

HD-XXXX-1

HrBasic Reference Manual



Ver. 5.50

The information contained herein is the property of Hirata Corporation and shall not be reproduced in whole or in part without prior written approval of Hirata Corporation. The information contained herein is subject to change without notice and should not be constructed as a commitment by Hirata Corporation.

Hirata Corporation assumes no responsibility for any errors or omissions in this document.

Warranty

All of Hirata's products which are passed our formal inspection test shall be guaranteed against faults due to the negligence of Hirata for either earlier period of one year or four thousand hours of operation from the day of shipment from Hirata Factory.

This warranty shall be applicable to the parts replacement and/or labor for repair in our factory and transportation cost shall not be applied.

We will charge the repair of faults caused by the following reasons:

- * Wrong usage which is prohibited in the instruction manual.
- * After the expiration of guarantee period.
- * Earthquake, fire, riot, violence, war and other force majeure.
- * Modification, repair or adjustment is performed by unauthorized person.

Contact your sales agent for individual warranty coverage.

HrBasic Reference Manual

Ver. 5.50

USER'S GUIDE (HD-XXXXE-1)

Copyright 2004-2005 by Hirata Corporation All right reserved.

First published in January 2005

Printed in Japan

Hirata Corporation

Tokyo Head Quarters

3-9-20 Togoshi, Shinagawa, Tokyo 142-0041 JAPAN

Phone (03) 3786-1226

Facsimile (03) 3786-1264

Robotics Division

1016-6 Kusuno, Kumamoto 861-5511 JAPAN

Phone (096) 245-1333

Facsimile (096) 245-0816

1. Introduction

1.1 Hirata Robot System

- HNC and HAC

There are mainly two kinds of controllers named as “HNC” and “HAC” produced by Hirata Corporation.

A HNC (Hirata Numerical Controller) can control a Hirata robot servo system by the instructions using remote I/O or communication.

A HAC (Hirata Assembly Controller) is programmable by HrBasic language in addition to HNC functions. A HrBasic execution component of HAC is named as “STP”. Therefore, functionally, HAC = HNC + STP.

- STP and WinSTP

STP (STation Processor) is the software that can execute the program developed by robot control language HrBasic.

A standard HAC-8XX controller is equipped with STP. After the HrBasic program developed on a PC is downloaded to STP, STP interprets and executes HrBasic program.

WinSTP is STP that runs on a Windows PC. WinSTP is one of software components of HBDE and it can execute the HrBasic program on a PC.

- HrBasic

HrBasic is the language based on BASIC to learn easily that includes the extended statements for robot control, I/O control and timer control and that can run as maximum 32 jobs simultaneously. You can develop and debug the HrBasic program on a Windows PC using HBDE.

- HBDE

HBDE (HrBasic Developing Environment) is the integrated developing software for a robot control system using HrBasic on a Windows PC. You can operate and manage projects, developing programs, compiling, linking, downloading to STP, debugging, monitoring I/O, robot setting data.

1.2 HrBasic and STP

This manual describes the programming language HrBasic which runs in STP. STP controls a Hirata robot system or various peripheral devices with communication of RS232C, internal bus or fieldbus and Ethernet network.

HrBasic can operate global variables, various control memories and timer. And it can communicate easily with PC, robots and deices through the file system architecture.

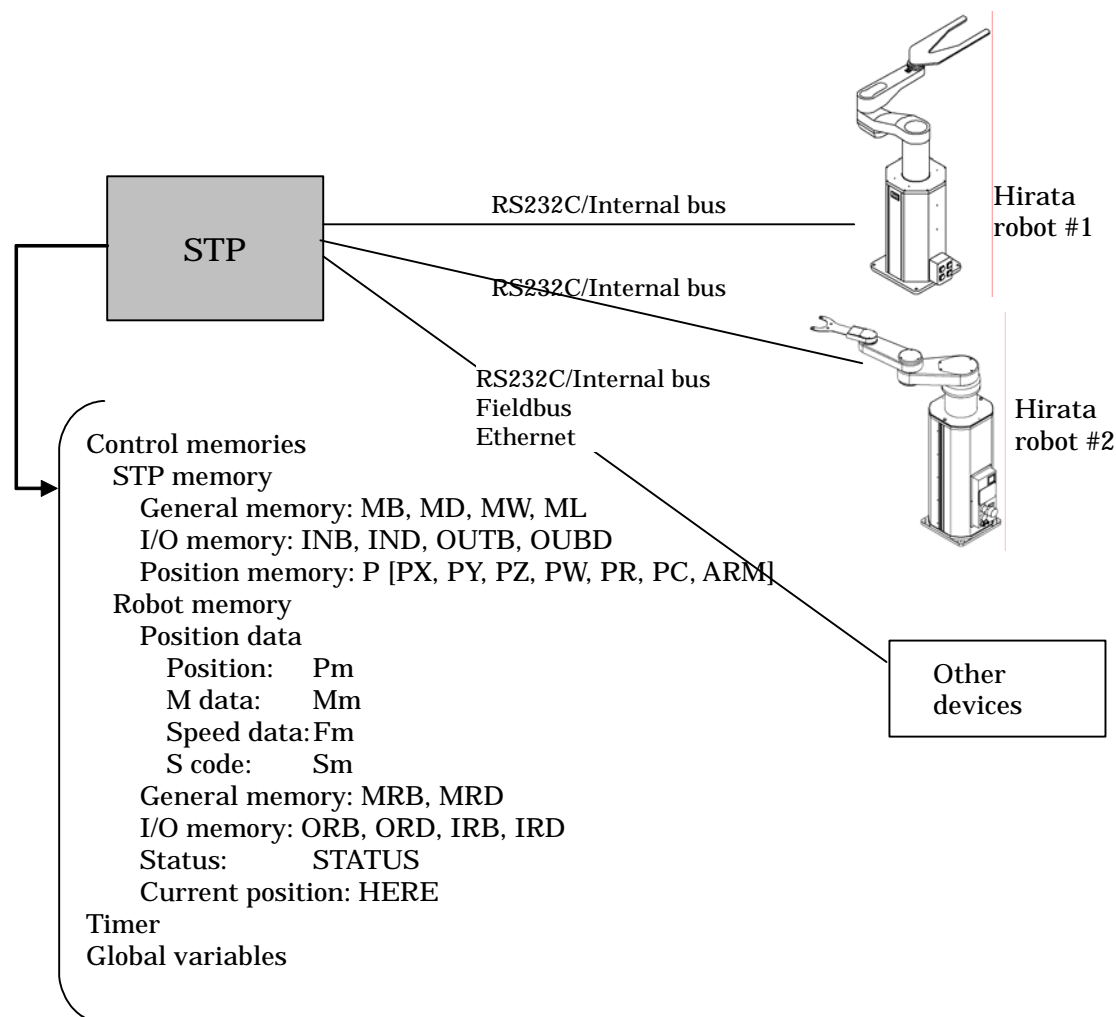
The control memories are categorized to two types which are STP memories and robot memories.

The STP memories include the followings.

- General memory
- I/O memory
- Temporary position memory

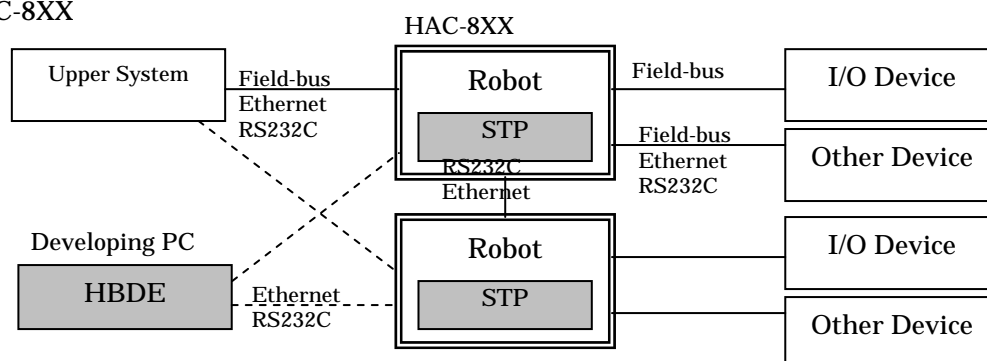
The program can access the connected robot information such as the followings.

- Current robot position
- Teaching position data
- Robot Status

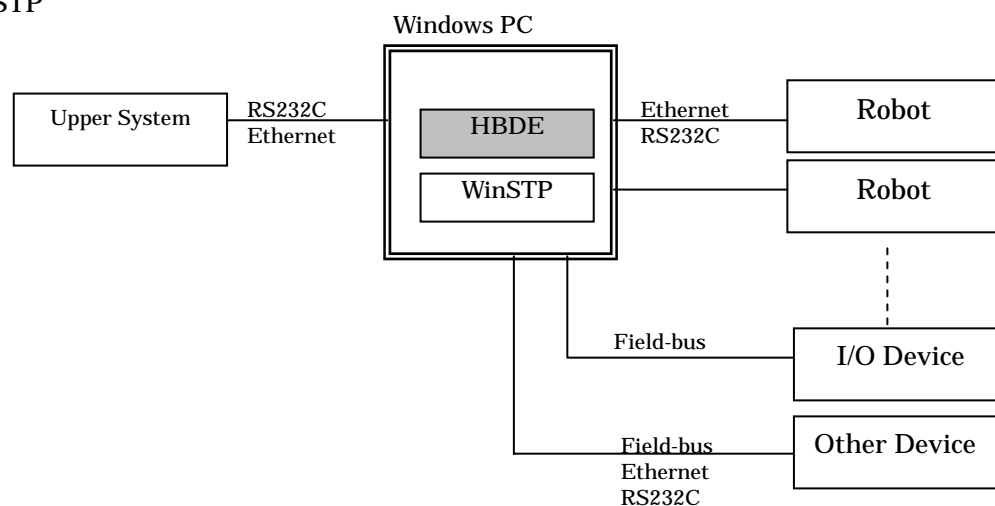


1.3 Example of Robot System

Using HAC-8XX



Using WinSTP



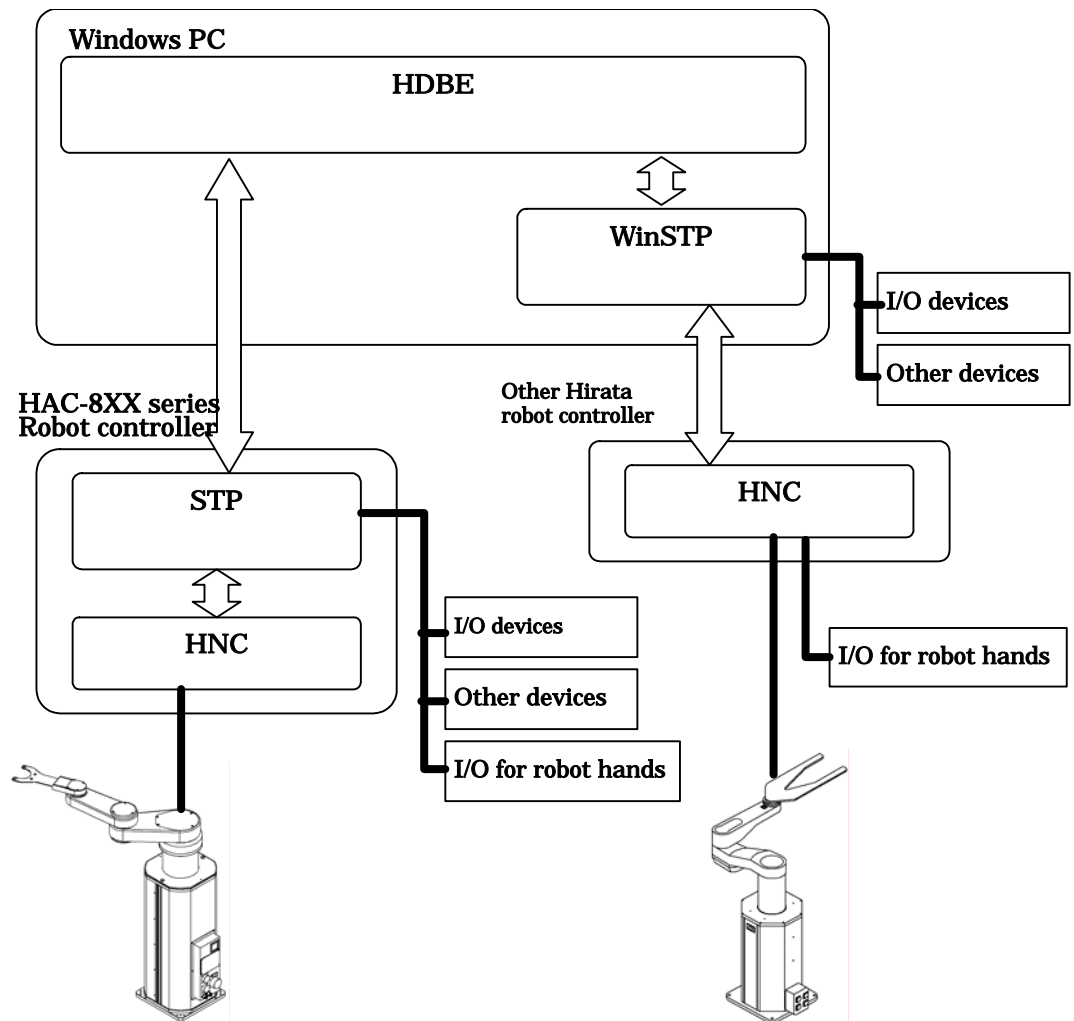
2. Program Developing/Running Environment

2.1 System Structure

The following figure shows the system structure of HrBasic developing or running environment.

HrBasic can be executed on the following platforms

- STP in HAC-8XX
- WinSTP



(1) HBDE --- HrBasic Developing Environment

HBDE provides the environment to develop HrBasic program and to maintenance the robot system and the program. HBDE works on the Windows PC.

The following functions are available by HBDE.

- Project management of HrBasic program
- Editing of HrBasic program
- Compiling and linking of HrBasic program
- Downloading or uploading of HrBasic program

- Debugging of HrBasic program
- Maintenance tool of HrBasic program

(2) STP/WinSTP --- Execution engine of HrBasic program

STP (Station Processor) is the environment that executes HrBasic programs.

- HAC-8XX/STP

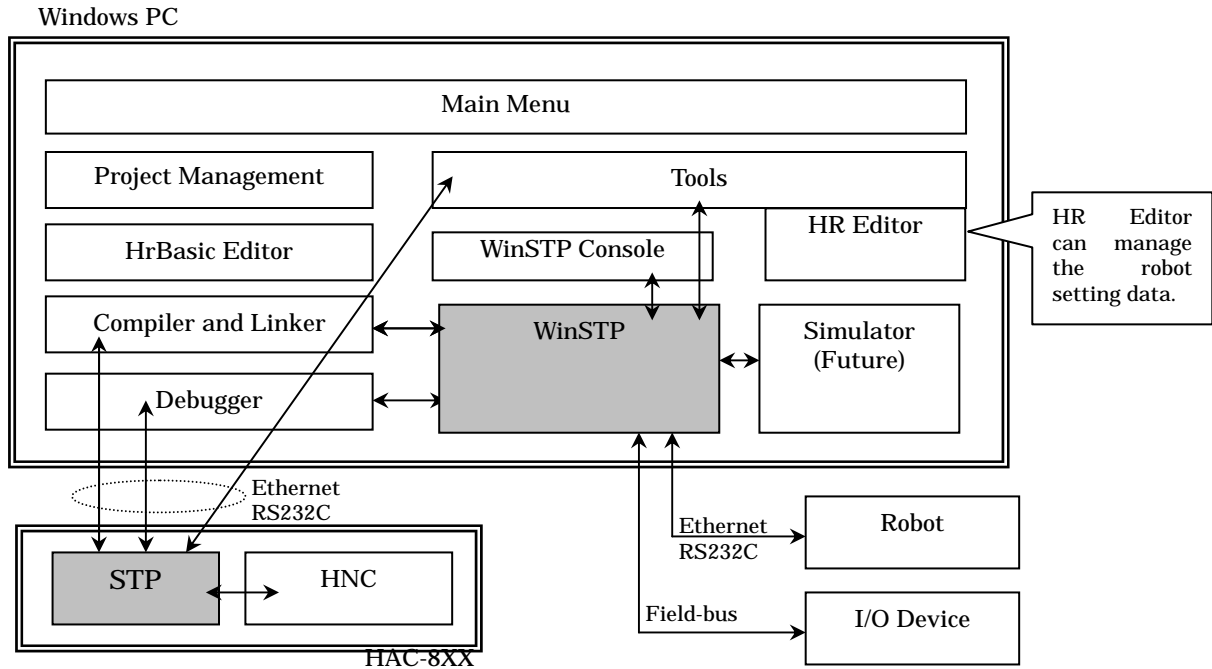
STP is equipped normally in HAC-8XX series.

- WinSTP

WinSTP is the STP for Windows that can execute HrBasic on a Windows PC.

2.2 Software Components of HBDE

The following figure shows the software components and structure of HBDE.



2.3 Specifications

2.3.1 STP Hardware Specifications

Item	HAC-8XX/STP	WinSTP
Microprocessor	Hitachi SH-4 (240MHz)	According to using PC specification Recommended) CPU: above 200MHz Memory: above 64MB HDD: more than 40MB free space OS: Windows95/98/Me/NT4.0/2000/XP
Arithmetic co-processor	Built-in co-processor on SH-4	
Memory	Flash memory : 4MB SDRAM : 64 MB SRAM : 2MB (battery-backup)	
Serial communication	8 ports for standard (PC104 extension(future)) Baud rate : 115200 bps max.	PC COM1-COM9 available Note) Standard PC has only COM1 or COM2. PCI board or USB device of serial COM is needed for extension.
Real-time clock	Built-in calendar and timer on SH-4 Battery-backup	Windows timer and calendar
Precision of timer	1 msec	1 msec
Remote I/O	PC104 extension board (Hilscher GmbH) InterBus Master, Slave PROFIBUS Master, Slave DeviceNet Master, Slave In : 256 bits (max. 4096 bits) Out : 256 bits (max. 4096 bits)	PCI board (Hilscher GmbH) InterBus Master, Slave PROFIBUS Master, Slave DeviceNet Master, Slave In : 256 bits (max. 4096 bits) Out : 256 bits (max. 4096 bits)
Ethernet	10BASE-T * 1	According to using PC specification
Other interfaces	Compact flash card	VB, VC++ application interface MITSUBISHI MELSEC board interface
Monitor on board	7 segments LED	
Size	200 mm * 100 mm * 2 boards (CPU and extension interface board)	

2.3.2 STP Execution Time ⁱ

Item	HAC-8XX/STP (*1)	WinSTP (CPU:533Mhz)
One step execution interval of HrBasic	Average 0.050msec	Average 0.050msec
Max. interruption time by operating system	About 1msec	About 6 to 10msec

(*1) Using real-time operating system "Micro-C OS"

ⁱ Note: The value changes according to the running environment.

2.3.3 HrBasic Specifications

Item	Specification
Job	Max. 32 jobs running simultaneously
Max. program area	1MB (about 45000 to 57000 steps of all jobs)
Max. variable area	1MB
Max. position data memory	8000 points
MD memory (general purpose, battery-backup, byte memory)	1024 bytes
MW memory (general purpose, battery-backup, word memory)	16384 words
ML memory (general purpose, battery-backup, long word memory)	1024 long words
I/O	In : 256 bits (Max. 4096 bits) Out : 256 bits (Max. 4096 bits)
Available user timer	32 timers (Min. scale 1msec)
Available variable type	String, Integer, Long, Single float, Double float

2.3.4 HrBasic Statements and Functions

Kind	Usage	Description	Function
Pre-Processor	Definition	Define	Define the specified name as a constant.
	Macro	Macro	Define a format of macro call.
	Header file	Include	Read the specified header file.
Definable instruction	Definition	Dim	Define as array variable.
		DimNet	Define as network global variable
		Global	Define as global variable.
		DimPos	Define the number of position memory.
		Rem	Define the comment line.
	Flow control	GoTo	Jump to a specified line, then execute.
		GoSub	Call subroutine.
		Return	Terminate subroutine, then resume the former process.
		For - Next	Repeat the instruction between For and Next.
		If Then Else	Decide the condition of logical expression.
		Delay	Break temporarily the execution of job.
		Wait	Wait until conditions are satisfied.
		TimeOut	Get the result of timeout by Wait command.
		On GoTo	Jump one of specified step.
		On GoSub	Call one of specified subroutines.
		Select Case	Evaluate an expression and execute the processing block.
		InitGoSub	Initialize the subroutine-call stack.
Interrupt control instruction	Error control	On Error GoTo	Specify the destination at error.
		Resume	Terminate error process, then resume the former process.
		Err	Hold error code.
Control instruction	Job control	Job Start	Control job execution.
		Job On	
		Job Off	
	Robot control	GetPriority	Get the running priority of the current job.
		SetPriority	Set the running priority of the current job.
		Move	Move a robot to specified coordinates.
		Set	Set operating characteristic data of a robot.
		Ref	Deal data inside of a robot.
		Seq - SeqEnd	Set or terminate robot sequence mode.
		Finish	Complete MOVE in sequence mode.
		Hold	Specify or cancel the servo lock of the robot.
		Disable	Inhibit robot movement.
		Enable	Allow robot movement

Kind	Usage	Description	Function
		Calib	Execute automatic origin calibration.
		SetRobNo	Set a robot number for the current job.
		ClearRobNo	Clear the robot number for the current job.
		GetRobNo	Get the robot number for the current job.
		EnableOnlineErr	Enable robot ONLINE mode check.
		DisableOnlineErr	Disable robot ONLINE mode check.
		RobCheckBpZone	Check robot position within BP/ZONE.
		RobCheckCurPos	Check robot position nearby teaching data.
		RobCheckStop	Check robot stopped.
		RobClearErr	Clear robot errors.
		RobSetPosRange	Define allowable margin of position.
		Inching	Execute inching motion.
		AxesPara	Make axes parameter.
		PosRec	Make one robot position record.
		CollisionCheck	Enable or disable collision check between robots.
		RobWorldPos	Get current robot position in the world coordinates system.
		RobDistance	Get the distance between two robots.
		RobGetCurSpeed	Get the current robot speed.
		RobGetCurTorq	Get the current robot torque.
		RobGetCurAveTorq	Get the current effective torque of a robot.
		RobGetCurPos	Get the current encoder position of a robot.
		RobReadSvoPara	Read servo parameter of a robot.
		RobWriteSvoPara	Write servo parameter of a robot.
		RobReadSG	Read system generation data of a robot.
		RobWriteSG	Write system generation data of a robot.
	File control	Open	Open a communication file.
		Close	Close a file.
		Input\$	Read the specified length of the string from a file.
		Input #	Substitute data of a sequential file to a variable.
		Line Input #	Read one line from a sequential file.
		Print #	Output data to a file.
		Eof	Examine the termination code of a file.
		FreeFile	Get unused file number.
		RchkHrcs	Check a HRCS protocol frame received.
		ReadHrcs	Read a HRCS protocol frame.
		WriteHrcs	Write a HRCS protocol frame.
		EnableDSRCheck	Enable DSR signal check of RS232C.
		DisableDSRCheck	Disable DSR signal check of RS232C.
		EnableRTSAuto	Enable automatic RTS signal control of RS232C.
		DisableRTSAuto	Disable automatic RTS signal control of RS232C.
		ComFunction	Control RS232C signal.
		GetComStatus	Get signal status of RS232C.
	Pulse generation	Pulse	Generate pulse. (Substitute a value for the specified period.)
	Clock control	Time\$	Get or set the current system time.
		Date\$	Get or set the current system date.
Network instruction	Network communication	NetOpen	Open a network communication.
		NetClose	Close a network communication.
		NetRead	Read data from a network communication.
		NetWrite	Write data from a network communication.
Conversion instruction	Arithmetic function	Sin	Get sine.
		Cos	Get cosine.
		Tan	Get tangent.
		Atn	Get arctangent.
		Sgn	Get the sign of value.
		Abs	Get absolute value.
		Int	Remove decimals
		Fix	Remove decimals
		Log	Get natural logarithms.

Kind	Usage	Description	Function
	Operator	Exp	Get e raised to a power.
		Sqr	Get square root.
		Mod	Execute arithmetic division and get the remainder.
		Not	Execute negation.
		And	Execute logical multiplication.
		Or	Execute logical addition.
		Xor	Execute exclusive logical addition.
		Eqv	Execute logical equivalence.
		Imp	Execute logical implication.
	Arithmetic Constant	Pai	Get the value of pi.
	Character	Left\$	Pick out arbitrary length from the left of a string.
		Mid\$	Specify one part of a string.
		Right\$	Pick out arbitrary length from the right of a string.
		Space\$	Get a string with the arbitrary length blank.
		Chr\$	Get the character of specified character code.
		String\$	Get the character string connected one arbitrary character.
		Hex\$	Get the character string converted decimal into hexadecimal.
		Str\$	Convert numerical value into a string.
		Val	Convert the number of a character string into actual value.
		Asc	Get the character codes of characters.
		Len	Get the length of a string.
		InStr	Get the first position of the string in another string.
		ScanStr	Scan string data according to specified format. And get the value as parameter from string by operator in the format.
		PrintStr	Print string data according to specified format. And put the data string of specified parameter by operator in the format.
Initialization	Operation of position memory in STP	InitPos	Initialize position memory in STP.
Message	Print to STP console	ConsoleMsgOn	Enable to print message to STP console.
		ConsoleMsgOff	Disable to print message to STP console.
		ConsoleMsg	Print specified message to STP console.

3. Program Development Guideline

The development process of the HrBasic program is overviewed below.

- (1) Design the functions of the system.

The output is

- System functional specifications

- (2) Design the interface for the peripheral equipment if necessary.

