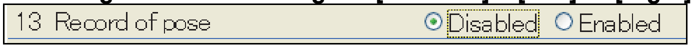
```
[1] Robot Program                                  UNIT1
Pose File No     =   777   Record No=  5
   1         1834.0    0.0  2030.0    0.00  -90.00-180.00
   2         1834.0   -0.0  2000.0-179.53  -90.00   -0.47
   3         1834.0   -0.0  1900.0-119.82  -90.00  -60.18
   4         1834.0   -0.0  1800.0-128.44  -90.00  -51.56
[EOF]
```

5 **The positions can be checked by selecting the step as with regular tasks and then using CHECK GO/BACK.**

6 **After recording the last pose, just proceed to the steps described in the next section "Ending the pose recording declaration."**
There is no need to record the END or other instructions. (In the pose recording status, it is not possible to record any application instructions.)

Exiting the pose recording mode

1 **Select <Teach/Playback Condition> and [13 Record of pose], and return pose recording to "disable" using the [ENABLE] + [Left] or [Right] cursor key.**



13 Record of pose ◉ Disabled ○ Enabled

2 **Press the f12 <Complete> key.**
>>From this point on, regular task programs can be taught. Pose files cannot be recorded. This state is retained even when the main power of the controller is turned off.

How to enable/disable the Pose recording mode using a short cut R code

Instead of using the “Teach / Playback condition” screen, the following Shortcut R-code can be used to enable/disable the Pose recording mode.

Teach

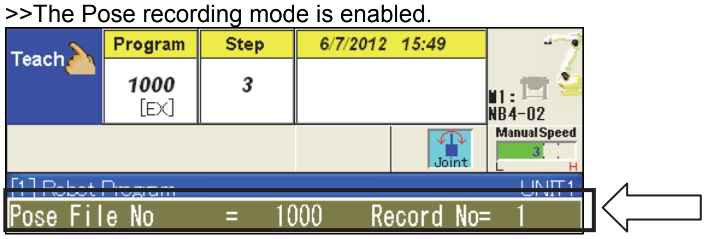
1 **Select the Teach mode.**

R

2 **To enable the Pose recording mode, enter the following code.**

R 1 1 ↵ 1 ↵

>>The Pose recording mode is enabled.



Teach	Program	Step	6/7/2012 15:49
	1000 [EX]	3	

M1: NB4-02

Manual Speed 3

Joint

[1] Robot Program UNIT1

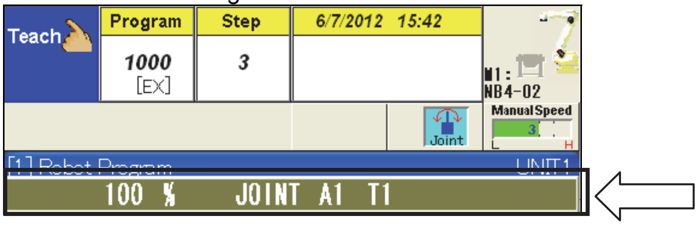
Pose File No = 1000 Record No= 1

R

3 **To disable the Pose recording mode, enter the following code.**

R 1 1 ↵ 0 ↵

>>The Pose recording mode is disabled.



Teach	Program	Step	6/7/2012 15:42
	1000 [EX]	3	

M1: NB4-02

Manual Speed 3

Joint

[1] Robot Program UNIT1

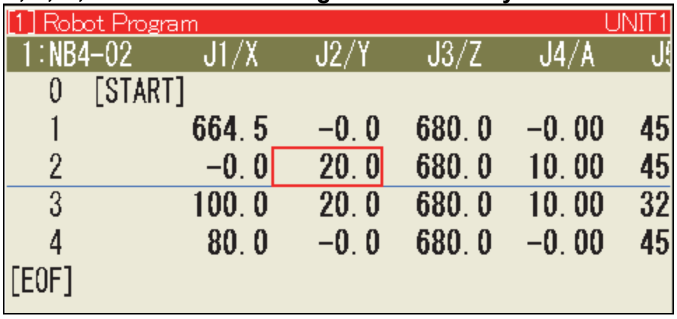
100 % JOINT A1 T1

3.3.3 Modifying pose files

1 **Interim poses can be deleted using [ENABLE] + [DEL].**
However, even when an interim pose is deleted, the subsequent poses will not be adjusted forward. A deleted pose number will become a missing number.

2 **Pose positions can be modified using [ENABLE] + [MOD Position].**

3 **Modifications can be made using manual input on an axis by axis basis by screen editing.**
Press the [EDIT] key, open the screen editing screen, and input directly the X, Y, Z, R, P and Y values using the number keys.



```
[1] Robot Program                                  UNIT1
1:NB4-02        J1/X     J2/Y     J3/Z     J4/A     J5
   0  [START]
   1            664.5    -0.0    680.0    -0.00    45
   2             -0.0    20.0    680.0    10.00    45
   3            100.0    20.0    680.0    10.00    32
   4             80.0    -0.0    680.0    -0.00    45
[EOF]
```

POINT

- Although the pose variable data is displayed in a format of (X, Y, Z, roll, pitch, yaw), the internal data format is encoder value of each axis. So when a big modification is added to the pose variable data, the robot posture may change in unexpected way. Please do not forget to check the robot posture using CHECK GO operation.

- The Pose variables are automatically converted to encoder values though the displayed values are in the format of (X, Y, Z, roll, pitch, yaw). So the CONF setting is ignored when the MOVEX command is executed.

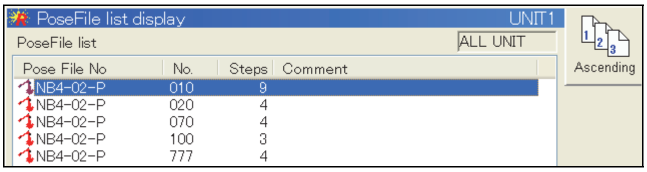
3.3.4 Displaying a list of pose files

A list of only pose files can be displayed.

This is useful for checking the numbers of empty files. It is also possible to select an already recorded pose file from the list.

1 **Set pose recording to "Enable."**

2 **Input the R17 program number list shortcut.**
>>A list of only pose files such as the one shown below appears.



PoseFile list display UNIT1

PoseFile list ALL UNIT

Pose File No	No.	Steps	Comment
NB4-02-P	010	9	
NB4-02-P	020	4	
NB4-02-P	070	4	
NB4-02-P	100	3	
NB4-02-P	777	4	

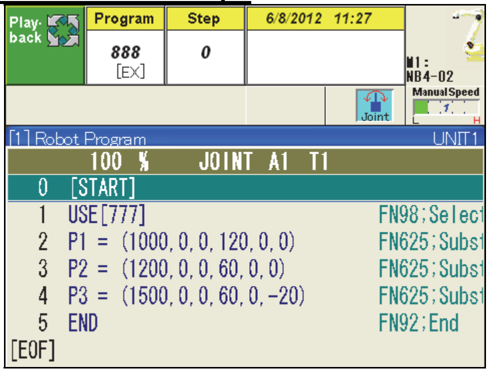
3 **Move the cursor to the designated pose file and press [Enter].**
≫ Pose file selected is displayed. If pressing [Reset] instead of [Enter], then back to the previous teach screen.

3.3.5 Renewing of pose file

This section describes the robot behavior or the pose file access operation when pose calculation or pose assignment command was executed in the pose recording mode (☞ Refer to "3.3.2 Recording pose files").

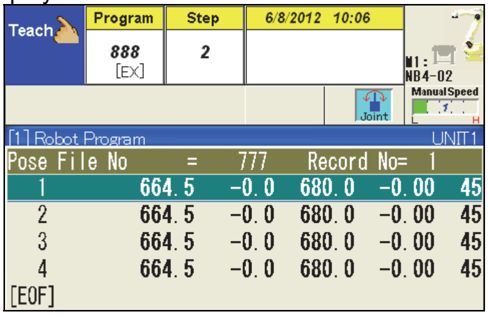
For example, consider about executing “robot program 888” which includes pose assignment command. As the result, “pose 1 to 3” of “pose file 777” **in the work memory** is changed. Next time when executing “pose file 777”, robot moves to the new “pose 1 to 3” which pose assignment is completed.

But in this moment, this new “pose 1 to 3” is just stored in the work memory, **so “pose file 777” itself is not renewed yet**.



```
Play-back  Program 888 [EX]  Step 0  6/8/2012 11:27
M1:NB4-02  ManualSpeed  Joint
[1] Robot Program                              UNIT1
        100 %      JOINT  A1   T1
   0  [START]
   1  USE[777]                         FN98;Select
   2  P1 = (1000,0,0,120,0,0)          FN625;Subst
   3  P2 = (1200,0,0,60,0,0)           FN625;Subst
   4  P3 = (1500,0,0,60,0,-20)         FN625;Subst
   5  END                              FN92;End
[EOF]
```

Switch to the teach mode after executing “robot program 888”, then “pose file 777” is already displayed in the monitor screen.

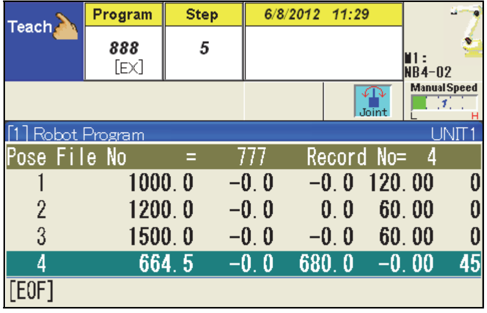


```
Teach  Program 888 [EX]  Step 2  6/8/2012 10:06
M1:NB4-02  ManualSpeed  Joint
[1] Robot Program                              UNIT1
Pose File No    =    777    Record No=  1
   1        664.5   -0.0   680.0   -0.00   45
   2        664.5   -0.0   680.0   -0.00   45
   3        664.5   -0.0   680.0   -0.00   45
   4        664.5   -0.0   680.0   -0.00   45
[EOF]
```

POINT

The result of executing “robot program 888” is just affected to the internal work memory. The displayed pose data on the screen is just the content of the pose file so the data does not show the content in the internal work memory.

However, when pressing [RECORD] key after selecting pose 4, the current robot position will be recorded in “pose 4” and “pose 1 to 3” is renewed to the new data that pose assignment command is completed. At this moment, “pose file 777” is overwritten.



```
Teach  Program 888 [EX]  Step 5  6/8/2012 11:29
M1:NB4-02  ManualSpeed  Joint
[1] Robot Program                              UNIT1
Pose File No    =    777    Record No=  4
   1       1000.0   -0.0   -0.0  120.00   0
   2       1200.0   -0.0    0.0   60.00   0
   3       1500.0   -0.0   -0.0   60.00   0
   4        664.5   -0.0  680.0   -0.00   45
[EOF]
```

POINT

Beware that all pose (1 to 4) is overwritten at the same time although only “pose 4” is selected. Because the pose 1- 3 calculated during the playback operation will also be overwritten.

IMPORTANT

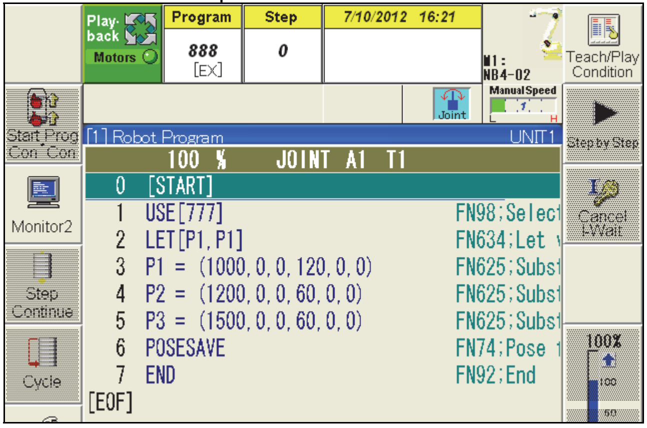
As described above, when executing robot program while in the pose recording mode, monitor display and real robot position may not match. So while in the pose recording mode, robot program had not better been executed to avoid miss-operation.

3.3.6 Command to save pose file

This command is to save the result of pose calculation or pose assignment onto the pose file that is designated by `USE` command.

Normally the result of pose calculation or pose assignment is not saved onto the pose file (☞ Refer to "3.3.5 Renewing of pose file"), so FN74 `POSESAVE` (Pose File Save) is necessary to save these result to pose file.

Following example shows that “robot program 888” includes the pose save command (FN74 `POSESAVE`) and the pose assignment. As the result of playback “robot program 888”, “pose 1 to 3” data is saved onto “pose file 777” and affected the monitor display data in Teach mode.



```
[1] Robot Program                              UNIT1
        100 %      JOINT  A1  T1
   0  [START]
   1  USE[777]                          FN98;Selec
   2  LET[P1,P1]                        FN634;Let
   3  P1 = (1000,0,0,120,0,0)           FN625;Subs
   4  P2 = (1200,0,0,60,0,0)            FN625;Subs
   5  P3 = (1500,0,0,60,0,0)            FN625;Subs
   6  POSESAVE                          FN74;Pose
   7  END                               FN92;End
[EOF]
```

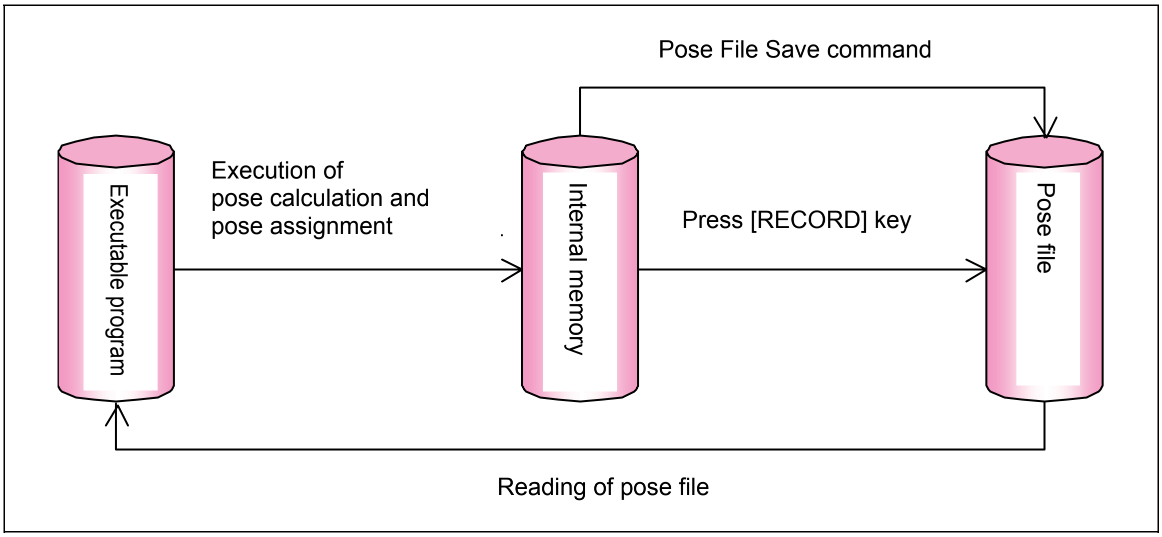


Fig 3.3 Mechanism of pose file renewing

NOTE

Chapter 4 Compiling programs

This chapter describes how to compile robot language programs. Once compiled, the programs are checked for syntax errors, etc., and they are converted into an executable format to enable them to be played back.

4.1 Compiling

The robot language programs (ASCII files) which have been created cannot be executed unless they are first compiled. The objective of compiling is not only to convert the files into a format which can be executed by the robot but also to find syntax errors. Compiling is accomplished by the following the steps below.

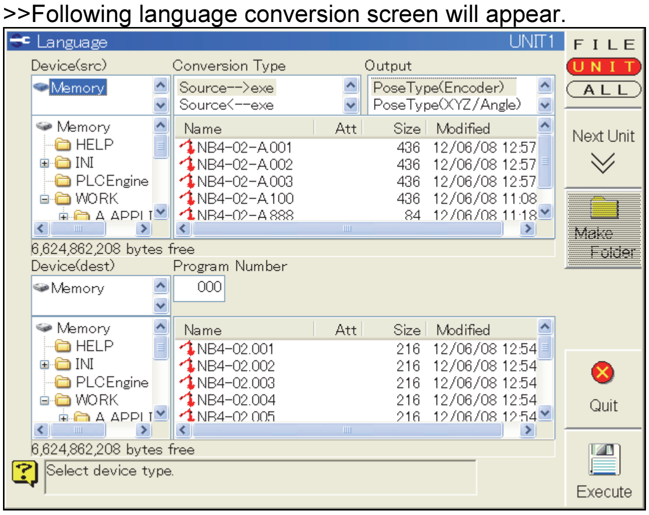
Service Utilities

1 **Press <Service Utilities> f key.**

≫ Service menu list will appear.

2 **Among the service menu list, select [9 Program conversion] - [8 Language] and press [Enter].**

>>Following language conversion screen will appear.



Language UNIT1

Device(src) Conversion Type Output

Memory | Source-->exe / Source<--exe | PoseType(Encoder) / PoseType(XYZ/Angle)

Memory: HELP, INI, PLCEngine, WORK, A APPLI

Name	Att	Size	Modified
NB4-02-A.001		436	12/06/08 12:57
NB4-02-A.002		436	12/06/08 12:57
NB4-02-A.003		436	12/06/08 12:57
NB4-02-A.100		436	12/06/08 11:08
NB4-02-A.888		84	12/06/08 11:18

6,624,862,208 bytes free

Device(dest) Program Number

Memory 000

Memory: HELP, INI, PLCEngine, WORK, A APPLI

Name	Att	Size	Modified
NB4-02.001		216	12/06/08 12:54
NB4-02.002		216	12/06/08 12:54
NB4-02.003		216	12/06/08 12:54
NB4-02.004		216	12/06/08 12:54
NB4-02.005		216	12/06/08 12:54

6,624,862,208 bytes free

Select device type.

3 **Select "Source → exe" as the conversion type.**

4 **The output type box is an option selected for reverse compiling ("Source ← exe") and, as such, it is not specified to compile programs.**

FILE UNIT ALL

5 **Press f7<FILE> to switch the file view so that your desired file will appear.**

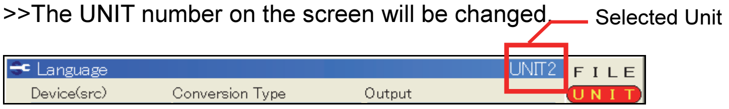
- * To switch the file view, the operator qualification of ***EXPERT*** or higher is required.
- * This is available only when compiling.

FILE UNIT ALL	The task programs in the selected unit are displayed.
FILE UNIT ALL	All the task programs are displayed. This even enables to display task programs in the unit which is not registered in the current robot controller.

Next Unit

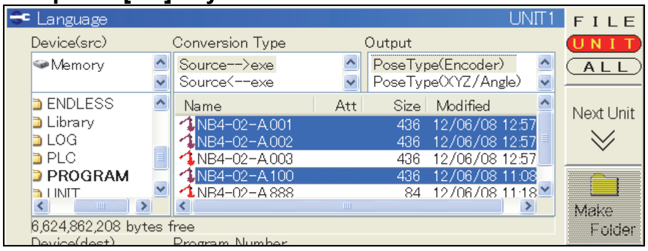
6 **Press f8<Next Unit> to switch the UNIT No.**
When the selected file view is "UNIT", the file list will be updated.
The UNIT selected here is to be used as the target UNIT in conversion operation.

>>The UNIT number on the screen will be changed.



The unit switched at this point is intended for the manual operation

7 **Specify the file to be compiled. To cancel the selection, select the target file and press [BS] key.**



Language UNIT1

Device(src): Memory

Conversion Type: Source-->exe / Source<--exe

Output: PoseType(Encoder) / PoseType(XYZ/Angle)

Folders: ENDLESS, Library, LOG, PLC, PROGRAM, UNIT

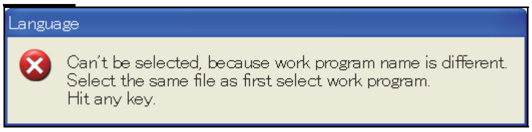
Name	Att	Size	Modified
NB4-02-A.001		436	12/06/08 12:57
NB4-02-A.002		436	12/06/08 12:57
NB4-02-A.003		436	12/06/08 12:57
NB4-02-A.100		436	12/06/08 11:08
NB4-02-A.888		84	12/06/08 11:18

6,624,862,208 bytes free

Device(dest) Program Number

However, when the setting is "ALL" and the files like ones shown as below are selected at the same time, a warning message will appear, failing to select those files.

SRA166-1-A.001
UNIT2-A.1002



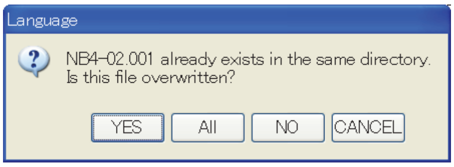
Language

Can't be selected, because work program name is different.
Select the same file as first select work program.
Hit any key.

8 **Next, in the same way, specify the device, folder and file which serve as the compiling destination.**
At the compiling stage, it is possible to change the numbers of the programs. To do so, align the cursor with the program number field of the device (destination), and input the program number using the number keys.
However, it is not possible to change the program number when two or more files have been selected. In this case, the execution file will be created by a program number of the selected file.

Execute

9 **Press the f12 [Execute] key.**
>>Compiling now starts for the selected file.
When a program number already existing in the memory has been selected, a pop-up message such as the one below appears. Select [YES] or [NO] using the [Left] or [Right] cursor key, and press the [Enter] key.

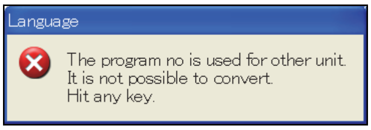


Language

NB4-02.001 already exists in the same directory.
Is this file overwritten?

YES	Overwrites.
All	Overwrites all files if already existing. This is displayed when selecting two or more files, while not displayed once after selected.
NO	Does not overwrite, but converts the next file.
CANCEL	Cancels conversion.

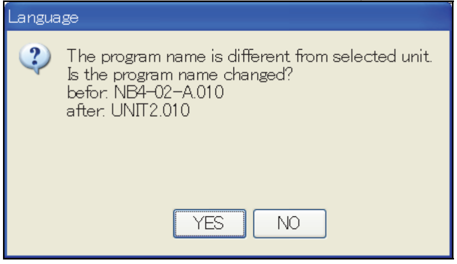
When the program number after conversion has been already used in the program in a different unit, the following pop-up message appears, which fails to execute the conversion. In this case, delete the file in advance or change the program number in advance to convert it.



Language

The program no is used for other unit.
It is not possible to convert.
Hit any key.

The following pop-up message appears if executing a file of which name differs from the task program in the currently selected unit. In this case, select the appropriate one with [←][→] key, and press [Enter].

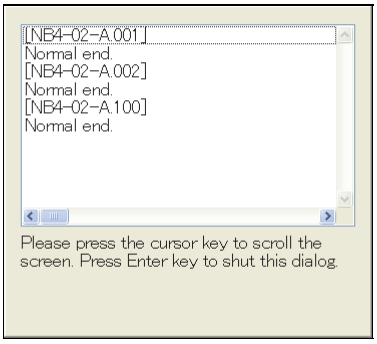


Language

The program name is different from selected unit.
Is the program name changed?
befor: NB4-02-A.010
after: UNIT2.010

YES	Changes a name of the created file.
All	Changes a name of all the created files. This is displayed when selecting two or more files, while not displayed once after selected.
CANCEL	Cancels conversion.

10 **When the compiling has been completed successfully, the "Successfully completed" message appears at the screen center.**
An execute form file which can be played back has now been created.

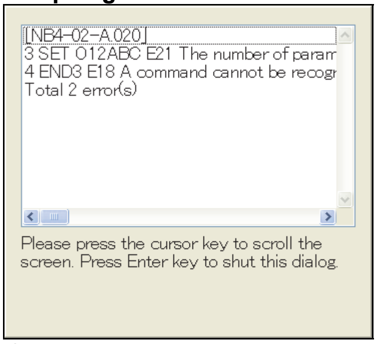


```
[NB4-02-A.001]
Normal end.
[NB4-02-A.002]
Normal end.
[NB4-02-A.100]
Normal end.
```

Please press the cursor key to scroll the screen. Press Enter key to shut this dialog.

[Displayed item]	
[NB4-02-A.1000]	→Conversion File Name
Normal end.	→Result

11 **If errors have been detected during compiling, all the parts where the compiling errors were detected are displayed at the screen center.**



```
[NB4-02-A.020]
3 SET 012ABC E21 The number of param
4 END3 E18 A command cannot be recogr
Total 2 error(s)
```

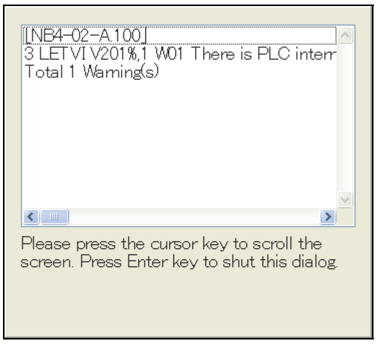
Please press the cursor key to scroll the screen. Press Enter key to shut this dialog.

If there are so many errors that they cannot be displayed on one screen, the error displays can be scrolled using the [Up] or [Down] cursor key.

[Example of a compiling error]	
[NB4-02-A.1006]	→Conversion File Name
2 END3	→This indicates the number of the line where the error was found and a description of the error.
E18 A command Cannot be recognized	→This indicates what kind of error has been found. (☞ Refer to Table 4.1.1 Robot language compiling errors)

12 **When an error has been detected, correct the place indicated by editing the ASCII file, and proceed with compiling again.**
An execute form file is not created until the compiling is successfully completed.

13 **If warnings have been detected during compiling, all the parts where the compiling warnings were detected are displayed at the screen center.**



```
[NB4-02-A.100]
3 LETVI V201%,1 W01 There is PLC interr
Total 1 Warning(s)
```

Please press the cursor key to scroll the screen. Press Enter key to shut this dialog.

[Example of a compiling warning]

[NB4-02-A.1007] →Conversion File Name

3 LETVI V201%,1 → The content and number of the line where warnings were detected.

W01 There is PLC internal variable access. → Displays the content of warning. (☞ Refer to “Table 4.1.2 Robot language compiling warnings”.)

14 **An execute form file will be created when only a warning has been detected. Check the indicated part to see whether the created file can be used or not.**

Table 4.1.1 Robot language compiling errors

Error description	Explanation
E01 Too many characters in one line.	Use up to 254 characters per line.
E02 Illegal line number.	Check that the line number or label name is correct.
E03 Syntax error.	Check that there are no syntax errors.
E04 Numerical value outside prescribed range.	The parameter of the application instruction is outside the prescribed range. Alternatively, the FLOAT or INT variable is outside the input range.
E05 Error in description of calculation expression.	There is an error in an assignment expression. • When there is an error in the argument of a general-purpose function • When the right side is missing in an assignment expression (V1%=) • When the expression ends with an operator (V1%=1+)
E06 Too many characters in an identifier.	The maximum number of characters that can be used for a label is 16.
E07 Illegal label.	Check for symbols that cannot be used for labels.
E08 Too many labels.	Reduce the number of labels or divide up the program.
E09 Same label has been defined twice.	Check for a label with the same name.
E10 Program size has been exceeded.	Divide up the program using program call, etc.
E11 Illegal type of operation expression.	Check whether any numerical values have been assigned to character strings.
E12 Missing "(".	Check that "(" and ")" are paired up.
E13 Missing ")".	Check that "(" and ")" are paired up.
E14 Missing "[".	Check that "[" and "]" are paired up.
E15 Missing "]".	Check that "[" and "]" are paired up.
E16 Missing "$".	A character string type is required.
E17 Illegal register.	Check the register numbers.
E18 Cannot recognize instruction.	Check for incorrect command names and function names.
E19 Illegal step number.	Check whether the specified step number exists. Step numbers range from 0 to 999.
E20 Illegal program number.	Program numbers range from 0 to 9999.
E21 Number of parameters do not match.	Check the functions or commands.
E22 No THEN.	Check THEN.
E23 Must be ELSE.	Check for instructions in the IF statements.
E24 Excessively long program on one line.	The instruction on one line is too long due to the functions and parentheses used.
E25 No GOTO.	No GOTO corresponding to ON instruction.
E26 Error in interpolation description.	Check the interpolation parameter settings.

Error description	Explanation
E27 No comma.	Check whether there are not enough multiple parameters.
E28 Missing "=".	There is no part to which the initial value is to be assigned in the FOR statement.
E29 Error in accuracy description.	Check the accuracy number.
E30 Error in speed description.	Check whether the maximum speed is exceeded.
E31 Error in tool description.	Check for errors in the tool numbers.
E32 Too many MOVE instructions (999).	The maximum number of MOVE instructions is 999.
E33 No LET statement in FOR.	There is no part to which the initial value is to be assigned in the FOR statement.
E34 Too many nesting levels for FOR.	Up to 4 nesting levels may be used for FOR.
E35 Number of NEXT's do not match.	Check the number of FOR's and NEXT's.
E36 Illegal input/output device number.	
E37 No TO.	Check that there is a TO in the FOR statement.
E38 Must be STEP.	Check whether a command other than STEP, that indicates the increment, has been described in the FOR statement.
E40 Illegal line number.	Some lines have a line number and some do not.
E41 Instruction which cannot be used in FOR is present.	In the loop between FOR and NEXT, there is a command that jumps out of the loop.
E42 Error in conveyor description.	Check the conveyor register.
E43 Error in CONF description.	Check what is specified for the configuration.
E44 Cannot open temporary file.	A temporarily file could not be opened during conversion.
E45 This speed specification cannot be reverse converted.	An attempt has been made to reverse convert a speed specification existing solely in this controller into the AW format.
E46 This interpolation format cannot be reverse converted.	An attempt has been made to reverse convert an interpolation format existing solely in this controller into the AW format.
E47 Cannot open ASCII file.	The ASCII file to be converted could not be opened.
E48 Cannot open robot program.	The robot program to be converted could not be opened.
E49 Conversion task was forcibly terminated.	The forced termination key was pressed.
E51 Error in acceleration description.	Check what has been specified for the acceleration.
E52 Error in smoothness description.	Check what has been specified for the smoothness.
E53 The label is not found.	
E54 Error is in description of the mechanism.	
E55 Pose calculation failed.	Check the value of pose constant
E56 The error occurred while saving.	
E58 Other compilation processing is executed.	Try to execute after the other ongoing compiling processes are finished.
E59 This interpolation specification is not convertible.	
E60 The translation table overflowed.	The size of conversion table of the INCLUDE command is too large.
E61 A file does no open.	Failed to open the conversion table.
E62 MOVE and MOVEJ are uncorrespondence to seven axis robot.	
E63 ENDW does not match.	Number of WHILE and ENDW does not match. Check them.
E64 Too many nesting levels for WHILE to ENDW.	Up to 4 nesting levels may be used for WHILE to ENDW.
E65 Inadequate usage of flow control statements.	Inadequate command is included.
E66 ENDIF does not match.	Number of IF and ENDIF does not match. Check them.
E67 Too many nesting levels for IF to ENDIF.	Up to 4 nesting levels may be used for IF to ENDIF.
E69 Too many nesting levels for SWITCH to ENDS.	Up to 4 nesting levels may be used for SWITCH to ENDS.
E70 Inadequate usage of SWITCH to ENDS.	This is not allowed to use right after SWITCH command.

Table 4.1.2 Robot language compiling warnings

Warning description	Explanation
W01 There is PLC internal variable access.	The integer and actual variables for the PLC internal variable access are being used. Check whether there are any problems for usage.

4.2 Reverse compiling

The term "reverse compiling" refers to converting an execute form program into a robot language program (ASCII file).

Since plural form are permitted for the movement instructions in robot language programs, so this from must be specified to perform reverse compiling. Except this, all operation is same as compiling.

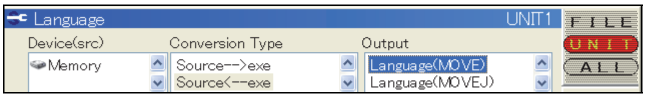
Note that user task programs cannot be reverse compiled.

Automatic indent is not performed when reverse compiling.

Only those operations that differ from compiling are described below.

1 **Select "Source → exe" as the conversion type.**

2 **When the cursor is moved to the output format box, the selection options shown below appear.**
Use the [Up] or [Down] cursor key to align the cursor with the desired movement instruction format, and then press the [Enter] key.



Language UNIT1

Device(src): Memory

Conversion Type: Source-->exe / Source<--exe

Output: Language(MOVE) / Language(MOVEJ)

Table 4.2.1 Form of move command

Output Form	Position expression
MOVEX-X	TCP coordinate (X, Y, Z, r, p, y)
MOVEX-J	Angle of each axis (J1, J2, …, J6)
MOVEX-E	Axis encoders (E1, E2, …, E6)

For further details, refer to "2.4.3 Pose constant" or the instruction manual "Command Reference".

POINT

Notes for the move command form :

When executing the reverse compiling of executable program that was originally created by robot language, its move command form is the original form that was used in the original robot language, irrelevant to the output form (MOVE/MOVEJ/MOVEX and or so) designated in this screen. Because executable program contains the original move command of robot language in its internal information.

Example :
Original robot language program: was “MOVEX-X”.
-> Compiling to create the executable program
-> Reverse compiling to create the robot language program while designating output form as “MOVE-J” in this screen.
-> Created robot language program is “MOVE-X” irrelevant to the designated output form.

4.3 Force execute language conversion

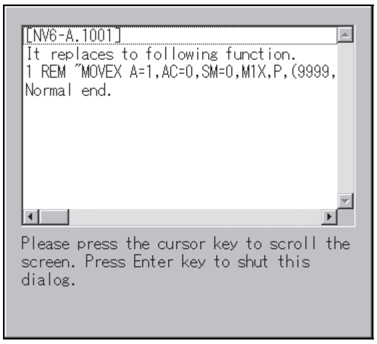
Even if a particular error (☞Refer to “Table 4.3.1 Errors of Force conversion”) occurs in compiling, the compiling process can be forcibly continued. The step that causes an error can be replaced with Comment (REM command). Note that this is available only when compiling ([Language format] → [Execution format]).
The following describes the operation different from compiling process only.

1 **Confirm that "Force execute language conversion" setting is ”ON”.**
*** For details, see**☞ Refer to “4.4Force execute of language conversion“

Execute

2 **Select a desired file to compile, and press f12<Execute>.**

3 **If the language conversion is forcibly executed in compiling, the message will tell you in the center of screen. However when the other type of errors occur, the file of execution format is not to be created.**



```
[NV6-A.1001]
It replaces to following function.
1 REM ˝MOVEX A=1,AC=0,SM=0,M1X,P,(9999,
Normal end.
```

Please press the cursor key to scroll the screen. Press Enter key to shut this dialog.

[Display example in force conversion]

[NB4-02-A.1006]	→**Conversion File Name**
It replaces to following function.	→Message in force conversion
1 REM “MOVEX A=1,AC=0,SM=0,･･･	→Forcibly converted line number and command after conversion
Normal end.	→Result

Table 4.3.1 Errors of Force conversion

Error description	Explanation
E55 Pose calculation failed.	The value of pose constant is not correct.

4.4 Force execute of language conversion

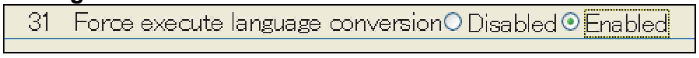
This concerns the setting whether to execute the language conversion forcibly when a particular error occurs. (☞ Refer to “Table 4.3.1 Errors of Force conversion“) Follow the procedures below.

* To change this setting, the operator qualification of ***EXPERT*** or higher is required.

Constant Setting

1 **Press <Constant Setting>.**
>>The setting menu screen will appear.

2 **Select [5 Operation Constants] - [1 Operation condition] in <Constant Setting> menu.**



31 Force execute language conversion ○ Disabled ● Enabled

+ 🟩 + ⇨

3 **Align the cursor with "Force execute language conversion", and press [→] with holding [ENABLE] to switch to "Enabled".**

[Setting value]
Disabled → Forced conversion is not executed.
Enabled → Forced conversion is executed.

Complete

4 **Press f12<Complete>.**
>>The setting will be saved.

Chapter 5 Command

"Commands" are the statements such as MOVEX to operate (move) robot and SETM to turn ON/OFF the output signals.

Commands of executable program are classified into move commands recorded by using [RECORD] key and application commands recorded by using [FN] key, but in robot language programs all of these are treated as "commands".

For details of them, please refer to the online manual of the teach pendant or the "COMMAND REFERENCE" manual or the respective option manuals.

5.1 MOVEX (movement command)

"MOVEX" is the most basic command to move the robot towards a teaching point.
1 MOVEX command equals 1 teach point and the robot moves to the designated point.

> Please see the following explanations also.
> - Instruction manual ***"BASIC OPERATIONS"*** - "Chapter 4 Teaching"
> - "2.4.3 Pose constants"
> - "2.4.4 Shift constants"
> - "2.4.5 MOVEX-X with User coordinate system (position and direction)"
> - "2.5.9 Pose variables"
> - "2.5.10 Shift variables"
> - "2.7.5 Pose operations"

5.1.1 In case of single mechanism

For example, in case of a 6-axes mechanism (manipulator), the MOVEX should be written like this example.

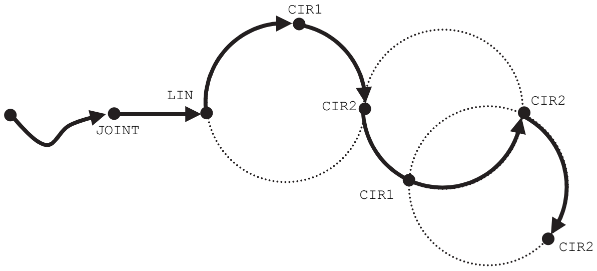
```
MOVEX A=1P,AC=1,SM=1,F,M1X,P,(1200,0,1800,0,0,-180),R=10.0,H=1,MS,CONF=0000
       (1)  (2)  (3) (4)(5)(6)          (7)           (8)   (9)(10)  (11)
```

No.	Mark	Name	value	Description
1	A	Accuracy	1-8 or 1P-8P	Set the shortcut motion accuracy level. If this parameter is not written, the previous value will be used. If "P" is attached, the robot does not make the shortcut locus and try to make a pause (positioning) motion.
2	AC	Acceleration	0 – 3	Acceleration level. (0 – 3) If this parameter is not written, the value is regarded as "0". (0 is the same withe the default acceleration)
3	SM	Smoothness	0 – 3	Smoothness level. (0 - 3) If this parameter is not written, the value is regarded as "0". (0 is the same withe the default smoothness)
4	F	Fine motion	–	If "F" is written, the fine motion function is enabled. If not written, the fine motion function is disabled. See the instruction manual ***"FINE MOTION"*** also.
	HM	Synchromotion	–	If "HM" is written, the synchro motion control is enabled. If not written, the synchro motion control is disabled (= simultaneous control). See the instruction manual ***"SYNCHROMOTION CONTROL"*** also.
5	M1X M1J M1E M2X M2J M2E M3X M3J M3E :	Mechanism selection	–	Set the mechanism number and the pose constant format type. (EXAMPLE) M1X : MOVEX-X is used for the mechanism 1 M1J : MOVEX-J is used for the mechanism 1 M1E : MOVEX-E is used for the mechanism 1 M2X : MOVEX-X is used for the mechanism 2 - For details of the pose constants, refer to "2.4.3 Pose constants". - MOVEX-X is available only when the concerned mechanism is a manipulator (articulated robot). In case of a slider or a positioner, it is not available. - If X,J,E are not written, it is regarded as "X".

> When trying to move the robot using MOVE-X, the robot may take unexpected posture because of e.g. sigular point etc. To determine the robot posture completely, MOVEX-J or MOVEX-E are recommended.

POINT

> Some functions (e.g. fine motion etc.) are option function.

No.	Mark	Name	value	Description
6	P L C1 C2 LE C1E C2E	Interpolation type	–	This is the "Interpolation type" that is used when the robot TCP moves between 2 teach points. 7 types are available. P JOINT Joint interpolation(linear interpolation OFF) L LIN Linear interpolation C1 CIR1 Circular interpolation(middle point) C2 CIR2 Circular interpolation(end point) LE S-LIN LIN with a stationary tool C1E S-CIR1 CIR1 with a stationary tool C2E S-CIR2 CIR2 with a stationary tool  JOINT Each axis will move without considering the other axis, the TCP locus does not get linear. LIN Each axis will move with considering the other axis to make the TCP locus linear. CIR1 By using the previous point and the next point, an imaginary circle is generated and the TCP will draw the first half arc. CIR2 By using the previous 2 points, an imaginary circle is generated and the TCP will draw the last half arc. In case of the stationary tool interpolation (LE, C1E, and C2E), the interpolation control will be done based on the pre-defined user coordinate system's location and the direction. For details, refer to the online help of the "FN67 STOOL" and the <ServiceUtilities> - [10 User Coordinates Definition].