The output is

- Interface specifications

- (3) Design the program specifications.

Design the following assignments.

- MB/MD/MW/ML memory assignments
- I/O assignments
- Timer assignments
- Robot position assignments

Design the jobⁱ structure of the system. The output is

- Job structure diagram

Note)

The volume, maintainability and quality of a HrBasic program depend on how to design job components by dividing the system functions to HrBasic jobs. The guideline of this is described later.

Design the system state flow if necessary. The output is

- State flow diagram

Design the process flow of jobs. The output is

- Job flow diagram

- (4) Create program header files.

Define the following assignments to header files.

- MB/MD/MW/ML memory assignments
- I/O assignments
- Timer assignments
- Robot position assignments

- (5) Create source program files.

Program the procedure of each job according to the program specifications.

- (6) Debug the program on the target system.

Using HBDE, download the program to the target system and then check that the all functions described in the functional specifications work without a problem.

If the program has a bug, refine the program and check again.

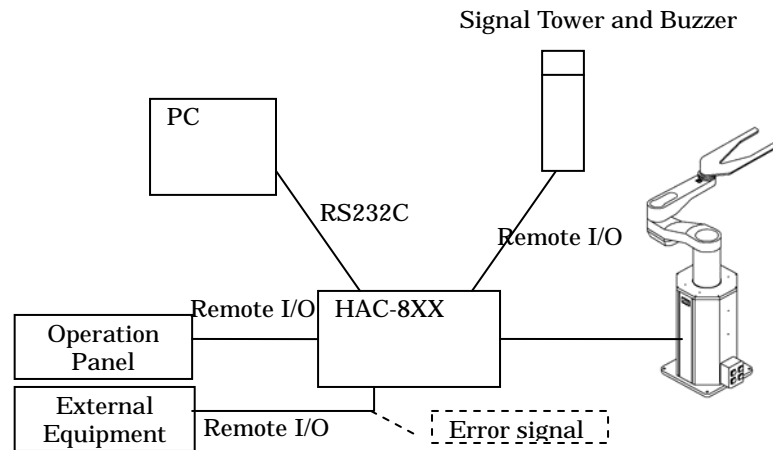
This chapter describes the above-mentioned developing process with a sample system.

ⁱ A job is the HrBasic program component that STP executes concurrently. See Chapter 4 about details.

3.1 Functional Specifications

The functional specifications of the sample system are shown below.

(1) System configuration



- Operation Panel

A Manual button, Ready button and a Start button are implemented.

- PC

PC monitors the running state and statistics of the system. The displaying items are Operation Mode, Total Running Time, Total Stopping Time, Cycle Time and Cycle Counts.

Total Running Time means the sum of the Running mode time. Total Stopping Time means the sum of the time except the Running mode. Cycle Time means that the time during the motion; point A -> point B -> point A. Cycle Counts means the number of the motions; point A -> point B -> point A. Point A and B are described later.

- Signal Tower

A blue lamp, red lamp and a buzzer are implemented.

(2) Operation and motion specifications

- After the power-on, the system starts in Manual mode. Manual mode accepts only a Ready button.
- To press a Ready button, the system is transferred to Ready mode and then a robot moves to the origin position. Ready mode accepts a Start button and a Manual button.
- To press a Start button in Ready mode, the system is transferred to Running mode and then a robot repeats to goes to the point B and back to the point A. The point A is the position to pick a part and the point B is the position to place it, but the sample program omits the motion to pick and place. Running mode accepts only a Manual button.
- In Ready mode or Running mode, to press a Manual button, the system is transferred to Manual mode and a robot stops immediately.

- In Ready mode or Running mode, a error signal becomes high, the system is transferred to Error mode and then a robot stops immediately. Error mode accepts only a Manual button.
- (3) Signal tower and buzzer
- Manual mode
A blue lamp is blinked and other lamp is off.
 - Ready mode
A blue lamp is blinked and other lamp is off.
 - Running mode
A blue lamp is lighted and other lamp is off.
 - Error mode
A red lamp is lighted and other lamp is off. A buzzer makes sound.

3.2 Interface Specifications

The sample system has the following interface specifications for PC.

- PC reads ML memory in HAC/STP and displays the value of the memory.
- See “ML memory assignment” about the items to display.

Note)

The PC application can access ML memory to use our software “Hirata Robot System Interface Library”.

3.3 Program Design Specifications

(1) MB/MD/MW/ML memory assignments

The sample system uses only ML memory.

The ML memory assignments of the sample system are shown below.

ML memory assignment					
No	Name	Explanation	Set	Reset	Note
0					Not used
1	ML.MODE	Operation mode =0: Manual mode =10: Ready mode =11: Running mode =12: Error mode	HAC	HAC	
2					Not used
3	ML.RUN.TIME	Total running time (sec)	HAC	PC	Sum of Running mode time 1sec = 1
4	ML.STOP.TIME	Total stopping time (sec)	HAC	PC	Sum of time except Running mode 1sec = 1
5	ML.CYCLE.TIME	Cycle time (msec)	HAC	HAC	1s = 1000
6	ML.COUNT	Number of cycles	HAC	PC	

“No” --- Index number of ML memory

“Name” --- Name defined in header file

“Explanation” --- Explanation of content and value

“Set” --- Equipment to set value; “HAC” or “PC”

“Reset” --- Equipment to reset or clear value; “HAC” or “PC”

(2) I/O assignments

The I/O assignments of the sample system are shown below.

Input assignments				
Byte No	Bit No	Name	Explanation	Note
0	0			Not used
	1	I.READY	Ready button	
	2	I.START	Start button	
	3	I.MANUAL	Manual button	
	4	I.ERROR	Error	
	5			Not used
	6			Not used
	7			Not used
1	8			Not used
	9			Not used
	10			Not used
	11			Not used
	12			Not used
	13			Not used
	14			Not used
	15			Not used

Output assignments

Byte No	Bit No	Name	Explanation	Note
0	0			Not used
	1	O.BLUE	Signal tower: blue lamp	
	2	O.RED	Signal tower: red lamp	
	3			Not used
	4			Not used
	5	O.BUZZER	Buzzer	
	6			Not used
	7			Not used
1	8			Not used
	9			Not used
	10			Not used
	11			Not used
	12			Not used
	13			Not used
	14			Not used
	15			Not used

“Byte No” --- Index number of I/O area as byte blocks

“Bit No” --- I/O bit number

“Name” --- Name defined in header file

“Explanation” --- Explanation of meaning and value

(3) Timer assignments

The timer assignments of the sample system are shown below.

No	Name	Explanation	Note
0			Not used
1	TIM.STOP	Timer for stopping time	
2			Not used
3	TIM.CYCLE	Timer for cycle time and running time	

“No” --- Index number of TIM variable

“Name” --- Name defined in header file

“Explanation” --- Explanation of meaning and value

(4) Robot position assignments

The robot position assignments of the sample system are shown below.

No	Name	Explanation	Note
0			Not used
1	PM.ORIGIN	Origin position	
2-99			Not used
100	PM.PICK	Point A to pick	
101	PM.PLACE	Point B to place	

“No” --- Index number of PM variable to access teaching position

“Name” --- Name defined in header file

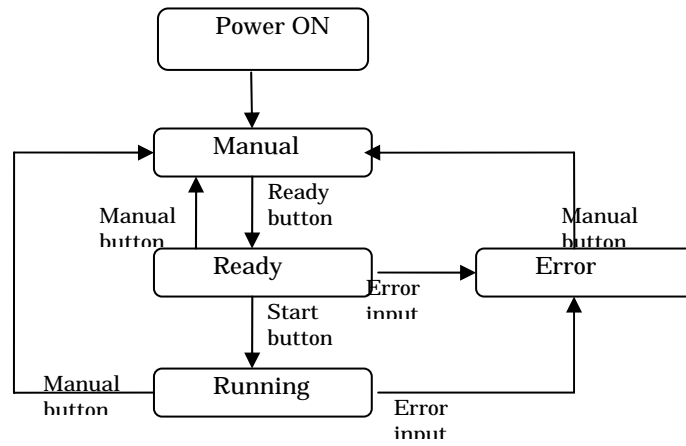
“Explanation” --- Explanation of meaning and value

(5) Job structure

See “3.4 Job Component Structure”.

(6) State flow diagram

The state flow of the sample system is shown below.



3.4 Job Structure

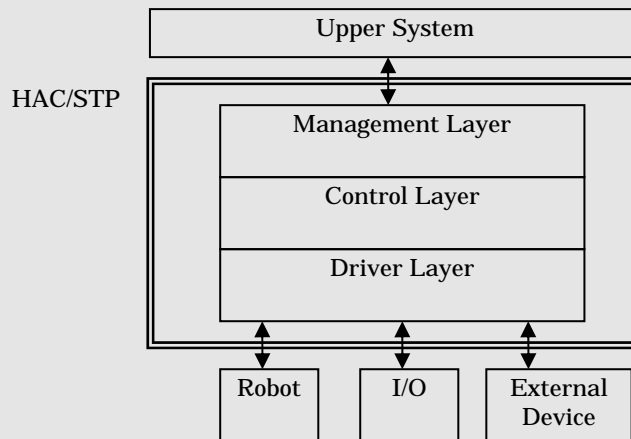
Total 32 jobs are available in a HrBasic program. The job structure is very important because it determines what functions a job executes in the system. And it influences the volume, maintainability and quality of the HrBasic program.

The guideline to decide the job structure is shown below.



< Guideline of Job Structure >

For the purpose of reusing a job program, the job structure has to be hierarchical. The hierarchical structure realizes the software packaging and the combination of the packaged programs can be applied to the various system easily.



Management Layer

This layer job manages the whole motion process of the system. A main job of motion controls instructs an abstract composite operation, such as “Move and pick an object at point A” or “Move and place an object at point B”, to a control layer job. And this layer includes system initialization, mode management, error management, upper system interface, system diagnostics and data management.

Control Layer

Generally, this layer job always waits for an instruction from a management layer job. If an instruction of an abstract composite operation is received, a job resolves the instruction into more primitive operations, such as “Move to point A”, “Grip an object”, “Move to point B” or “Release an object”, and then instructs primitive operations to a driver layer job. When the execution of instructions is finished, this layer job returns a result to a management layer job.

Driver Layer

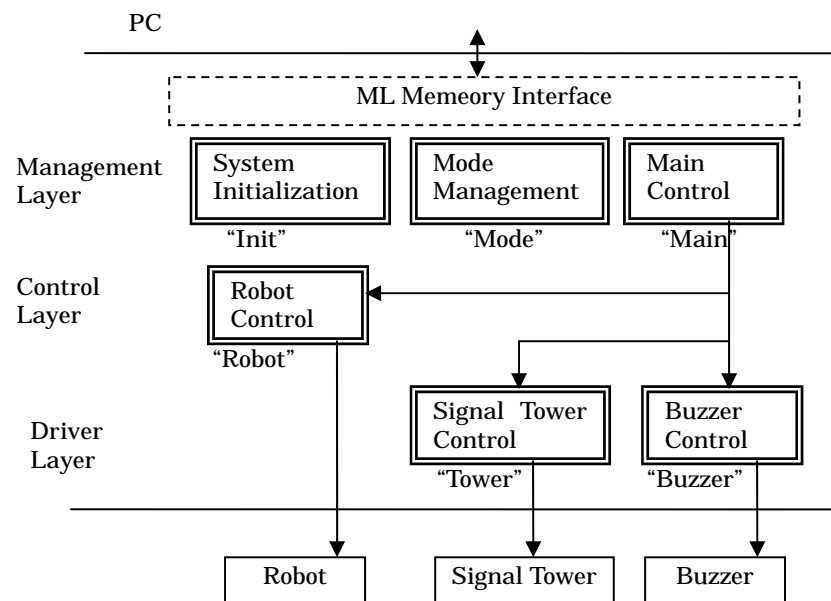
Generally, this layer job always waits for an instruction from a control layer job. If an instruction of a primitive operation is received, a job controls a device with the dependence of hardware. When the control is finished, this layer job returns a result to a control layer job. For example, if an instruction “Move to point A” is received, a job controls a motor driver with the communication, and returns a motion result after the control is finished.

To adopt this structure, for example, in case that device hardware is changed with the same control, the exchange of only the driver layer job applies the new system. Moreover, the stock of packaged programs in control layer and driver layer results in rapid development of various systems by the combination of the packages.

The sample program contains the following six jobs.

- (1) Init --- Management layer
The job initializes the system. During the initialization, the job inhibits other jobs from running.
- (2) Mode --- Management layer
The job watches input signals and change operation mode.
- (3) Main --- Management layer
The job is the main process which controls the whole motion of the system. According to the current operation mode, the job instructs operations to Robot job, Tower job and Buzzer job.
- (4) Robot --- Control layer
As the received instruction, the job controls the robot motion to move an object.
Note)
There is not a job in driver layer for robot control because the driver software is embedded in the operating system of HAC.
- (5) Tower --- Driver layer
As the received instruction, the job controls the lamps on the signal tower.
- (6) Buzzer --- Driver layer
As the received instruction, the job controls the buzzer.

The job structure diagram of the sample system is shown below.



3.5 Header File



Note

< Importance of Using Header File >

To define various constants in a header file with centralization, the only modification of the header file and recompiling the program can change the system easily. This programming method reduces the cost of program modification and prevents the decrement of the program quality after modification. Therefore, it is strongly recommended to define all constants which have a possibility to change in the future or which is coded two times or over in the program.



Guideline for
Programming

< Guideline of Header File Coding >

- Define all constants which have a possibility to change in the future or which is coded two times or over in the program.
- Define all index numbers of MB/MD/MW/ML memory referring to "MB/MD/MW/ML memory assignments". The file name of a header file has to be "MB.hed", "MD.hed", "MW.hed" or "ML.hed" respectively. And the defined name in a header file has to be "MB.XXXX", "MD.XXXX", "MW.XXXX" or "ML.XXXX" respectively.
- Define all index numbers of I/O referring to "I/O assignments". The file name of a header file has to be "IO.hed". And the defined name in a header file has to be "I.XXXX" for input or "O.XXXX" for output.
- Define all index numbers of a TIM variable referring to "Timer assignments". The file name of a header file has to be "TIM.hed". And the defined name in a header file has to be "TIM.XXXX".
- Define all index numbers of a PM variable referring to "Robot position assignments". The file name of a header file has to be "PM.hed". And the defined name in a header file has to be "PM.XXXX".

The sample system header files based on the above mentioned guideline are shown below. There are five header files.

Contents	File Name
ML memory assignments	ML.hed
I/O assignments	IO.hed
Timer assignments	TIM.hed
Robot position assignments	PM.hed
System constants	System.hed

< ML.hed >

```
'=====
'  ML memory assignment
'  Header File
'  ML.hed
'=====
Define ML.MODE           1  'Operation mode
Define ML.RUN.TIME       3  'Total running time (sec)
Define ML.STOP.TIME      4  'Total stopping time (sec)
Define ML.CYCLE.TIME     5  'Cycle time (msec)
```

Define ML.COUNT 6 'Number of cycles

< IO.hed >

```
'=====
'   I/O Assignment
'   Header File
'   IO.hed
'=====
***Input***
Define I.READY      1   'Ready button
Define I.START      2   'Start button
Define I.MANUAL     3   'Manual button
Define I.ERROR      4   'Error
***Output***
Define O.BLUE       1   'Signal tower blue lamp
Define O.RED        2   'Signal tower red lamp
Define O.BUZZER     5   'Buzzer
```

< TIM.hed >

```
'=====
'   Timer assignment
'   Header File
'   TIM.hed
'=====
*** Timer number ***
Define TIM.STOP      1   'For stopping time
Define TIM.CYCLE     3   'For cycle time
*** Timer constants ***
Define TMAX.STOP     1728000   'Maximum stopping time--20 days
Define TMAX.CYCLE    3600      'Maximum cycle time--1 hour
```

< PM.hed >

```
'=====
'   Robot position assignment
'   Header File
'   PM.hed
'=====
Define PM.ORIGIN     1   'Origin
Define PM.PICK       100 'A point (pocking position)
Define PM.PLACE      101 'B point (placing position)
```

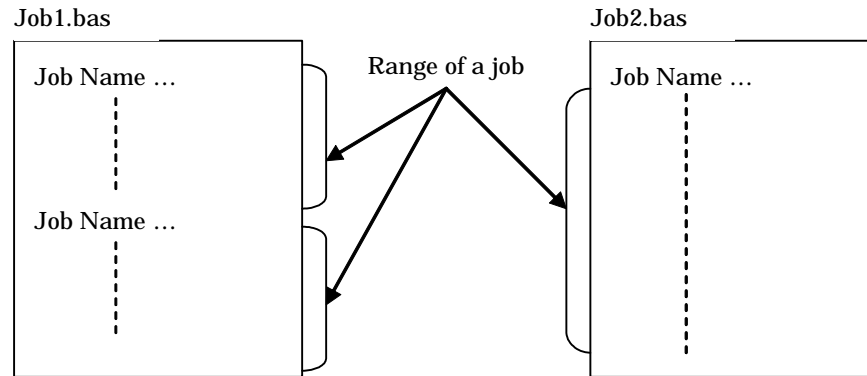
< System.hed >

```
'=====
'   System Constants
'   Header File
'   System.hed
'=====
```

```
*** COM port for robot control ***
Define ROBOT.COM.PARA    "COM0"  'OPEN parameter
Define ROBOT.FNO         1        'File number
Define ROBOT.NO          1        'Robot number
*** Operation mode ***
Define MODE.MANUAL       0        'Manual mode
Define MODE.READY        10      'Ready mode
Define MODE.RUN          11      'Running mode
Define MODE.ERROR        12      'Error mode
*** Signal tower control command ***
Define RED.BLINK         1        'Blink red
Define BLUE.BLINK        2        'Blink blue
Define BLUE.LIGHT        3        'Light blue
*** Buzzer control command ***
Define BUZZER.STOP       0        'Stop buzzer
Define BUZZER.START      1        'Start buzzer
*** Robot control command ***
Define ROBCMD.STOP       1        'Stop robot
Define ROBCMD.ORIGIN     2        'Origin
Define ROBCMD.PICK       3        'Pick
Define ROBCMD.PLACE      4        'Place
*** Delay constants ***
Define BLINK.INTERVAL    1.0      'Blinking time of signal tower (sec)
Define BUZZER.INTERVAL   0.2      'Buzzer control time (sec)
```

3.6 Job Programming

The range of one job program is coded program steps from the “Job Name” statement to the next one or to the end step of a source file. Therefore, both only one job and two or over jobs can be coded in a source file.



**Guideline for
Programming**

< Guideline of number of jobs in a source file >

As described in “3.4 Job Structure”, considering the reusability of a program and the easiness of debugging, only one job has to be coded in one source file.

The sample source files based on the above mentioned guideline are created as one job for one source file as follows.

Job Name	File Name
Init	Init.bas
Mode	Mode.bas
Main	Main.bas
Robot	Robot.bas
Tower	Tower.bas
Buzzer	Buzzer.bas

The explanation of each job is described below.



**Guideline for
Programming**

< Coding Guideline >

A Hrbasic program can be coded in a free style. But for the purpose of the maintainability and quality of a program, it is necessary to adopt the common style to the coding.

We made such coding guideline and it is strongly recommended that you are programming in accordance with the guideline. See “HrBasic Coding Guideline” about details.

3.6.1 Init Job

< Explanation >

The job initializes the ML memory, the global variablesⁱ and the output signals.

MB/MD/MW/ML memory continues to hold the previous value after the power on. A Timer, an I/O variable and a global variable are cleared with zero value after the power on.

If it is necessary to set initial value to MB/MD/MW/ML memory, a timer, an I/O variable and a global variable, it has to be executed in this job.

(1) How to start Init job first in the system

Init job has to start first of all jobs. After power on, jobs starts according to the order which is defined in a make file. So, Init job program has to be registered as the first program in a make fileⁱⁱ.

(2) How to inhibit the execution of other jobs until Init job is completed

STP starts all jobs in order as defined in a make file. During the initialization, the execution of jobs except Init job has to be inhibited. The sample program uses the global variable g.InitEnd% which becomes 1 when the initialization is completed. Other job waits that g.InitEnd% becomes 1 at the top of program.

(3) Job Off when Init terminates.

If a job never runs anymore, it is better that the job terminates by Job Off statement. The terminated job by Job Off never uses the resource of STP.

< Sample Program >

```
'=====
'   System Initialization
'   Init.bas
'=====
Job Name "Init"      'Job name

'*** Header File ***
Include "ML.hed"
Include "IO.hed"
Include "System.hed"

'*** Global Variables ***
Global g.InitEnd%    'Initialization completed

'*** Executable Program ***

g.InitEnd% = 0        'Initialization not completed
ML(ML.MODE) = MODE.MANUAL 'Set initial mode
ML(ML.CYCLE.TIME) = 0  'Clear cycle time

OUTB(O.BLUE) = 0      'Intialize outputs
OUTB(O.RED) = 0
OUTB(O.BUZZER) = 0
```

ⁱ Global variable: See “6.2.3 Local variable and global variable” about details.

ⁱⁱ Make file: Configuration file of compilation. Refer to “HBDE operation manual” or HBDE help about details.

```
g.InitEnd% = 1      'Initialization completed

Job "Init" Off      'Terminate this job
```

3.6.2 Mode Job

< Explanation >

Mode job change operation mode by watching remote inputs of operation buttons.

The job waits for g.InitEnd%=1 as the completion of Init job and runs infinitely until power off.

< Sample Program >

```
'=====
'  Mode Management
'  Mode.bas
'=====

Job Name "Mode"      'Job name

'*** Header File ***
Include "ML.hed"
Include "IO.hed"
Include "System.hed"

'*** Global Variables ***
Global g.InitEnd%    'Initialization completed
Global g.ModeChange% 'Mode changed
Global g.Error%      'Error code

'*** Executable Program ***
'---Job Initialization---
Wait g.InitEnd%=1    'Wait for initialization completed

'---Main Loop---
*LOOP
'Check already mode changed
If g.ModeChange% = 1 Then GoTo *LOOP

'Change mode by operation
Select Case ML(ML.MODE)
Case MODE.ERROR 'Error mode
    GoSub *CHECK.MANUAL 'Check manual button
    If ret% = 1 Then
        g.Error% = 0 'Clear error information
        GoTo *LOOP 'Mode changed
    EndIf
Case MODE.MANUAL 'Manual mode
    GoSub *CHECK.READY 'Check ready button
    If ret% = 1 Then GoTo *LOOP 'Mode changed
Case MODE.READY 'Ready mode
```

```
GoSub *CHECK.ERROR 'Check error input
If ret% = 1 Then GoTo *LOOP 'Mode changed
GoSub *CHECK.MANUAL 'Check manual button
If ret% = 1 Then GoTo *LOOP 'Mode changed
GoSub *CHECK.RUN 'Check start button
If ret% = 1 Then GoTo *LOOP 'Mode changed
Case MODE.RUN      'Running mode
GoSub *CHECK.ERROR 'Check error input
If ret% = 1 Then GoTo *LOOP 'Mode changed
GoSub *CHECK.MANUAL 'Check manual button
If ret% = 1 Then GoTo *LOOP 'Mode changed
Case Else
ML(ML.MODE) = MODE.MANUAL
End Select

GoTo *LOOP
```

'Procedure: CHECK.ERROR

'Summary: Check error input

'Return: [OUT] ret% =1:Mode changed

'Argument: [IN] Nothing

'Caution:

*CHECK.ERROR

ret% = 0 'Clear return value

If INB(I.ERROR) = 1 or g.Error% <> 0 Then 'Error input ON or job error

ML(ML.MODE) = MODE.ERROR 'Change mode

g.ModeChange% = 1

ret% = 1

EndIf

Return

'Procedure: CHECK.MANUAL

'Summary: Check manual button

'Return: [OUT] ret% =1:Mode changed

'Argument: [IN] Nothing

'Caution:

*CHECK.MANUAL

ret% = 0 'Clear return value

If INB(I.MANUAL) = 1 Then 'Manual button ON

ML(ML.MODE) = MODE.MANUAL 'Change mode

g.ModeChange% = 1

ret% = 1

EndIf

Return

'Procedure: CHECK.READY

```
'Summary:   Check ready button
'Return:    [OUT] ret% =1:Mode changed
'Argument:  [IN] Nothing
'Caution:

*****

*CHECK.READY
  ret% = 0 'Clear return value
  If INB(I.READY) = 1 Then 'Ready button ON
    ML(ML.MODE) = MODE.READY 'Change mode
    g.ModeChange% = 1
    ret% = 1
  EndIf
  Return

*****

'Procedure: CHECK.RUN
'Summary:   Check start button
'Return:    [OUT] ret% =1:Mode changed
'Argument:  [IN] Nothing
'Caution:

*****

*CHECK.RUN
  ret% = 0 'Clear return value
  If INB(I.START) = 1 Then 'Start button ON
    ML(ML.MODE) = MODE.RUN 'Change mode
    g.ModeChange% = 1
    ret% = 1
  EndIf
  Return
```

3.6.3 Main Job

< Explanation >

Main job controls the whole motion process of the system after the initialization is completed.

Waiting for the completion of Init job is the same as Mode job.

The job controls lamps of a signal tower and controls a buzzer in Error mode. The control of lamps and a buzzer is executed only once after mode changed. The actual control is executed by the command instructed to Tower job and Buzzer job using a global variable.

In Ready mode or Running mode, similarly, the job controls a robot by the command instructed to Robot job using a global variable. And then the job waits for the completion of a robot control.

In Error mode or Manual mode, the job instructs stopping a robot to Robot job.

And the job calculates the total running time, the total stopping time, the cycle time and the number of cycles, and then sets the values to ML memory for PC.

The job runs infinitely until power off.

< Sample Program >

```
'=====
```

```
' Main Control
' Main.bas
'=====
Job Name "Main"      'Job name

'*** Header File ***
Include "ML.hed"
Include "TIM.hed"
Include "System.hed"

'*** Global Variables ***
Global g.InitEnd%    'Initialization completed
Global g.ModeChange% 'Mode changed
Global g.RobotCmd%   'Robot control command
Global g.TowerCmd%   'Signal tower control command
Global g.BuzzerCmd%  'Buzzer control command

'*** Executable Program ***
'---Job Initialization---
Wait g.InitEnd%=1    'Wait for initialization completed

'---Main Loop---
*LOOP
'Only one time after mode changed
If g.ModeChange% = 1 Then 'Mode changed
  'For each mode
  Select Case ML(ML.MODE)
    Case MODE.ERROR      'Error mode
      g.TowerCmd% = RED.BLINK 'Blink red
      g.BuzzerCmd% = BUZZER.START 'Start buzzer
      g.RobotCmd% = ROBCMD.STOP 'Stop robot
      Wait g.RobotCmd% = 0 'Wait for completion
    Case MODE.MANUAL      'Manual mode
      g.TowerCmd% = BLUE.BLINK 'Blink blue
      g.BuzzerCmd% = BUZZER.STOP 'Stop buzzer
      g.RobotCmd% = ROBCMD.STOP 'Stop robot
      Wait g.RobotCmd% = 0 'Wait for completion
    Case MODE.READY       'Ready mode
      g.TowerCmd% = BLUE.BLINK 'Blink blue
      g.RobotCmd% = ROBCMD.ORIGIN 'Move to origin
      Wait g.RobotCmd% = 0 'Wait for completion
    Case MODE.RUN         'Running mode
      g.TowerCmd% = BLUE.LIGHT 'Light blue
    Case Else
  End Select
  g.ModeChange% = 0 'Not mode changed
  'Measure stopping time
  If ML(ML.MODE) <> MODE.RUN Then
    TIM(TIM.STOP) = TMAX.STOP 'Start timer
  Else
    t.stop! = TMAX.STOP - TIM(TIM.CYCLE) 'Calculate time
```

```
        ML(ML.STOP.TIME) = ML(ML.STOP.TIME) + t.stop! 'Set time
    EndIf
EndIf

'All the time
Select Case ML(ML.MODE)
Case MODE.RUN      'Running mode
    'Robot control
    TIM(TIM.CYCLE) = TMAX.CYCLE 'Start timer for cycle time
    g.RobotCmd% = ROBCMD.PICK 'Request robot to pick
    Wait g.RobotCmd% = 0 'Wait for completion
    g.RobotCmd% = ROBCMD.PLACE 'Request robot to place
    Wait g.RobotCmd% = 0 'Wait for completion
    t.cycle! = TMAX.CYCLE - TIM(TIM.CYCLE) 'Calculate cycle time
    'Time measurement
    ML(ML.CYCLE.TIME) = t.cycle! * 1000 'Set cycle time (msec)
    ML(ML.RUN.TIME) = ML(ML.RUN.TIME) + t.cycle! 'Set total
running time
    ML(ML.COUNT) = ML(ML.COUNT) + 1 'Count up number of cycles
Case Else
End Select

GoTo *LOOP
```

3.6.4 Robot Job

< Explanation >

The job controls a robot motion according to a received command from Main job.
In Error mode or Manual mode, the job never controls a robot for safety.
Waiting for the completion of Init job is the same as Mode job.
The job runs infinitely until power off.

< Sample Program >

```
'=====
'  Robot Control
'  Robot.bas
'=====
Job Name "Robot"      'Job name

*** Header File ***
Include "ML.hed"
Include "PM.hed"
Include "System.hed"

*** Global Variables ***
Global g.InitEnd%      'Initialization completed
Global g.RobotCmd%     'Robot control command
Global g.Error%        'Error code

*** Executable Program ***
```

```
'---Job Initialization---
On Error GoTo *ERR.HANDLER 'Register error handler
Wait g.InitEnd%=1 'Wait for initialization completed
'Open COM port for robot
Open ROBOT.COM.PARA As #ROBOT.FNO RobType=580 RobNoList=1
SetRobNo(ROBOT.NO) 'Set default robot number
Enable #ROBOT.FNO 'Enable robot motion

'---Main Loop---
*LOOP
'Never execute in error or manual mode for safety
Select Case ML(ML.MODE)
Case MODE.ERROR, MODE.MANUAL
    g.RobotCmd% = 0 'Clear command
    GoTo *LOOP
Case Else
End Select

'For each command
Select Case g.RobotCmd%
Case ROBCMD.STOP 'Stop robot
    GoSub *ROB.STOP
Case ROBCMD.ORIGIN 'Move to origin
    GoSub *ROB.ORIGIN
Case ROBCMD.PICK 'Move to picking
    GoSub *ROB.PICK
Case ROBCMD.PLACE 'Move to placing
    GoSub *ROB.PLACE
Case Else
End Select
g.RobotCmd% = 0 'Clear command for response

GoTo *LOOP

'---Error Handler---
*ERR.HANDLER
Disable #ROBOT.FNO 'Disable motion
g.Error% = Err 'Get job error code
Wait ML(ML.MODE)=MODE.ERROR 'Wait for error mode
RobClearErr #ROBOT.FNO 'Clear robot error
Resume *LOOP

'*****

'Procedure: ROB.STOP
'Summary: Stop robot
'Return: [OUT] Nothing
'Argument: [IN] Nothing
'Caution:
'*****

*ROB.STOP
Disable #ROBOT.FNO 'Disable motion
```

Return

'Procedure: ROB.ORIGIN

'Summary: Move to origin position

'Return: [OUT] Nothing

'Argument: [IN] Nothing

'Caution:

*ROB.ORIGIN

Move #ROBOT.FNO, PM(PM.ORIGIN)

Return

'Procedure: ROB.PICK

'Summary: Move to picking position

'Return: [OUT] Nothing

'Argument: [IN] Nothing

'Caution:

*ROB.PICK

Move #ROBOT.FNO, PM(PM.PICK)

Return

'Procedure: ROB.PLACE

'Summary: Move to placing position

'Return: [OUT] Nothing

'Argument: [IN] Nothing

'Caution:

*ROB.PLACE

Move #ROBOT.FNO, PM(PM.PLACE)

Return

3.6.5 Tower Job

< Explanation >

The job controls a signal tower according to a received command from Main job.

Waiting for the completion of Init job is the same as Mode job.

The job runs infinitely until power off.

< Sample Program >

'=====

' Signal tower control

' Tower.bas

'=====

Job Name "Tower" 'Job name

*** Header File ***


```
Include "IO.hed"
Include "System.hed"

'*** Global Variables ***
Global g.InitEnd%    'Initialization completed
Global g.TowerCmd%  'Signal tower control command

'*** Executable Program ***
'---Job Initialization---
Wait g.InitEnd%=1    'Wait for initialization completed

'---Main Loop---
*LOOP
  'For each command
  Select Case g.TowerCmd%
    Case RED.BLINK 'Blink red
      GoSub *BLINK.RED
    Case BLUE.BLINK 'Blink blue
      GoSub *BLINK.BLUE
    Case BLUE.LIGHT 'Light blue
      GoSub *LIGHT.BLUE
    Case Else
      GoSub *ALL.OFF 'All lamp off
  End Select

  GoTo *LOOP

'*****

'Procedure: BLINK.RED
'Summary:   Blink red
'Return:    [OUT] Nothing
'Argument:  [IN] Nothing
'Caution:
'*****

*BLINK.RED
  OUTB(O.BLUE)=0 'Blue lamp off
  Delay BLINK.INTERVAL
  OUTB(O.RED)=1
  Delay BLINK.INTERVAL
  OUTB(O.RED)=0
  Return

'*****

'Procedure: BLINK.BLUE
'Summary:   Blink blue
'Return:    [OUT] Nothing
'Argument:  [IN] Nothing
'Caution:
'*****

*BLINK.BLUE
  OUTB(O.RED)=0 'Red lamp off
```

```
    Delay BLINK.INTERVAL
    OUTB(O.BLUE)=1
    Delay BLINK.INTERVAL
    OUTB(O.BLUE)=0
    Return

*****

'Procedure: LIGHT.BLUE
'Summary:   Light blue
'Return:    [OUT] Nothing
'Argument:  [IN] Nothing
'Caution:

*****

*LIGHT.BLUE
    OUTB(O.RED)=0 'Red lamp off
    OUTB(O.BLUE)=1 'Blue lamp on
    Return

*****

'Procedure: ALL.OFF
'Summary:   All lamp off
'Return:    [OUT] Nothing
'Argument:  [IN] Nothing
'Caution:

*****

*ALL.OFF
    OUTB(O.RED)=0 'Red lamp off
    OUTB(O.BLUE)=0 'Blue lamp off
    Return
```

3.6.6 Buzzer Job

< Explanation >

The job controls a buzzer according to a received command from Main job.
Waiting for the completion of Init job is the same as Mode job.
The job runs infinitely until power off.

< Sample Program >

```
'=====
'   Buzzer control
'   Buzzer.bas
'=====
Job Name "Buzzer"      'Job name

*** Header File ***
Include "IO.hed"
Include "System.hed"

*** Global Variables ***
Global g.InitEnd%      'Initialization completed
Global g.BuzzerCmd%    'Buzzer control command
```

```
*** Executable Program ***
'---Job Initialization---
Wait g.InitEnd%=1 'Wait for initialization completed

'---Main Loop---
*LOOP
'For each command
Select Case g.BuzzerCmd%
Case BUZZER.STOP 'Stop buzzer
    GoSub *STOP.BUZZER
Case BUZZER.START 'Start buzzer
    GoSub *START.BUZZER
Case Else
    GoSub *STOP.BUZZER 'Stop buzzer
End Select

GoTo *LOOP

*****

'Procedure: STOP.BUZZER
'Summary:    Stop buzzer
'Return:     [OUT] Nothing
'Argument:   [IN] Nothing
'Caution:
*****

*STOP.BUZZER
    OUTB(O.BUZZER)=0 'Buzzer OFF
    Return

*****

'Procedure: START.BUZZER
'Summary:    Start buzzer
'Return:     [OUT] Nothing
'Argument:   [IN] Nothing
'Caution:
*****

*START.BUZZER
    OUTB(O.BUZZER)=1
    Delay BUZZER.INTERVAL
    OUTB(O.BUZZER)=0
    Delay BUZZER.INTERVAL
    Return
```

3.7 Debug

After the programming is finished, do the test and the debugging using HBDE and STP. The debugging flow which includes the programming is shown below.

(1) Edit source programs

- Create and edit HrBasic source files to use a text editor or HrBasic Editor.
- Create and edit HrBasic header or macro files to use a text editor or HrBasic Editor if necessary.
- Create and edit a make file to manage programs that will be downloaded to STP.

(2) Compile and link

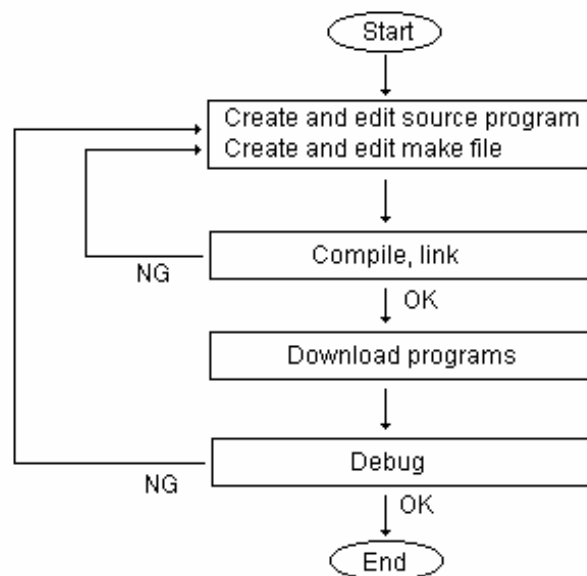
- Compile and link the programs to specify the make file.
- If compiling or linking errors have occurred, modify the programs and compile and link them again.

(3) Download the program

- Download the program without compiling or linking errors to STP.

(4) Debug the program

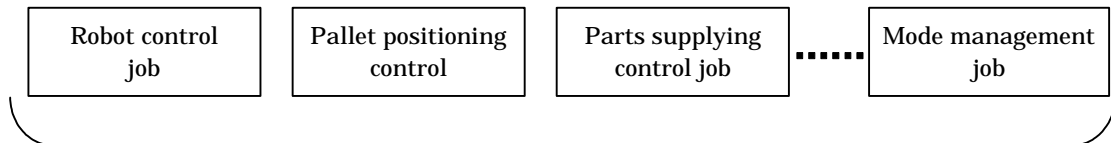
- Check the programs in STP run correctly and debug them.
- If an error occurs or the programs run with the unexpected execution, modify the program and retry 2. 3. 4.



4. Individual Functions

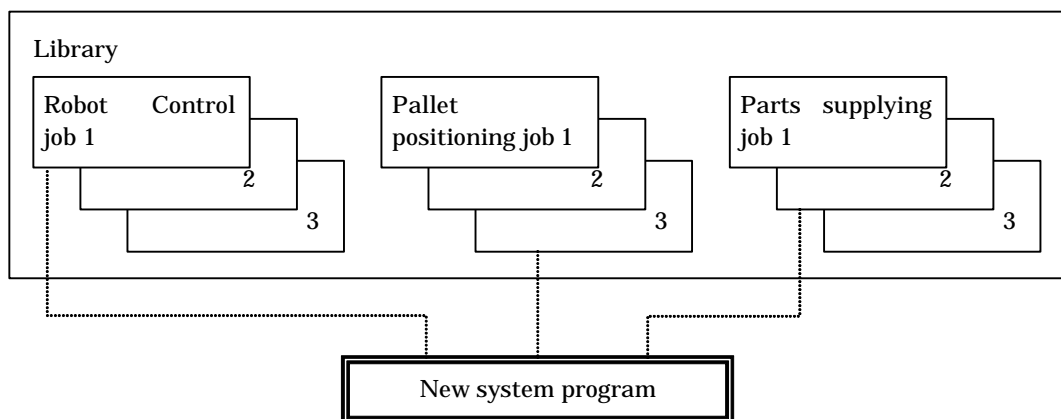
4.1 Job

In HrBasic, as described in chapter 3, the system control programs are divided and coded into jobs from the point of view of operations, functions with the hierarchical structure. Each job is executed concurrently in the multi job method. (Max 32 jobs)



Maximum 32 jobs runs concurrently.

By the multi job method, a program can be structured by dividing the jobs according to the functions and/or devices. Therefore, a program would be more simplified and can be packaged to form a job. You can create libraries of application systems easier.



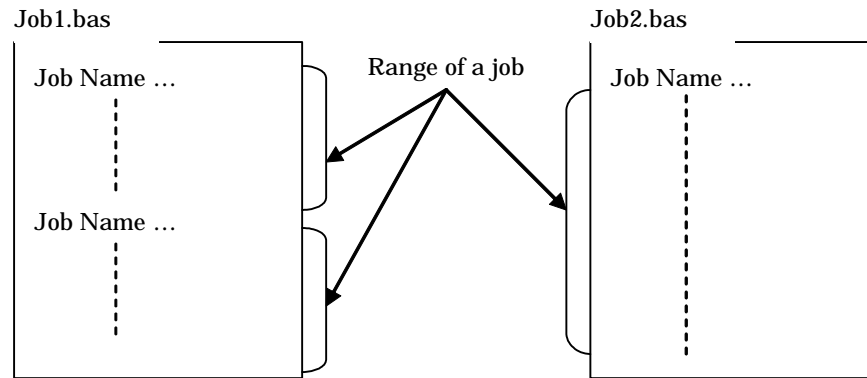
HrBasic has the following features.

- Maximum amount of jobs and steps
 - Job : 32
 - Step : about 45000 to 57000 steps of all jobs in 1 M bytes memory
- Starting jobs

In Hrbasic, all jobs defined in a make file starts automatically in order of the definitions after STP system starts or a program is downloaded.

If the starting of a job has to be controlled by other job, the job has to stop at the first step of the program by Job Off statement.

The range of one job program is coded program steps from the “Job Name” statement to the next one or to the end step of a source file. Therefore, both only one job and two or over jobs can be coded in a source file.



**Guideline for
Programming**

< Guideline of number of jobs in a source file >

As described in Chapter 3, considering the reusability of a program and the easiness of debugging, only one job has to be coded in one source file.

Basically, jobs are independent from each other and a job never influences another job. But, the following functions are common in all jobs or available for a job to control other job.

- Reserved memory See “4.2 Reserved Memory”.
- Timer See “4.3.1 TIM”.
- File and communication See “4.4 File and Communication”.
- Global variable See “6.2.3 Local variable, Global Variable and Network Global Variable”.
- Job Start/On/Off statement ---- See “9.3 Reference”.
- Time\$ and Date\$ ----- See “9.3 Reference”.

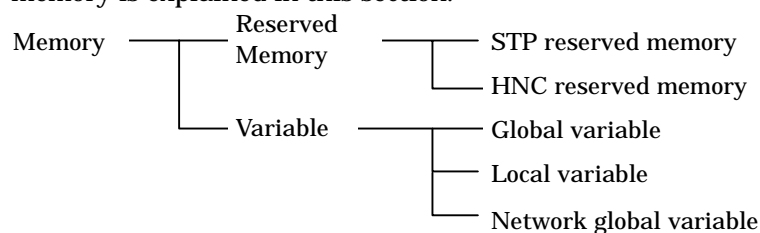


**Guideline for
Programming**

See “3.4 Job Structure” about how to decide job structure.

4.2 Reserved Memory

The memory which HrBasic can access contains the following types. Reserved memory is explained in this section.



Reserved memory has the two kinds, STP reserved memory and HNCⁱ reserved memory. They are explained in detail below.

(1) STP reserved memory

The list of STP reserved memory is shown below.

Memory Type	Format	Index Number	Meaning
Bit memory	MBn	n=0 to 1023	General purpose bit memory
Byte memory	MDn	n=0 to 1023	General purpose byte memory
Word memory	MWn	n=0 to 16383	General purpose word (2 bytes) memory
Long word memory	MLn	n=0 to 1023	General purpose long word (4 bytes) memory
Input bit	INBn	n=0 to 255 Max 4096 bits can be extended.	Remote input bit
Input byte	INDn	n=0 to 31 Max 512 bytes can be extended.	Remote input byte
Output bit	OUTBn	n=0 to 255 Max 4096 bits can be extended.	Remote output bit
Output byte	OUTDn	n=0 to 31 Max 512 bytes can be extended.	Remote output byte
Position memory	Pn	n=0~7999	Position data memory
X axis data	PXn	Using range can be defined by program.	X axis data of Pn
Y axis data	PYn		Y axis data of Pn
Z axis data	PZn		Z axis data of Pn
W axis data	PWn		W axis data of Pn
R axis data	PRn		R axis data of Pn
C axis data	PCn		C axis data of Pn
ARM	PARMn		ARM data of Pn
M data	PDMn		M data of Pn
F code	PDFn		F code of Pn
S code	PDSn		S code of Pn

ⁱ HNC is a component of robot motion control. See “1.1 Hirata Robot System”.



Note

MB/MD/MW/ML memory holds the last value after the power reset. But other memory is cleared by zero.

(2) HNC reserved memory

The list of HNC reserved memory is shown below.

Memory Type	Format	Index Number	Meaning
HNC input bit	IRBn	n=0 to 31	HNC remote input bit
HNC input byte	IRDn	n=0 to 3	HNC remote input byte
HNC output bit	ORBn	n=0 to 31	HNC remote output bit
HNC output byte	ORDn	n=0 to 3	HNC remote output byte
Robot position memory	PMn	n=0 to 999	Robot position data memory Teaching data is held here.
M data	MMn		M data of PMn
F code	FMn		F code of PMn
Robot status	STATUSn	n=0 to 9	Robot status information
Robot current position	HERE		Robot current position data
Robot expanded parameter	EXPARAn	n=0~1099	Robot expanded parameter



Note

- HNC reserved memory holds the last value after the power reset.
- Ref function has to be used to access HNC reserved memory. Ref function is available after Open statement to open the connection for HNC.
- MM and FM can be read but cannot be written.

4.2.1 MB/MD

MB/MD memory is the reserved memory of STP.



Note

MB/MD holds the last value after the power reset.

● MB

This memory is general purpose bit memory accessed by MB0 to MB1023 which can hold the value of zero or one.

● MD

This memory is general purpose byte (8 bits) memory accessed by MD0 to MD1023 which can hold the value of zero to 255 (FFh).

MB memory is the bit assignment memory overlapped by MD0 to MD127.

For example, MD0 includes 8 bits of MB0 to MB7. So, MB0 to MB7 represents each bit of the MD0 value which can be 0 to 255 (FFh). The following table shows the relation of bit number and the exponential value.

Bit	7	6	5	4	3	2	1	0
MD0	MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0
Exponential value	2 ⁷ (128)	2 ⁶ (64)	2 ⁵ (32)	2 ⁴ (16)	2 ³ (8)	2 ² (4)	2 ¹ (2)	2 ⁰ (1)

In case that the value of MD0 is 0, 150 or 255, the bits of MB0-MB7 are shown below.

MB MD0 value	MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0
0	0	0	0	0	0	0	0	0
150	1	0	0	1	0	1	1	0
255	1	1	1	1	1	1	1	1

Exponential value	2^7 (128)	2^6 (64)	2^5 (32)	2^4 (16)	2^3 (8)	2^2 (4)	2^1 (2)	2^0 (1)
-------------------	----------------	---------------	---------------	---------------	--------------	--------------	--------------	--------------

In case that MD0 is 150, the following equation shows how MD0 value is derived from MB0-MB7.

$$\begin{aligned}
 150 &= 2^7 \times 1 + 2^6 \times 0 + 2^5 \times 0 + 2^4 \times 1 + 2^3 \times 0 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 0 \\
 &= 128 \times 1 + 64 \times 0 + 32 \times 0 + 16 \times 1 + 8 \times 0 + 4 \times 1 + 2 \times 1 + 1 \times 0
 \end{aligned}$$

Explained by the sample of MD0, MB0-MB7, the same relation of MD and MB are applied to MD1-MD127 as follows.

MD	MB							
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8
2	23	22	21	20	19	18	17	16
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
126	1015	1014	1013	1012	1011	1010	1009	1008
127	1023	1022	1021	1020	1019	1018	1017	1016
128								
129								
⋮								
⋮								
1023								



Note

- MB(n), MD(n) are equivalent to MBn, MDn.
e.g.) MB(3) MD(5)
- “n” of MBn, MDn can be specified by the indirect expression such as a formula, variable or reserved memory. But in this case, parentheses are necessary for “n”.
e.g.) MB(MD5) --- The MB with the index number of MD5 value.

4.2.2 MW

MW memory is the reserved memory of STP.

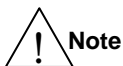


Note

MW holds the last value after the power reset.

MW is general purpose word (2 bytes) memory accessed by the hex value of 0000h to FFFFh.

MW is independent from MB/MD and has no overlapped area.



Note

- MW(n) is equivalent to MWn.
e.g.) MW(3)
- “n” of MWn can be specified by the indirect expression such as a formula, variable or reserved memory. But in this case, parentheses are necessary for “n”.
e.g.) MW(MD5) --- The MW with the index number of MD5 value.

4.2.3 ML

ML memory is the reserved memory of STP.



Note

ML holds the last value after the power reset.

ML is general purpose long word (4 bytes) memory accessed by ML0 to ML1023 which can hold the decimal value of -2147483648 to 2147483647 and the hex value of 00000000h to FFFFFFFFh.

ML is independent from MB/MD/MW and has no overlapped area.



Note

- ML(n) is equivalent to MLn.
e.g.) ML(3)
- “n” of MLn can be specified by the indirect expression such as a formula, variable or reserved memory. But in this case, parentheses are necessary for “n”.
e.g.) ML(MD5) --- The ML with the index number of MD5 value.

4.2.4 INB/IND/OUTB/OUTD

INB/IND/OUTB/OUTD is the reserved memory of STP.

- INB

INB is bit memory accessed by INB0 to INB255 as default, to INB4095 as extension which holds a bit state of remote input.

- IND

IND is byte memory accessed by IND0 to IND31 as default, to IND511 as extension which is overlapped by INB area and holds a byte state of remote input.

- OUTB

OUTB is bit memory accessed by OUTB0 to OUTB255 as default, to OUTB4095 as extension which controls an on/off signal to remote output.

- OUTD

OUTD is byte memory accessed by OUTD0 to OUTD31 as default, to OUTD511 as extension which is overlapped by OUTB area and controls a byte data to remote output.



Note

Extended I/O area (INB256-INB4095, IND32-IND511, OUTB256-OUTB4095, and OUTD32-OUTD511) can be access by a program though hardware is not connected to the area. In this case, input data to read is always zero and output data is never controlled.

INB and OUTB represent the bit expression of IND and OUTD like MB/MD.

For example, IND0 includes 8 bits of INB0 to INB7. So, INB0 to INB7 represents each bit of the IND0 value which can be 0 to 255 (FFh). The following table shows the relation of bit number and the exponential value.

Bit	7	6	5	4	3	2	1	0
IND0	INB7	INB6	INB5	INB4	INB3	INB2	INB1	INB0
OUTD0	OUTB7	OUTB6	OUTB5	OUTB4	OUTB3	OUTB2	OUTB1	OUTB0
Exponential value	2^7 (128)	2^6 (64)	2^5 (32)	2^4 (16)	2^3 (8)	2^2 (4)	2^1 (2)	2^0 (1)

In case that the value of IND0 is 0, 150 or 255, the bits of INB0-INB7 are shown below.