No.	Mark	Name	value	Description

7	–	Pose constants
		or
		Pose variables
			–	Set the teach point using a pose constant.
				See the "2.4.3 Pose constants" also.
				
				MOVEX-X format pose constant
				Write a teach point in `(X,Y,Z,roll,pitch,yaw)`.
				`MOVEX A=1,`**`M1X`**`,P,`**`(1465,0,1500,0,0,-180)`**`,R=5.0,H=1,MS`
				
				- After converting to the executable format, the data itself is kept in this format. (It is also possible to edit the data in the screen editor.)
				- When applying the reverse compiling, the step data will return to MOVEX-X format inspite of the output format designation.
				
				- It is possible to use a user-defined coordinate system.
				`MOVEX A=1,`**`M1X`**`,P,`**`(100,0,200,0,0,-180)U`**`,R=5.0,H=1,MS`
				
				MOVEX-J format pose constant
				Write a teach point in `(J1,J2,J3,J4,J5,J6)`.
				`MOVEX A=1,`**`M1J`**`,P,`**`(0,90,0,0,0,0)`**`,R=5.0,H=1,MS`
				
				- After converting to the executable format, the data itself is kept in this format. (It is also possible to edit the data in the screen editor.)
				- When applying the reverse compiling, the step data will return to MOVEX-J format inspite of the output format designation.
				
				MOVEX-E format pose constant
				Write a teach point in encoder values.
				`MOVEX A=1,`**`M1E`**`,P,`**`(&H80000,&H80000,&H80000,&H80000, &H80000, &H80000)`**`,R=5.0,H=1,MS`
				
				- When compiling this, the data is converted to **the same data format with a movement command that is recorded with key using the actual robot.** When applying the reverse compiling, the step data will be converted to the robot language following to the output format designation.

				It is also possible to set a teach point usgin pose variables.
				See "2.5.9 Pose variables" also.
				
				Pose variable
				`USE 1`
				`P1 = (1200,0,1800,0,0,-180)`
				`MOVEX A=1,`**`M1X`**`,P,`**`P1`**`,R=5.0,H=1,MS`
				
				- To use pose variables, it is necessary to load a pose file in advance. For details, refer to the online help of "`FN98 USE`".
				- To edit (record / modify) the pose variables, refer to "3.3 Creating pose files".
				- It is also possible to use the pose variables that were calculated via "Pose operation". (Refer to "2.7.5 Pose operations")
				- After converting to the executable format, the data is kept in the pose variable format. It is also possible to change the pose variable number etc.
				- When applying the reverse compiling, the step data will return to the original format (pose variable format) inspite of the output format designation.
				- It is also possible to write the pose variable number like the following using an integer variable.
				`P[V1%]`

No.	記号	名前	値	説明

				Shift motion
				Only when a pose variable is being used, it is possible to use the shift variable or shift constant with this movement command.
				`USE 1`
				`P1 = (1200,0,1800,0,0,-180) '(X,Y,Z,roll,pitch,yaw)`
				`R1 = (10,0,0,0,0,0)         'X+10[mm] shift`
				`MOVEX A=1,M1X,P,`**`P1+R1`**`,R=5.0,H=1,MS`
				`MOVEX A=1,M1X,P,`**`P1+(10,0,0,0,0,0)`**`,R=5.0,H=1,MS`
				`MOVEX A=1,M1X,P,`**`P*+R1`**`,R=5.0,H=1,MS`
				`MOVEX A=1,M1X,P,`**`P*+(10,0,0,0,0,0)`**`,R=5.0,H=1,MS`
				
				- Pose variable "`P*`" stands for the present position.
				- The shift motion robot language and the movement command that includes "`P*`" cannot be edited in the screen editor after the compile operation. To edit them, please edit the robot language source codes in a text editor and compile them to the executable format.
				- For the shift constant and the shift variable, only the format of `(X,Y,Z,rol,pitch,yaw)` can be used.

8	S T R D	Speed	–	This is the robot movement speed. "`R=10.0`" stands for the speed of "10.0%". Following 4 types are available. S Linear speed (1.0[mm/s]～5000[mm/s]) T Movement time (0.01[sec]～100[s]) R Speed rate (1.0[%]～100[%]) D Tool angle change speed (1～500[deg/s])
9	H	Tool number	1-32	The tool number that is used for the movement command.
10	MS	Speed standard	–	Attach this mark at the last of the mechanism that is the "Speed standard" This mark can be attached to plural mechanisms.

11	CONF	Configuration	0000～1112	Sometimes, plural robot postures that can realize the designated `(X,Y,Z,roll,pitch,yaw)` exist. This parameter "`CONF`" can determine only 1 among those postures. The `CONF` parameter can be written in the following format.
				
				`CONF = ijkl`
				
				`i  0:FLIP / 1:NONFLIP` Wrist unit FLIP / NON-FLIP
				`j  0:ABOVE / 1:BELOW` Elbow above / below
				`k  0:LEFTY / 1:RIGHTY` Left side arm / right side arm
				`l  0:±180 deg or less` Flange axis rotation direction
				`   1:0～360 deg`
				`   2:0～-360 deg`
				
				- This parameter works only in case of `MOVEX-X` format pose constant. In case of pose variable, the `CONF` is ignored because the internal data is processed in encoder value format that can determine the robot posture uniquely.
				- If `CONF` is not written, the robot will select the posture that is the closest to the previous step posture automatically. But, if force posture selection command like `FLIP` etc. is executed in advance, the robot will follow the command.
				- See the online help for the following function commands also.
				`FN160 POSAUTO`
				`FN161 LEFTY`
				`FN162 RIGHTY`
				`FN163 ABOVE`
				`FN164 BELOW`
				`FN165 FLIP`
				`FN166 NONFLIP`
				`FN202 FRANGE`

WARNING

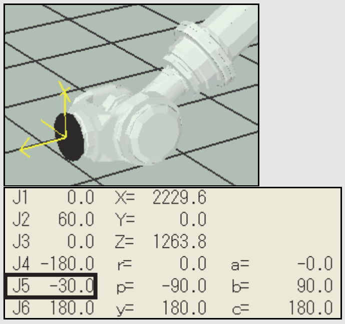
If the CONF parameter is wrong, the robot may make unexpected motion or posture and result in breakage of the gripper and its wirings or serious accidents etc. Therefore, when trying to run the work-program using the real robot, please pay special attention. Generally, CONF parameter should be omitted to let the robot to take the natural posture automatically. Of course, even in that case, careful program pre-check is still necessary. (The simple simulation software "FD on Desk" can be used to check the robot motion visually without using the real robot.)

INFO.

<FLIP / NONFLIP>

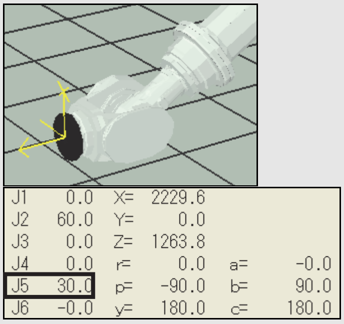
If there are 2 patterns of J5 angle for the identical (X,Y,Z,roll,pitch,yaw), these functions select either of them forcibly.

0:FLIP(J5<0)



```
J1     0.0  X=  2229.6
J2    60.0  Y=     0.0
J3     0.0  Z=  1263.8
J4  -180.0  r=     0.0  a=    -0.0
J5   -30.0  p=   -90.0  b=    90.0
J6   180.0  y=   180.0  c=   180.0
```

1:NONFLIP(J5>0)



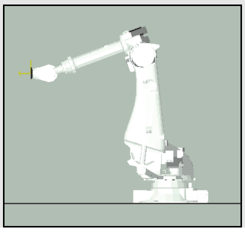
```
J1     0.0  X=  2229.6
J2    60.0  Y=     0.0
J3     0.0  Z=  1263.8
J4     0.0  r=     0.0  a=    -0.0
J5    30.0  p=   -90.0  b=    90.0
J6    -0.0  y=   180.0  c=   180.0
```

INFO.

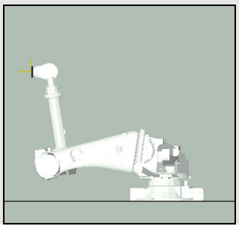
< ABOVE / BELOW >

If there are 2 patterns of "elbow position" for the identical (X,Y,Z,roll,pitch,yaw), these functions select either of them forcibly. ABOVE is upper side and BELOW is lower side. But if the robot does not supprt the reverse posture, the posture of BELOW can not be made. Even if BELOW command is used, the robot will stop in half way and display an error message. (These pictures were made ignoring the motion range limit setting.)

0:ABOVE



1:BELOW

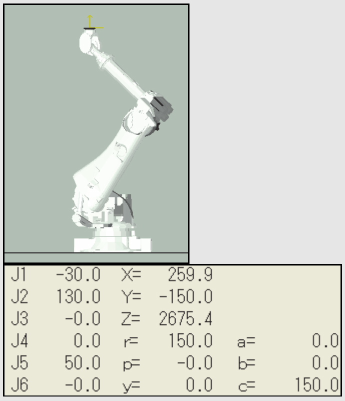


INFO.

< LEFTY / RIGHTY >

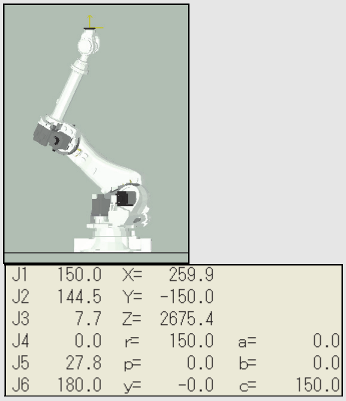
If there are 2 patterns of J1 angle for the identical (X,Y,Z,roll,pitch,yaw), these functions select either of them forcibly.

1:RIGHTY(J1<0)



```
J1   -30.0  X=   259.9
J2   130.0  Y=  -150.0
J3    -0.0  Z=  2675.4
J4     0.0  r=   150.0  a=     0.0
J5    50.0  p=    -0.0  b=     0.0
J6    -0.0  y=     0.0  c=   150.0
```

0:LEFTY(J1>0)



```
J1   150.0  X=   259.9
J2   144.5  Y=  -150.0
J3     7.7  Z=  2675.4
J4     0.0  r=   150.0  a=     0.0
J5    27.8  p=     0.0  b=     0.0
J6   180.0  y=    -0.0  c=   150.0
```

INFO.

To disable the force posture selection commands e.g. FLIP etc., execute POSAUTO. After executing this command, the posture selection will be executed automatically.

<Singular point and dead zone>
Generally, in case of 6-axes articulated robots, for 1 target point `(X,Y,Z,roll,pitch,yaw)`, the number of available combinations of `(J1,J2,J3,J4,J5,J6)` to make the position may become infinity.
A point like this is called as ***"Singular point"***. At this point, there are some problems;

- The TCP speed gets very slow.
- Accurate interpolation control gets impossible.
- The wrist angle changes too much when going through the point.

Therefore, please avoid the sigular point as much as possible.

(Example of singular point)
1: A posture in which J4 axis and J6 axis are parallel.(J5 is 0 [deg])
2: A posture in which J5 axis rotation center point is on the J1 axis

(Supplement)
In this controller, the specific range around the singular point is called as ***"Dead zone (Wrist singularity zone)"***.
The dead zone range is defined as 10 [deg] when shipping. Inside this range, the accuracy of the interpolation calculation (including the tool direction) gets worse.

<Constant Setting> - [3 Machine Constants] [8 Posture Control] ***"Wrist singularity zone"***

For the "Speed", variables can be used.

```
R = V1!
```

The available variables are followings;

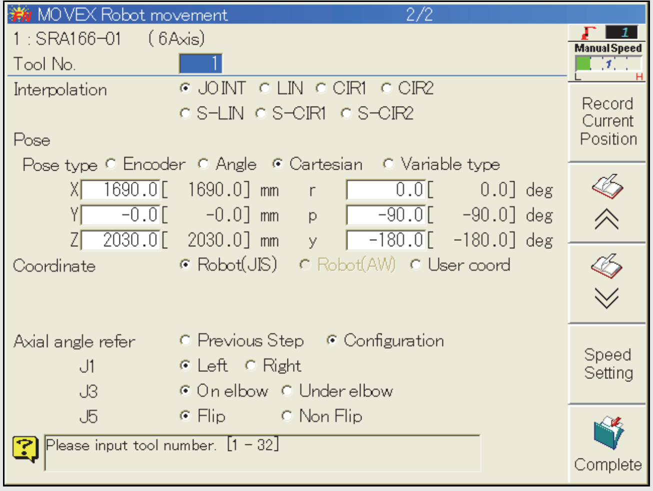
`Vn%`	Global variable (integer)
`Vn!`	Global variable (real)
`Ln%`	Local variable(integer)
`Ln!`	Local variable(real)

After compiling robot languages written in `MOVEX-X` or `MOVEX-J`, it is possible to edit the respective parametes later using a screen like the following.

(Robot language)

```
MOVEX A=1,M1X,P,(1690,0,2030,0,-90,-180),R= 5.0,H=1,MS, CONF=0000
```

(Screen editor)



MOVEX Robot movement 2/2

1 : SRA166-01 (6Axis)

Tool No. 1

Interpolation ● JOINT ○ LIN ○ CIR1 ○ CIR2 ○ S-LIN ○ S-CIR1 ○ S-CIR2

Pose

Pose type ○ Encoder ○ Angle ● Cartesian ○ Variable type

X	1690.0	[1690.0]	mm	r	0.0	[0.0]	deg
Y	-0.0	[-0.0]	mm	p	-90.0	[-90.0]	deg
Z	2030.0	[2030.0]	mm	y	-180.0	[-180.0]	deg

Coordinate ● Robot(JIS) ○ Robot(AW) ○ User coord

Axial angle refer ○ Previous Step ● Configuration

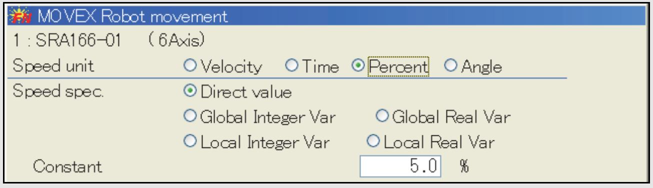
J1 ● Left ○ Right

J3 ● On elbow ○ Under elbow

J5 ● Flip ○ Non Flip

Please input tool number. [1 - 32]

(“User coordinate” is displayed in case of ***SPECIALIST***)



MOVEX Robot movement

1 : SRA166-01 (6Axis)

Speed unit ○ Velocity ○ Time ● Percent ○ Angle

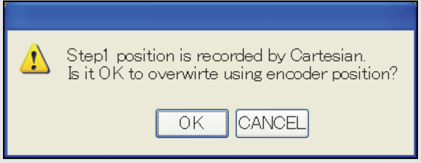
Speed spec. ● Direct value ○ Global Integer Var ○ Global Real Var ○ Local Integer Var ○ Local Real Var

Constant 5.0 %

(Note)

- It is also possible to make this step by inputting `"FN645 MOVEX"` from the teach pendant.

- If the operation of [Enable] + [Position modify] is used to modify the position data of the step, a warning message is displayed. If [OK] is selected, the present encoder value of each axis of the robot will be written to the current movement command and the original data format will be lost. Be careful.



Step1 position is recorded by Cartesian.
Is it OK to overwirte using encoder position?

"Cartesian"	=	A step created by MOVEX-X format
"Angle"	=	A step created by MOVEX-J format
"Pose Variable"	=	A step created by pose variable format

5.1.2 In case of multi mechanism

If there are plural mechanisms like a servo gun robot, MOVEX command should be written like the followings.

■Pose constant

```
MOVEX A=1,M1X,P,(1690,0,2030,0,-90,-180),R= 10,H=1,MS,CONF=0000,M2J,P,(-20.0),R=10,H=1
MOVEX A=1,M1J,P,(0,90,0,0,0,0),R= 10,H=1,MS,CONF=0000,M2J,P,(-20.0),R=10,H=1
```

- For the mechanism 1 (robot), the coordinates are set in `MOVEX-X` or `MOVEX-J` format.
- For the mechanism 2 (servo-gun), the gun axis position is set in `MOVEX-J` format. (20 mm open in this case)

■Pose variable

```
USE 1
P1 = (1690,0,2030,0,-90,-180,-20)
MOVEX A=1,M1X,P,P1,R= 10,H=1,MS,CONF=0000,M2J,P,P1,R=10,H=1
```

- For pose variable, set 7 parameters including the gun-axis position.
- The robor reads the first 6 parameters in the pose variable and the servo-gun reads the 7th parameter.

Also in case of a slider (traverse unit), the description is the same.

```
MOVEX A=1,M1X,P,(1690,0,2030,0,-90,-180),R= 10,H=1,MS,CONF=0000,M2J,P,(-100.0),R=100,H=1
```

In case of 2-axes positioner, 2 axis values for the concerned mechanism are required.

```
MOVEX A=1,M1X,P,(1690,0,2030,0,-90,-180),R= 10,H=1,MS,CONF=0000,M2J,P,(0,0),R=100,H=1
```

5.2 Output signal

By using the general output signals, it is possible to take an interlock with other devices via an external PLC etc. or open/close a gripper etc. To turn ON/OFF the general output signals, please use the following commands.

```
FN0      ALLCLR     Output signal all reset
FN32     SET        Output signal set
FN34     RESET      Output signal reset
FN35     SETMD      Output signal (ON/OFF/delay/pulse)
FN43     OUTDIS     Discrete output signal
FN44     OUT        Binary output signal
FN100    SETO       Consecutive output signal
FN105    SETM       Output signal
FN264    MULTIM     Multi output signal
FN280    DPRESETM   Output signal (Distance)
```

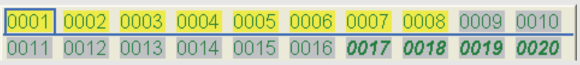
For details, refer to the followings;
The Online help or the instruction manual ***"COMMAND REFERENCE"***

<Pulse pattern and delay pattern>
It is possible to assign pulse pattern or delay pattern for general output signals. For details, refer to the instruction manual ***"EXTERNAL INPUT / OUTPUT"***. And it is also possible to apply delay and pulse length setting for output signals by using `SETMD` function command.

<Combination output signal O5101 to O5196>
The ***"Combination outpuut signal"*** is an output signal to control output signals up to 16 at the same time. For example, 8 signals can be turned ON/OFF at the same time by using only 1 command like the following example. (In this example, O1 – O8 are assigned to the O5101)

```
MULTIM O5101,255
```

Result



0001	0002	0003	0004	0005	0006	0007	0008	0009	0010
0011	0012	0013	0014	0015	0016	***0017***	***0018***	***0019***	***0020***

The setting can be done in the following menu;
<Csonstant Setting> - [6 Signals] [3 Output Signal Assignment] [7 Combination Outputs]

The function command "`FN105 SETM`" can be inputted using key.

POINT

Because the output signals that are assigned to specific function (those signals are called as "Status output signal") are controlled by the CPU of this robot controller, it is impossible to turn ON/OFF using these function commands.

5.3 Input signal

To make an inter-lock control using the external input signals, use the following commands.

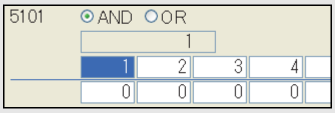
```
FN525   WAITI    Wait Input cond
FN526   WAITJ    Wait not Input cond
FN528   FETCH    Fetch Input cond
FN552   WAIT     Wait I-cond with timer
FN553   WAITA    Wait I-group(AND) with timer
FN554   WAITO    Wait I-group(OR) with timer
FN555   WAITE    Wait I-group with timer
FN557   WAITL    Wait I-group with timer2
FN558   WAITAD   Wait I-group BCD(AND) with timer
FN559   WAITOD   Wait I-group BCD(OR) with timer
FN560   WAITED   Wait I-group BCD with timer
```

For example, if the `I1` signal is OFF when executing `WAITI` command, the robot will stop (this is "Inter-locked status"), and then the robot will restart when the `I1` signal turns ON (this is Inter-lock release operation).

```
MOVEX
MOVEX
WAITI I1
MOVEX
MOVEX
END
```

For details, refer to the followings;
☞The Online help or the instruction manual ***"COMMAND REFERENCE"***

<Combination input signal >
Up to 16 input signals can be used as 1 input signal by combining those signals with AND or OR condition. For example, if you want to regard it as "interlock- release" only when the all of `I1`, `I2`, `I3`, and `I4` signals are ON, please make the setting like the following picture.