INB IND0 value	INB7	INB6	INB5	INB4	INB3	INB2	INB1	INB0
0	0	0	0	0	0	0	0	0
150	1	0	0	1	0	1	1	0
255	1	1	1	1	1	1	1	1

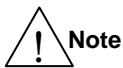
Exponential value 2^7 (128) 2^6 (64) 2^5 (32) 2^4 (16) 2^3 (8) 2^2 (4) 2^1 (2) 2^0 (1)

In case that IND0 is 150, the following equation shows how IND0 value is derived from INB0-INB7.

$$\begin{aligned}
 150 &= 2^7 \times 1 + 2^6 \times 0 + 2^5 \times 0 + 2^4 \times 1 + 2^3 \times 0 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 0 \\
 &= 128 \times 1 + 64 \times 0 + 32 \times 0 + 16 \times 1 + 8 \times 0 + 4 \times 1 + 2 \times 1 + 1 \times 0
 \end{aligned}$$

Explained by the sample of IND0, INB0-INB7, the same relation of IND and INB are applied to all of IND area. And OUTB/OUTD is the same.

IND/ OUTD	INB/ OUTB							
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8
.
.
30	247	246	245	244	243	242	241	240
31	255	254	253	252	251	250	249	248
.
.
510	4087	4086	4085	4084	4083	4082	4081	4080
511	4095	4094	4093	4092	4091	4090	4089	4088



Note

- INB(n), IND(n), OUTB(n), OUTD(n) are equivalent to INBn, INDn, OUTBn, OUTDn.
e.g.) INB(3) OUTD(5)
- “n” of INBn, INDn, OUTBn, OUTDn can be specified by the indirect expression such as a formula, variable or reserved memory. But in this case, parentheses are necessary for “n”.
e.g.) INB(MD5) --- The INB with the index number of MD5 value.

4.2.5 P and Its Structure

P memory is the reserved memory of STP.

P memory is useful for the case that a program calculates a robot position data and accessed by P0 to P7999.

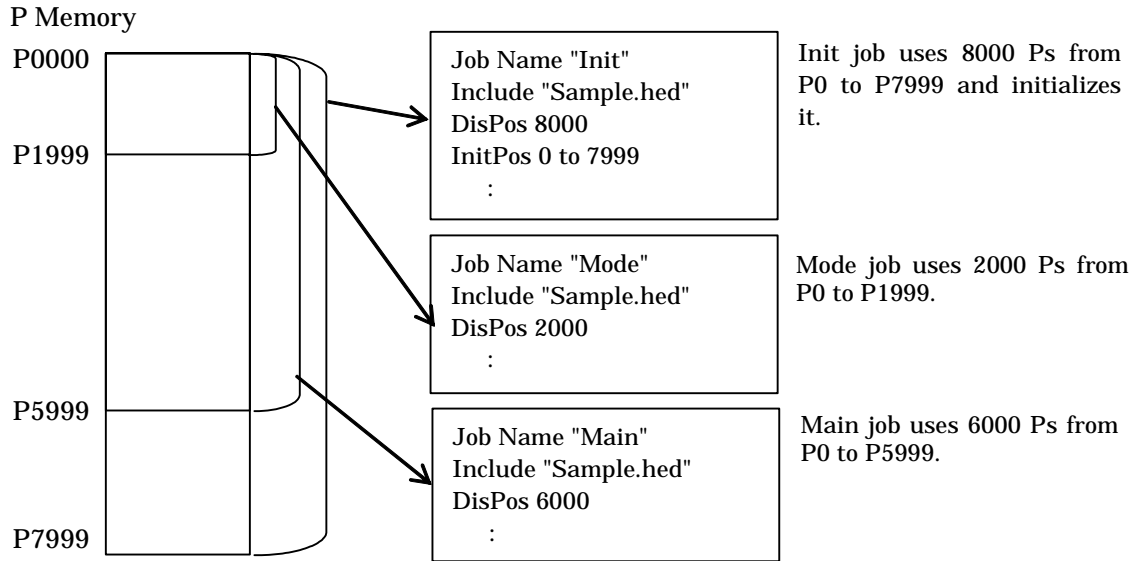
In typical case, a program reads a position data from HNC to P memory and adds some calculation to it and writes it back to HNC.

Before accessing P memory in a job, declaration DimPos is needed to declare how many P memories the program uses.

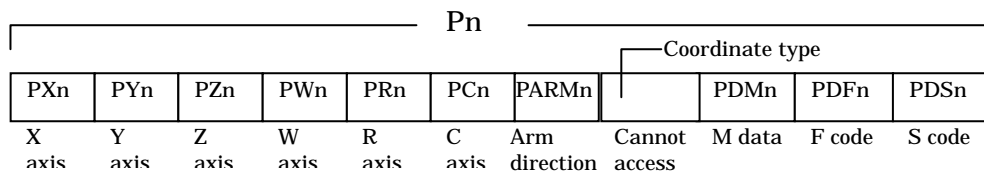
For example, the following declares one hundred P memories to uses in a job.

DimPos 100

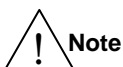
After this declaration, a program can use one hundred Ps from P0 to P99.
 The number of Ps declared by DimPos is effective only in a declared job.
 P memory has to be initialized by InitPos statement before use.



P memory has the following data structure.



Data Item	Format	Data Range	Initial	Note
X axis data	PXn	-2147483.648 to 2147483.647	0.0	Single precision floating value of 4 bytes
Y axis data	PYn	-2147483.648 to 2147483.647	0.0	
Z axis data	PZn	-2147483.648 to 2147483.647	0.0	
W axis data	PWn	-2147483.648 to 2147483.647	0.0	
R axis data	PRn	-2147483.648 to 2147483.647	0.0	
C axis data	PCn	-2147483.648 to 2147483.647	0.0	
Arm direction	PARMn	LEFTY(0) / RIGHTY(1)	0	1 byte area. Only used for SCARA type robot.
M data	PDMn	0 to 99, 255	255	2 bytes area. The value 255 means the end point.
F code	PDFn	0 to 99	0	2 bytes area.
S code	PDSn	0 to 99	0	2 bytes area.
Coordinate type	—	—	0	Cannot access



Note

- P(n), PX(n), PY(n)..... are equivalent to Pn, PXn, PYn.....
e.g.) P(5) PX(3) PY(1)
- "n" of Pn, PXn, PYn..... can be specified by the indirect expression such as a formula, variable or reserved memory. But in this case, parentheses are necessary for "n".
e.g.) P(MD5) --- The P with the index number of MD5 value.

PX(MD3) --- The PX with the index number of MD3 value.

The examples of P memory usage are shown below.

- Read the robot position data (robot #1, address 110) to P10.

P10 = Ref(#fno%[rno:1], PM110)

After this program is executed, value of each item is set as follows.

PX10 X axis data of PM110

PY10 Y axis data of PM110

PZ10 Z axis data of PM110

PW10 W axis data of PM110

PR10.....R axis data of PM110

PC10.....C axis data of PM110

PARM10..... RIGHTY or LEFTY of PM110

PDM10 M data of PM110

PDF10 F code of PM110

PDS10 S code of PM110

A program can treat PX10, PY10... like a variable. A coordinate type in P10 area is also set from PM110 but it cannot be access by a program.

- Read the current position of robot #1 to P0 and change the Z, W axis position, and then move to the position.

P0 = Ref(#fno%[rno:1], HERE) 'Get current position

'M,F,S code cleared

PDM0=1:PDF0=99:PDS=1 '!! Set M,F,S code !!

PZ0 = PZ0 - 10 'Z axis: 10mm up

PW0=200 'W axis: 200 degree

Move #fno%[rno:1], PTP,P0 'Move changed position

Ref(#x, HERE) can get only the current axis position, arm component and dimension code. M,F,S code cannot be got and the its value is set to zero because the robot controller cannot decide M,F,S code for the current position. Therefore, a program has to set valid M,F,S code to Pn memory after REF(#1, HERE) is executed.

- Read the current position of robot #1 to P0 and write it to HNC position memory.

P0 = Ref(#fno%[rno:1], HERE) 'Read current position

'M,F,S code cleared

PDM0=1:PDF0=99:PDS=1 '!! Set M,F,S code !!

'Write P0 to robot position PM100

Ref(#fno%[rno:1], PM100) = P0

Ref(#x, HERE) can get only the current axis position, arm component and dimension code. M,F,S code cannot be got and the its value is set to zero because the robot controller cannot decide M,F,S code for the current position. Therefore, a program has to set valid M,F,S code to Pn memory after REF(#1, HERE) is executed.

If only MM100 is set, program as follows.

Ref(#1[rno:1],MM100)=50 'Set 50 to M data of PM100

- Read the position data of robot #1 and change the position to write it back.

P0 =Ref(#fno%[rno:1], PM0) 'Read address 0 to P0

PX0 = PX0 + 100.0 'X axis +100mm

$PY0 = PY0 - 50.0$ 'Y axis -50mm
 $PDM0 = 80$ 'Set 80 to M data
 $PDF0 = 30$ 'Set 30 to F code
 $Ref(\#fno\%[rno:1], PM0) = P0$ 'Write back

Read the robot position PM0 to P0 in STP and change some value of P0, and the write P0 back to the robot position.



Note

PosRec function is available to set the data to P memory.
 $P0 = PosRec(10, 20, 30, 40, 50, 60, LEFTY, 1, 1, 99, 1)$
 See the PosRec reference about details.

- Read the position data of robot #1 and copy it to another robot position data.

$P1 = Ref(\#fno\%[rno:1], PM100)$ 'Read PM100 to P1
 $P2 = Ref(\#fno\%[rno:1], PM200)$ 'Read PM200 to P2
 $PX2 = PX1$ 'Copy X axis data from P1 to P2
 $PY2 = PY1$ 'Copy Y axis data from P1 to P2
 $PZ2 = PZ1$ 'Copy Z axis data from P1 to P2
 $PW2 = PW1$ 'Copy W axis data from P1 to P2
 $PARM2 = PARM1$ 'Copy arm data from P1 to P2
 $Ref(\#1[rno:1], PM200) = P2$ 'Write P2 to PM200

Read the robot position data PM100, PM200 to P1, P2. Then copy X, Y, Z, W data of P1 to P2 and then write P2 to PM200.

After this, X, Y, Z, W and arm data of PM200 equals to PM100 but M data, F code and S code of PM200 has not been changed.

- Set M data and F code by indirection.

For $i\% = 0$ to 49
 $P(i\%) = Ref(\#fno\%[rno:1], PM(i\%))$
 $PDM(i\%) = 2$
 $PDF(i\%) = i\% + 10$
 $Ref(\#fno\%[rno:1], PM(i\%)) = P(i\%)$
 Next $i\%$

The following procedure is executed from position address 0 to 49.

Read the robot position data to P memory and set 2 to M data and set the address number + 10 to F code. Then write P memory to the robot position data.

4.2.6 IRB/IRD/ORB/ORD

IRB/IRD/ORB/ORD is the reserved memory of robot (HNC).

- IRB

IRB is bit memory accessed by IRB0 to IRB31 which holds a bit state of robot remote input connected with HNC.

- IRD

IRD is byte memory accessed by IRD0 to IRD3 which is overlapped by IRB area and holds a byte state of robot remote input connected with HNC.

- ORB

ORB is bit memory accessed by ORB0 to ORB31 which controls an on/off signal to robot remote output connected with HNC.

- ORD

ORD is byte memory accessed by ORD0 to ORD3 which is overlapped by ORB area and controls a byte data to robot remote output connected with HNC.

IRB and ORB represent the bit expression of IRD and ORD.

For example, IRD0 includes 8 bits of IRB0 to IRB7. So, IRB0 to IRB7 represents each bit of the IRD0 value which can be 0 to 255 (FFh). The following table shows the relation of bit number and the exponential value.

Bit	7	6	5	4	3	2	1	0
IRD0	IRB7	IRB6	IRB5	IRB4	IRB3	IRB2	IRB1	IRB0
ORD0	ORB7	ORB6	ORB5	ORB4	ORB3	ORB2	ORB1	ORB0
Exponential value	2^7 (128)	2^6 (64)	2^5 (32)	2^4 (16)	2^3 (8)	2^2 (4)	2^1 (2)	2^0 (1)

In case that the value of IRD0 is 0, 150 or 255, the bits of IRB0-IRB7 are shown below.

IRB IRD0 value	IRB7	IRB6	IRB5	IRB4	IRB3	IRB2	IRB1	IRB0
0	0	0	0	0	0	0	0	0
150	1	0	0	1	0	1	1	0
255	1	1	1	1	1	1	1	1

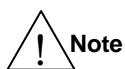
Exponential value	2^7 (128)	2^6 (64)	2^5 (32)	2^4 (16)	2^3 (8)	2^2 (4)	2^1 (2)	2^0 (1)
-------------------	----------------	---------------	---------------	---------------	--------------	--------------	--------------	--------------

In case that IRD0 is 150, the following equation shows how IRD0 value is derived from IRB0-IRB7.

$$\begin{aligned}
 150 &= 2^7 \times 1 + 2^6 \times 0 + 2^5 \times 0 + 2^4 \times 1 + 2^3 \times 0 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 0 \\
 &= 128 \times 1 + 64 \times 0 + 32 \times 0 + 16 \times 1 + 8 \times 0 + 4 \times 1 + 2 \times 1 + 1 \times 0
 \end{aligned}$$

Explained by the sample of IRD0, IRB0-IRB7, the same relation of IRD and IRB are applied to all of IRD area. And ORB/ORD is the same.

IRD/ ORD	IRB/ ORB							
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8
2	23	22	21	20	19	18	17	16
3	31	30	29	28	27	26	25	24



Note

- Ref function has to be used to access IRB/IRD/ORB/ORD. And Ref function has to be already opened to connect with HNC.
Sample)
Open "com1:4800,E,7,1" as #fno%
:
dat% = Ref(#fno%[rno:1], IRD(5))
Ref(#fno%[rno:1], ORD(3)) = 10
- IRB(n), IRD(n), ORB(n), ORD(n) are equivalent to IRBn, IRDn, ORBn, ORDn.

- e.g.) IRB(3) ORD(5)
- “n” of IRBn, IRDn, ORBn, ORDn can be specified by the indirect expression such as a formula, variable or reserved memory. But in this case, parentheses are necessary for “n”.
- e.g.) IRB(MD5) --- The IRB with the index number of MD5 value.

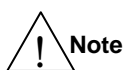
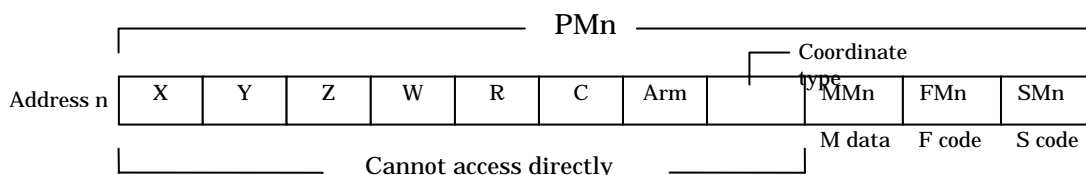
4.2.7 PM/MM/FM/SM

PM/MM/FM/SM is the reserved memory of robot (HNC).

PM represents a block of addressed position data held in HNC. Generally, teaching data using a teaching pendant connected with a robot controller is saved to PM memory.

MM, FM or SM is the element of position data, which represents M data, F code or S code respectively.

Each axis element in PM memory can not be access directly but can be read to P memory in STP.



Note

Ref function has to be used to access PM/MM/FM/SM. And Ref function has to be already opened to connect with HNC.

Sample)

Open “com1:4800,E,7,1” as #fno%

:

P0 = Ref(#fno%[rno:1], PM100)

● PM

Using PM, a robot position data block with a specified address can be accessed. But a program cannot access an axis element in PM memory directly. It is necessary that a program reads PM memory to P memory in STP as follows.

P0 = Ref(#fno%[rno:1], PM100) ‘Read robot #1 PM100 to STP P0

Ref(#fno%[rno:2],PM10) = P20 ‘Write STP P20 to robot #2 PM10

After read position data to STP Pn, a program can access or change an element of position data using the following format. (See “4.2.5 P and Its Structure”.)

X axis data	PXn
Y axis data	PYn
Z axis data	PZn
W axis data	PWn
R axis data	PRn
C axis data	PCn
Arm direction	PARMn
M data	PDMn
F code	PDFn
S code	PDSn
Coordinate type	Cannot access

- MM

MM can directly access M data contained in robot position data at the specified address.

M data is the motion parameter of robot output signal or motion type. The range of available setting is 0 to 99. The value 255 has the special meaning, the end position. Refer to robot operation manual about M data.

MM can be read or written as follows.

MD1 = Ref(#fno%[rno:1], MM1) 'Read robot #1 MM1 to MD1

Ref(#fno%[rno:2], MM200) = 99 'Write 99 to robot #2 MM200

- FM

FM can directly access F code contained in robot position data at the specified address.

F code is the speed parameter of robot. The range of available setting is 0 to 99. Refer to robot operation manual about F code.

FM can be read only as follows.

MD1 = Ref(#fno%[rno:1], FM1) 'Read robot #1 FM1 to MD1

- SM

SM can directly access S code contained in robot position data at the specified address.

S code is the extended parameter of robot motion. The range of available setting is 0 to 99. Refer to robot operation manual about S code.

SM can be read only as follows.

MD1 = Ref(#fno%[rno:1], SM1) 'Read robot #1 SM1 to MD1



Note

- PMn(n), MMn(n), SM(n), FM(n) are equivalent to PMn, MMn, SMn, FMn. e.g.) PM(5) MM(3)
- “n” of PMn, MMn, SMn, FMn can be specified by the indirect expression such as a formula, variable or reserved memory. But in this case, parentheses are necessary for “n”. e.g.) MM(MD5) --- The M code with the index number of MD5 value.

4.2.8 STATUS

STATUS is the reserved memory of robot (HNC).



Note

Ref function has to be used to access STATUS. And Ref function has to be already opened to connect with HNC.

Sample)

Open “com1:4800,E,7,1” as #fno%

:

ecode% = Ref(#fno%[rno:1], STATUS0)

STATUS memory is read-only memory to monitor robot status and it always contains robot motion status and error information.

STATUS memory has the ten items named STATUS0, STATUS1...STATUS9 as follows.

Name	Explanation
STATUS0	Robot error code
STATUS1	X axis error information

Name	Explanation
STATUS2	Y axis error information
STATUS3	Z axis error information
STATUS4	W axis error information
STATUS5	R axis error information
STATUS6	C axis error information
STATUS7	未使用
STATUS8	Robot status #1
STATUS9	Robot status #2

Detail of each STATUS is described below.

(1) STATUS0

STATUS0 holds a current error code shown in the following list.

Error Code		Explanation
Dec	Hex	
0	&H00	No error
9	&H09	Positioning error (See (2) STATUS1-STATUS6.)
16	&H10	Emergency stop
32	&H20	A-CAL not completed (See (2) STATUS1-STATUS6.)
48	&H30	Specified address number is out of range
49	&H31	Move to end point
50	&H32	FAN alarm (Only when FAN ALARM enabled)
64	&H40	Position out of area limit (See (2) STATUS1-STATUS6.)
81	&H51	Overrun error (See (2) STATUS1-STATUS6.)
97	&H61	Communication command error
98	&H62	Command not accepted
99	&H63	System data (SG/SP) destroyed
100	&H64	Cannot read position data from memory card
103	&H67	Servo parameter destroyed
112	&H70	Low voltage of encoder battery
128	&H80	Duplicated command received
130	&H82	Sensor input not ON (Only when SENSOR STOP enabled)
132	&H84	Measurement result out of range (Only when GLASS ALIGNMENT enabled)
144	&H90	Move before A-CAL completion
149	&H95	Coordinate conversion error or invalid position data
160	&HA0	Servo driver error (See (2) STATUS1-STATUS6.)
172	&HB0	Cannot servo-lock
192~207	&HC0~&HCF	HARL-U2 program error (Only when HARL-U2 enabled)
208~220	&HD0~&HDC	Alignment error (Only when ORIFLA enabled)
224	&HE0	Axis interlocked (Only when AXIS INTERLOCK enabled)

(2) STATUS1 - STATUS6

STATUS1 to STATUS6 hold error information of X axis, Y axis, Z axis, W axis, R axis and C axis respectively. If the following error code is set to STATUS0, the axis error information is set at the same time.

- Positioning error : Error code 9 (&H09)
The error existence of the axis is set as the following value.
0 No error
1 Error
- A-CAL error : Error code 32 (&H20)
The error information of the axis is set as the following value.

- 0 An origin sensor does not become ON while the axis is moving in the direction of origin.
 - 1 An origin sensor does not become OFF or the axis cannot move back to the working area.
 - 2 A limit sensor becomes ON while the axis is moving in the direction of origin.
 - 4 Counter is under the regulation when reset.
 - 5 Counter is over the regulation when reset.
 - 7 Other error (Not occurred generally)
- Position out of area limit : Error code 64 (&H40)
The error existence and information of the axis is set as the following value.
 - 0 No error
 - 1 Error on the side of lower limit
 - 2 Error on the side of upper limit
 - Overrun error : Error code 81 (&H51)
The error existence and information of the axis is set as the following value.
 - 0 No error
 - 1 Error on the side of origin
 - 2 Error on the side of overrun
 - 3 Error on the both side
 - Servo driver error : Error code 160 (&HA0)
The error existence of the axis is set as the following value.
 - 0 No error
 - 1 Error

(3) STATUS8

In STATUS8, the following bits are assigned as robot status.

7	6	5	4	3	2	1	0	Bit number
&H80	&H40	&H20	&H10	&H8	&H4	&H2	&H1	Value when bit ON
								1: ON-LINE mode
								1: MANUAL mode
								1: AUTO mode
								0: Not used
								1: SEQ mode
								1: STOP signal ON / 0: STOP signal OFF
								1: ES (Emergency Stop) / 0: Not ES
								0: Not used

- MANUAL mode is one of KEY-IN, TEACH and CHECK mode after the auto/manual switch on the controller is selected to MANUAL.
- See SEQ-SEQEND statement in Chapter 9 about SEQ mode.
- STOP signal represents DI(IN5) signal using in AUTO mode.

(4) STATUS9

In STATUS9, the following bits are assigned as robot status.

7	6	5	4	3	2	1	0	Bit number
&H80	&H40	&H20	&H10	&H8	&H4	&H2	&H1	Value when bit ON

- The bit of Z axis zone becomes “1” when the z axis is under the following condition.

The value of the z axis is less than the value defined in [RESPONSE]-[RESPONSE]-[SAFE.ZONE] of System Parameter of a robot.

- The bit of command execution becomes “1” while a robot is executing the command received by the communication.



Note

- STATUS(n) are equivalent to STATUSn.
e.g.) STATUS(8)
- “n” of STATUSn can be specified by the indirect expression such as a formula, variable or reserved memory. But in this case, parentheses are necessary for “n”.
e.g.) STATUS(MD5) --- STATUS with the index number of MD5 value.

4.2.9 HERE

HERE is the reserved memory of robot (HNC).



Note

Ref function has to be used to access HERE. And Ref function has to be already opened to connect with HNC.

Sample)

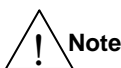
Open “com1:4800,E,7,1” as #fno%

:

P0 = Ref(#fno%[rno:1],HERE)

HERE is the read-only memory which holds the current position of a robot. The each axis data of the current position cannot be accessed directly. Generally, a program reads HERE to P memory in STP by Ref function and then accesses it.

P0 = Ref(#fno%[rno:1], HERE) ‘Read current robot #1 position to P0



Note

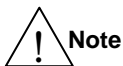
Position data read by HERE does not contain M data, F code and S code. After the above sample program is executed, M data, F code and S code of P0 remains the last value.

After read position data to STP Pn, a program can access or change an element of position data using the following format. (See “4.2.5 P and Its Structure”.)

X axis data	PXn
Y axis data	PYn
Z axis data	PZn
W axis data	PWn
R axis data	PRn
C axis data	PCn
Arm direction	PARMn
M data	Not read by HERE Last data before read
F code	Not read by HERE Last data before read
S code	Not read by HERE Last data before read
Coordinate type	Cannot access

4.2.10 EXPARA

EXPARA is the reserved memory of robot (HNC).



Note

Ref function has to be used to access HERE. And Ref function has to be already opened to connect with HNC.

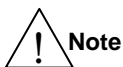
Sample)

```
Open "com1:4800,E,7,1" as #fno%
:
para% = Ref( #fno%[rno:1],EXPARA10 )
```

Using EXPARA, a program can read or write an expanded parameter of a robot. Expanded parameter is the extension of robot system data, by which special motion and control is enabled.

Five hundred items for integer value, five hundred items for real value and one hundred items for common parameter are prepared per a robot.

Refer to robot operation manual about details.



Note

- EXPARA(n) are equivalent to EXPARAn.
e.g.) STATUS(8)
- “n” of EXPARAn can be specified by the indirect expression such as a formula, variable or reserved memory. But in this case, parentheses are necessary for “n”.
e.g.) EXPARA(MD5) --- EXPARA with the index number of MD5 value.

4.3 Timer

HrBasic supports the following four types of timer.

(1) TIM

TIM is general purpose timer.

(2) Delay statement

This stops job execution for a time.

(3) Calendar in STP

A program can reads or write current time and date by Time\$ and Date\$ statement.

(4) Wait statement

Wait waits that the specified condition is true or the specified time passes. And TimeOut function checks the time has passed or not.

See “9. HrBasic Command Reference” about (2), (3), (4).

4.3.1 TIM

TIM is timer which can count down time with the range of 0.000 to 2147483.647 sec by the precision of 1 msec accessed by TIM0 to TIM31.

TIM is independent from jobs. So, any job can use a TIMn with an arbitrary number. Even if a job terminates by Job Off, the TIMn continues to count.

How to start a timer is to set the time as follows.

TIM0 = 10.5

‘TIM0 starts to count down 10.5 sec.

The value of TIMn decreases to zero at 1 msec intervals. When the value becomes zero as the time is up, the value of TIMn is immediately changed to -1 which means TRUE. Therefore, the following program can check TIMn is up.

```
If TIMn Then
    'Time-up
Else
    'Not time-up
EndIf
```



Note

When the time is up, the value of TIM is changed to -1 without delay.

So, the example (1) runs correctly, but (2) incorrectly.

```
(1) If TIM5 Then GoTo *TIMEUP    'Correctly
    If TIM5 = -1 Then GoTo * TIMEUP    'Correctly
(2) If TIM5 = 0 Then GoTo * TIMEUP    'Incorrectly
```

TIM can restart by setting a new value to a timer even if the time has not been up. To set a new value can be executed both by own job or other job.

MD(3)=0

‘Set zero to MD3.

TIM1=20

‘Start TIM1 by 20 sec.

Wait TIM1=-1 or MD(3)=1

‘Wait for TIM1 time-up or MD3 equals 1.

If MD(3)=1 then TIM1=30

‘If MD(3) equals 1, restart TIM1 by 30 sec.

**Note**

- TIM(n) are equivalent to TIMn.
e.g.) TIM(3)
- “n” of TIMn can be specified by the indirect expression such as a formula, variable or reserved memory. But in this case, parentheses are necessary for “n”.
e.g.) TIM(MD5)=60.00
--- Set 60.0s to TIM with the index number of MD5 value.

4.4 File and Communication

Generally, “File” is the logical unit which contains data, typically located in a hard disk, CD or memory card. And a program can read or write it.

In addition to such “File”, in HrBasic, a peripheral device connected with STP can be treated as “File” and a program receives or sends data as if a program reads or writes “File”.

Files which HrBasic can operate are shown bellow.

- (1) Data file in a hard disk, floppy disk and CD.



Note

These files are supported only by Windows STP (WinSTP).

- (2) Communication with a peripheral device

HrBasic can access a peripheral device communicated with STP such as robot, motor driver, PC system, barcode reader and so on as the file operation. HrBasic program send or receive data with such device as if it reads or writes data.

4.4.1 How to Access Data File

How to access a data file by HrBasic is described below.

- (1) Open a data file

Open a data file using Open statement. Parameters of Open statement are the followings.

- File name
Specify the file name to open.
- File mode
Specify file mode as one of “Append”, “Binary”, “Input”, “Output” or “Random”.
- Access type
Specify file access type as one of “Read”, “Write” or “Read Write”.
- File number
Specify the number assigned for the opened file. After the file is opened, this number has to be used to access the file. Available number is 0 to 47.



Note

The same file number is not available at the same time for the file access even if the files are different and even if different jobs access the file.

- (2) Read and write the data file

Read and write the data file using the file number specified to Open statement.

For a data file access, the following commands are available.

- Print statement
- Input\$ function
- Input statement

- Line Input statement
- (3) Close the data file
Close the data file using the file number.

**Note**

If the file number is omitted in Close statement, all opened files are closed automatically.

4.4.2 How to Communicate with Peripheral Device

**Note**

See “8. Robot Control by HrBasic” about communication with a robot.

Before a program accesses a data file, a program has to open it specifying the file name. Before a program communicates with a peripheral device, a program has to open it specifying the COM port name connected with the device and specify the parameters of the communication. A COM port is a communication port which has is available in STP and it has the COM port number from 0 to 14. Open statement can open a COM port.

Some HrBasic statement or function supports the following communication protocols.

- Terminal protocol

This protocol is generally used for terminal mode and CR/LF is added to the end of communication data.

TEXT	C _R	L _F
------	----------------	----------------

- CR : Carriage Return (&H0D)
- LF : Line Feed (&H0A)

- HRCS protocol

This protocol has the following communication format.

S _{TX}	TEXT	E _{TX}	L _{RC}
-----------------	------	-----------------	-----------------

- STX : Start of Text (&H02)
- ETX : End of Text (&H03)
- LRC : Longitudinal Redundancy Check

LRCⁱ is a check code for communication data calculated by exclusive OR of bytes in TEXT to ETX.

In case of the protocol except the above, the procedure for the protocol has to be programmed using Print statement and Input\$ function.

How to access a COM port by HrBasic is described below.

(1) Open a COM port

Open a COM port using Open statement. Parameters of Open statement are the followings.

- COM port name
Specify the COM port name to use by the format of “COMn”. “n” is the COM port number.
- Parameters

ⁱ See “Appendix B LRC Calculation”.

Specify the communication parameters such as communication speed, parity, data length, stop bit and so on.

- File number

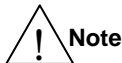
Specify the number assigned for the opened file. After the file is opened, this number has to be used to access the file. Available number is 0 to 47.

- Robot type

Specify a robot controller type for robot communication.

- Robot list

In case of the communication with a robot controller which has four virtual robots, specify a robot number list for robot communication.



Note

- The same COM port is not opened multiply at the same time even if different file number is used.
- The same file number is not available at the same time for the file access even if the files are different and even if different jobs access the file.

(2) Send or receive data to access the COM port

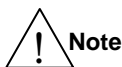
Send and receive data with the COM port using the file number specified to Open statement. For communication, the following commands are available.

- Print statement
- Input\$ function
- Input statement
- Line input statement
- RchkHrcs statement
- WriteHrcs statement
- ReadHrcs statement

In addition, in case of the port connected with a robot, robot control commands are available.

(3) Close the COM port

Close the COM port using the file number.



Note

- If the file number is omitted in Close statement, all opened files are closed automatically.
- RS232C COM numbers of Windows PC are automatically assigned according to hardware implementation. In WinSTP, this assigned COM has to be specified to Open statement.

Each COM of HAC or WinSTP has the different function as shown below.

< HAC COM Port Functions >

COM No.	Interface	Function
COM0	Inner memory interface	Only for robot control. This port is connected with HNC component through a memory interface.
COM1	RS232C	General purpose port which can communicate with an arbitrary external device or HNC. Available speed: 1200 to 115200 bps
COM2	RS232C	
COM3	RS232C	
COM4	RS232C	
COM5	RS232C	
COM6	RS232C	
COM7	---	Not available.
COM8	RS232C	HBDE debugging or general purpose port. Available speed: 1200 to 115200 bps
COM9	RS232C	Only for HBDE debugging. HrBasic cannot access this port. Available speed: 1200 to 115200 bps
COM10	Ethernet	10BASE-T
COM11	---	Not available.
COM12	---	Not available.
COM13	---	Not available.
COM14	---	Not available.

< WinSTP COM Port Functions >

COM No.	Interface	Function
COM0	Inter-process communication	Only for HBDE debugging. HrBasic cannot access this port.
COM1	RS232C	General purpose port which can communicate with an arbitrary external device or HNC. To be available or not depends on the hardware implementation. Available speed: 1200 to 115200 bps
COM2	RS232C	
COM3	RS232C	
COM4	RS232C	
COM5	RS232C	
COM6	RS232C	
COM7	RS232C	
COM8	RS232C	
COM9	RS232C	
COM10	Ethernet	To be available or not depends on the hardware implementation. 10BASE-T.
COM11	Inter-process communication	Windows application port. HrBasic in WinSTP can communicate with other application in Windows.
COM12	Inter-process communication	
COM13	Inter-process communication	
COM14	Inter-process communication	

4.5 Error Handling

In STP, multiple jobs run simultaneously. If an error occurs in a job, the job stops at the error detected step. After an error occurs, a job is not restarted automatically. To restart a job, it is necessary that other job program terminates the stopped job by Job Off statement and then restarts the job by Job Start statement, or that debugging environment can restart the stopped job.

There are three types of “job error”ⁱ as follows.

(1) System error

This error is mainly caused by STP hardware trouble. After this error occurs, the error procedure has to be executed according to the system specification and then check the hardware.

(2) Illegal usage of HrBasic command

This error is caused by the illegal usage of statement, function or variable. And this error has to be removed during the system development.

(3) Communication error and robot error

This error is caused by trouble of communication or robot which may occurs when the system is running. After this error occurs, the error procedure has to be executed according to the system specification.

When these errors occur, without stopping a job, On Error GoTo statement is available for an error procedure. On Error GoTo statement registers the entry of the error handler to jump when a job error occurs. The error handler is the program module which controls the error procedure.

If the error handler has been registered once, a program jumps to the error handler immediately when a job error occurs.



Note

On Error GoTo statement is not a declaration but executable anywhere and anytime in a job. In the following case, the different error handler is used according to the error state.

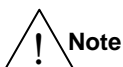
```
If err.flag% = 0 Then
    On Error GoTo *ERR.HANDLER.1
Else
    On Error GoTo *ERR.HANDLER.2
EndIf
```

And the following program deletes the registered error handler.

```
On Error GoTo 0
```

After this step, a program does not jump to the error handler even if a job error occurs.

In an error handler, the current job error code is referred by reserved variable Err and Resume statement terminates the error handler to go back to a normal program.



Note

Reserved variable Err contains code of the job error which has occurred.

ⁱ See “Appendix D Running Job Errors”.



Note

A job error occurs in the error handler, a program stops at the error step not to execute the error handler again.
To recover this state, other job has to terminate the job by Job Off statement and then restart the job by Job Start statement.

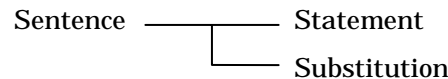
You can see “3. Development Guideline of HrBasic Program” about the sample of error handling

In the sample, Robot job detects a job error and Mode job transfers to error mode.

5. Syntax Rules

5.1 Sentence

Sentence is the smallest unit of program. A sentence consists of a statement or a substitution. One sentence corresponds to one executable program step and STP interprets program steps to execute one by one.



< Statement example >

On Error GoTo *ERR.HANDLER 'On Error GoTo statement
 Move #fno%, PM(100) 'Move statement
 If y% = 0 Then 'If -- Then statement
 For i%=1 To 10 'For statement

< Substitution example >

z.axis! = z.axis! + 10.0!
 a\$ = Mid\$("abcd", 1, 1)

5.2 Line

A line consists of one or more sentences. A line terminates with a carriage-return character or an end-of-file.

Maximum 254 bytes of characters except a carriage-return or end-of-file are able to code in one line.

Note) End-of-file is the code that indicates the end of file.

One line may consist of multiple sentences separated by colon " : ". This line is called "multi-statements", several sentences can be written in one line.

< Line example >

Move #fno%, PM(100)

< Multi-statement example >

x.axis! = 10.0: y.axis! = 20.0

5.3 Statement

A statement contains one of the following formats.

Command Arguments Sub-command

Command (Arguments) Sub-command

It is separated to Command, Argument and Sub-command by Blank space or tab.

Command is the reserved string and it represents the name of embedded procedure.

An argument is separated each other by comma " , " and specified as a constant, variable, expression or label. It may be omitted for some commands.

For some commands, Sub-command may be added to specify the detail parameter.

A statement executes an embedded procedure using the specified parameters.

< Statement example >

Move #fno%, PM(100) 'Move statement

5.4 Function

A function contains the following format.

Command (Arguments)

It is separated to Command, Argument and Sub-command by Blank space or tab.

Command is the reserved string and it represents the name of embedded procedure.

An argument is separated each other by comma “ , ” and specified as a constant, variable, expression or label. It may be omitted for some commands.

A function executes an embedded procedure using the specified parameters and returns the value to a program as the result of procedure. So, a function is the statement which has the returned value.

The returned value is used in a substitution, or as a part of an expression.

< Function example >

a\$ = Mid\$(“abcd”, 1, 1)	'Mid\$ function for substitution
If Eof(fno%) Then	'Eof function for expression
y! = a! + (b! * c! / Sin(x!))	'Sin function for expression

5.5 Comment

Comment is phrases as the notation of a program and it is effective in understanding a program.

A part of one line after Rem statement or an apostrophe “ ’ ” is regarded as comment.

Comment is neglected for compilation and its content is never checked.

< Comment example >

err.flag% = 0 'This is comment in this line.

5.6 Label

HrBasic program does not have a line number to specify the line, but label is available instead of it. A Label is a mark of the line specified in a program.

Generally, a label is used for the following purposes.

- Destination to jump by GoTo statement
- Entry name of subroutineⁱ

In HrBasic, a label has to be written at the top of one line as the following format.

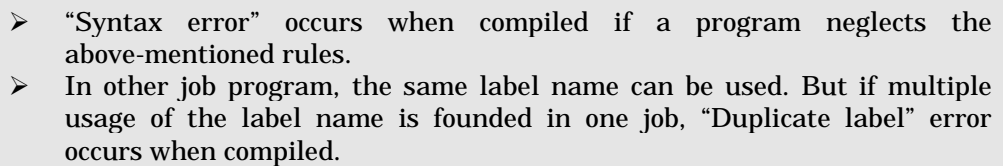
*Label-Name

There are the following rules for using a label.

- The top of label name has to be an asterisk “ * “.
- Except asterisk, the first character of label name has to be alphabetic.
- Except asterisk, available characters in label name are alphabetic, numerical or period “ . “, regardless of upper or lower case.

ⁱ See “7. Structured Programming of HrBasic Language”.

- Example#1) *LOOP i%= 1
Example#2) *LOOP
i%= 1
Example#3) *LOOP i%=1: j%=1
Example#4) *LOOP: i%=1: j%=1



A header file is a text file which is read into a source program when a source program is compiled. Define statements can be written in a header file. It defines the alias of description such as the system constant in a source program. File extension of a header file is “.hed”.

In a job program file, Include statement has to be described as the following format.

Generally, Include statement is written at the top of a job program. In the typical usage, Include statement is written after Job Name statement.

```
Job Name "Test"
Include "Sample.hed"
```

HrBasic compiler replaces the string in a source program with the defined string in a header file.

HrBasic compiler replaces String-A in a source program with String-B.



Sample program is shown below.

- Header file
 'IO.hed

 ' Remote I/O

 Define O.BLUE 1 'Signal tower blue lamp
 Define O.RED 2 'Signal tower red lamp
 Define O.BUZZER 5 'Buzzer on
- Job program
 'Init.bas
 Job Name "Init" 'Job name
 Include "IO.hed" 'Header file

 Global g.InitEnd% 'Global variables
 Global g.Mode%

 g.InitEnd% = 0 'Initialize global
 g.Mode% = 0

 'Initialize output
 OUTB(O.BLUE) = 0 'Replace O.BLUE with 1
 OUTB(O.RED) = 0 'Replace O.RED with 2
 OUTB(O.BUZZER) = 0 'Replace O.BUZZER with 5

 g.InitEnd% = 1 'Initialize completed

 Job Off 'Terminate job

5.9 Character Set

Available character is shown below.

- Alphabet in upper case
- Alphabet in lower case
- Numeral
- Special symbols

HrBasic treats words regardless of both cases except the special cases.

5.10 Special Symbols

HrBasic language uses a symbol for an operator such as arithmetic operator (+, -, *, /, ^), comparing operator (=, <, >).

Moreover there are the following usage of a symbol.

- (1) Colon " : "
 It is used to separate the sentences of multi-statement as a terminator.
 < Example >
 A=B+C : X=A
- (2) Comma " , "
 It is used to separate parameters.
 < Example >
 Move #1, ptp, PM0

(3) Semicolon “ ; ”

It is used to separate parameters of Print statement.

< Example >

Print #1, "A=";A

(4) Apostrophe “ ‘ ”

It is used as the start of comment.

(5) Asterisk “ * ”

It is used as the start of label.

(6) Blank space or tab

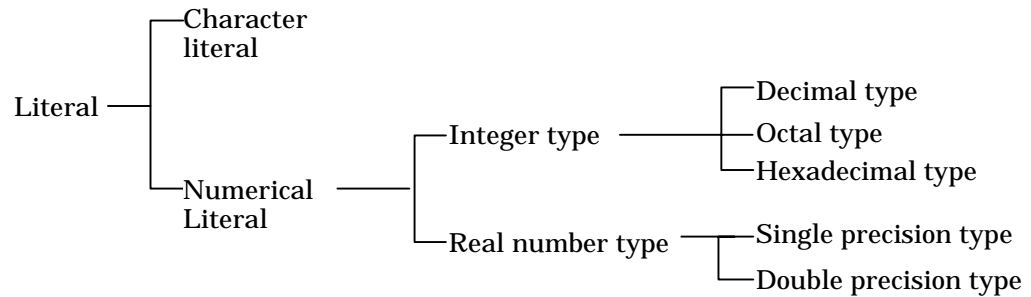
It is used for character constant or separator in a sentence.

6. Elements of Language

6.1 Literal (Constant)

Direct data coded in a program such as 100, 3.14 or "Hollow World" is called "literal" or "constant". In HrBasic, "literal" and "constant" have the same meanings.

HrBasic language can treat the following literals.



6.1.1 Character Literal

Character literal is the string of 255 or less bytes which consists of alphabets, numerals and symbols enclosed by double quotations marks("). The double quotation marks can not be expressed directly in the character literal. If double quotation marks are required in the character literal, express them as shown the example below.

< Example >

- "Good Morning"
- "123456789"
- CHR\$(&H22) + "1234" + CHR\$(&H22)

6.1.2 Numerical Literal

Numerical literals contain integer type and real number type and each type has positive, negative or zero. Minus symbol must be put the top of negative numbers, but plus symbol can be omitted in case of positive numbers.

6.1.3 Integer Type Literal

Integer type literals contain the following expressions. And each expression contains the short integer (16 bits) type and the long integer (32 bits) type.

(1) Decimal type

Decimal type literal begins with numerical character. Decimal point cannot be used. The zero of the top of literal is neglected for the value.

Short integer type literal can expressed the value of -32768 through +32767 with percent mark "%".

Long integer type literal can expressed the value of -2147483648 through +2147483647 with ampersand mark "&".

< Example >

- Short integer type
32767
-123%

32767%

- Long integer type
2147483647
32767&
-123&
2147483647&

(2) Octal type

Octal type literal begins with “&O” or “&” and numbers of 0 through 7 are followed.

If number of 8 or 9 appears in octal type literal, an error occurs when compiled.

The following table shows the range of the value.

- Short integer type value

A percent mark “%” can be added at the end of a literal.

Decimal	Octal	Binary
32767	&O77777	0111 1111 1111 1111
32766	&O77776	0111 1111 1111 1110
:	:	:
2	&O00002	0000 0000 0000 0010
1	&O00001	0000 0000 0000 0001
0	&O00000	0000 0000 0000 0000
-1	&O177777	1111 1111 1111 1111
-2	&O177776	1111 1111 1111 1110
:	:	:
-32767	&O100001	1000 0000 0000 0001
-32768	&O100000	1000 0000 0000 0000

- Long integer type value

An ampersand mark “&” can be added at the end of a literal.

Decimal	Octal	Binary
2147483647	&O17777777777	0111 1111 1111 1111 1111 1111 1111 1111
2147483646	&O17777777776	0111 1111 1111 1111 1111 1111 1111 1110
:	:	:
2	&O10000000002	0000 0000 0000 0000 0000 0000 0000 0010
1	&O10000000001	0000 0000 0000 0000 0000 0000 0000 0001
0	&O10000000000	0000 0000 0000 0000 0000 0000 0000 0000
-1	&O37777777777	1111 1111 1111 1111 1111 1111 1111 1111
-2	&O37777777776	1111 1111 1111 1111 1111 1111 1111 1110
:	:	:
-2147483647	&O30000000001	1000 0000 0000 0000 0000 0000 0000 0001
-2147483648	&O30000000000	1000 0000 0000 0000 0000 0000 0000 0000

< Example >

- Short integer type
&12345
&O7777%
- Long integer type
&O1777777777
&12345&
&1777777777&

(3) Hexadecimal type

Hexadecimal type literal begins with “&H” and numbers of 0 through 9, or hexadecimal numbers of “a” or “A” through “f” or “F” are followed. “a” or “A” to “f” or “F” corresponds with decimal value of 10 to 15.

The following table shows the range of the value.

- Short integer type value

A percent mark “%” can be added at the end of a literal.

Decimal	Octal	Binary
32767	&H7FFF	0111 1111 1111 1111
32766	&H7FFE	0111 1111 1111 1110
:	:	:
2	&H0002	0000 0000 0000 0010
1	&H0001	0000 0000 0000 0001
0	&H0000	0000 0000 0000 0000
-1	&HFFFF	1111 1111 1111 1111
-2	&HFFFE	1111 1111 1111 1110
:	:	:
-32767	&H8001	1000 0000 0000 0001
-32768	&H8000	1000 0000 0000 0000

- Long integer type value

An ampersand mark “&” can be added at the end of a literal.

Decimal	Octal	Binary
2147483647	&H7FFFFFFF	0111 1111 1111 1111 1111 1111 1111 1111
2147483646	&H7FFFFFFE	0111 1111 1111 1111 1111 1111 1111 1110
:	:	:
2	&H00000002	0000 0000 0000 0000 0000 0000 0000 0010
1	&H00000001	0000 0000 0000 0000 0000 0000 0000 0001
0	&H00000000	0000 0000 0000 0000 0000 0000 0000 0000
-1	&HFFFFFFF	1111 1111 1111 1111 1111 1111 1111 1111
-2	&HFFFFFFFE	1111 1111 1111 1111 1111 1111 1111 1110
:	:	:
-2147483647	&H80000001	1000 0000 0000 0000 0000 0000 0000 0001
-2147483648	&H80000000	1000 0000 0000 0000 0000 0000 0000 0000

< Example >

- Short integer type
&H12345
&HFFF%
- Long integer type
&H7FFFFFFFE
&H12345&
&H7FFFE &



Note

The output of Print statement has always the decimal format even if the octal or hexadecimal type literal is specified to Print.

6.1.4 Real Number Type Literal

Real number type literal contain single precision type and double precision type.

(1) Single precision type

The value of single precision type literal has 7 significant figures. Single precision type literal can express the value of -3.402823E+38 through 3.402823E+38.

One of the following conditions decides a literal single precision type.

- The value has the above-mentioned range with 7 or less significant figures.

- An exclamation mark “!” is added at the end of a literal.

< Example >

1.23
-7.09E-06
3525.68
3.14!

(2) Double precision type

The value of double precision type literal has 16 significant figures. Double precision type literal can express the value of -1.7976931348623158E+308 through 1.7976931348623158E+308.

One of the following conditions decides a literal is double precision type.

- The value has the above-mentioned range with 8 or more significant figures.
- A number sign “#” is added at the end of a literal.

< Example >

1234567890
-1.09432E-06+0.3141592653E+01
56789.0#
8657036.1543976

6.2 Variable

A variable is a memory area which keeps the calculated value. A program reads or writes the value in the memory area to specify the variable name that consists of alphabets and numerals.

Assignment from variable name to memory area is automatically done when a program is compiled.

The value of variable is changed by substitution and a program can refer to the value at any time.

When STP starts or a program is downloaded to STP, all variables are cleared by zero.

6.2.1 Variable Name and Type Declaration Character

Variable name has the following rules.

- It has to consist of alphabets, numerals and period “ . ”.
- It has to begin with alphabets.
- It has to be maximum 16 bytes including type declaration character.

The following variables are compiled to two different variables.

A1234560

A1234568

Variable name cannot be the reserved nameⁱ, but a part of variable name can be the reserved name.

It is not cared which case of alphabets variable name has.

In case that the two variable names equal, if type declaration characters differ, the compiler distinguishes the two variables. Type declaration character is added to the end of variable name.

If type declaration character is omitted, the compiler decides that the variable is single precision real-number type as if “ ! ” is added.

Type declaration character	<div style="display: inline-block; vertical-align: middle;"> <div style="display: inline-block; vertical-align: middle; font-size: 4em; line-height: 1;">{</div> <div style="display: inline-block; vertical-align: middle; padding-left: 10px;"> % Integer or 16bits integer type (2bytes) & Long integer or 32bits integer type (4bytes) ! Single precision real-number type (4bytes) # Double precision real-number type (8bytes) \$ String type (max 255bytes) </div> </div>
<div style="display: inline-block; vertical-align: middle;"> <div style="display: inline-block; vertical-align: middle; font-size: 4em; line-height: 1;">}</div> <div style="display: inline-block; vertical-align: middle; padding-left: 10px;"> A A% A& A# A\$ </div> </div>	These differs, but A and A! are same.



Note

To substitute a literal for a variable, if the variable type differs from the literal type, there is the case that an error occurs when compiled.

< OK >

A% = 32767

A% = 222%

A& = 123

< Compiling error >

A% = 32768 'Overflow for variable

A% = 32768% 'Overflow of literal

ⁱ Reserved name is keyword of HrBasic language, such as name of statement (e.g. Mid, If), name of function (e.g. Len, Abs), and operator (e.g. Or, Mod).

A% = 3272& 'Different type
 A% = "cat" 'Different type
 See "6.3 Type conversion" about type conversion in substitution.

6.2.2 Array Variable

In HrBasic, continuous memory areas can be assigned to one variable name. This structured variable is called array variable.

One memory area in array variable is called element. Each element can be indicated by integer number called index.

Array variable has to be declared by Dim statement with specifying the volume of array.

Dim Array-variable-name(max-index-number)

The volume of array can be specified by the maximum number of index with parentheses.

The following example shows the array which consists of seven elements with index numbers, zero through six.

Dim a%(6)

Element	Memory area
a%(0)	2 bytes
a%(1)	2 bytes
a%(2)	2 bytes
a%(3)	2 bytes
a%(4)	2 bytes
a%(5)	2 bytes
A%(6)	2 bytes

Maximum three dimensions of array are available in HrBasic.

The following example shows the structure of two dimensions array.