5101 ● AND ○ OR

1			
1	2	3	4
0	0	0	0

```
WAITI I5101
```

The setting can be done in the following menu;
<Csonstant Setting> - [6 Signals] [2 Input Signal Assignment] [7 Combination Inputs]

The function command "`FN525 WAITI`" can be inputted using key.

There are some commands that require input signal as a condition if the command is going to be executed or not. The commands like those have "`I`" as the last letter of the command name.

(Example) "`FN81 CALLPI`" Program call (I-condition)

"`FN557 WAITL`" is a command in which the escape destination step can be set using a LABEL.

5.4 Step jump / step call

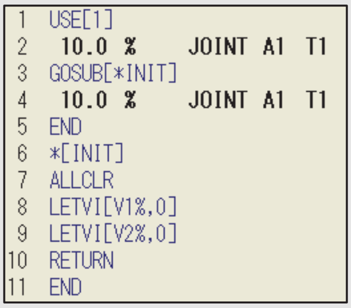
To perform step jump or step call, use the following commands.

FN20	JMP	Step jump
FN21	CALL	Step call
FN22	RETURN	Step return
FN23	JMPI	Step jump (I-condition)
FN24	CALLI	Step call (I-condition)
FN25	RETI	Step return (I-condition)
FN26	JMPN	Step jump (freq. condition)
FN27	CALLN	Step call (freq. condition)
FN28	RETN	Step return (freq. condition)
FN86	FCASEN	Case jump (freq. condition)
FN87	FCASEI	Case jump (I-condition)
FN88	FCASEEND	Case jump end
FN90	GOTO	Line jump
FN91	GOSUB	Line call
FN601	*	Label
FN603	ON	ON GOTO Jump

For details, refer to the followings;
The Online help or the instruction manual ***"COMMAND REFERENCE"***

For the jump or the call commands, "label" can be used.
(Example1) In this example, GOSUB is used with a label. To return to the original step, please use RETURN.

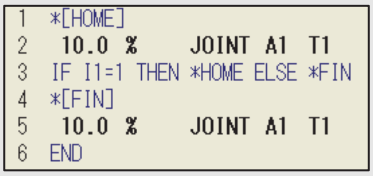
```
USE 1
MOVEX A=1,M1X,P,P1,R=10,H=1,MS
GOSUB *INIT
MOVEX A=1,M1X,P,P1,R=10,H=1,MS
END
*INIT
ALLCLR
LETVI V1%,0
LETVI V2%,0
RETURN
END
```



```
USE[1]
 10.0 %     JOINT  A1  T1
GOSUB[*INIT]
 10.0 %     JOINT  A1  T1
END
*[INIT]
ALLCLR
LETVI[V1%,0]
LETVI[V2%,0]
RETURN
END
```

(Example2) Label can be used as the jump destination of IF command.

```
*HOME
MOVEX A=1,M1X,P,P1,R=10,H=1,MS
IF I1=1 THEN *HOME ELSE *FIN
*FIN
MOVEX A=1,M1X,P,P2,R=10,H=1,MS
END
```



```
*[HOME]
 10.0 %     JOINT  A1  T1
IF I1=1 THEN *HOME ELSE *FIN
*[FIN]
 10.0 %     JOINT  A1  T1
END
```

(Example3) The jump destination can be selected via the value of V1% (1-10).

```
ON V1% GOTO *LB1,*LB2,*LB3,*LB4,*LB5,*LB6,*LB7,*LB8,*LB9,*LB10
```

When using labels instead of line numbers or step numbers for step jump or step call, it would get easier to maintain the work-program. However, if there are many jumps or calls in 1 program, it would be difficult to read the program.

5.5 Program jump / program call

To perform program jump or program call, use the following commands.

FN80	CALLP	Program call
FN81	CALLPI	Program call (I-condition)
FN82	CALLPN	Program call (freq. condition)

FN83	JMPP	Program jump
FN84	JMPPI	Program jump (I-condition)
FN85	JMPPN	Program jump (freq. condition)

FN102	CALLPR	Relative program call
FN103	CALLPRI	Relative program call (I)
FN104	CALLPRN	Relative program call (freq.)

FN400	JMPPBCD	Program jump (to ext. BCD prog)
FN401	JMPPBIN	Program jump (to ext. BIN prog)
FN402	CALLPBCD	Program call (external BCD prog)
FN403	CALLPBIN	Program call (external BIN prog)

FN590	LCALLP	call program with argument
N591	LCALLPI	call program with args(I)
FN592	LCALLPN	call program with args(freq)

FN680	JMPPV	Program jump (Variable)
FN681	JMPPIV	Program jump (I-cond.) (Variable)
FN682	JMPPNV	Program jump (freq.) (Variable)

FN690	CALLPV	Program call (Variable)
FN691	CALLPIV	Program call (I-dond.) (Variable)
FN692	CALLPNV	Program call (freq.) (Variable)

For details, refer to the followings;
☞The Online help or the instruction manual ***"COMMAND REFERENCE"***

INFO.

When executing a program call, the robot will return to the next step in the original program after executing the END command of the destination program. In case of the program jump, the robot will not return to the original program.

INFO.

JMPPV, JMPPIV, JMPPNV, CALLPV, CALLPIV, and CALLPNV can select one of the following items as the parameter. When using the integer variable, the content of the variable is regarded as the program number. The other specification is the same with JMPP, JMPPI, JMPPN, CALLP, CALLPI, and CALLPN.

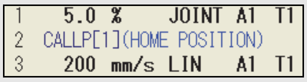
1: Program number (1～9999)
2: Global integer variable Vn% (n=1～200,301～500)
3: Local integer variable Ln% (n=1～200,301～500)

INFO.

If the destination program has "FN99 REM" at the beginning step, the content of the comment will be displayed on the screen like the followings.

Program call command

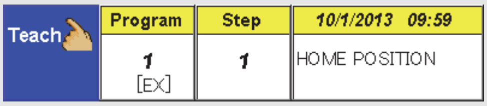
CALLP 1



```
1    5.0 %      JOINT  A1  T1
2  CALLP[1](HOME POSITION)
3    200  mm/s  LIN    A1  T1
```

The destination program

REM "HOME POSITION"



Teach	Program	Step	10/1/2013 09:59
	1 [EX]	1	HOME POSITION

5.6 FORK / CALLFAR

In case of multi-unit system, to call or start the other unit, please use the following commands.

FN450	FORK	Fork Program	
FN451	FORKI	ForkI Program	(NOTE) "I" stands for input signal waiting condition
FN452	FORKN	ForkN Program	(NOTE) "N" stands for frequency condition
FN453	FORKWAIT	Wait Fork-Program	
FN454	CALLFAR	CallFar Program	
FN455	CALLFARI	CallFarI Program	(NOTE) "I" stands for input signal waiting condition
FN456	CALLFARN	CallFarN Program	(NOTE) "N" stands for frequency condition

For details, refer to the followings;
☞The Online help or the instruction manual ***"COMMAND REFERENCE"***
☞Instruction manual ***"MULTI-UNIT"***

5.7 Variables and calculation

For the calculation using the variables, please use the following commands.

```
FN75    LETVI        Set integer variable
FN76    LETVF        Set real variable
FN77    LETVS        Set strings variable

FN142   GETP         Set real variable(pos)
FN143   GETPOSE      Set real variable(pose)
FN144   LETPOSE      Set pose variable
FN145   GETSFT       V! Set real var.(shift)
FN157   GETANGLE     Set real variable(ang)
FN158   GETFIGURE    Set real variable(figure)

FN628   LETLI        Set local integer variable
FN629   LETLF        Set local real variable

FN634   LET          Let variable
FN635   ADDP         Add pose variable

FN637   ADDVI        Add integer variable
FN638   ADDVF        Add real variable
FN639   SUBVI        Subtract integer variable
FN640   SUBVF        Subtract real variable
FN641   MULVI        Multiply integer variable
FN642   MULVF        Multiply real variable
FN643   DIVVI        Divide integer variable
FN644   DIVVF        Divide real variable

FN648   ASIN         Let ASIN function
FN649   ACOS         Let ACOSfunction
FN650   TIMER        Let TIMER function
FN651   SQR          Let SQR function
FN652   SIN          Let SIN function
FN653   COS          Let COS function
FN654   TAN          Let TAN function
FN655   ATN          Let ATN function
FN656   ATN2         Let ATN2 function
FN657   ABS          Let ABS function
FN658   MIN          Let MIN function
FN659   MAX          Let MAX function
```

For details, refer to the followings;
☞The Online help or the instruction manual ***"COMMAND REFERENCE"***
☞"2.7 Statements"

It is also possible to create a new original procedure.
☞"2.9 User procedure"

5.8 Shift

To execute shift motion, please use the following commands.

FN29	RINT	Robot interrupt(I-condition)
FN30	RINTA	Robot interrupt(Analog)
FN51	SREQ	Shift data request
FN52	SHIFTR	Shift
FN53	LOCCVT	Coord. trans(shift value)
FN54	LOCCVT1	Coord. trans(posi. value)
FN58	SHIFTA	XYZ shift
FN59	SEA	Search
FN68	LETR	Set shift value
FN69	ADDR	Add shift value
FN101	PRINT	Strings output
FN111	RSCLR	RS232C Buffer clear
FN113	CHGCOORD	Change coord. No.(shift)
FN127	WAITR	Wait shift value receive
FN145	GETSFT	V! Set real var.(shift)
FN224	REGC	Shift register copy
FN271	INPUT	Strings input
FN275	LOCCVT3	Base angle shift
FN315	SREQ2	Binary shift data request
FN606	PRINT	Print String
FN633	LETRE	Let shift element
FN634	LET	Let variable
FN669	PRINTF	Print string with format
FN699	CLRREGWR	Clear register of written sts
FN723	SIGREQ	Shift value get(signal)

For details, refer to the followings;
☞The Online help or the instruction manual ***"COMMAND REFERENCE"***
☞Instruction manual ***"SHIFT FUNCTIONS BY EXTERNAL INPUT"***

INFO.

While executing the shift motion, the force posture selection functions are enabled.

(See also)
"5.1 MOVEX (movement command)"
"5.9 Force posture selection "

5.9 Force posture selection

These are the functions to select the posture of the robot forcibly in the following cases;

(1) When executing a **MOVEX** command that is written in pose constant of the **"MOVEX-X"** format.

(2) While executing the shift motion. SHiFT
(In this case, the posture selection becomes possible in spite of the data format of the MOVEX teach point.)

FN160	POSAUTO	Posture controll disable
FN161	LEFTY	Arm config.(left/front)
FN162	RIGHTY	Arm config.(right/back)
FN163	ABOVE	Elbow config.(above)
FN164	BELOW	Elbow config.(below)
FN165	FLIP	Wrist config.(flip)
FN166	NONFLIP	Wrist config.(non-flip)
FN202	FRANGE	Flange axis rot. config.

For details, refer to the followings;
☞The Online help or the instruction manual ***"COMMAND REFERENCE"***
☞Instruction manual ***"SHIFT FUNCTIONS BY EXTERNAL INPUT"***
☞"5.1 MOVEX (movement command)"

INFO.

- (If exists) The **CONF** parameter in a **MOVEX** command precedes these commands.
- Once these commands are used, the posture selection will be applied for the all of the following steps. To cancel the force posture selection, please execute **POSAUTO**.

5.10 Coordinate calculation and pose variable

These are the commands to calculate the robot postures, coodinates etc. using the real variables etc. And there are the commands related to the pose variable and pose file.

FN71	LETX	Pose X
FN72	LETY	Pose Y
FN73	LETZ	Pose Z
FN74	POSESAVE	Pose file save
FN94	GETPELR	Set real variable(Euler pos)
FN98	USE	Select pose file
FN142	GETP	Set real variable(pos)
FN143	GETPOSE	Set real variable(pose)
FN144	LETPOSE	Set pose variable
FN157	GETANGLE	Set real variable(ang)
FN158	GETFIGURE	Set real variable(figure)
FN171	NRLCRD	Change coord. for R-Lang.
FN626	MODUSRCOORD	Modify User coordinate
FN630	LETCOORDP	Let pose variable
FN632	LETPE	Let pose element
FN634	LET	Let variable
FN635	ADDP	Add pose variable

For details, refer to the followings;
☞The Online help or the instruction manual ***"COMMAND REFERENCE"***
☞"2.6 Any variables" (User variables)

INFO.

To use the pose variables in a robot language program, please execute the **"FN98 USE"** in advance to load the pose file that contains the pose variables to be used. If the pose file is not loaded, error will occur.

5.11 USER TASK

"USER TASK" is a sort of user macro program that can be executed in the back ground of the robot work-program. In this function, it is also possible to create an original window (user screen) by using draw commands. Although the user task can be started when turing ON the controller power, it is also possible to call it from the robot work-program using the "`FN671 CALLMCR`" commands etc. The user-task related commands are shown as below.

PRINT	Print String
WINDOW	Open/Close user display
TITLE	Set the title on user display
CLS	Clear user display
LOCATE	Locate the diplay pos
GLINE	Draw the line
GBOX	Draw the box
BARC	Draw the arc
GPAINT	Paint
GSETP	Draw the pixel
COLOR	Set the color
BGCOLOR	Set the back ground color
EXIT	Exit usermacro
PAUSE	Pause usermacro
GARC	Display ellipse
GFONT	Set the font
GSOFTKEY	Create soft key
GMSGBOX	Create message box
PRINTF	Print string with format

FN593	LCALLMCR	Call UT Program with args
FN670	FORKMCR	Fork User Task Program
FN671	CALLMCR	Call User Task Program
FN672	FORKMCRTM	Fork User Task Program(Time)
FN673	FORKMCRDST	Fork User Task Program(Distance)

For details, refer to the followings;
☞The Online help or the instruction manual ***"COMMAND REFERENCE"***
☞Instructino manual ***"User Task"***

- The commands related to the User-Task can be executed only in a user-task program.
- The **PRINT** and **PRINTF** can be used in a robot work-program only if the output destination is not the user window

5.12 Any variable (user variable)

"Any variable" (user variable) is a variable that can be defined by the user with free name. Because the any variable support the POSITION type etc. that can be used for the 6-axes articulated robot's coordinate calculation, it is possible to generate complicated teaching poiint by combining the various commands and pose variables.

FN801	DIM	Any variable
FN809	POS2POSE	Set pose variable (position)
FN810	ANG2POSE	Set pose variable (angle)
FN811	ENC2POSE	Set pose variable (encoder)
FN812	POSE2POS	Set position variable (pose)
FN813	ANG2POS	Set position variable (angle)
FN814	ENC2POS	Set position variable (encoder)
FN815	POSE2ANG	Set angle variable (pose)
FN816	POS2ANG	Set angle variable (position)
FN817	ENC2ANG	Set angle variable (encoder)
FN818	POSE2ENC	Set encoder variable (pose)
FN819	POS2ENC	Set encoder variable (position)
FN820	ANG2ENC	Set encoder variable (angle)
FN821	CVTCOORDPOS	Coord. trans(position)
FN822	GETPOS	Set position variable (pos.data)
FN823	GETANG	Set angle variable (pos.data)
FN824	GETENC	Set encoder variable (pos.data)
FN825	OPEPOSE	Extraction pose variable
FN826	OPEPOS	Extraction position variable
FN827	OPEANG	Extraction angle variable
FN828	OPEENC	Extraction encoder variable

For details, refer to the followings;
☞The Online help or the instruction manual ***"COMMAND REFERENCE"***
☞"2.6 Any variables" (User variables)

5.13 User procedure

"User procedure" is a procedure (function) that users can freely create.

`FN802`	`UserProc`	User procedure
`FN803`	`ExitProc`	Exit User procedure
`FN804`	`EndProc`	End User procedure
`FN805`	`RetProc`	Return User procedure
`FN806`	`CallProc`	Call User procedure

For details, refer to the followings;
☞The Online help or the instruction manual ***"COMMAND REFERENCE"***
☞"2.9 User procedure"

5.14 Socket communication

Using a user-task program, it is possible to perform socket communication with an external device that is connected using the Ethernet.

SOCKCREATE	Creating socket
SOCKCLOSE	Closing socket
SOCKBIND	Binding socket
SOCKWAIT	Waiting for reception
SOCKCONNECT	Connecting server
SOCKSEND	Transmitting data
SOCKSENDSTR	Transmitting character string
SOCKRECV	Receiving data
SETSTR	Set string to the buffer
SETINT	Set integer to the buffer
SETREAL	Set real to the buffer
SETBYTE	Set bytes to the buffer
GETSTR	Get string from the buffer
GETINT	Get integer from the buffer
GETREAL	Get real from the buffer
GETBYTE	Get bytes from the buffer

For details, refer to the followings;
☞Instruction manual ***"Socket Communication"***.

5.15 Analog I/O

These are the commands to input / output the analog signals using option board.

```
FN169   SPDDOWNA       Analog input speed override
FN46    AOUT           Analog output
FN319   AUTOZERO       Analog input auto zero set
```

For details, refer to the followings;
☞The Online help or the instruction manual ***"COMMAND REFERENCE"***
☞Instruction manual ***"TCP Velocity Data Output (Including Analog Output)"***
☞Instruction manual ***"VELOCITY OVERRIDE BY INPUT SIGNAL (INCLUDING ANALOG INPUT)"***

INFO.

(NOTE) Digital I/O signal version commands are also provided.

```
FN277   SPDDOWND     Digital input speed override
FN278   DOUT         Digital output
```

5.16 Others

And, there are various commands. Concerning the commands related to the respective applications or the optional functions, refer to their respective manuals also.

FN41	STOP	Robot stop
FN42	STOPI	Robot stop(I-condition)
FN50	DELAY	Timer delay
FN67	STOOL	Select the stationary tool No.
FN92	END	End
FN99	REM	Comment
FN230	COLSEL	Set interferense detection level
FN252	PAUSEINPUT	Pause Input
FN438	SPN	Servo ON
FN439	SPF	Servo OFF
FN467	USRERR	User error
FN600	NOP	NOP(**N**o **OP**eration)
FN697	INCLUDE	Translate table included(file)
FN698	INCLUDEIO	Translate table included(I/O)

For details, refer to the followings;

☞The Online help or the instruction manual ***"COMMAND REFERENCE"***
☞Instruction manual "APPLICATION MANUAL: SPOT WELDING" (FN119 SPOT etc.)
☞Instruction manual "APPLICATION MANUAL : ARC WELDING" (FN414 AS etc.)
☞Instruction manual "Palletize function" (FN249 PALLET3 etc.)
☞Instruction manual "PALLET2 Palletize function" (FN47 PALLET2 etc.)
☞Instruction manual "Seam Welding" (FN245 SEAMST etc.)
☞Instruction manual "FLEXhand Function" (FN362 FHCLAMP etc.)
☞Instruction manual "ADAPTIVE MOTION" (FN364 ADAPTON etc.)
☞Instruction manual "MECHANISM-by-MECHANISM SERVO ON/OFF FUNCTION"
(FN438 SPN etc.)
☞Instruction manual "Conveyor tracking" (FN550 CNVI etc.)
☞Instruction manual " MECHANISM CHANGE "
(FN95 CHGGUN/FN301 CHGMEC etc.)
☞Instruction manual "FORCE CONTROL" (FN326 FORCECTRL etc.)
☞And other option manuals

INFO.

When STOP or STOPI is executed, the robot will stop. To restart the work program, the operation of the start button or the external start signal is necessary.

INFO.

DELAY can be inputted using END ⌛ key.

INFO.

When using the stationary tool interpolation motion, the robot will make the interpolation based on the specific user-coordinate system (defined in advance). Concerning the user coordinate system, please refer to the online help of <Service Utilities> - [10 User Coord. Definition].

NOTE

Chapter 6 How to Use the Robot Language

This chapter describes the application and the manner to use the robot language.

6.1 Notes on Using the Robot Language

6.1.1 Positioning accuracy

In order to teach the MOVE command by the robot language, the accuracy in each item is necessary as below. The accuracy respectively affects the accidental error between the position designated by the robot language and the actual TCP.