Dim x&(2,2)

Element	Memory area
x&(0, 0)	4 bytes
x&(0, 1)	4 bytes
x&(0, 2)	4 bytes
x&(1, 0)	4 bytes
x&(1, 1)	4 bytes
x&(1, 2)	4 bytes
x&(2, 0)	4 bytes
x&(2, 1)	4 bytes
x&(2, 2)	4 bytes

The following example shows the structure of three dimensions array.

Dim y#(2,2,2)

Element	Memory area
y#(0, 0, 0)	8 bytes
y#(0, 0, 1)	8 bytes
y#(0, 0, 2)	8 bytes
y#(0, 1, 0)	8 bytes
y#(0, 1, 1)	8 bytes
y#(0, 1, 2)	8 bytes
y#(0, 2, 0)	8 bytes
y#(0, 2, 1)	8 bytes
y#(0, 2, 2)	8 bytes
y#(1, 0, 0)	8 bytes

y#(1, 0, 1)	8 bytes
y#(1, 0, 2)	8 bytes
y#(1, 1, 0)	8 bytes
y#(1, 1, 1)	8 bytes
y#(1, 1, 2)	8 bytes
y#(1, 2, 0)	8 bytes
y#(1, 2, 1)	8 bytes
y#(1, 2, 2)	8 bytes
y#(2, 0, 0)	8 bytes
y#(2, 0, 1)	8 bytes
y#(2, 0, 2)	8 bytes
y#(2, 1, 0)	8 bytes
y#(2, 1, 1)	8 bytes
y#(2, 1, 2)	8 bytes
y#(2, 2, 0)	8 bytes
y#(2, 2, 1)	8 bytes
y#(2, 2, 2)	8 bytes

The following example shows number of elements for each dimension.

< Example >

Dim a%(10)	'One dimension --- 11 elements
Dim aa&(10,50)	'Two dimensions --- 11x51=561 elements
Dim aaa\$(2,5,3)	'Three dimensions --- 3x6x4=72 elements



Note

Memory size of variables in STP is 1 Mbytes. When a program is downloaded, the area that all variables use is checked whether it exceeds.

The following expression can access an array element

Array-variable-name(index-to-access).

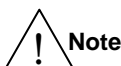
Literal, reserved memory, variable or expression is available for the index to access an array element.

< Example >

If a%(i%+1) = 10 Then

6.2.3 Local Variable, Global Variable and Network Global Variable

In HrBasic, variables are categorized into three types, local variable, global variable and network global variable by scope in which a variable is available. Each type is described below.



Note

Maximum 500 local variables are available per one job.
Maximum 500 global variables are available in all jobs.
Maximum 100 network global variables are available in all jobs.

(1) Local variable

A variable that can be accessed only in a job is called local variable. It is not necessary to declare local variables and the compiler automatically assign the memory area of used variables by analyzing a program.

A local variable in a job is independent of a variable in other job. If there is the local variable which has the same name in other job, there is no

influence with each other. So, a program in a job cannot read or write a local variable in the other job.

In the following example, the usage of portno% is restricted in its own job.

Job Name "Port1" Include "loader.hed" portno%=1 : :	Job Name "Port2" Include "loader.hed" portno%=2 : :
---	---

(2) Global variable

A variable that can be shared by multiple jobs is called global variable. A global variable has to be declared by Global statement.

A global variable in a job shares the memory area with other jobs that declare the same global variable name. The jobs that declare a global variable can read or write it at any time.

If an array variable is used as global, the array does not need to be declared by Dim statement, but has to be declared by Global statement.

Global g.Array%(10, 20, 30)

When a global variable is declared in some jobs, if the same name variable is not declared as global in a job, the variable is treated as local.

In the following example, all jobs access g.Mode%(2) as global. And Port1 and Port2 job accesses Err.no% as global, but Mode job accesses it as local.

Job Name "Port1" Include "loader.hed" Global g.Mode%(2) Global Err.no% portno%=1 : g.Mode%(portno%)=1 :	Job Name "Port2" Include "loader.hed" Global g.Mode%(2) Global Err.no% portno%=2 : g.Mode%(portno%)=1 :	Job Name "Mode" Include "loader.hed" Global g.Mode%(2) : If g.Mode%(2)=3 Then Err.no%=1 EndIf :
--	--	--



**Guideline for
Programming**

For the purpose of increasing the maintainability of program, it is recommended that the same name is not used for a local variable and a global variable. And it is recommended to use the variable name which can clearly distinguish a local variable or a global variable.

(3) Network global variable

STP supports one of the following three fieldbus.

- InterBus
- PROFIBUS
- DeviceNet

Remote I/O of STP is realized by the fieldbus and STP can communicate with each other using the fieldbus network.

Note) This STP communication is not supported for DeviceNet yet.

**Note**

The STP fieldbus communication requires defining the network definition by HBDE.
Refer to HBDE help or “HBDE Operation Manual” about details.

A variable that can be shared by multiple STPs communicated with each other by fieldbus network is called network global variable. A network global variable has to be declared by DimNet statement.

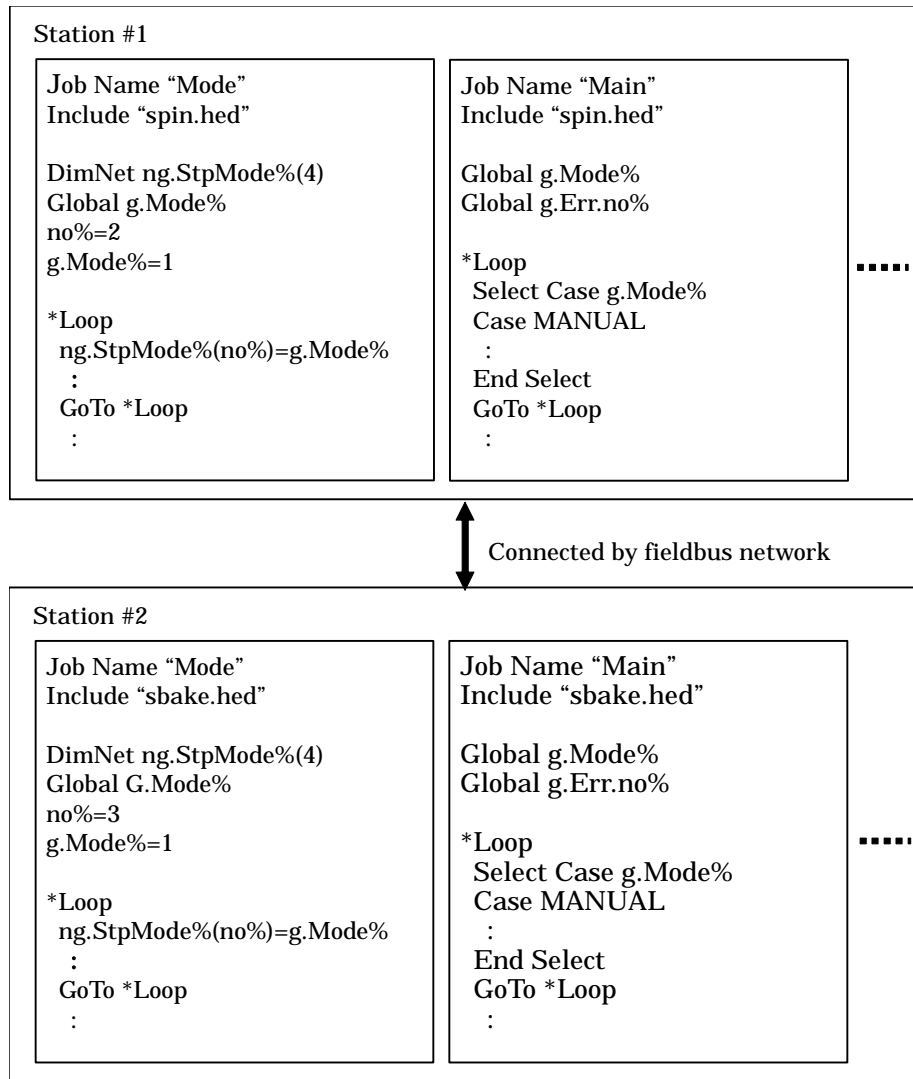
A global variable in a STP job shares the content with other jobs which declares the same name of a network global variable in a different STP. The jobs which declare a network global variable can read or write it at any time.

If an array variable is used as network global, the array does not need to be declared by Dim statement, but has to be declared by DimNet statement.

DimNet ng.Array%(10, 20, 30)

When a network global variable is declared in some jobs, if the same name variable is not declared as network global in a job, the variable is treated as local.

In the following example, STP Station#1 and Station#2 are connected with each other by fieldbus network. Network global variable, ng.StpMode%(4) is declared in Station#1 Mode job and Station#2 Mode job. And these jobs write the value to the network global variable.



**Guideline for
Programming**

For the purpose of increasing the maintainability of program, it is recommended that the same name is not used for a local variable and a network global variable. And it is recommended to use the variable name which can clearly distinguish a local variable or a network global variable.

6.3 Type Conversion

In HrBasic, numerical data can be converted to other type. But the conversion between a string type and a numerical type is not available.

- (1) When the numeric data of some type is substituted for numeric variable of other type, the value is converted to the type declared by its variable name.

Example)

abc% = 1.234 '1 is substituted for abc%

- (2) In case of the operation between different precision, the value is converted to higher precision at operation. 10# /3 is operated as 10# / 3#.

Example)

a# = 10# / 3 '3.33333333333333 is substituted for a#

b# = 10# / 3# '3.33333333333333 is substituted for b#

- (3) In case of logical operation, all numerical values are converted to integers and the results are shown by integers.

Example)

a! = 12.34 '12.34 is substituted for a!

b! = Not a! '-13 is substituted for b!

- (4) In case of conversion from real number to integer, the value under decimal point is rounded to the nearest whole number. In this case, if the value is over the integer type, the error is indicated.

Example)

a% = 34.4 '34 is substituted for a%

b% = 34.5 '35 is substituted for b%

a# = 1.234E+07 '1.234E+07 is substituted for a#

c% = a# 'Overflow error at this step

- (5) When the double precision variable is substituted for the single precision variable, the value is expressed as significant 7 columns. The precision variable is 7digits and the absolute value of the error against the original value is less than 5.96E-8.

Example)

a# = 1.23456789# '1.23456789 is substituted for a#

b! = a# '1.234567 is substituted for b!

When the operation is mixed with the double precision variable (or constant) and the single precision variable (or constant) or the value of the single precision is substituted for the double precision variable, the conversion error happens at the digits after significant columns.

Example)

- a) The operation between different precision (Conversion error happens in the operation result)

Bad example a# = 1.41421356#+0.12

Good example a# = 1.41421356#+0.12#

- b) When lower precision value is substituted for higher precision

Bad example a# = 3.1415

Good example a# = 3.1415#

6.4 Operator

HrBasic has the following types of operator.

- (1) Arithmetic operator
- (2) Relational operator
- (3) Logical operator
- (4) Character string operator

6.4.1 Arithmetic Operator

Arithmetic operator is used for the arithmetic calculation and there are the following operators in HrBasic.

Operator	Operation	Example	Explanation
+	Plus	+A	Same as A
-	Minus	-A	Minus A
^	Exponent	3^4	3 to 4 th power
*	Multiplication	2 * A	2 multiplied by A
/	Division	3 / 5	3 divided by 5
+	Addition	A + 3	A plus 3
-	Subtraction	A - B	A minus B
mod	Remainder	17 mod 5	The remainder of 17 divided by 5. The result is 2.



Note

In case of changing priorityⁱ, use parentheses. Operator enclosed by parentheses is processed earlier than other operation. In parentheses, operation is executed in the sequence.

The programming sample is shown below.

	Arithmetic expression	HrBasic expression
1)	$2X + Y$	$2 * X + Y$
2)	$\frac{X}{Y} + 2$	$X / Y + 2$
3)	$\frac{X + Y}{2}$	$(X + Y) / 2$
4)	$X^2 + 2X + 1$	$X^2 + 2 * X + 1$
5)	X^{Y^2}	$X^(Y^2)$
6)	$(X^Y)^2$	X^Y^2
7)	$Y(-X)$	$Y * -X$

The remarks of operation are described below.

- (1) Division by zero

When the expression is divided by 0, the maximum number processed internally is substituted as quotient and it is processed as error. In case of minus exponentiation to 0, the process is the same as the division by 0.

ⁱ See "6.5.1 Priority of Operations".

Example)

$a\% = 2 / 0$

$b! = 0 \wedge -3$

(2) Overflow

When the result of operation or substitution is over the allowed range, overflow occurs. When the overflow happens, overflow error is output and the maximum number is given as result and it is processed as error.

Example)

$a\% = 32000 + 10000$

$b! = 3 \wedge 1000$

(3) Exponent

Exponent operation can not be calculated by negative real number. (positive number or negative integer is possible.)

Example)

$a! = b! \wedge -1.23$

6.4.2 Relational Operator

Relational operation is to compare two numerals. The result is given by true (-1) or falsehood (0) and used to branch the program flow conditionally. (See If statement or Select statement.)

Relational operators are listed below.

Operator	Operation	Example	Explanation
\leq	Smaller or equal	$A \leq B$	If A is smaller than B or equal, the result is true. If A is larger than B, it is false.
\geq	Larger or equal	$A \geq B$	If A is larger than B or equal, the result is true. If A is smaller than B, it is false.
$<$	Smaller	$A < B$	If A is smaller than B, the result is true. If A is larger than B or equal, it is false.
$>$	Larger	$A > B$	If A is larger than B, the result is true. If A is smaller than B or equal, it is false.
\neq	Not equal	$A \neq B$	If A is not equal to B, the result is true. If A is equal to B, it is false.
$=$	Equal	$A = B$	If A is equal to B, the result is true. If A is not equal to B, it is false.

Relational operators have the two cases of numerical comparison and character string comparison.



Note

- Note that “=” is used for substitution also.
- Comparison between string and numeral is not allowed.

(1) Comparison of numerals

In case of comparison of different precision numerals, the result is calculated by conversion to the most precise type of both sides.

Example#1)

$a\% = 1$

If $a\% < 1.21!$ Then ‘Comparison of 1.00! and 1.21!’

Example#2)

$a! = 1.23!$

$b\# = 2.3345\#$

If $a! < b\#$ Then ‘Comparison of 1.2300# and 2.3345#’

Relation of numerical type and precision is shown below.

Type declaration	Type name	Precision
#	Double precision real	Most precise ↕ Least precise
!	Single precision real	
&	Long integer	
%	Integer	

(2) Comparison of character strings

Character strings of both sides are compared from the top of strings by comparison of each character one by one.

If the lengths of two strings and all characters are same, two strings are equal.

If the lengths of two strings are same but characters of them are not same, the string that contains a bigger character code is bigger than another string.

If the lengths of two strings are not same, the longer string is bigger than another string.



Note

String comparison is executed to compare the value of ASCII code of a character including space or tab.

6.4.3 Logical Operator

Logical operator is used for logical calculation such as logical operation in the condition or bit operation in the expression.

(1) Logical operation in the condition

In the condition, a logical operator returns the result of true (-1) or false (0) to calculate the one or two logical value.

The program flow can be controlled by If statement to check the result of a logical operator.

In the condition, the following operators can be available.

Operator	Operation	Example	Explanation
not	Not (Negation)	not (A=B)	If not (A=B), the result is true. If (A=B), the result is false.
and	And (Logical multiplication)	(A=B) and (C=D)	If (A=B) and (C=D), the result is true. If not (A=B) or not (C=D), the result is false.
or	Inclusive or (Logical addition)	(A=B) or (C=D)	If (A=B) or (C=D), the result is true. If not (A=B) and not (C=D), the result is false.
xor	Exclusive or (Logical exclusion)	(A=B) xor (C=D)	If (A=B) and not (C=D), the result is true. If not (A=B) and (C=D), the result is true. If (A=B) and (C=D), the result is false. If not (A=B) and not (C=D), the result is false.
eqv	Logical equivalence	(A=B) eqv (C=D)	If (A=B) and not (C=D), the result is false. If not (A=B) and (C=D), the result is false. If (A=B) and (C=D), the result is true. If not (A=B) and not (C=D), the result is true.

Operator	Operation	Example	Explanation
imp	Logical implication	(A=B) imp (C=D)	If (A=B) and not (C=D), the result is false. In other case, the result is true.

(2) Bit operation

In bit operation, the following operators are available.

Operator	Operation	Truth table		
not	Not (Negation)	X	not X	
		1	0	
		0	1	
and	And (Logical multiplication)	X	Y	X and Y
		1	1	1
		1	0	0
		0	1	0
		0	0	0
or	Inclusive or (Logical addition)	X	Y	X or Y
		1	1	1
		1	0	1
		0	1	1
		0	0	0
xor	Exclusive or (Logical exclusion)	X	Y	X xor Y
		1	1	0
		1	0	1
		0	1	1
		0	0	0
eqv	Logical equivalence	X	Y	X eqv Y
		1	1	1
		1	0	0
		0	1	0
		0	0	1
imp	Logical implication	X	Y	X imp Y
		1	1	1
		1	0	0
		0	1	1
		0	0	1

In bit operation, value type for logical operator has to be integer or long integer. The following table shows values of decimal, hexadecimal and binary expression of each type

< Integer >

Decimal	Hexadecimal	Binary
32767	&H7FFF	0111 1111 1111 1111
32766	&H7FFE	0111 1111 1111 1110
:	:	:
2	&H0002	0000 0000 0000 0010
1	&H0001	0000 0000 0000 0001
0	&H0000	0000 0000 0000 0000
-1	&HFFFF	1111 1111 1111 1111
-2	&HFFFE	1111 1111 1111 1110
:	:	:
-32767	&H8001	1000 0000 0000 0001
-32768	&H8000	1000 0000 0000 0000

< Long integer >

Decimal	Hexadecimal	Binary
2147483647	&H7FFFFFFF	0111 1111 1111 1111 1111 1111 1111 1111
2147483646	&H7FFFFFFE	0111 1111 1111 1111 1111 1111 1111 1110
:	:	:
2	&H00000002	0000 0000 0000 0000 0000 0000 0000 0010
1	&H00000001	0000 0000 0000 0000 0000 0000 0000 0001
0	&H00000000	0000 0000 0000 0000 0000 0000 0000 0000
-1	&HFFFFFFF	1111 1111 1111 1111 1111 1111 1111 1111
-2	&HFFFFFFFE	1111 1111 1111 1111 1111 1111 1111 1110
:	:	:
-2147483647	&H80000001	1000 0000 0000 0000 0000 0000 0000 0001
-2147483648	&H80000000	1000 0000 0000 0000 0000 0000 0000 0000

In logical bit operation, binary bits of each value are operated according to the truth table. As the result, for example, “and” operator can clear some bits and “or” operator can mix some bits.

Examples of logical bit operation are shown below.

Logical bit operation	Result	Explanation
63 and 8	8	$63 = (11111)_2$ $8 = (001000)_2$ $63 \text{ and } 8 = (001000)_2 = 8$ Therefore, 63 and 8 = 8
-1 and 8	8	$-1 = (1111111111111111)_2$ $8 = (0000000000001000)_2$ $-1 \text{ and } 8 = (0000000000001000)_2 = 8$ Therefore, -1 and 8 = 8
12 or 11	15	$12 = (1100)_2$ $11 = (1011)_2$ $12 \text{ or } 11 = (1111)_2 = 15$ Therefore, 12 or 11 = 15
32767 or -32768	-1	$32767 = (0111111111111111)_2$ $-32768 = (1000000000000000)_2$ $32767 \text{ or } -32768 = (1111111111111111)_2 = -1$ Therefore, 32767 or -32768 = -1
12 xor 11	7	$12 = (1100)_2$ $11 = (1011)_2$ $12 \text{ xor } 11 = (0111)_2 = 7$ Therefore, 12 xor 11 = 7
10 xor 10	0	$10 = (1010)_2$ $10 = (1010)_2$ $-1 \text{ and } 8 = (0000)_2 = 0$ Therefore, 63 and 8 = 8
not X-1	-X	If $X = 10$, $10 = (0000000000001010)_2$ $10-1 = (0000000000001001)_2$ $\text{not } 10 = (111111111110110)_2 = -10$ Therefore, not 10-1 = -10

6.4.4 Character string operator

Character string “+” operator joins a string to another.

Example)


a\$ = “HIRATA” : b\$ = “INDUSTRIAL” : c\$ = “ROBOT”

d\$ = a\$ + “#” + b\$ + “#” + c\$

--- “HIRATA#INDUSTRIAL#ROBOT” is substituted for d\$.

6.4.5 Priority of Operations

Priority of operations is show below. Calculation is executed in order of this priority.

Priority	Operator
High	()
	Exponent (^)
	Plus sign (+), Minus sign (-)
	*, /
	mod
	+, -
	Relational operators (<=, >=, <, >, <>, =)
	not
	and
	or
	xor, eqv, imp
Low	

6.5 Expression

Expression is the general numeric expression as constants, variables and functions connected with operator. Also, the characters and numerals or any variables are regarded as expression.

Example)

“BASIC”

3.14

$10 + 3 / 5$

$a! + b! / c! - d!$

$\text{Tan}(x\#)$

7. Structured Programming

Structured programming is the programming method developed to increase productivity, reliability and maintainability of programs in the software engineering.

The followings are two main fundamentals of structured programming.

- Element of program structure
- Subroutine as program module

HrBasic is able to program by means of these fundamentals. So, HrBasic has the ability to develop a structured program.

Element of program structure, module and remarks in HrBasic are described below.

7.1 Element of Program Structure

Structured programming is based on the combination of the following programming elements to describe a procedure.

- (1) Sequence structure
- (2) Selection structure
- (3) Iteration (Repetition) structure

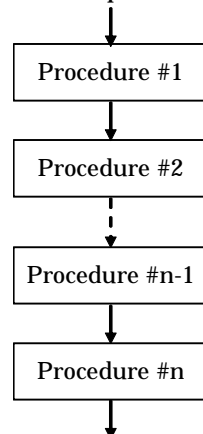
In a structured program, there are one entrance and one exit of a procedure using the above-mentioned elements. This structure makes a program easier to understand, maintain and test.

If GoTo statement is frequently used in a program, process flow jumps irregularly. This causes difficulty to understand program flow with decrement of maintainability and makes testing too complex. Therefore, GoTo statement is not used in structured programming generally. But, in some case, local usage of GoTo statement is effective to simplify a program.

Each element is explained below.

7.1.1 Sequence Structure

Sequence structure means the program structure which is executed from top to end of program. In the following figure, the program is executed in order of the number from procedure #1 to procedure #n.



In HrBasic, a source program is executed from the first line to the end line sequentially. In case of multi-statement, a sentence in one line is executed from left to right.

Example)

```

Job Name "arm"
Global g.ArmCmd%
arm.robot% = 1 : table.robot% = 0

*POWER.ON
g.ArmCmd% = &HFF
Job "arm" Off

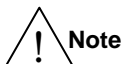
```

7.1.2 Selection Structure

Selection structure means to select a procedure by the logical condition. There are some patterns of selection structures. The following figure shows the pattern of selection structure and the description of HrBasic.

Type	Structure	Program
Two branches #1	<pre> graph TD Entry(()) --> Cond{Condition} Cond -- True --> Proc[Procedure] Cond -- False --> Join(()) Proc --> Join Join --> Exit(()) </pre>	<pre> If Condition Procedure EndIf </pre>
Two branches #2	<pre> graph TD Entry(()) --> Cond{Condition} Cond -- True --> Proc1[Procedure #1] Cond -- False --> Proc2[Procedure #2] Proc1 --> Join(()) Proc2 --> Join Join --> Exit(()) </pre>	<pre> If Condition Procedure #1 Else Procedure #2 EndIf </pre>
Multiple selected branches	<pre> graph TD Entry(()) --> Cond[Condition] Cond -- #1 --> Proc1[Proc.#1] Cond -- #2 --> Proc2[Proc.#2] Cond -- #n --> Procn[Proc.#n] Cond -- Else --> Procn1[Proc.#n+1] Proc1 --> Join(()) Proc2 --> Join Procn --> Join Procn1 --> Join Join --> Exit(()) </pre>	<pre> Select Case Condition Case #1 Procedure #1 Case #2 Procedure #2 : Case #n Procedure #n Case Else Procedure #n+1 End Select </pre>

In this figure, "Procedure" may contain not only one step but multiple steps. And it can contain selection structures.



Note

In case of two branches, maximum number of nestsⁱ is allowed up to 20. In case of multiple selected branches, it is allowed up to 8.

"Condition" is the expression which has the logical result of true or false. And it has to be programmed in one line.

Sample programs of each pattern are shown below.

ⁱ Nest: Recursive usage of a program structure

- (1) Two branches #1
If arm.pos% <> 0 then 'Arm position --- upper.
 'Move arm down
 OUTB(O.ARM.UP) = SWITCH.OFF
 OUTB(O.ARM.DOWN) = SWITCH.ON
 arm.pos%=0 'Arm position --- origin
EndIf
- (2) Two branches #2
If INB(I.SHUT.OPEN) = 1 then 'Shutter opened
 OUTB(O.SHUT.CLOSE) = SWITCH.OFF
 OUTB(O.SHUT.OPEN) = SWITCH.OFF
Else
 OUTB(O.SHUT.CLOSE) = SWITCH.OFF
 OUTB(O.SHUT.OPEN) = SWITCH.ON
EndIf
- (3) Multiple selected branches
Select Case object.plate.no%
Case PLATE1
 Move #1,PTP,PM(PM.PLATE1)
Case PLATE2
 Move #1,PTP,PM(PM.PLATE2)
Case PLATE3
 Move #1,PTP,PM(PM.PLATE3)
Case PLATE4
 Move #1,PTP,PM(PM.PLATE4)
Case PLATE5
 Move #1,PTP,PM(PM.PLATE5)
Case Else
 error.flag%=13
End Select
- (4) Combination of selections
sys.err%=Err
Select Case sys.err%
Case 39
 error.flag% = 2 'Receiving error
Case 43
 error.flag% = 3 'Not connected
Case 80
 error.flag% = 4 'Timeout
Case 81
 If (Ref(#1,STATUS8) and &H40) <> 0 then
 error.flag% = 11 'Emergency stop
 Else
 If (ref(#1,STATUS8) and &H01) <> &H01 then
 error.flag% = 10 'Not online
 Else
 error.flag% = 5 'Robot error
 EndIf
 EndIf
Case 82
 error.flag% = 6 'Robot response error
Case 83
 error.flag% = 7 'Robot memory error
Case Else
 If sys.err% <= 14 then
 error.flag% = 12 'System error
 Else

```

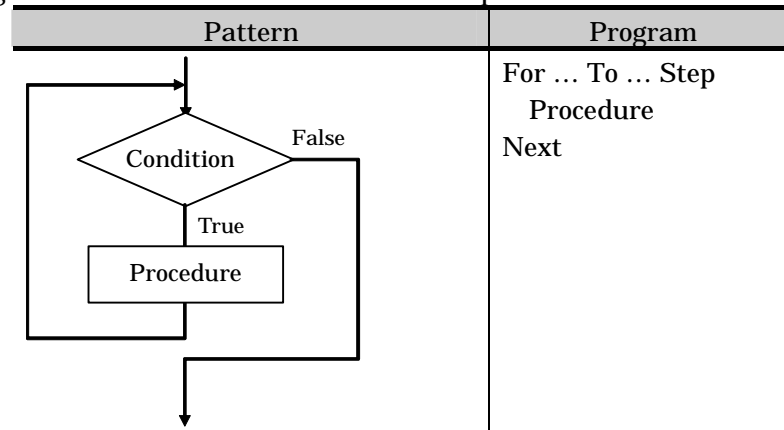
        error.flag% = 08  'Application error
    EndIf
End Select

```

7.1.3 Iteration Structure

Iteration structure means that a procedure repeats while a condition is fulfilled.

In HrBasic, For-Next statement is supported for this structure. The following figure shows the structure and the description of HrBasic.



In this figure, “Procedure” may contain not only one step but multiple steps. And it can contain selection structures.



Note

In For-Next statement, maximum number of nests is allowed up to 16.

In For-Next statement, “Condition” is described to count the number of repetitions.



Note

GoTo statement is needed when a program exits the loop without the condition of For statement. See “7.1.4 Usage of GoTo statement”

Sample program is shown below.

```

For i% = 0 to data.cnt% - 1
    data.box%( i%, 0 ) = data.bax( i%+1, 0 )
    data.box%( i%, 1 ) = data.bax( i%+1, 1 )
    Select Case data.box%(i%, 0)
        Case PATTERN.1
            MD(MD.PATNO1)=i%
        Case PATTERN.2
            MD(MD.PATNO2)=i%
        Case PATTERN.3
            MD(MD.PATNO3)=i%
        Case Else
            error.flag% = 10
    End Select
Next i%

```


7.1.4 Usage of GoTo Statement

In structured programming, GoTo statement is not used generally. But some case of repetition structure has to use GoTo statement. And in some case, a program is difficult to understand without GoTo statement.

These cases are shown below.

(1) Repetition structure without For-Next statement

In case that a procedure repeats while a condition except the count of repetitions is fulfilled, GoTo statement has to be used.

Example)

'Flicker blue lamp by 1 sec interval

*BLUE.BLINK.LOOP

Delay 1

OUTB(O.BLUE)=1

Delay 1

OUTB(O.BLUE)=0

'Loop while manual or ready mode

If mode%=MANUAL.MODE or mode%=READY.MODE Then

GoTo * BLUE.BLINK.LOOP

EndIf

(2) Repetition structure with For-Next statement adding some condition

In addition to the count of repetitions, if some condition is fulfilled, a program exits the For-Next loop using GoTo statement.

Exxample)

For i = 0 to pate.no%

If plate.mode(i%) = BUSY Then

equip.mode% = BUSY

GoTo *NEXT.CHECK

EndIf

next

* NEXT.CHECK

:

:

(3) Infinite loop

If a program never terminates except the power-off or Job Off by another job, GoTo statement is used for the infinite loop.

Example)

*LOOP

Select Case mode%

Case ERROR.MODE

'Error mode

GoSub *RED.BRINK

Case MANUAL.MODE

'Manual mode

GoSub *BLUE.BLINK

Case READY.MODE

'Ready mode

GoSub *BLUE.BLINK

Case AUTO.MODE

'Running mode

GoSub *BLUE.LIGHT

Case Else

Select End

GoTo *LOOP

(4) Distinction between exception procedure and normal procedure

An exception procedure can be programmed using If-Then-Else statement without GoTo statement, but if an exception procedure is complex, there is a case that a program becomes simpler to use GoTo statement.

< Example without GoTo statement >

```
If plate.mode%(HP1) <> AUTO or plate.mode%(HP2) <> AUTO Then
'Exception #1
    error.flag% = 14
Else
    If plate.mode%(HMDS) <> AUTO Then 'Exception #2
        error.flag% = 37
    Else
        If plate.mode%(CP1) <> AUTO Then 'Exception #3
            error.flag% = 38
        Else
            If plate.mode%(CP2) <> AUTO Then 'Exception #4
                error.flag% = 39
            EndIf
        EndIf
    EndIf
EndIf
'Normal procedure
:
:
:
```

< Example with GoTo statement >

```
If plate.mode%(HP1) <> AUTO or plate.mode%(HP2) <> AUTO Then
'Exception #1
    error.flag% = 14
    GoTo *PLATE.CHK.ERR
EndIf
'Exception #2
If plate.mode%(HMDS) <> AUTO Then
    error.flag% = 37
    GoTo *PLATE.CHK.ERR
EndIf
'Exception #3
If plate.mode%(CP1) <> AUTO Then
    error.flag% = 38
    GoTo *PLATE.CHK.ERR
EndIf
'Exception #4
If plate.mode%(CP2) <> AUTO Then
    error.flag% = 39
    goto *PLATE.CHK.ERR
EndIf

'Normal procedure
*PLATE.CHK.ERR
:
:
```

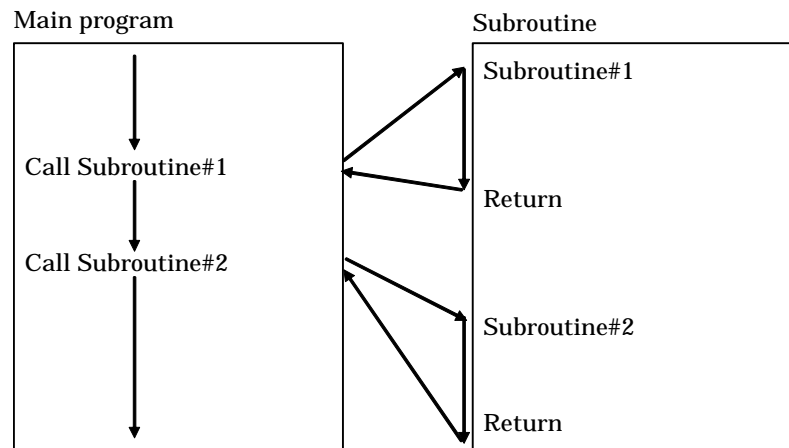
For a simple program, it is recommended that GoTo statement is used in the cases of (1) through (4).

7.2 Subroutine as Program Module

7.2.1 Merit of Subroutine

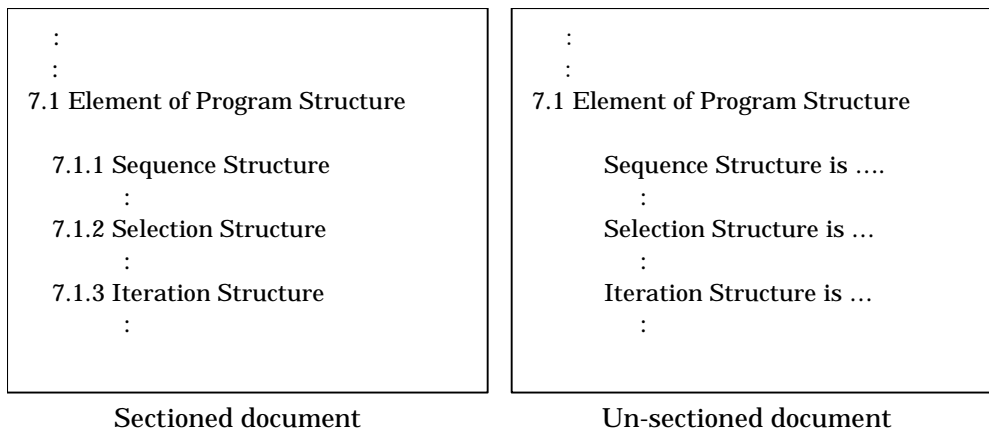
As a system is bigger, its program is larger and more complex with the difficulty to understand. One of the solutions is that a program is divided to small subroutines.

If there is a common procedure in a program, that procedure should be described as a subroutine which is independent from a main program. And the subroutine has to be called in a main program. After the call of subroutine, a procedure of the subroutine is executed and then exits and returns to a main program.



This structure avoids the loss of some procedures in a program. And if the trouble of the procedure happens, only the subroutine has to be modified.

Even if there is no common procedure, to divide a program to small subroutines are effective to increase maintainability and quality of a program. This is likened to make a sectioned document such as a manual. A sectioned document is easier to understand than un-sectioned document.



If a procedure runs once, it is better that the procedure “Initialization”, for example, is extracted from a main program as a subroutine. This causes that a main program is simpler and a HrBasic user concentrates on only the “Initialization” subroutine when the system initialization is tested or checked.

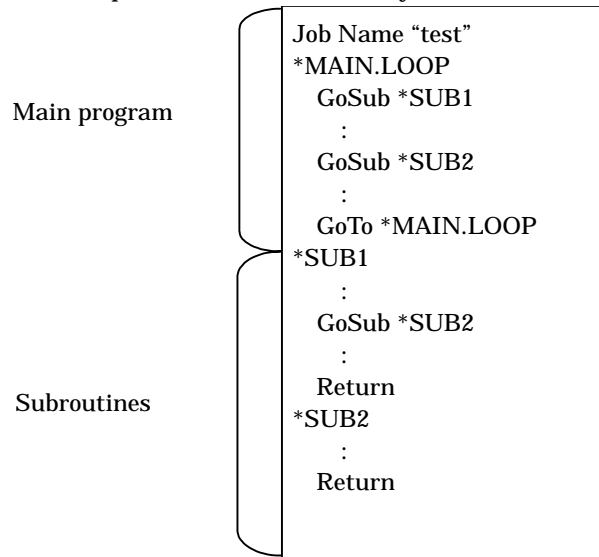
Generally, merits to divide a program to subroutines are the followings.

- (1) Compact to develop
- (2) Easy to debug

- (3) Easy to understand

7.2.2 Practice and Note of HrBasic Subroutine

In HrBasic, a subroutine is called by GoSub statement and it returns to the the next step of GoSub statement by Return statement.



There are some notes to use subroutines as follows.

- (1) Local variable in a subroutine

In HrBasic, a local variable can be accessed at any step in a job. A local variable which is accessed in a main program is also able to be read or written in a subroutine. Therefore, after a local variable is written in a subroutine, a main program cannot get the last value of the variable and there is the bad case that a main program cannot work correctly.

It is necessary that usage of a local variable is independent from a main program.

In the following example, i% is changed in AUTO.IO.SET subroutine and the main program does not run with the unexpected result.

Example)

```

Job Name "sample1"
Include "sample.hed"
Global g.PortMode%(3)
*MAIN.LOOP
For i% = 0 To 3 'Set i%
  Select Case g.PortMode%(i%)
  Case AUTO.MODE:
    GoSub *AUTO.IO.SET
    :
  End Select
  GoTo *MAIN.LOOP

*AUTO.IO.SET
For i%=0 to 2 '!!! i% overwritten !!!
  OUTB( O.BASE+i%) = 1
Next i%
:
Return
:

```

- (2) Limit of subroutine nests

The maximum nests of subroutines are 16. If the nests are overflow, Job error “Out of memory” occurs.

In the following example, SUB1 through SUB16 are called in nests.

Example)

Job Name “sample2”

```
*MAIN.LOOP
  GoSub *SUB1
  GoTo *MAIN.LOOP

*SUB1
  GoSub *SUB1
  Return
*SUB2
  GoSub *SUB2
  Return
  :
  :
*SUB15
  GoSub *SUB16
  Return
*SUB16
  :
  Return
```

(3) Pairing of GoSub and Return

A subroutine has to be programmed as entry by GoSub and exit by Return. To use GoTo statement, there may be a problem in case that a program jumps to enter a subroutine or exits from a subroutine.

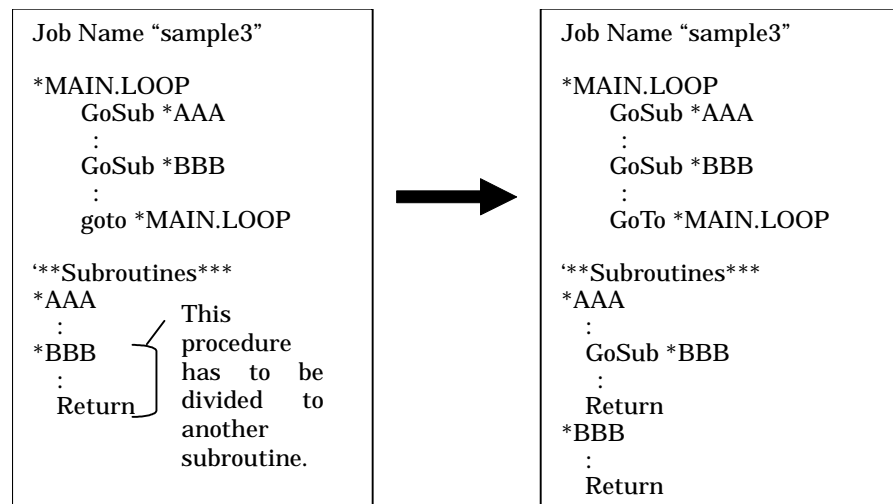
Case	Example	Explanation
Entry by GoSub and exit by Return	<pre>*LOOP GoSub *AAA : GoTo *LOOP *AAA 'Subroutine : Return</pre>	Correct usage without problem.
Entry by GoSub and exit by GoTo	<pre>*LOOP GoSub *AAA *BBB : GoTo *LOOP *AAA 'Subroutine : GoTo *BBB</pre>	When a program is running, STP checks the pairing of GoSub and Return. In this case, job error “Nests of GOSUB-RETURN overflow” occurs.

Case	Example	Explanation
Entry by GoTo and exit by GoSub	<pre> *LOOP GoTo *AAA : GoTo *LOOP *AAA 'Subroutine : Return </pre>	When a program is running, STP checks the pairing of GoSub and Return. In this case, job error "RETURN without GOSUB" occurs.
Entry by GoTo and exit by GoTo	<pre> *LOOP GoTo *AAA *BBB : GoTo *LOOP *AAA 'Subroutine : GoTo *BBB </pre>	In this case, a program runs normally. But, this usage of GoTo statement causes the difficulty to understand the program. Such complex program is called "spaghetti program" ⁱ .

(4) Independency of subroutine

A subroutine has to be functioned simply and it is necessary to have high independency from other program.

In the following example, the left program has to be modified as the right program.

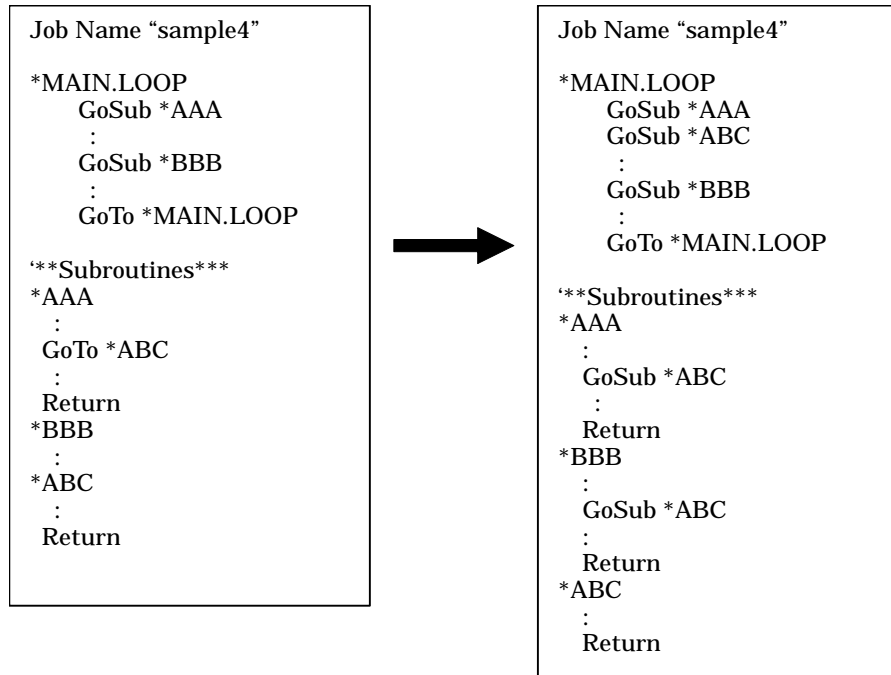


(5) Go back to the program which has called the subroutine.

When a program exits from a subroutine, to return into the program which has called the subroutine is necessary. The following left figure is a bad example that a program jumps to another subroutine. This example is a just "spaghetti program".

In this case, the program structure needs to be designed properly by functional division and then it has to be modified to the right figure.

ⁱ Spaghetti program: complex and tangled program like spaghetti

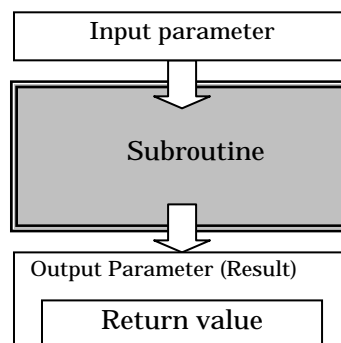


7.2.3 Input Parameter and Output Parameter

A function is programmed in a subroutine. There is a case that a function needs some parameters before execution and the result after a subroutine has executed a function to return.

A parameter specified before a subroutine is executed is called "input parameter" and a parameter as the result of a subroutine is called "output parameter". A input parameter has to be set in a main program calling a subroutine and a output parameter has to be set in a subroutine. Parameter for a subroutine is often called "argument".

In case that a subroutine returns with the value of the result, the value is called "return value" which generally contains error information. A return value is a kind of output parameter.



In HrBasic, a local variable is generally used as a parameter to increase the independency of a subroutine.

For example, the following shows the subroutine which calculates triangle area by its base length and height.

< Example of subroutine >

'Procedure: GET.TRI.AREA

'Summary: Get triangle area

'Return: [OUT] ret% =0:OK =-1:parameter error

'Argument:

```

' [IN] base! --- Base length of triangle (0 to 100cm)
' [IN] height! --- Height of triangle (0 to 100cm)
' [OUT] area! --- Area of triangle (cm2)
'Caution:
*****
*GET.TRI.AREA
  'Clear return value
  ret% = 0
  'Check parameter
  If base! < 0.0! or base! > 100.0! Then
    ret% = -1 'parameter error
    Return
  EndIf
  If height! < 0.0! or height! > 100.0! Then
    ret% = -1 'parameter error
    Return
  EndIf
  'Get area
  area! = (base! * height!) / 2.0!
  'Normal return
  Return
< Example of program to call the subroutine >
  'Get triangle area with base 21.3cm, height 3.5cm
  base! = 21.3!; height! = 3.5! 'Set input parameters
  GoSub *GET.TRI.AREA
  If ret% <> 0 Then 'Error
    GoTo *ERROR.HANDLER
  EndIf
  g.TriArea! = area! 'Set result to global variable

```



**Guideline for
Programming**

It is recommended that the subroutine specification is described like the following format using comments before the subroutine program is described.

```

'Procedure: Subroutine-Name
'Summary: Function-Overview
'Return: [OUT] Return-Value-Explanation
'Argument: [IN] Input-Parameter-Explanation
'          [OUT] Output-Parameter-Explanation
'Caution: Note

```

```

*Subroutine-Name
  Subroutine-Program
  Return

```



Note

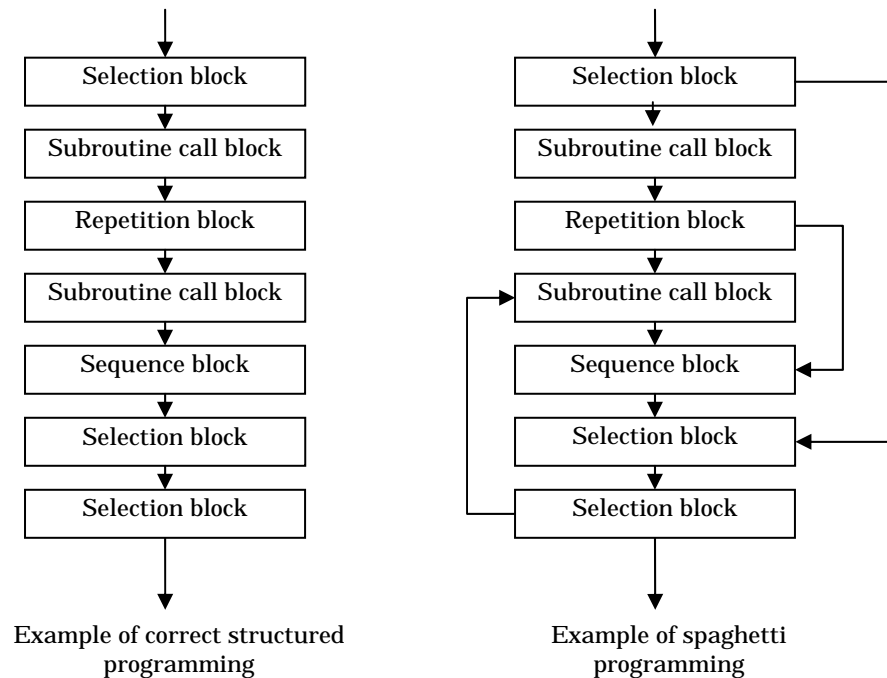
A subroutine without parameters is allowed to program.

7.3 The Point of Structured Programming

As simple explanation, structured programming is the combination of procedure blocks. A program is built by blocks of sequence structures, selection structures, repetition structures and subroutines.

In structure programming, it keeps strictly that each block has only one entry and one exit.

Generally, a program is not allowed to enter or exit from the middle of a block. A program which frequently enter or exit from the middle of a block is called “spaghetti program”, complex, difficult to understand and maintain, and its quality becomes lower.



The following figure shows bad example and how to modify it.

Example #1) Entry to the middle of selection block

Bad example	Modified
<pre> If addr% < 10 Then GoTo *RB.MOVE EndIf If addr% = 10 Then OUTB(addr% + 100) = 1 *RB.MOVE Move #1,PTP,PM(addr%) EndIf </pre>	<pre> If addr% <= 10 Then If addr% = 10 Then OUTB(addr% + 100) = 1 EndIf Move #1,PTP,PM(addr%) EndIf </pre>

Example #2) Exit from the middle of selection block

Bad example	Modified
<pre> If addr% = 20 Then OUTB(addr% + 100) = 1 Wait INB(addr% + 200) ,10 OUTB(addr% + 100) = 0 If TimeOut Then GoTo *NEXT.STEP EndIf EndIf Move #1,PTP,PM(addr%) *NEXT.STEP : </pre>	<pre> flag%=0 If addr% = 20 Then OUTB(addr% + 100) = 1 Wait INB(addr% + 200), 10 If TimeOut Then flag% = 1 OUTB(addr%+100) = 0 EndIf If flag% = 0 Then Move #1,PTP,PM(addr%) EndIf </pre>

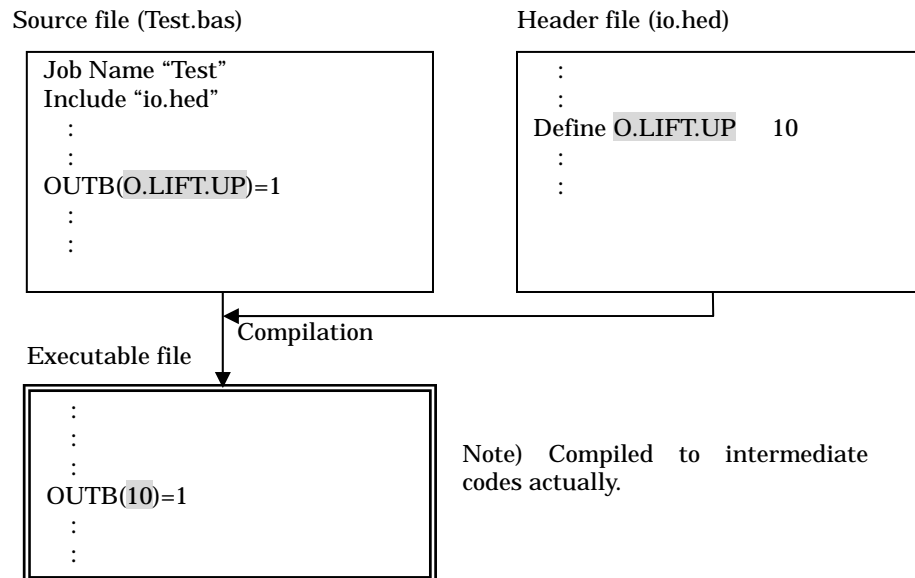
Example #3) Entry to and Exit from the middle of selection block

Bad example	Modified
<pre> If mode% = AUTO Then arm.rbt% = 1 : table.rbt% = 0 If (Ref(#3,SATTUS9) and &h01)=0 Then GoTo *SKIP.SEQ EndIf arm.rbt% = 0 : table.rbt% = 0 Else *SKIP.SEQ arm.rbt% = 0 : table.rbt% = 1 Disable #2 SeqEnd #2 EndIf </pre>	<pre> flag% = 0 If mode% = AUTO Then arm.rbt% = 1 : table.rbt% = 0 If (Ref(#3,STATUS9) and &h01)=0 Then flag% = 1 Else arm.rbt% = 0 : table.rbt% = 0 EndIf EndIf If mode%<>AUTO or flag% =1 Then arm.rbt% = 0 : table.rbt% = 1 Disable #2 SeqEnd #2 EndIf </pre>

7.4 Header File

A header file is the definition file and it has the filename suffix “.hed”.