(1) Tool installation accuracy (e.g. arc welding torch etc.) (*)
(2) Encoder correction accuracy / Mastering accuracy
(3) Robot positioning accuracy
(4) Workpiece processing accuracy
(5) Workpiece installation accuracy

(*) LIN, CIR1 and CIR2 are executed based on the TCP position. The error between the TCP position data (Tool length) and the actual TCP position affects the path accuracy.

6.1.2 How to play back the program created by the robot language

To perform the check operation or the automatic operation for the program created by the robot language, operate from the step 1 in principle.

However, it is fine to operate from the step once stopped when restarting after a temporary stop during playback.

⚠ WARNING

The teaching point of MOVE command described by the robot language is not a position stored by teaching but the one read out from the pose variable in the prior step or the one calculated by the computation expression in most of the cases.

If executing from the step halfway, the robot does not work as desired because no correct positional information can be obtained.

6.1.3 Check operation

Before playback of the program created by the robot language, be sure to check the robot operation beforehand by the check operation.

⚠ WARNING

Even if normally finishing the conversion from the robot language format to the execution format (compile), the robot may not work as expected when there is an error in the calculation manner or the procedures for obtaining the position.

6.1.4 Rewriting to the encoder value

MOVE command of program taught by the robot language is positionally recorded by the coordinate value, articular angle value or variables.

When correcting the position of MOVE command of program taught by the robot language using the teach pendant <Positional correction>, the record data of step where the position were corrected are to be rewritten to the present encoder value for each axis.

6.1.5 Restoring the pose variable

The pose variable is restored in the inner memory of the robot controller. Since the pose variable is deleted if restarting the robot controller, it is necessary to perform a calculation.

In order to retain the present pose variables, restore them in the pose file using the POSESAVE command. The pose variable for pose files can be read out to the inner memory using the USE command.

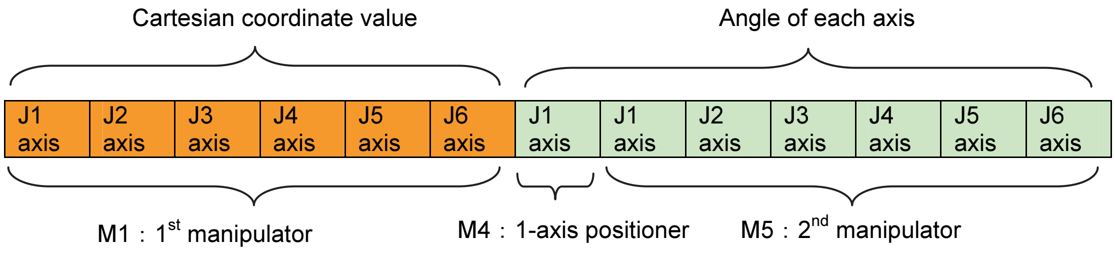
6.1.6 Notes on the use of multi-mechanism

When the system has two or more mechanisms in the unit, bear in mind the following cautions and limitations regarding the function command that deals with pose variables.

(1) Set the record data of pose variables to the multi-mechanism specification.

Go to <Constant settings> - [5 Operation Constants] - [1 Operation condition] - [12 robot language (GETP, GETPOSE)], and set to “ON”.

(2) The record data of pose variables are recorded by the order of mechanism number in the unit from the front.
For instance, the record data used in the unit where three mechanisms exist; such as M1: 1st manipulator, M4: 1-axis positioner, M5: 2nd manipulator, are as follows.



(3) When using the function command that obtains or substitutes the recorded data of pose variables, be sure that only the data from the 1st manipulator can handle the Cartesian coordinate values, while all the other mechanisms handle the angle of each axis. See the figure shown in (2).

Function commands that obtain and substitute the recorded data of pose variables are as follows.

- GETP, GETPOSE, LETPOSE → The manipulators other than the 1st one shall handle the angle data of each axis.
- ADDP, LETPE → Does not support the 2nd or later manipulator.
- LETCOORDP → Supports all mechanisms.

6.2 What Can be Performed by the Robot Language

The robot language is available for the workpiece on which the teach points can be numerically designated on the robot coordinates. Just like as the NC (Numeric control), it is available to designate the position of welding seam line and the trajectory by the figures or arithmetic calculus.

Even in the case where there are a variety of welding positions or welding seam lengths on a different length of workpiece, use of the robot language would manage it with specifying a different figure by the same task program, which makes it unnecessary to teach every workpiece.

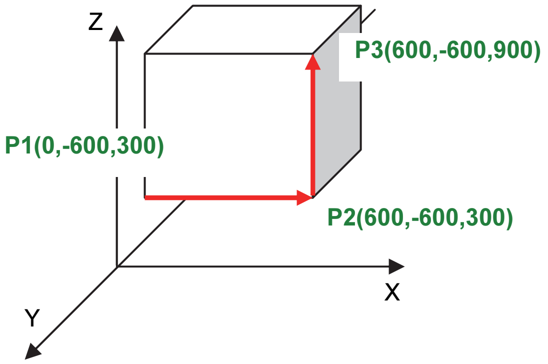
6.2.1 Numerically designating the target position on the robot coordinates

When moving on a line connecting three points; P1→P2→P3 as below:

1） In the teaching playback system, manually operate the robot to each point and record the position.

【Teaching playback system】

```
Step1 MOVE Joint   P1 teaching position
Step2 MOVE Line    P2 teaching position
Step3 MOVE Line    P3 teaching position
```



2） In the robot language system, describe the coordinate values of each point (including the posture) in the task program. It is unnecessary to manually operate the robot.

【Robot language system】Offline teaching system

```
Step1 MOVEX P   (0, -600, 300, 0,-90,0)
Step2 MOVEX L   (600, -600 ,300, 0, -90, 0)
Step3 MOVEX L   (600, -600, 900, 0, -90, 0)
```

※ Details of the description are simplified in part.

6.2.2 Numerically designating the move distance

P1 start position (700.0,-300.0,600.0)
→ Linear motion of 100mm from P1 in the X axis (+) direction to reach P2
→ Linear motion of 100mm from P2 in the Z axis (-) direction to reach P3

This control can be described in the robot language as below.

【Robot language system】Calculates the target position (using the pose variable Pn) and then describes the MOVE command.

1） Determines the calculation of target position using the pose variable Pn.

```
Step1 P1 = (700.0,-300.0,600.0,-150,0,150)
Step2 P2 = P1 + (100,0,0,0,0,0)     ← Adding the move distance. (+100mm in the X direction)
Step3 P3 = P2 + (0,0,-100,0,0,0)    ← Adding the move distance. (-100mm) in the Z axis.)
```

2） Moves linearly in the sequence of the position of pose variable; P1 →P2→P3.

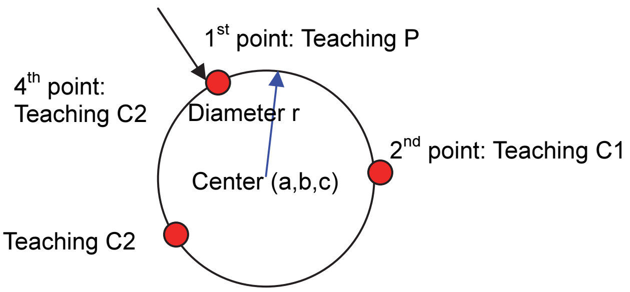
```
Step4 MOVEX A=1,AC=0,SM=0,M1X, P, P1, R= 100,H=1,MS, CONF=0020
Step5 MOVEX A=1,AC=0,SM=0,M1X, L, P2, S= 16.6,H=1,MS, CONF=0020
Step6 MOVEX A=1,AC=0,SM=0,M1X, L, P3, S= 16.6,H=1,MS, CONF=0020
Step7 END
```

6.2.3 Calculating the circular trajectory

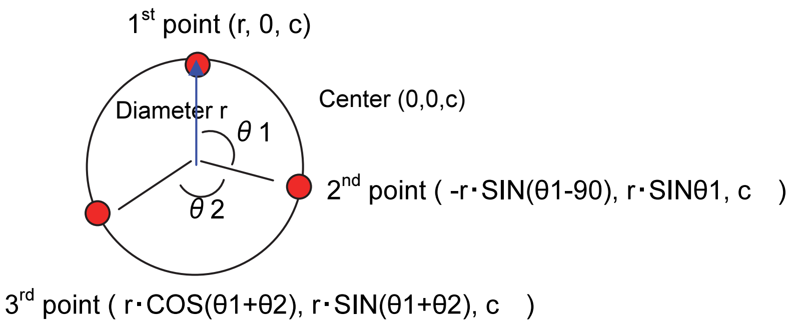
When teaching the circular trajectory of the center position and diameter already known:

1） In the teaching playback system, manually operate the robot to three representative points on the circumference and record the position.

【Teaching playback system】
Step1 MOVE Joint P1 teaching position
Step2 MOVE CIR1 P2 teaching position
Step3 MOVE CIR2 P3 teaching position
Step4 MOVE CIR2 P1 teaching position



2） In the robot language system, it is available to calculate a specific position on the circumference using the center and the diameter of circle. Here we use the circle on the XY plane for explanation (Z position is fixed).


1st point (r, 0, c)
Center (0,0,c)
Diameter r
θ 1
θ 2
2nd point (-r･SIN(θ1-90), r･SINθ1, c)
3rd point (r･COS(θ1+θ2), r･SIN(θ1+θ2), c)

【Robot language system】

1) Set the value required for the calculation formula (r,c,θ1,θ2) to the real number variable (Vn! or Ln!).

```
Step1 V1! = r
Step2 V2! = c
Step3 V3! = θ1
Step4 V4! = θ2
```

2) Substitute the calculation position for three points on the circumference in advance into the pose variable P1 (1st point), P2 (2nd point) and P3 (3rd point).

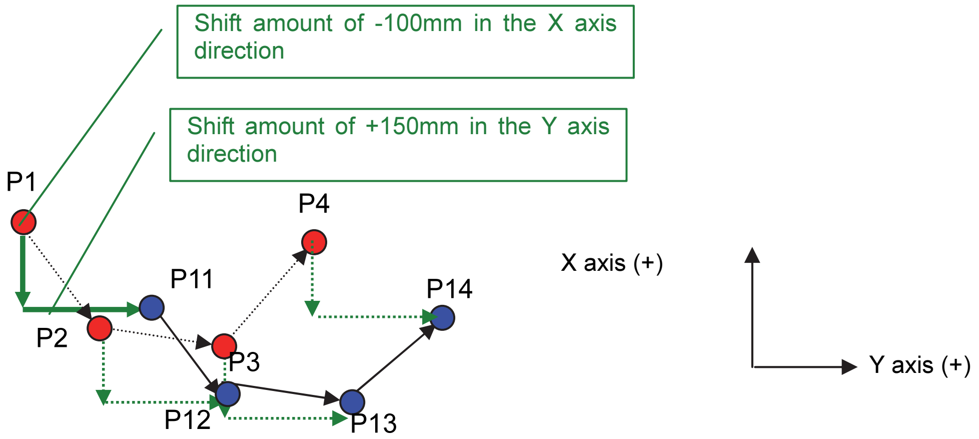
```
Step5 P1=( V1!, 0, V2!, 0, -90, 0)
Step6 L1! = V3! – 90
Step7 L2! = V3! + V4!
Step8 P2=(-1 * V1! * SIN(L1!), V1! * SIN(V3!), V2!, 0, -90, 0)
Step9 P3=(V1! * COS(L2!),V1! * SIN(L2!), V2!, 0, -90, 0)
```

3) Exercise circular control in the sequence of the position of pose variable P1→P2→P3→P1.

```
Step10 MOVEX P   P1
Step11 MOVEX C1   P2
Step12 MOVEX C2   P3
Step13 MOVEX C2   P1
```

6.2.4 Numerically shifting the task programs

Shift the task program (P1,P2,P3,P4) for -100mm in the X axis direction and +150mm in the Y axis direction.


Shift amount of -100mm in the X axis direction
Shift amount of +150mm in the Y axis direction
P1
P4
P11
P14
P2
P3
P12
P13
X axis (+)
Y axis (+)

Here the pose variables (P1～P4, P11～P14) are used as the teaching position. Set the shift amount to the shift variable R in the coordinates (X,Y,Z,r,p,y), and add the shift variable R to the original teaching position.

【Robot language description】

```
    Step1 R1 = ( -100.0, 150.0, 0, 0, 0, 0 )   → Setting the shift amount of each coordinate axis to the
  shift variable.
    Step2 P11 = P1 + R1 → Add the shift amount to the original position P1.
    Step3 P12 = P2 + R1 → Add the shift amount to the original position P2.
    Step4 P13 = P3 + R1 → Add the shift amount to the original position P3.
    Step5 P14 = P4 + R1 → Add the shift amount to the original position P4
    Step6 MOVEX P P11
    Step7 MOVEX P P12
    Step8 MOVEX P P13
    Step9 MOVEX P P14
```

6.2.5 Responding to two or more workpieces by a single task program using variables

In the normal teaching playback system, the robot positional data (hereinafter called record data) recorded in the task program are the position taught by the actual work (encoder value).

Where there are two or more types of shape or dimension of workpiece, it is inconvenient to perform teaching every time the workpieces is changed.

To respond to such a case, it is useful to use a single task program only for calculating the teaching position (shift calculus, arithmetic calculus) based on the change of shape or dimension.

First numerically calculate the teaching position in the robot language. Set the shape and dimensional data to the variable as parameters. Then, the calculation results are substituted into the variables, and set the variable data to the positional data of MOVE command.

The variable data are TCP(X,Y,Z,r,p,y) in the robot coordinates.

The robot coordinates are as follows.

Machine coordinate system, Work coordinate system, User coordinate system, World coordinate system, Tool coordinate system.

TCP position X,Y,Z (mm) and the tool angle r,p,y (degree) can be designated in any of the coordinate systems.

Following functions are realized by using the variables.

- The task program can be created without actual work based on the installation position, shape and dimension of the workpiece.
- Flexibility is improved so that it enables to support two or more workpieces by a single task program.
- High level of trajectory calculation is available by combining with the arithmetic calculation of the robot language.

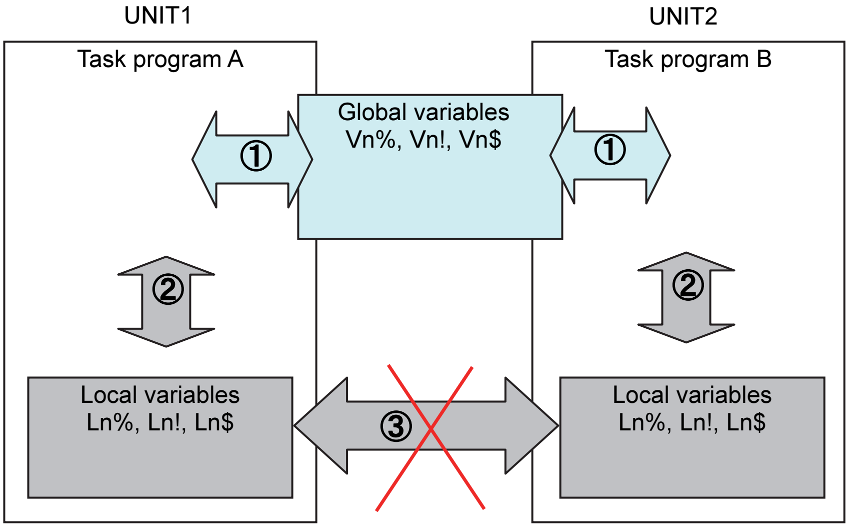
6.3 Variables (Inner Variable, Pose Variable)

This section describes the variables used in the robot language.

6.3.1 Inner variable

(1) Global variable and Local variable

The inner variable is the area that stores numeric values used in the robot language. There are the integer variable (%), the real number variable (!) and the character string variable ($), each of which respectively has the Global variable (V) and the Local variable (L).



The global variable is the common area in the system, which is common among all the task programs regardless of within or without the unit.

The integer variable V1% described by the task program A and V1% described by the task program B are the same variable (area).

The local variable is the common area in the UNIT

The integer variable L1% described by the task program A of UNIT1 and L1% described by the task program B of UNIT2 are different variables (area).

The local variables described by the task program A of UNIT1 cannot be accessed from the task program B of UNIT2. And vice versa, the local variables described by the task program B of UNIT2 cannot be accessed from the task program A of UNIT1.

(2) Shift variable

The shift variables are the local variables. The variable R1 described by the task program A and R1 described by the task program B are different variables (area).

6.3.2 Pose variable

(1) The pose variables are available from P1 to P9999.

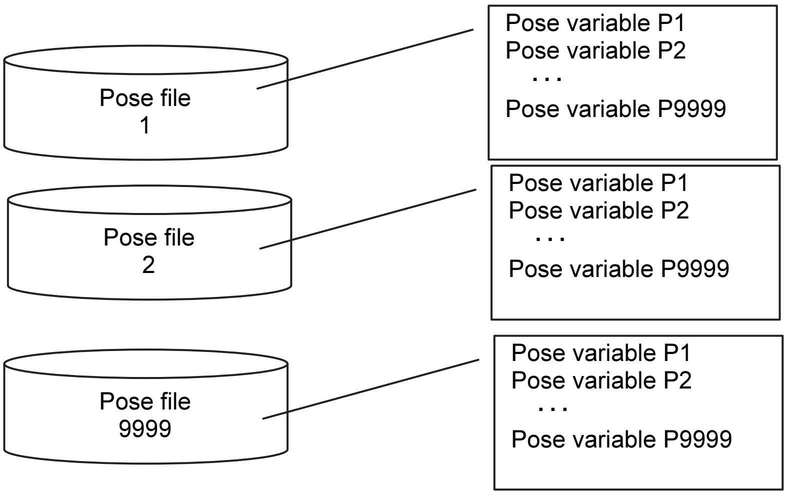
(2) The pose variables are controlled in the memory by the same structure as the step data of task program.

(3) The editable pose variables are recorded by POSESAVE command in the pose file. (They will be cleared if turning off the main power without saving the file because of no power retention.)

(4) The pose file can be created for the same number of files as the task program (9999), however only one file can be called at a time.

(5) File name of pose file
“Unit name - P. Pose file number”
For example, it is “UNIT01-P.1000” when the pose file number of UNIT1=1000.

(6) Structure of pose file
The pose variable and pose file are related as shown in the figure below.



6.3.3 How to edit the pose variable

The pose variables can be set by the following two manners.

- Editing by the teach pendant.
- Reading and writing in the robot language.

Editing by the teach pendant

(1) Proceeding to <Service Utilities> - [1 Teach/Playback Condition] and setting [13 Recording of Pose] to “Enabled”, the program monitor screen is switched to the pose file screen.

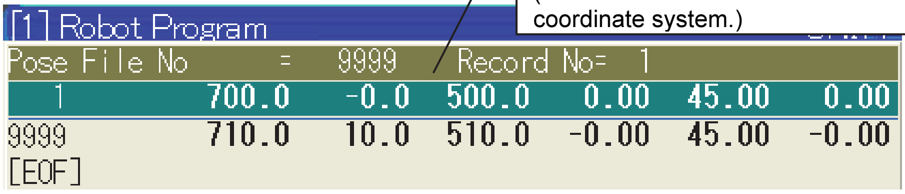
(2) Program No. ＝Pose file No., Step No. ＝ Pose variable No.

(3) As the same as reading operation of the task program, inputting the pose file No. to read out the pose file.

(4) The pose variable (positional data) recorded in the target file is displayed.

(5) As the same as the task program, the pose variable value can be changed using [EDIT].

When the pose variable P1 and P9999 are recorded in the pose file 9999.
(Positional data are the values on the base coordinate system.)



```
[1] Robot Program
Pose File No     =   9999    Record No=  1
   1          700.0    -0.0   500.0    0.00   45.00    0.00
9999          710.0    10.0   510.0   -0.00   45.00   -0.00
[EOF]
```

Reading/writing in the robot language

(1) Designating the pose file No. by “USE” command. (USE command has a role to switch the pose file, too.) The pose variable (P1～P9999) described by the step after “USE” command is controlled as the step data (Step1～Step9999) of pose file specified by “USE”.

【Example】Record the value to the step 3 in the pose file 2 (in the memory).

```
USE   2
P3=(100, 10, -200, 0, 90, 0)
```

(2) Recording the pose variable currently saved in the pose file specified by “USE” using “POSESAVE” command.

【Example】Save the value of the step 3 in the pose file 2.

```
USE   2
P3=(100, 10, -200, 0, 90, 0)
POSESAVE
```

(3) It is available to read out the pose variable using “GETPOSE” command from the pose file specified by “USE”.

【Example】Read out the pose variable saved in the step 3 of the pose file 2 into the inner real number variable V11!～V16!.

```
USE   2
GETPOSE   V11!, 3
```

(4) It is available to write the pose variable in the pose file specified by “USE” using “LETPOSE” command.

【Example】Write the pose variable of the inner real number variable V11!～V16! in the step 4 of the pose file 2.

```
USE   2
LETPOSE   4, V11!
```

6.4 Example of How to Use the Robot Language

6.4.1 How to read out the current position of robot <Method 1>

The GETP command ＜FN142＞ is available for reading out the current position of robot. Also, it is available to teach on the function list of the task program.

【Example】

Read out the current position (orthogonal position X,Y,Z,a,b,c) to the real number variable V11!～V16!. The positional data read out are the current position of the time when GETP command is executed.

```
GETP   V11!
```

※1）Normally, it is read by the orthogonal position in the machine coordinate system.
※2）Teaching NRLCRD command ＜FN171＞ as a step prior to GETP command, it is read by the orthogonal position in the user coordinate system. (See the section 6.4.3 .)

6.4.2 How to read out the current position of robot <Method 2>

The SYSTEM!() function is available for reading out the current position of robot. As this function is the macro command, it is available only by the user task. The positional data read out by the SYSTEM!() function are stored by the real number variable (Vn!).

For instance, when constantly monitoring the robot current position in the machine coordinate system:

(1) Create the user task program.

(2) Convert the user task into the executable format by the program convert function after created by the text file.

(3) Create the text file of “USERTASK-A.100” for the user task P100.

The following is a sample description for “USERTASK-A.100”. It reads out the XYZ position in the machine coordinate system of Mech 1 into the real number variable V11!～V13 by a 50msec cycle.

```
REM "Reading the robot current position (TCP in the machine coordinate system)"
REM "V11! ← Current TCP of Mech 1 (X coordinate)"
V11! = SYSTEM!(150)
REM "V12! ← Current TCP of Mech 1 (Y coordinate)"
V12! = SYSTEM!(151)
REM "V13! ← Current TCP of Mech 1 (Z coordinate)"
V13! = SYSTEM!(152)
PAUSE 50
GOTO 1
END
```

※ For the argument of SYSTEM function (value in （　）), see the section ‘2.6.7 System functions’.

(4) Convert “Source-->exe” using the Program Conversion – Language function.
+
(5) “USERTASK.100” is created.

(6) Start the user task P100.

【Starting method 1】Start P100 in <Service Utilities> - [12　User Task].

【Starting method 2】Start P100 from the task program using FORKMCR command.

6.4.3 Controlling the pose variable by the positional data on user coordinates

When using the pose variable, the following commands are often applied.

・LETX（FN71）: Substituting the pose X component.
・LETY（FN72）: Substituting the pose Y component.
・LETZ（FN73）: Substituting the pose Z component.
・GETP（FN142）: Substituting the real number variable (coordinate value).
・GETPOSE（FN143）: Substituting the real number variable (pose variable).
・LETPOSE（FN144）: Substituting the pose variable.

The pose variable used by the above command is the orthogonal value in the machine coordinate system. In order to control the pose variable in the user coordinate system, use NRLCRD command to bind up the target steps.

NRLCRD is the command to designate (switch) the user coordinate system. These function commands can be taught from the list of function command using teach pendant.

【Example】To operate by the orthogonal value in the user coordinate system 2.

```
...
NRLCRD 2        ← Designating the user coordinate system 2.
LETX   [P1,100] ← Substituting the X coordinate value 100 on the user coordinate system 2 into the X
                  coordinate value of the variable P1.
LETY   [P2,100] ← Substituting the Y coordinate value 100 on the user coordinate system 2 into the Y
                  coordinate value of the variable P2.
LETZ   [P3,100] ← Substituting the Z coordinate value 100 on the user coordinate system 2 into the Z
                  coordinate value of the variable P3.
GETP   [V11!] ← Obtaining the current position X,Y,Z,a,b, and c on the user coordinate system 2 to the
                real number variableV11!～V16!.
GETPOSE   [V21!,1] ← Reading out the data of variable P1 to the orthogonal value of V21!～V26! on
                     the user coordinate system 2.
LETPOSE   [2,V31!] ← Writing the orthogonal value of V31!～V36! to the variable P2 as the orthogonal
                     value on the user coordinate system 2.
NRLCRD 0 ← Releasing the designated user coordinate system (later pose variables are dealt with the
           machine coordinate system.)
...
```

POINT

To correct the user coordinate system by function command:

The designated user coordinate system can be corrected by executing the MODUSRCOORD ＜Fn626＞. For details of the function command, see the HELP.

6.4.4 Numerically controlling the robot position

(1) Numerically designate the positional data on the machine coordinate system.

```
MOVEX A=8,AC=0,SM=0,M1X,P,( 710, 50, 300, 150, -45, -135),R= 3.0,H=1,MS
                        |           | ─   Numerically designating X,Y,Z,a,b,and c.
                        | ————————        MnX : Designating the machine coordinate system.
```

(2) Designate the positional data on the machine coordinate system by the real number variable.

```
MOVEX A=8,AC=0,SM=0,M1X,P,( V11!, V12!, V13!, V14!, V15!, V16!),R= 3.0,H=1,MS
                        |           | ─   Designating X,Y,Z,a,b, and c by the real number variable.
                        | ————————        MnX : Designating the machine coordinate system.
```

※The robot TCP can be moved by changing the value of real number variable V11!～V16!.

(3) Designate the positional data on the machine coordinate system by the pose variable.

```
P11=(800, -20, 190, 150, -45, -135)
MOVEX A=8,AC=0,SM=0,M1X,P,P11,R= 3.0,H=1,MS
```

(4) Substitute the positional data on the user coordinate system into the pose variable.

Add the identification symbol ‘U’ of the coordinate system after the direct value (   ). It is dealt as the direct value on the coordinate system of the user coordinate system No. designated by the user coordinate conversion command CHGCOORD. Addition of the identification symbol ‘U’ without execution of CHGCOORD will lead to no effect.

In the following case, it is considered as the orthogonal value on the user coordinate system 2.

※In substituting into the pose variable P11, it is converted to the orthogonal value on the machine coordinate system.

```
CHGCOORD 2
P11=(800, -20, 190, 150, -45, -135)U
MOVEX A=8,AC=0,SM=0,M1X,P,P11,R= 3.0,H=1,MS
CHGCOORD 0   ←   Be sure to close by “CHGCOORD 0” when the user coordinate system operation is
                 finished.
```

(5) Numerically designate the data of each axis angle.

```
MOVEX A=8,AC=0,SM=0,M1J,P,(0,90,0,0,-90,0),R= 3.0,H=1,MS
                        |           | ─   Numerically designating the angle for 6 axes.
                        | ————————        MnJ : Designating each axis angle.
```

(6) Designate the data of each axis angle by the real number variable.

```
MOVEX A=8,AC=0,SM=0,M1J,P,(V11!,V12!,V13!,V14!,V15!,V16!),R= 3.0,H=1,MS
                        |           | ─   Numerically designating the angle for 6 axes by the real
                                          number variable Vn!.
                        | ————————        MnJ : Designating each axis angle.
```

※The motion angle of the robot can be moved by changing the value of real number variable V11!～V16!.

6.4.5 Numerically controlling the move position of the external axis

(1) Numerically control the positioning of the orthogonal 2-axis. (Allocating the orthogonal 2-axis slider to Mech No.=3.)

```
MOVEX A=1,AC=0,SM=0,M3J,P,( 100, -300),R= 100,H=1,MS
                        |       | ─   Numerically designating the move position of 2-axis (mm).
                        | —————       MnJ : Designating each axis angle.
```

(2) Control the positioning of the orthogonal 2-axis by designating the V variable. (Allocating the orthogonal 2-axis slider to Mech No.=3.)

```
MOVEX A=1,AC=0,SM=0,M3J,P,( V11!,   V12!),R= 100,H=1,MS
                          |       | ─   Designating the move position of 2-axis (mm) by the real
                                        number variable.
                        | —————         MnJ : Designating each axis angle.
```

※The motion distance of the slider can be moved by changing the value of real number variable V11!,V12!.

6.4.6 Externally receiving the value by the input signal

GETSIGB(i1,i2)	Data are obtained by a bite unit from the I/O port specified by i1. (i1 is the group No. of the I/O port.) When i2=0, input signals will be got. When i2=1, output signals will be got. V1% = GETSIGB(10, 0) In the above description, the 8-bit input signal between the input signal IN49 and IN56 are converted to the binary integer and set to the integer variable V1%.	The integer value is returned.

【Example】When transferring the real number variable of the first decimal place from the external jig to the robot controller:

(1) Deal with the 16-bit consisted of the lower 8-bit of input signal I0009～I0016 and the upper 8-bit of input signal I0017～I0024 as the binary integer.
(2) The binary integer defined in (1) should be the integer value which is ten times of the real number variable used in the system.
(3) On the robot controller side, the received 16-bit binary integer is to be made a tenth part and then to be a real number.
(4) Separately prepare the sign bit to the input signal I0100. When I0100=1, it should be a minus value.

Describe the process to convert the external input signal to the real number by the above specification in the robot language. (Below)

```
V101! = GETSIGB(5,0)+2^8*GETSIGB(6,0)     ← I0009～I0016 (Lower), I0017～I0024 (Upper)
V111! = V101!/10                          ← Divide the binary integer fetched in V101! By 10.
V121! = -1^I0100*V111!                    ← Sign check
```

As a result, what value is to be stored is the real number of the first decimal place transmitted from the external jig to the real number variable V121!.

Argument "i1" of input function GETSIGB

The board internal I/O and input/output numbers from 1 to 2048 are grouped by every eight inputs/outputs. The same numbers are assigned to input/output signals.

i1	Input/output signals	i1	Input/output signals	i1	Input/output signals	i1	Input/output signals
0	Board internal I/O	8	0033～0040	～	***～***	252	1985～1992
1	Board internal I/O	9	0041～0048	245	1929～1936	253	1993～2000
2	Board internal I/O	10	0049～0056	246	1937～1944	254	2001～2008
3	Board internal I/O	11	0057～0064	247	1945～1952	255	2009～2016
4	0001～0008	12	0065～0072	248	1953～1960	256	2017～2024
5	0009～0016	13	0073～0080	249	1961～1968	257	2025～2032
6	0017～0024	14	0081～0088	250	1969～1976	258	2033～2040
7	0025～0032	～	***～***	251	1977～1984	259	2041～2048

For details of the usage, see the Chapter 2: Syntax.

6.4.7 Shifting the teaching position by the value received by input signals

(1) Shift control by SIGREQ command

Receive the shift amount by an input signal. For details, see the option manual “SHIFT FUNCTIONS BY EXTERNAL INPUT”; “2.25 SIGREQ ; Shift value get (signal) (FN723)”. (It also requires the synchronization signals such as the shift amount requirement signal and the shift setting completion signal and the parity bit to increase the reliability for the setting data.)

(2) Shift control by SHIFTA command

For SHIFTA command, each shift amount of XYZ can use the real number variable V!. As the real number variable is allowed to use the variable from 1 – 194 and 301 – 494, there is no restriction.
Substituting each shift amount fetched by GETSIGB into the real number variable V!, the shift control is enabled.

(3) Shift control by SHIFTR command

SHIFTR command uses the shift variable R. The number for shift variable R is R1 ~ R9. Substitute each shift amount fetched by GETSIGB into the real number variable V!, and then set the value of real number variable V! to the shift variable R.

```
R1=(V101!, 0, 0, 0, 0, 0) ← Substituting the value of real number variable V101! into the shift amount of X
                            coordinate system.
SHIFTR   R1, ….
```

This control enables to shift the X coordinate positional data in the SHIFTR section by the value stored in V101!.

(4) Shift control by SF8＋SF3 command
SF3 command reads out the deviation amount file (DEVxxx) in which the deviation correction amount is recorded, and corrects (shifts) the target position for the stored amount of deviation (XYZrpy).

Obtaining the external shift amount in the deviation amount file, the shift control by SF3 is enabled.

The shift amount externally specified can be obtained by GETSIGB to the real number variable V!, however the value of real number variable is stored in the deviation amount file, it is necessary to use SF8 command.

In order to use SF3 and SF8 command, it is necessary to use a sensor. In the system without sensor, register the “Touch sensor” as a dummy. This requires only a registration but not actual sensor equipment.

6.4.8 (Arc welding) How to switch the welding conditions and weaving conditions by external input signals

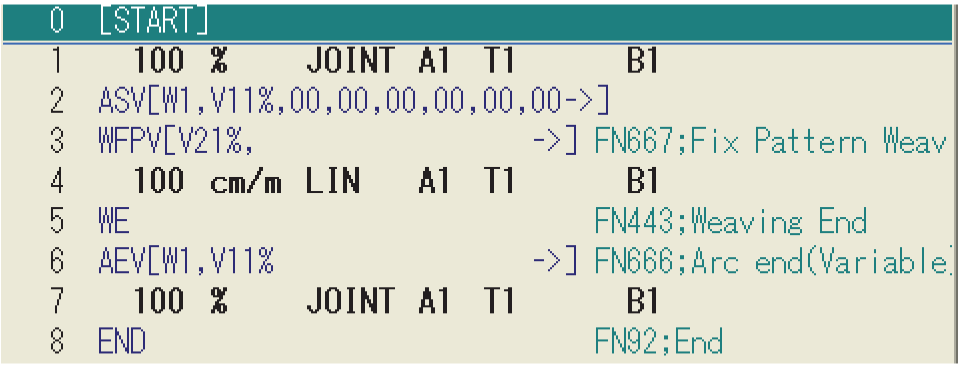
ASV, AEV, WFPV, and WAXV command are the commands that specify the welding conditions or weaving conditions by the variable Vn%.

(Procedure 1) The value specified by the integer variable Vn% is the condition file No.

Accordingly, create the AS condition file, AE condition file, or the weaving start condition file to use in advance.

(Procedure 2) Create the welding program.

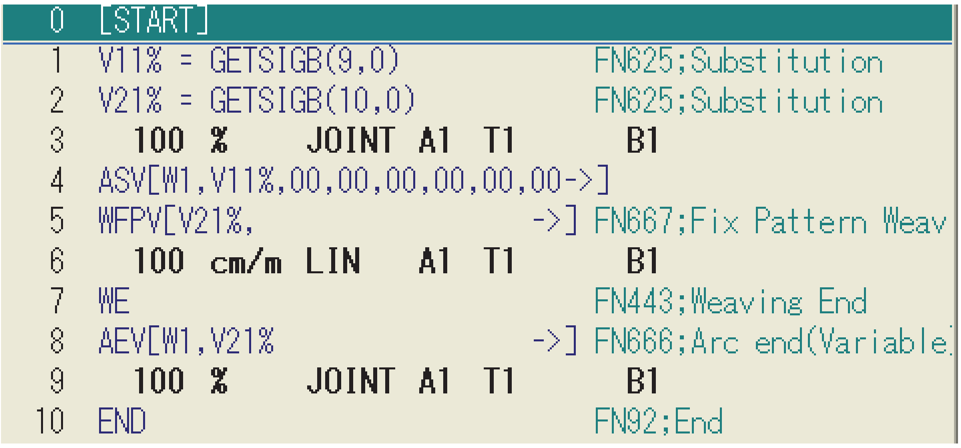
The following example shows that the AS condition file No. and AE condition file No. are specified by the integer variable V11%, and the weaving start condition file No. is specified by the integer variable V21%.



```
 0  [START]
 1    100  %     JOINT  A1   T1       B1
 2  ASV[W1,V11%,00,00,00,00,00,00->]
 3  WFPV[V21%,                     ->] FN667;Fix Pattern Weav
 4    100  cm/m  LIN    A1   T1       B1
 5  WE                                FN443;Weaving End
 6  AEV[W1,V11%                    ->] FN666;Arc end(Variable
 7    100  %     JOINT  A1   T1       B1
 8  END                               FN92;End
```

(Procedure 3) Set the file No. to the integer variable by using the external input signal.

(1) Example to fetch by GETSIGB
Fetch as IN41−IN48 → V11%, IN49−IN56 → V21%.



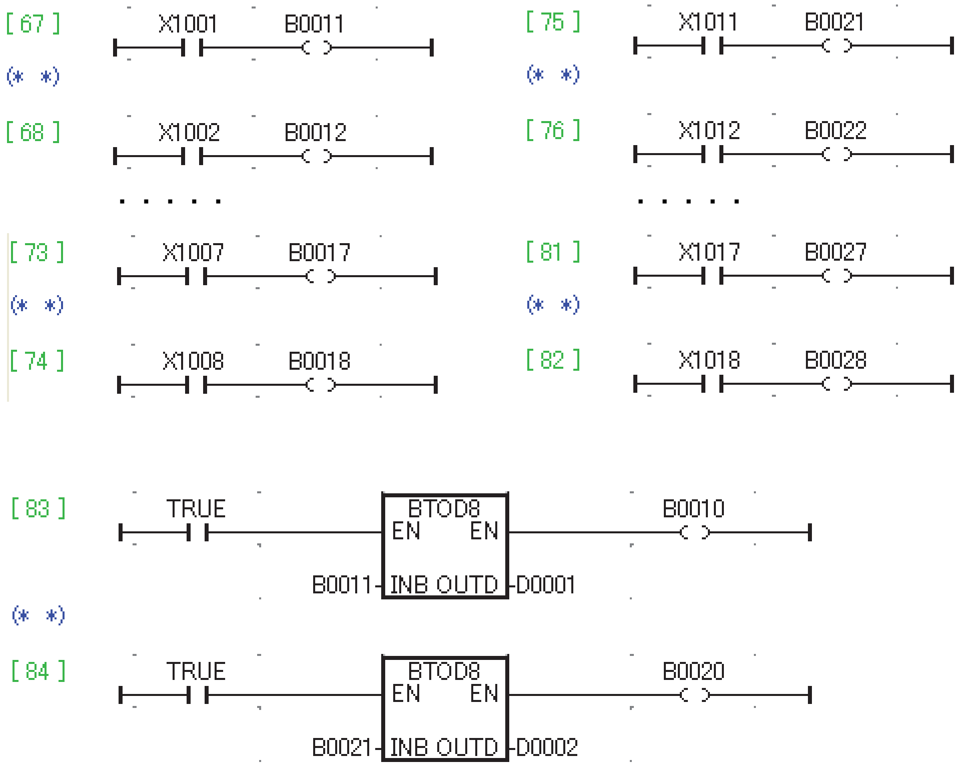
```
 0  [START]
 1  V11% = GETSIGB(9,0)               FN625;Substitution
 2  V21% = GETSIGB(10,0)              FN625;Substitution
 3    100  %     JOINT  A1   T1       B1
 4  ASV[W1,V11%,00,00,00,00,00,00->]
 5  WFPV[V21%,                     ->] FN667;Fix Pattern Weav
 6    100  cm/m  LIN    A1   T1       B1
 7  WE                                FN443;Weaving End
 8  AEV[W1,V21%                    ->] FN666;Arc end(Variable
 9    100  %     JOINT  A1   T1       B1
10  END                               FN92;End
```

(2) Example to fetch by the software PLC

① Use the common area of the integer variable Vn% and the integer variable Dxxxx in the software PLC. (D0001 → V201%, D0002 → V202%).

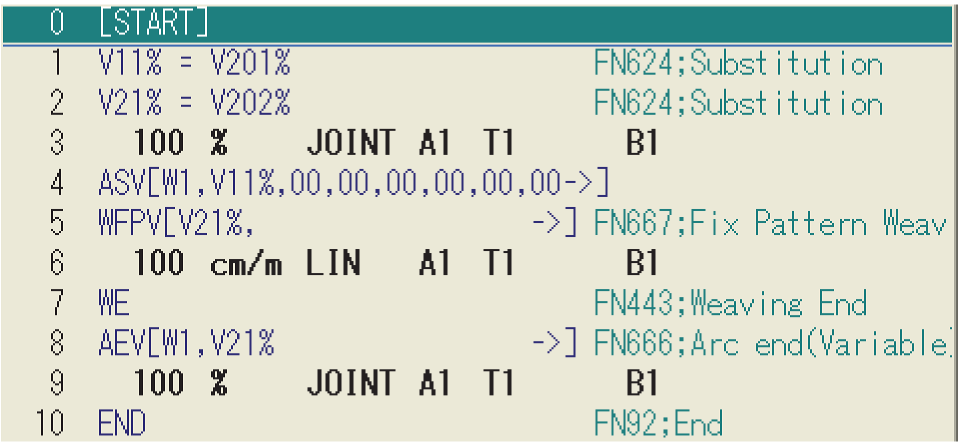
② As the condition file No. is 1 ~ 99, use 8-bit.

③ Fetch the 8-bit signal externally read in the software PLC into the integer variable. (Fetch as IN1002－IN1009 → integer variable D0001, and IN1012－IN1019 → integer variable D0002.)



④ Store the value fetched by PLC V201% and V202% to V11% and V21% at the head of welding program.

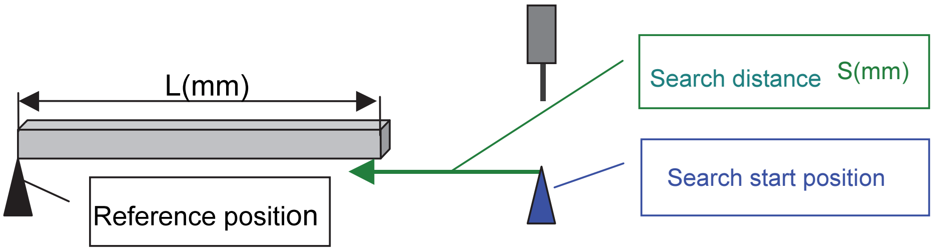
※ The integer variable V201% ~ V299% cannot be used for specifying the condition file No.



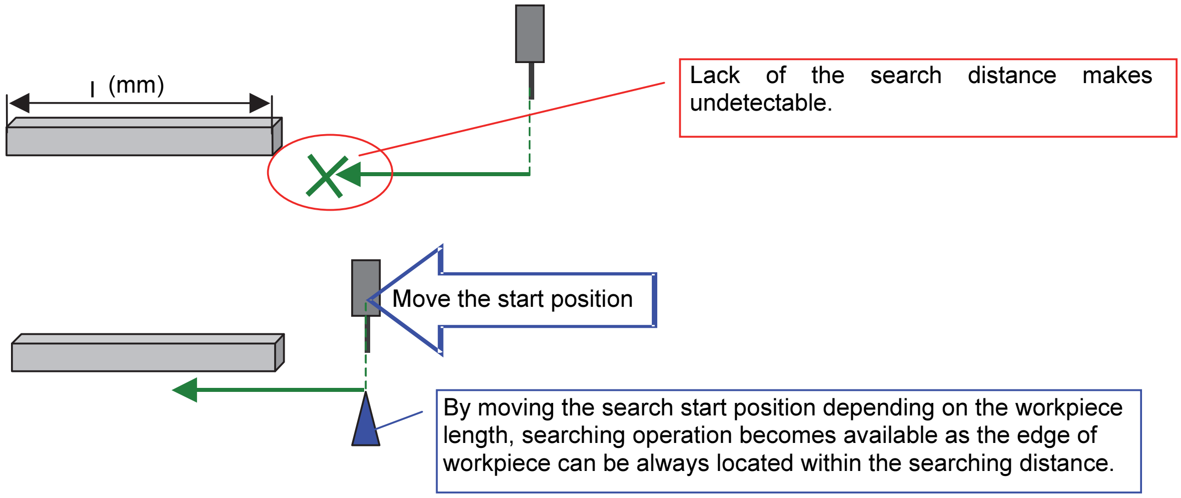
```
 0  [START]
 1  V11% = V201%                 FN624;Substitution
 2  V21% = V202%                 FN624;Substitution
 3    100 %    JOINT  A1  T1       B1
 4  ASV[W1,V11%,00,00,00,00,00,00->]
 5  WFPV[V21%,                 ->] FN667;Fix Pattern Weav
 6    100 cm/m LIN    A1  T1       B1
 7  WE                           FN443;Weaving End
 8  AEV[W1,V21%                ->] FN666;Arc end(Variable
 9    100 %    JOINT  A1  T1       B1
10  END                          FN92;End
```

6.4.9 (Arc welding) Detecting the workpiece edge of different length by Laser search

(1) Long length workpiece



(2) short length workpiece



Lack of the search distance makes undetectable.

By moving the search start position depending on the workpiece length, searching operation becomes available as the edge of workpiece can be always located within the searching distance.

Using the robot language, the start position depending on the workpiece length is automatically calculated.

＜Prerequisite＞

The searching distance S is the value taught in the parameter of the search command SF1. The workpiece length has been clear in advance.

The search distance S and the workpiece length L (l) can be obtained to the real number variable Vn! on the robot side by the external input signal.

For the method to obtain the external input signal to the real number variable, see “6.4.6 Externally receiving the value by the input signal.

【Method 1】

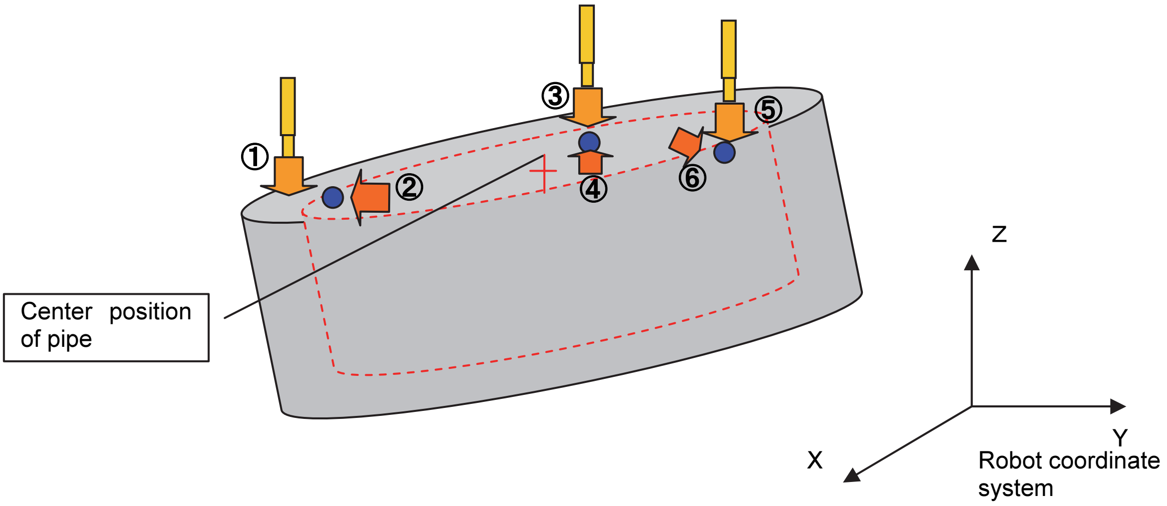
Set the position of “workpiece length + search distance –α” (mm) for the search start position as the left edge of workpiece a reference position. The reference position can be fixed in the system. The search start position is calculated by the robot language.

【Method 2】

Teach the search start position for the long length workpiece as a master workpiece, and then shift the search start position based on the change of workpiece length (a difference with the long length). Calculate the difference workpiece length in the robot language. Shift the search start position by the value of difference.

For details of the shifting method, see “6.4.7 Shifting the teaching position by the value received by input signals”.

6.4.10 (Arc welding) Calculating the center position of pipe by sensing 3 edge points on the inner face of cylindrical pipe



Using the touch sensor, detect three positions on the inner face of the cylindrical pipe workpiece.

It is available to calculate the center position of pipe from three points on the circumference.

(1) Create the master program to implement the search motion and the position detection shown in the above ①~⑥ for the master workpiece as a reference.

- The deviation in the vertical direction (= the height position of edge) is detected by ①, ③, and ⑤.
- Next, perform automatic adjustment for the height in detecting based on the position detected by ①, ③, and ⑤ not to mis-search in the horizontal direction of ②, ④, and ⑥.

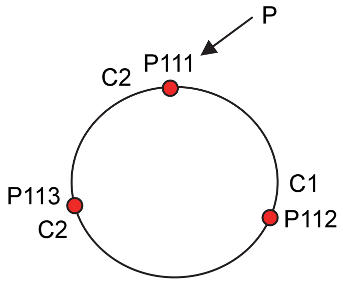
As this master program needs to obtain three points on the inner face of the cylindrical pipe, it is no problem even if creating it without using the robot language.

(2) Then, execute the master program for every workpiece to calculate the center position for each target workpiece.

In the next page, the sample of master program to calculate the center is described.

Furthermore, it is available to first record three points on the inner face detected by touch sensing in the pose variable, and then to create the circular arc passing through three points using those pose variables.
For example, if those three inner points detected by touch sensing are recorded in the pose variable P111, P112, and P113, the circular trajectory; Joint → P111 → circular arc C1 → P112 → circular arc C2 → P113 → circular arc C2 → P111 can be described by variables.
A single program can support the change of pipe size.

```
MOVEX A=1,AC=0,SM=0,M1X, P, P111, R= 100,H=1,MS, CONF=0020
MOVEX A=1,AC=0,SM=0,M1X, C1, P112, S= 16.6,H=1,MS, CONF=0020
MOVEX A=1,AC=0,SM=0,M1X, C2, P113, S= 16.6,H=1,MS, CONF=0020
MOVEX A=1,AC=0,SM=0,M1X, C2, P111, S= 16.6,H=1,MS, CONF=0020
```



From here, the outline of master program to calculate the center position of pipe is described. Some of the processes are skipped occasionally.

```
REM "Sensing 3 points on the pipe inner face & Calculating the center"
   ・・・
REM "Teaching by the pose record P7501"
USE 7501
   ・・・
REM "Searching the 1st point"
SF1 Detecting the touch sensing deviation in the vertical direction.
SF3 Start      ・・・Deviation correction in the vertical direction
SF1 Touch detection in the horizontal direction
SF3 End
REM "Touch sensing position in the horizontal direction→V111"
GETP V111!        ・・・ Obtaining the current position.
LETPOSE 111,V111!    ・・・ Record the 1st position in the pose file.
   ・・・
REM "2nd point search"
SF1 Detecting the touch sensing deviation in the vertical direction.
SF3 Start      ・・・Deviation correction in the vertical direction
SF1 Touch detection in the horizontal direction
SF3 End
REM " Touch sensing position   V121"
GETP V121!
LETPOSE 112,V121!    ・・・ Record the 2nd position in the pose file.
   ・・・
REM "Searching the 3rd point"
SF1 Detecting the touch sensing deviation in the vertical direction.
SF3 Start      ・・・Deviation correction in the vertical direction
SF1 Touch detection in the horizontal direction
SF3 End
REM "Touch sensing position V131"
GETP V131!
LETPOSE 113,V131!    ・・・ Recording the 3rd point in the pose file.
NOP
REM "Obtaining the circular center by 3 points"
GETPOSE V401!,111
GETPOSE V404!,112
GETPOSE V407!,113
REM "Calculation of the circular center by 3 points"
GETPOSE V441!,113         ← Retaining the positional posture of the 3rd point. (Updating XYZ after calculating
                            the center position.)
REM "Calculation of the circular center"
CALLP 5100   (※1)
REM "Recording the circular center in P1"
LETPOSE 1,V441!              ← Storing the circular center in the pose variable 1 of pose file P7501.
POSESAVE
END
```

(※1) P5100 is the common function to calculate the circular center by 3 points. If necessary, please contact us.

1) Input data to the function (Transmitting the orthogonal value XYZ of each three point.)

1st point (V401!,V402!,V403!)   2nd point (V404!,V405!,V406!)   3rd point (V407!,V408!,V409!)

2) Output data from the function (The calculated pipe center position XYZ are output.)

V441!＝Center X   V442!＝Center Y   V443!＝Center Z

6.4.11 (Arc welding) Creating the coordinates dedicated to the inner face of cylindrical pipe

It enables to detect the tilt of cylindrical pipe if the method described in the section 6.4.10 is allowed to calculate the center A and B of the cylindrical pipe. Based on this tilt line (＝ Center AB), the dedicated user coordinate system can be created.

In order to create the user coordinate system, use function command MODUSRCOORD to designate three reference points to form the coordinate axis.
There are three ways to designate three points; OXY／OZX／OYX.

The following is the example designated by **OZX method.**

First, perform sensing the position of center B to the point O (the origin of user coordinate system) in the direction of from the center B to A for the distance of Z (Z axis (+) direction in the user coordinate system). One of the sensed three points on the circumference is designated as X (X axis (+) direction in the user coordinate system).

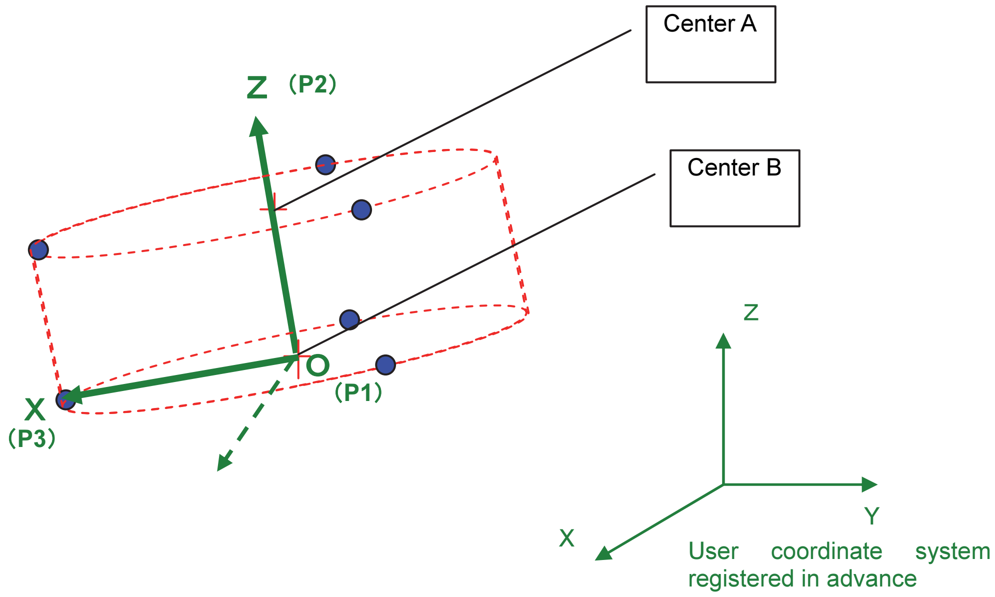
The procedure to create a new user coordinate system according to the workpiece shown in the figure below is described as follows.

(1) Register the user coordinate system by OZX method. Three reference points in registered are arbitrarily selected.
(2) Create the task program to execute the function command MODUSRCOORD of the user coordinate system corrected.

MODUSRCOORD (user coordinate system No., 1st pose variable No., 2nd pose variable No., 3rd pose variable No.)
In the following example:
User coordinate system No. : 1 (The coordinate system No. registered in advance)
The 1st pose variable No. : 1 (The pose variable equivalent to “O” of the OZX method ＝P1)
The 2nd pose variable No. : 2 (The pose variable equivalent to “Z” of the OZX method ＝P2)
The 3rd pose variable No. : 3 (The pose variable equivalent to “X” of the OZX method ＝P3)

(3) Store three reference points in the new coordinate system in each pose variable P1, P2, and P3.
(4) Execute the task program created in (2).

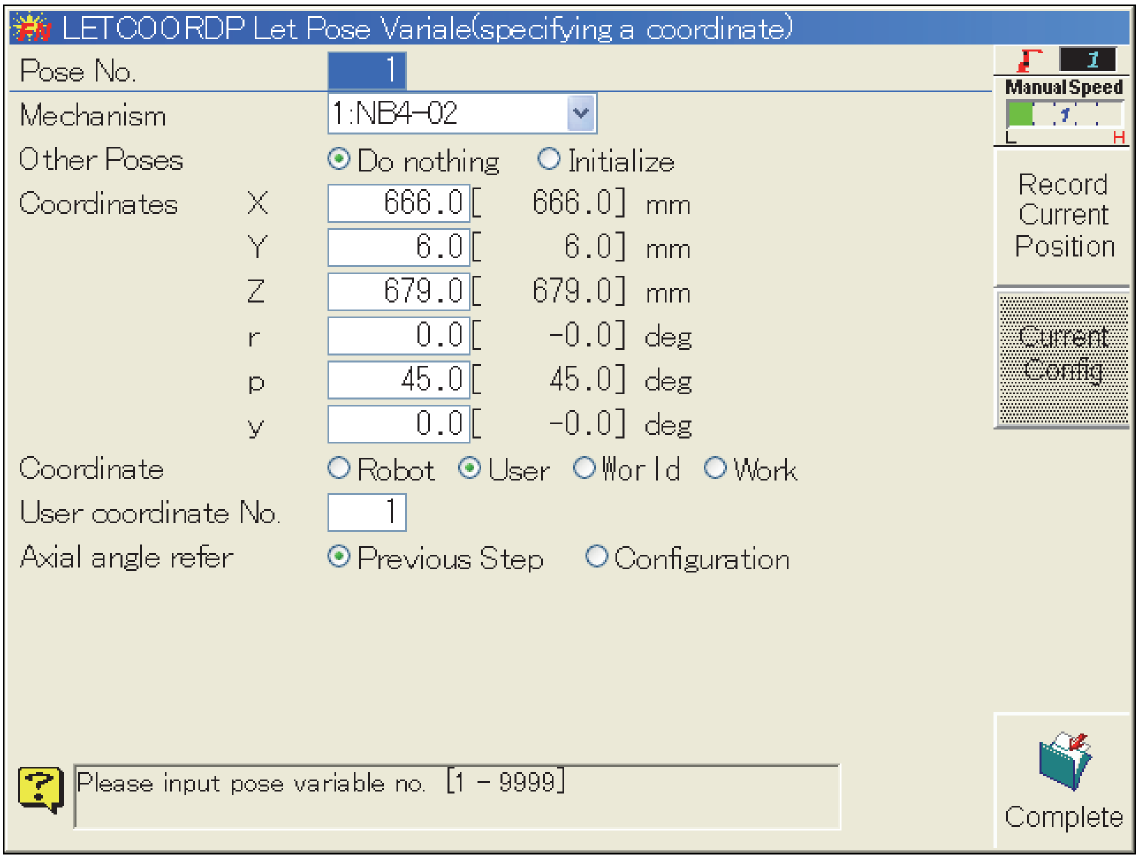
Thus, it enables to change the registered user coordinate system 1 to any coordinate systems according to the workpiece. Using this method, it enables to support by the same welding program even if the position or tilt angle of the workpiece changes as long as the welding program has been created based on the user coordinate system.



6.4.12 Function command that enables substituting the command value of arbitrary coordinates to pose variables

Function command Fn630 LETCOORDP allows to teach TCP and the tool angle in arbitrary coordinate systems by the teach pendant. TCP and the tool angle designated by LETCOORDP command are recorded in the “pose variable” of the pose variable No. specified on the setting screen.

After that, use the function command Fn645 MOVEX record to read out the “pose variable” recorded by LETCOORDP command, which enables to operate the robot to the designated position and tool angle.



LETCOORDP Let Pose Variale(specifying a coordinate)

Pose No.		1		
Mechanism		1:NB4-02		
Other Poses		● Do nothing	○ Initialize	
Coordinates	X	666.0	[666.0]	mm
	Y	6.0	[6.0]	mm
	Z	679.0	[679.0]	mm
	r	0.0	[-0.0]	deg
	p	45.0	[45.0]	deg
	y	0.0	[-0.0]	deg
Coordinate		○ Robot ● User ○ World ○ Work		
User coordinate No.		1		
Axial angle refer		● Previous Step	○ Configuration	

Please input pose variable no. [1 - 9999]

As for the value teaching by teach pendant, see the option manual “ROBOT LANGUAGE Operation by TP” for details.

This manual also describes the usage examples as below where the pose variables are applied.

- Obtaining the robot position posture
- Linear motion among 4 points specified by value
- Circular motion where the center position is designated (90 degrees division)
- Detection of the pipe center
- Circular motion where the center position is designated (45 degrees division)

6.4.13 Reading the current position of robot to externally output

(1) Monitoring the current position of robot

The following shows an example in using the macro function SYSTEM!(). The macro command (function) can be employed only by the user task program. The positional data read out are stored in the real number variable (Vn!).

For example, when always monitoring the current position of robot in the machine coordinate system:

① Create the user task program that operates in the background at constant cycle.

The user task is first created by the text file and then to be converted to the executable format by program conversion function.

Create the text file of “USERTASK-A.100” as the user task P100.

② Create the text file of “USERTASK-A.100”

Read out the XYZ position in the machine coordinate system of Mech 1 to the real number variable V11!∼V13! at 50msec cycle.

```
REM "Reading the current position of robot (TCP in the machine coordinate system)"
REM "V11! ←  Current TCP of Mech 1 (X coordinate)"
V11! = SYSTEM!(150)
REM "V12! ←  Current TCP of Mech 1 (Y coordinate)"
V12! = SYSTEM!(151)
REM "V13! ←  Current TCP of Mech 1 (Z coordinate)"
V13! = SYSTEM!(152)
V201% = V11! * 10
V202% = V12! * 10
V203% = V13! * 10
PAUSE 50
GOTO 1
END
```

The details are given in the following paragraph (2) “Output the current position to external signals”

③ Use the Program conversion ― Language conversion function to convert “Language format → Executable format” (either of encoder format or base/each axis format is fine). Thus, “USERTASK.100” is created.

④ Start the user task P100.

【Start method 1】Go to <Service Utilities> - [12 user task] and start P100.

【Start method 2】Use the task program to start P100 by “FORKMCR” command.

(2) Output the current position to external signals.

Output the value of read-out current position XYZ (V11!,V12!,V13!) to external signals.

As the external signals cannot express the real number, first multiply by ten to make it to the integer to be output.

While on the receiving side (jig side), make it a tenth part and return to the real number value.

The output value is stored in the common area with the software PLC and the inner variables (integer variable).

(Common area : V201%―V250% → Corresponding to the integer variable D0001―D0050 of the software PLC.)

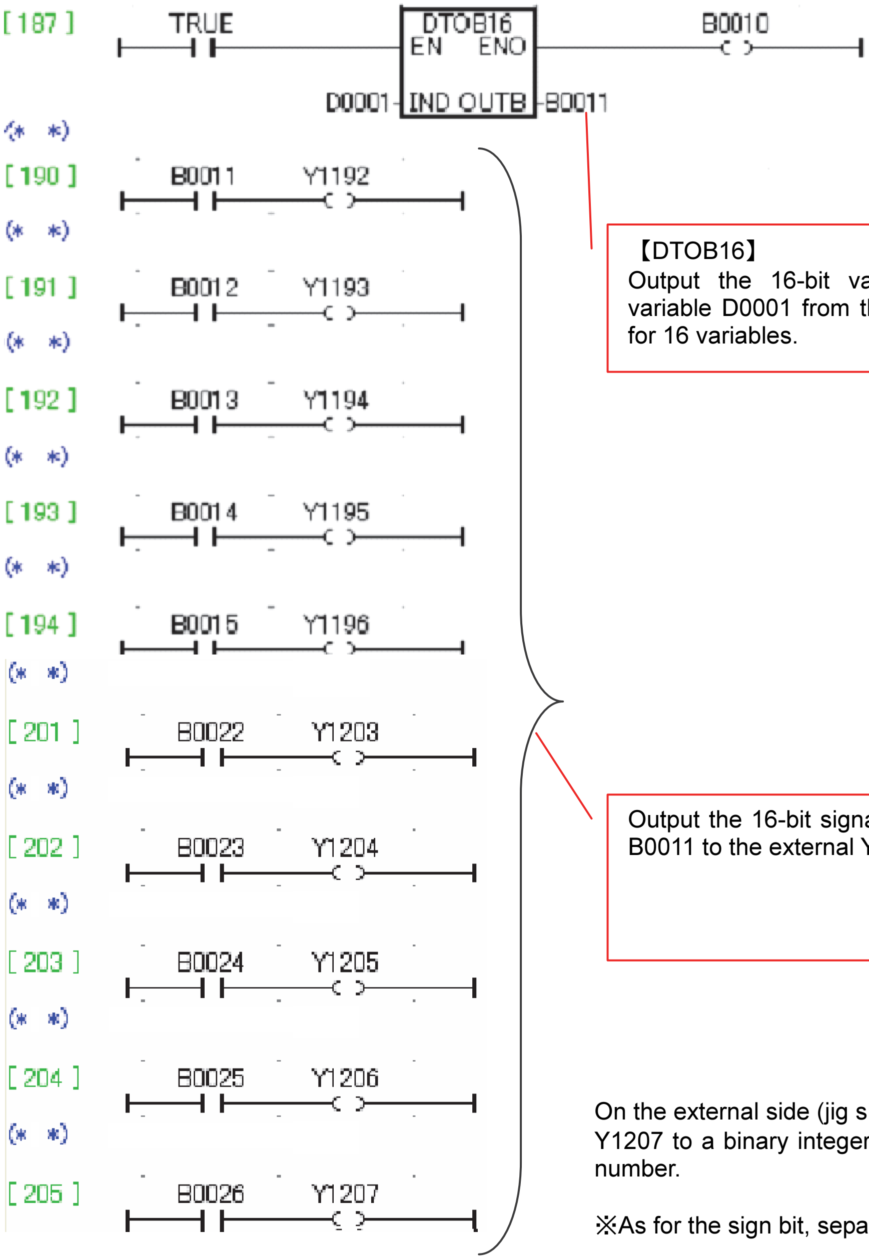
Multiply the real number of the read-out current position XYZ (V11!,V12!,V13!) by ten and set them to the common area with the software PLC (V201%,V202%,V203%).

```
V201% = V11! * 10
V202% = V12! * 10
V203% = V13! * 10
```

(3) Read the inner variable data of the robot language by the software PLC.

Since the variables are corresponding as follows; V201%－V250% → Integer variable D0001－D0050 of the software PLC, the below is applied in the same way.
V201% → D0001, V202% → D0002, V203% → D0003.

The following shows a sample logic of the software PLC.



【DTOB16】
Output the 16-bit value of the inner integer variable D0001 from the boolean variable B0011 for 16 variables.

Output the 16-bit signal from the boolean variable B0011 to the external Y1192 and after.

On the external side (jig side), make the 16-bit value of Y1192－Y1207 to a binary integer and make it a tenth part to be a real number.

※As for the sign bit, separately prepare one dedicated signal.

Appendix : Binary number / decimal number / hexadecimal number

Appendix 1）About Binary number／Decimal number／Hexadecimal number system

Decimal number system

The decimal number system expresses a number using ten distinct numeric characters from 0 to 9. A number is counted as 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 incremented by one in this order, and then to 10 in the next digit in the tens position. Thus, in the decimal number system, the digit is rounded up from 1 to such as 10, 100, 1000, 10000, and so forth.
The decimal number 1, in other words, is represented as the zero power of ten (10^0), 10 is the first power of ten (10^1), 100 is the second power of ten (10^2), 1000 is the third power of ten (10^3), and so on. The digit is rounded up in the decimal notation in the manner such as 10^0, 10^1, 10^2, and 10^3 in this regard.

For example, the decimal number 2976 is expressed as below.

Position of 10^3	Position of 10^2	Position of 10^1	Position of 10^0
2	9	7	6

This number is expressed by the formula below.

$2\times10^3 + 9\times10^2 + 7\times10^1 + 6\times10^0$
$= 2\times1000 + 9\times100 + 7\times10 + 6\times1$
$= 2976$

Binary number system

The binary number system expresses a number using two distinct numeric characters 0 and 1. A number is counted as 0 and 1, incremented by one in this order, and then to 10 in the next digit in the twos position. Thus, in the binary number system, the digit is rounded up in such as 2^0 (1), 2^1 (2), 2^2 (4) and 2^3 (8). (The number given in (   ) represents the decimal one.)

For example, the binary number 1101 is expressed as below.

Position of 2^3	Position of 2^2	Position of 2^1	Position of 2^0
1	1	0	1

This number is expressed by the formula below.

$1{\times}2^3 + 1{\times}2^2 + 0{\times}2^1 + 1{\times}2^0$
$= 1{\times}8 + 1{\times}4 + 0{\times}2 + 1{\times}1$
$= 13$ (the decimal number)

To convert a binary number to a decimal number, it is easier to vertically write down and count up as below.

$\mathbf{1} \times 2^3$	=	8
$\mathbf{1} \times 2^2$	=	4
$\mathbf{0} \times 2^1$	=	0
$\mathbf{1} \times 2^0$	=	$\underline{+)\quad 1}$
		13

To convert a decimal number to a binary number on the other hand, first divide a decimal number by 2, divide the result of division by 2, and then further the result of division by 2, and so forth. Repeat this calculation with the remainder left until the result of division becomes 0. And place all the remainders from the bottom upward with the last remainder put in undermost.
For example, the decimal number 19 is calculated as below.

19÷2	=	9	Remainder	**1**
9÷2	=	4	Remainder	**1**
4÷2	=	2	Remainder	**0**
2÷2	=	1	Remainder	**0**
1÷2	=	0	Remainder	**1**
	=	10011		

Conversion chart of decimal and binary number

Decimal number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Binary number	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111	10000

Hexadecimal number system

The hexadecimal number system expresses a number using numeric characters from 0 to 9 and alphabetic characters from A to F.

The number is counted as 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F incremented in this order, and then to 10 in the next digit.

The hexadecimal number A is represented as 10 in the decimal system, B is as 11, C is as 12, D is as 13, E is as 14, and F is as 15.

Thus, in the hexadecimal number system, the digit is rounded up in such as 16^0 (1), 16^1 (16), 16^2 (256) and 16^3 (4096). (The number given in (   ) represents the decimal one.)

For example, the hexadecimal number 4E5F is expressed as below.

Position of 16^3	Position of 16^2	Position of 16^1	Position of 16^0
4	E	5	F

This number is expressed by the formula below.

$$4\times16^3 + \mathrm{E}\times16^2 + 5\times16^1 + \mathrm{F}\times16^0$$
$$= 4\times4096 + \mathrm{E}(14)\times256 + 5\times16 + \mathrm{F}(15)\times1$$
$$= 20063 \text{ (the decimal number)}$$

To convert a hexadecimal number to a decimal number, it is easier to vertically write down and count up as below.

4×16^3	=	16384
E（14） $\times16^2$	=	3584
5×16^1	=	80
F（15） $\times16^0$	=	+) 15
		20063

To convert a decimal number to a hexadecimal number on the other hand, first divide a decimal number by 16, divide the result of division by 16, and then further the result of division by 16, and so forth. Repeat this calculation with the remainder left until the result of division becomes 0. And place all the remainders from the bottom upward with the last remainder put in undermost.

For example, the decimal number 1000 is calculated as below.

1000÷16	=	62	Remainder	**8**
62÷16	=	3	Remainder	**14 (E)**
3÷16	=	0	Remainder	**3**
	=	3E8	(Hexadecimal number system)	

A number system in computer uses the binary system to represent every number; using 0 and 1. It excels at the binary and the hexadecimal number system that is a multiplier factor of 2 comparing with the decimal number system which has been generally used in the human world. Therefore, the programming system often uses the hexadecimal number system.
Note that the following notation system is commonly used when it is necessary to make a distinction among a decimal, binary and hexadecimal number system.

Decimal number	15d	$15_{(10)}$
Binary number	1001b	$1001_{(2)}$
Hexadecimal number	10h	$10_{(16)}$

‘d’ = Capital letter of Decimal, ‘b’ = Capital letter of Binary, ‘h’ = Capital letter of Hexadecimal

Conversion chart of decimal, binary and hexadecimal number

Decimal number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Binary number	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111	10000
Hexadecimal number	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10

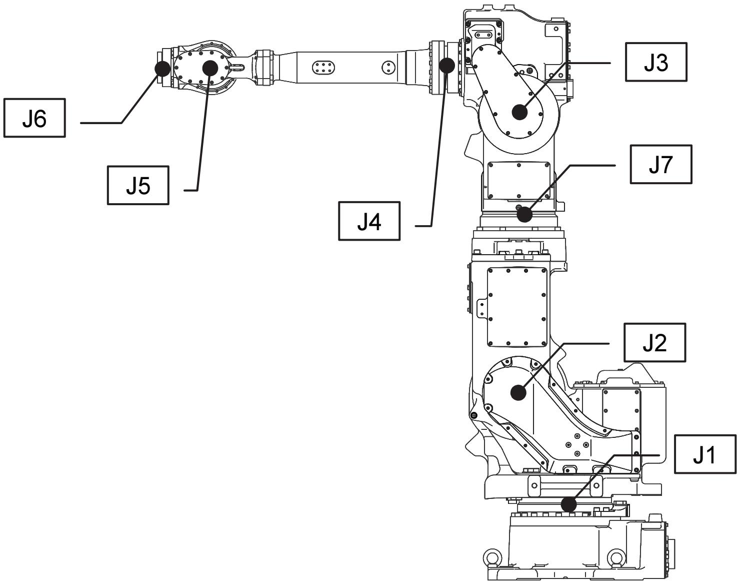
Appendix : Supplemental explanations for each robot

MR series

In case of the MR series, in robot language, from J2 to J7 axes belong to the mechanism1, and J1 axis belongs to the mechanism2. Therefore, please write the robot language movement command like the following. And please be sure that the axis order of the mechanism1 is J2, J7, J3, J4, J5, J6 when using MOVEX-J format.

```
USE 1
P1 = (X,Y,Z,roll,pitch,yaw,J1)
MOVEX A=1,M1X,P,(X,Y,Z,roll,pitch,yaw),R=10,H=1,MS,M2J,P,(J1),R=5,H=1
MOVEX A=1,M1J,P,(J2,J7,J3,J4,J5,J6),    R=10,H=1,MS,M2J,P,(J1),R=5,H=1
MOVEX A=1,M1X,P,P1,R=5.0,H=1,MS,M2J,P,P1,R=5,H=1
```

- From "J1" to "J7" stand for each axis's angle. Please refer to the following picture also.



POINT

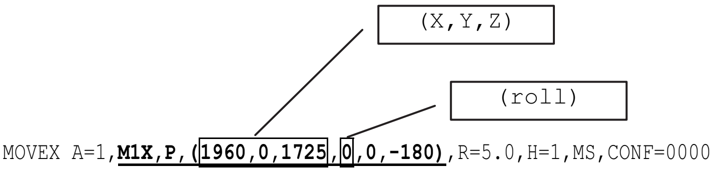
When operating this robot using a teach pendant, J7 axis is regarded as the mechanism number 2 to make the operation easier. However, in the robot language, the axis that belongs to the mechanism 2 is J1 axis.Be careful.

LP series

MOVEX-X format

Like other robots, MOVEX-X can be used in the style of (X,Y,Z,roll,pitch,yaw).

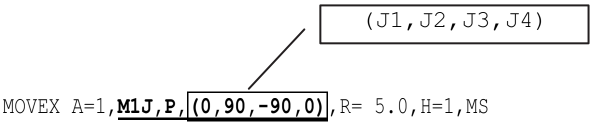
The initial value of (roll, pitch, yaw) depends on the tool setting parameter. But the value that changes via the J4 axis rotation is only roll and the other 2 values always keep their original values. It is strongly recommended to check the tool constant settings and those values in the axis data monitor window before writing a robot language program. Wrong values for the (roll,pitch,yaw) will cause wrong motion of the robot.



```
MOVEX A=1,M1X,P,(1960,0,1725,0,0,-180),R=5.0,H=1,MS,CONF=0000
```

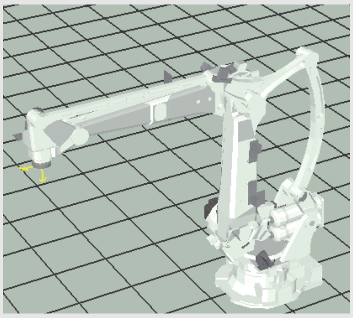
MOVEX-J format

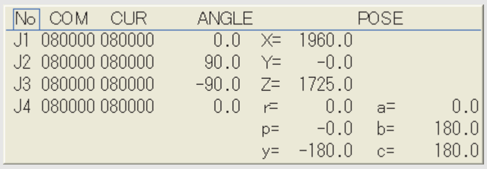
Please be sure that the number of the parameter is 4 because this robot is 4 axes robot. And the mechanical reference posture is (0, 90, -90, 0).



```
MOVEX A=1,M1J,P,(0,90,-90,0),R= 5.0,H=1,MS
```

LP series robot mechanical reference posture





```
No  COM    CUR       ANGLE             POSE
J1  080000 080000      0.0  X=  1960.0
J2  080000 080000     90.0  Y=    -0.0
J3  080000 080000    -90.0  Z=  1725.0
J4  080000 080000      0.0  r=     0.0  a=     0.0
                            p=    -0.0  b=   180.0
                            y=  -180.0  c=   180.0
```

(NOTE) In this example, the tool constant setting is default setting.

NACHI NACHI-FUJIKOSHI CORP.	http://www.nachi-fujikoshi.co.jp/		
Japan Main Office	Phone: +81-3-5568-5245	Fax: +81-3-5568-5236	Shiodome Sumitomo Bldg. 17F, 1-9-2 Higashi-Shinbashi Minato-ku, TOKYO, 105-0021 JAPAN
Nachi Robotic Systems Inc. (NRS)	http://www.nachirobotics.com/		
North America Headquarters	Phone: 248-305-6545	Fax: 248-305-6542	42775 W. 9 Mile Rd. Novi, Michigan 48375, U.S.A
Indiana Service Center	Phone: 248-305-6545	Fax: 248-305-6542	Greenwood, Indiana
Ohio Service Center	Phone: 248-305-6545	Fax: 248-305-6542	Cincinnati, Ohio
South Carolina Service Center	Phone: 248-305-6545	Fax: 248-305-6542	Greenville, South Carolina
Canada Branch Office	Phone: 905-760-9542	Fax: 905-760-9477	89 Courtland Ave., Unit No.2, Concord, Ontario, L4K 3T4, CANADA
Mexico Branch Office	Phone :+52-555312-6556	Fax:+52-55-5312-7248	Urbina No.54, Parque Industrial Naucalpan, Naucalpan de Juarez, Estado de Mexico C.P. 53489, MEXICO
NACHI EUROPE GmbH	http://www.nachi.de/		
Central Office Germany	Phone: +49-2151-65046-0	Fax: +49-2151-65046-90	Bischofstrasse 99, 47809, Krefeld, GERMANY
U.K. branch	Phone: +44-0121-423-5000	Fax: +44-0121-421-7520	Unit 3, 92, Kettles Wood Drive, Woodgate Business Park, Birmingham B32 3DB, U.K.
Czech branch	Phone: + 420-255-734-000	Fax: +420-255-734-001	Obchodni 132, 251 01 Cestlice, PRAGUE-EAST CZECH REPUBLIC
NACHI AUSTRALIA PTY. LTD.	http://www.nachi.com.au/		
Robotic Division & Victoria office	Phone: +61-(0)3-9796-4144	Fax: +61-(0)3-9796-3899	38, Melverton Drive, Hallam, Victoria 3803, , AUSTRALIA
Sydney office	Phone: +61-(0)2-9898-1511	Fax: +61-(0)2-9898-1678	Unit 1, 23-29 South Street, Rydalmere, N.S.W, 2116, AUSTRALIA
Brisbane office	Phone: +61-(0)7-3272-4714	Fax: +61-(0)7-3272-5324	7/96 Gardens Dr,Willawong,QLD 4110, , AUSTRALIA
NACHI SHANGHAI CO., LTD.	http://www.nachi.com.cn/		
Shanghai office	Phone: +86-(0)21-6915-2200	Fax: +86-(0)21-6915-2200	11F Royal Wealth Centre, No.7 Lane 98 Danba Road Putuo District, Shanghai 200062, China
NACHI KOREA	http://www.nachi-korea.co.kr/		
Seoul office	Phone: +82-(0)2-469-2254	Fax: +82-(0)2-469-2264	2F Dongsan Bldg. 276-4, Sungsu 2GA-3DONG, Sungdong-ku, Seoul 133-123, KOREA

Robot Division
1-1-1, FUJIKOSHIHONMACHI, TOYAMA CITY, JAPAN 930-8511
Phone +81-76-423-5137
Fax +81-76-493-5252

In case that an end user uses this product for military purpose or production of weapon, this product may be liable for the subject of export restriction stipulated in the Foreign Exchange and Foreign Trade Control Law. Please go through careful investigation and necessary formalities for export.

Original manual is written in Japanese.