Define statement can be described in a header file and it replaces a string to another one. When the string, defined at the first argument of Define statement, appears in a source program, the string is replaced with the string specified to the second argument of Define statement. And then the program is compiled.



By means of using the equivalent name by Define statement, the program modification becomes very easier in case that a constant needs to change to another value.

For the above example, only a header file has to be modified and then a program has to be re-compiled and linked, in case that I/O assignment needs to change.

If “OUTB(10)=1” is described one hundred times in a program, you have to find all the description and modify it. However, even if “OUTB(O.LIFT.UP)=1” is described one hundred times, only one sentence in the header file has to be modified.

In case that a constant is described once or few times, it is better that the constant which has a possibility of modification is defined in the header file.

7.5 Macro File

Macro is the efficient method in case that there are many repetitions of the same procedure in the program.

You can use a macro like a function call.

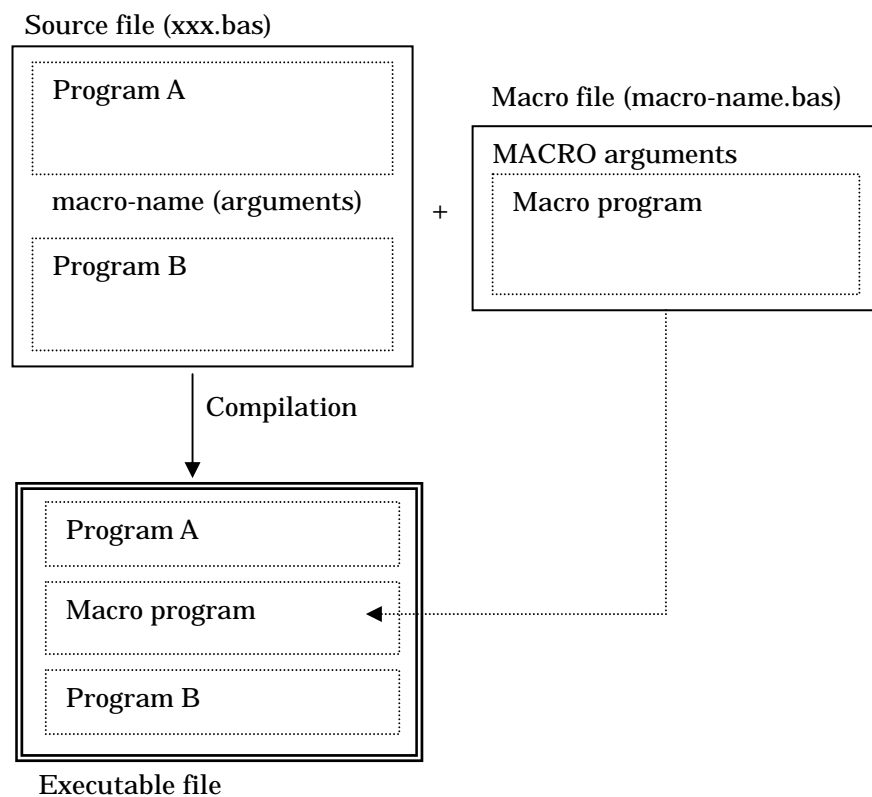
In a macro file, you can define a block of serial program steps as a macro.

The macro name to use like a function call is the name of macro file.

When calling the macro name is described in a program, compiler replaces the macro name to the block of program steps defined in the macro file.

You can use "arguments" like a function call as the interface parameters between a program and a macro.

The demerit is that an executable program becomes bigger in case that many macro-calls appear because the many macro procedures are embedded in a program.



● Macro file

A macro file is created or edited by a text editor as the same as a source file.

The file name except the file extension must consist of one to eight alphabets or numerical characters. And the first character of the file name must be an alphabet.

The file extension must be ".bas".

The macro file name must not be the same as the reserved word (e.g. Sin, Cos).

A macro file is necessary to locate in the directory defined at "Macro files" in [Set-up]-[Directory] of HBDE Main Menu.

A macro file does not need a Job Name statement at the top.

But it needs a Macro statement that defines arguments as parameters for a macro file.

You can define up to ten arguments, but the number of arguments in a source file and the one in a macro file defined by Macro statement must be the same.

Example)

< Source file: Test.bas >

Job Name "Test"

:

Display(1, 2, 3) ' Macro-call

:

< Macro file: Display.bas >

Macro para1%, para2%, para3%

:

} Number of arguments
must be the same.

Variables except reserved variables (e.g. MD, MW) in a macro file are local variables that can be access only in a macro file. Therefore, if a source program wants to get the return value as the execution result of a macro file, use a reserved global variable to set the return value.

Array variable cannot be specified as a argument of a macro.

● Example

< Source file: Test.bas >

Job Name "Test"

:

:

:

work%=4

weight%=0

limit%=10

Trans(work%, weight%, limit%, 8) ' Macro-call

:

:

Calling a macro file "Trans.bas" and specifying "work%" as 1st argument, "weight%" as the 2nd argument, "limit%" as 3rd argument and "8" as 4th argument.

< Macro file: Trans.bas >

Macro work.number%, work.weight%, work.limit%, work.lot%

:

:

:

"Macro" statement defines the arguments. In the example, the value of "work%" in a source file is handed over to "work.number%" and similarly "weight%" to "work.weight%", "limit%" to "work.limit%", "8" to "work.lot%".

8. Robot Control Programming

STP on HAC and WinSTP can control our HNC robots to execute HrBasic program.

A HNC-1XX, 2XX, 3XX, 544 type can handle max four axes as one robot.

A HNC-580 series or HAC-8XX can handle four virtual robots and max six axes of each robot.

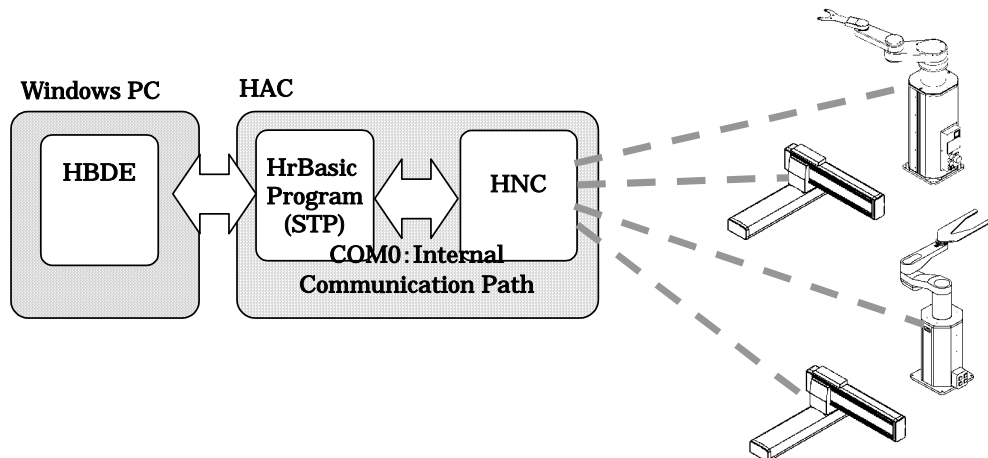
The following explanation shows how HrBasic program controls HNC robots.

8.1 Connection with HNC Robot

The connection method is different between STP on HAC and WinSTP shown as follows.

(1) Connection between STP and HNC on HAC

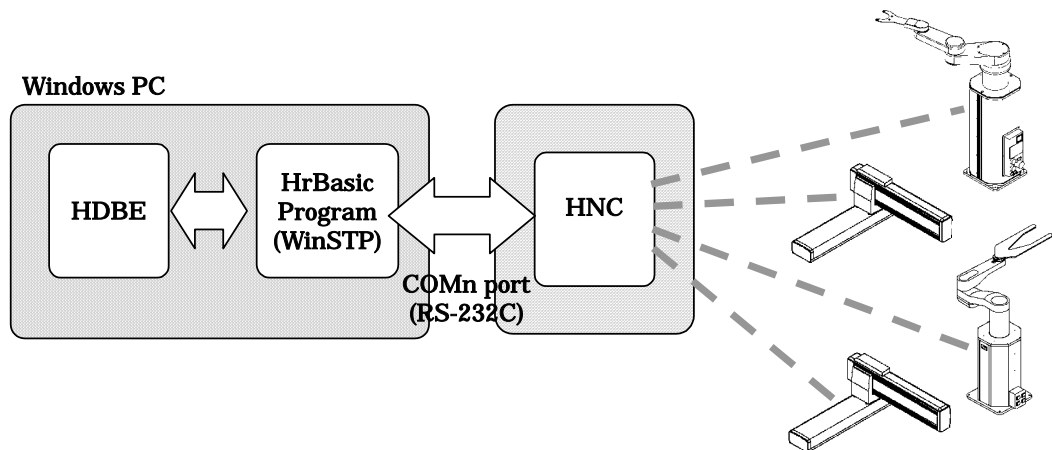
STP and HNC are implemented on the same HAC board. The connection between STP and HNC is realized as the internal communication path on the board. STP can communicate with HNC through this internal path. HrBasic program can access the internal path as "COM0" port.



(2) Connection between WinSTP and HNC robot

Standardly, WinSTP is connected with HNC by RS-232C interface.

HrBasic can communicate with HNC using "COMn" port of PC.



8.2 Procedure of Robot Communication

Procedure of robot communication is the same as the normal data communication as follows.

- (1) Open a port for robot communication
- (2) Access the port for robot communication
- (3) Close the port for robot communication

The details of each procedure are explained below.

8.2.1 Open a Port for Robot Communication

How to open a port for robot communication is the same in case of STP on HAC and WinSTP. But, the number of COM port to open is different in each case. The sample of OPEN statement is as follows.

STP type	Using COM port	OPEN statement
STP on HAC	COM0	open "COM0"
WinSTP	COMn (n>=1)	open "COMn: ..."

After a COM port is opened once, all jobs can access the port.



Note

- The same COM port cannot be opened multiply even if the different file number is specified.
- The same file number cannot be opened multiply even if the different COM is specified.

The parameters specified to OPEN statement are shown below.

- (1) COM port name
Specify the COM port name to open.
In case that STP will control HNC on HAC, "COM0" must be specified.
In case of WinSTP, "COMn" can be specified as RS-232C port. (n>=1)
- (2) RS-232C settings
In case of "COM0" for HAC, RS-232C settings are not needed.
In case of using RS-232C port, communication speed, parity, data length and stop bit must be specified.
- (3) File number
HrBasic treats a communication port (COM port) as a file. Specify the file number to assign for the opened file. After the file has been opened, the file number is used for all statements or functions to access the file.
- (4) Robot type
Specify the HNC type to communicate. If the type is omitted or zero is specified, HNC-1XX, 2XX, 3XX, 544 type that has no virtual robot with four axes max is selected. If "580" is specified, HNC-580 series type that has four virtual robots with max six axes for each robot is selected. In case of HAC-8XX, "580" must be specified because the type of HNC on HAC-8XX is the same as HNC-580 series.
- (5) Robot number list
If "580" is specified to "Robot type" described above, HrBasic can control up to the four virtual robots. The list of using robot numbers is needed for OPEN statement.

The robot number is the number that is set at HNC System Generation data as [MAINTENANNCE]-[MAINTENANCE DATA]-[STATION NO]. This data can be manipulated by teaching pendant or HrEditor installed as the software component of HBDE. [STATION NO] has the range of 1 to 999. Default setting of [STATION NO] is shown below.

Virtual Robot	Robot number [STATION NO]
ROBOT 1	1
ROBOT 2	2
ROBOT 3	3
ROBOT 4	4

The examples to open a robot communication port are shown below.

(Example 1 : HAC-8XX)

open "COM0" as #1 robtype=580 robnolist=1,2,3

- 1) Communication port COM0 (Internal interface port)
- 2) File number 1
- 3) HNC type 580
- 4) Robot number list 1, 2, 3

(Example 2 : WinSTP controls HNC-580 series)

open "COM1:19200,E,7,1" as #1 robtype=580 robnolist=1,2

- 1) Communication port COM1 (RS-232C port)
- 2) File number 1
- 3) RS-232C settings
 - Communication speed 19200 bps
 - Parity Even parity
 - Data length 7 bits
 - Stop bit 1 bit
- 4) HNC type 580
- 5) Robot number list 1, 2

(Example 3 : WinSTP controls HNC-1XX,2XX,3XX,544 type)

open "COM1:9600,E,7,1" as #1

- 1) Communication port COM1 (RS-232C port)
- 2) File number 1
- 3) RS-232C settings
 - Communication speed 9600 bps
 - Parity Even parity
 - Data length 7 bits
 - Stop bit 1 bit

8.2.2 Access the Port of Robot Communication

Using the file number assigned at OPEN statement, you can access the port to control the robot.

The file number must be specified to statements or functions (MOVE, REF, etc.) for robot control.

Example)

move #1, pm(100) 'move to teaching address #100

In addition, in case of HNC type "580", the robot number of the target that the program controls must be specified.

The robot number can be specified by the following two ways.

- (1) Specify the robot number in each statement or function for robot control. The description "[rno:robot_no]" can be added after the file number. "robot_no" is specified as the constant or the variable.

Example)

```
'Move the robot number #2 through file number #1 to address #100.
move #1[rno:2], pm(100)
```

- (2) Define the implicit robot number of current job using SETROBNO function. The implicit robot number will be used at robot control statements or functions without the description "[rno:robot_no]".

Example)

```
'Define implicit robot number is #2 for current job
setrobno(2)
'Move the robot number #2 through file number #1 to address #100.
move #1, pm(100)
```

8.2.3 Close the port for robot communication

Using the file number assigned at OPEN statement, close the port for robot communication. In case of HNC type "580", the robot number cannot be specified and the all communications of virtual robots are closed.

Example)

```
close #1 'Close port of file number #1
```



Note

If CLOSE statement is executed without the filename, all files that have been opened in STP will be closed.

8.3 Overview of Statements And Functions for Robot Control

HrBasic implements statements and functions that can control our HNC robot. In the following explanation, these statements and functions are overviewed. See "9.3 Language Reference" about the details.

- (1) MOVE statement

MOVE statement moves the robot to the specified position. By standard usage, MOVE does not return until the robot stops to complete positioning. But if an error occurs while the robot moving, MOVE finishes executing and a job error is raised.

If "nowait" option is added, MOVE returns immediately after the robot starts to move.

- (2) SET statement

SET statement sets data of the moving characteristics.

- (3) REF function

REF function returns the current status information of the robot.

- (4) SEQ-SEQEND statement

When MOVE is executed within SEQ-SEQEND block, MOVE returns immediately after the robot starts to move. But in case of the Z-axis down motion, the robot does not move Z-axis down waiting the execution of FINISH statement. FINISH statement allows that the robot moves Z-axis down. Even if there is not Z-axis down motion, FINISH statement is needed to complete positioning.

STP does not check the completion of positioning, the program have to confirm it.

In SEQ-SEQEND block, the program can check or control I/O while the robot is moving.

(5) FINISH statement

FINISH statement must be used within SEQ-SEQEND block.

FINISH statement allows that the robot moves Z-axis down after the robot starts to move by MOVE statement. If the execution of FINISH is late or there is not the execution of it, the robot is waiting at the position where Z-axis is up. Even if there is not Z-axis down motion, FINISH statement is needed for the completion of positioning

(6) HOLD ON/OFF statement

In case of HNC-1XX, 2XX, 3XX, 544 type, HOLD statement make the axis holding or not holding.

In case of HNC-580 series or HAC-8XX, HOLD statement is not supported since the all axes are always held after power on.

(7) DISABLE statement

While the robot is moving after MOVE statement, DISABLE statement can stop the robot moving. If the robot stops, MOVE statement by normal usage returns immediately and the next step of MOVE is executed.

(8) CALIB statement

For the robot system, A-CAL (Automatic Calibration) must be executed at least one time on purpose to make the origin point of the control system equivalent to the mechanical origin point.

CALIB statement executes A-CAL.

If using the absolute type of motor encoder, A-CAL data does not disappear after power off. In this case, A-CAL has to be executed only once normally.

If using the incremental type of motor encoder, A-CAL data disappears after power off. In this case, A-CAL has to be executed after every power on.

(9) SETROBNO function

SETROBNO function is used when the robot type "580" is specified at OPEN statement.

SETROBNO function defines the implicit robot number of current job. After SETROBNO is executed, specifying the robot number at a statement or function for robot control can be omitted.

Note) The implicit robot number is initialized to the value -1 for all jobs when STP or WinSTP starts.

(10) CLEARROBNO function

CLEARROBNO function is used when the robot type "580" is specified at OPEN statement.

CLEARROBNO function clears the current implicit robot number of the job. After CLEARROBNO is executed, the returned value of GETROBNO is -1.

(11) GETROBNO function

GETROBNO function is used when the robot type "580" is specified at OPEN statement.

GETROBNO returns the current implicit robot number of the job.

(12) ENABLEONLINEERR statement

ENABLEONLINEERR statement enables to check online mode of the robot during robot moving. If the robot is not online while robot moves, MOVE statement raises a job error.

After STP or WinSTP starts, online check is enabled as default for all jobs.

(13) DISABLEONLINEERR statement

DISABLEONLINEERR statement disables to check online mode of the robot during robot moving. Even if the robot is not online while robot moves, MOVE statement does not raise a job error and waits the completion of robot positioning.

(14) ROBCHECKBPZONE function

ROBCHECKBPZONE function is available when the robot type "580" is specified at OPEN statement.

ROBCHECKBPZONE checks current BP/ZONE status of the robot.

BP/ZONE status is the current positioning information of the robot as follows.

- BP (Base Position) If the current position of the robot is near the base position, BP status bit is ON.
- ZONE If the current position of the specified axis is within the range, ZONE status bit is ON.

ROBCHECKBPZONE function returns true value if the BP/ZONE status bit is ON.

Refer to robot operation manual about BP/ZONE settings.

(15) ROBCHECKCURPOS function

ROBCHECKCURPOS function checks that the current position of the robot is near the specified position address. ROBCHECKCURPOS function returns true value if the robot position is near.

The checking range of position can be defined by ROBSETPOSRANGE statement.

(16) ROBCHECKSTOP function

ROBCHECKSTOP function checks that the robot is stopping currently. ROBCHECKSTOP function returns true value if the robot is stopping.

(17) ROBCLEARERR statement

ROBCLEARERR statement clears error status of the robot. If the program restarts to move the robot after an error has occurred, error recovery must be executed by ROBCLEARERR statement.

Note) ROBCLEARERR statement cannot recover some errors.

(18) ROBSETPOSRANGE statement

ROBSETPOSRANGE statement defines the range to check the robot near the position address using by ROBCHECKCURPOS function.

Default setting is 1.0mm or 1.0deg. for all axes.

8.4 Sample Program

The sample program that two robots move parts from station #1 to station #2 via temporary table is shown below.

This sample assumes that HAC-8XX is used and two virtual robots have been configured.

8.4.1 Specification of Sample Program

(1) Motion

- 1) After the input signal to start is 1, two robots confirm to hold any parts.
- 2) Two robots move to the waiting position.
- 3) Robot #1 moves to the station #1.
- 4) Robot #1 gets a part with a chuck.
- 5) Robot #1 moves to the temporary table.
- 6) Robot #1 puts the part on the temporary table.
- 7) Robot #1 moves the waiting position and then robot #2 moves to the temporary table.
- 8) Robot #2 gets the part by vacuum.
- 9) Robot #2 moves to the station #2.
- 10) Robot #2 puts the part on the station #2
- 11) Robot #2 moves to the waiting position.

(2) Remote input signals

- Request to start
- Request to reset error
- Part presence on station #1
- Part presence on temporary table
- Part presence on station #2
- Part presence on robot #1
- Robot #1 chuck opened
- Robot #1 chuck closed
- Part presence on robot #2
- Robot #2 vacuum enabled

(3) Remote output signals

- System error
- Open/close robot #1 chuck
- Vacuum of robot #2
- Blow of robot #2

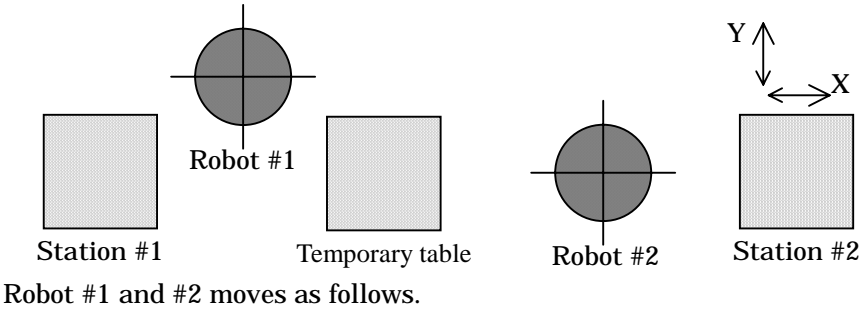
8.4.2 Job List of Sample Program

Job Name	Program Name	Function	Explanation
init	init.bas	System initialization	The job initializes variables, output signals and opens a communication port. The job must start first and the other jobs must wait for the completion of initialization.
main	main.bas	Process management	After "init" job completes initialization, the job accepts a request from the outside and manages the whole process of the system to indicate the process to the robot control jobs
robot1	robot1.bas	Robot #1 control	After "init" job completes initialization, the job controls robot #1 by indication from "main" job.

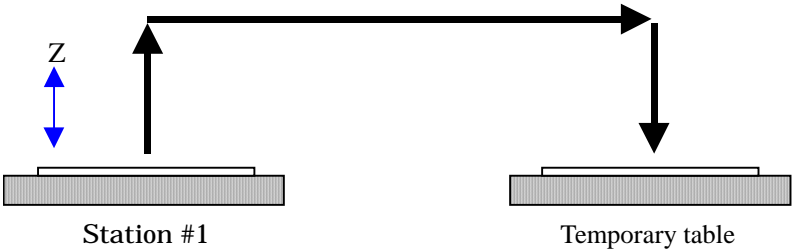
Job Name	Program Name	Function	Explanation
robot2	robot2.bas	Robot #2 control	After "init" job completes initialization, the job controls robot #2 by indication from "main" job.

8.4.3 Motion of Robots

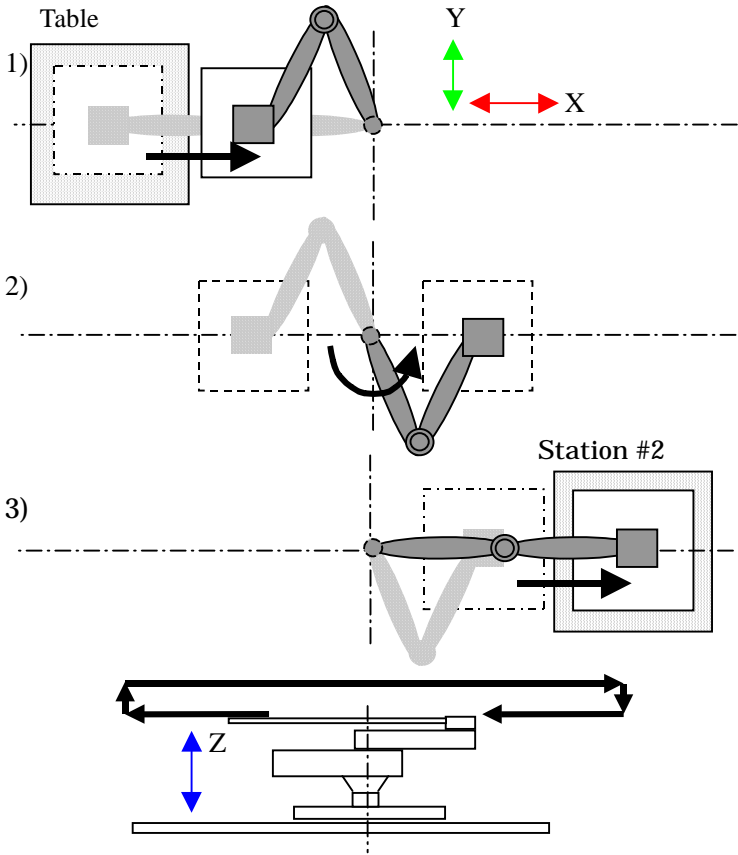
The position of robots and stations is shown below.



(1) Robot #1



(2) Robot #2



8.4.4 Header File

```
#####
' Robot.hed
#####

*****

' Robot Numbers
*****

define ROB.NUM          2
define RB1.NO           1
define RB2.NO           2

*****

' Robot Commands
*****

define CMD.MOVE.WAIT    1
define CMD.MOVE.GET     2
define CMD.MOVE.PUT     3
define CMD.CHK.OPEN    4
define CMD.CHK.CLOSE    5
define CMD.PART.GET     6
define CMD.PART.PUT     7

*****

' Addresses
*****

define RB1AD.WAIT       100   'Robot #1 waiting position
define RB1AD.GET        200   'Robot #1 getting position
define RB1AD.PUT        300   'Robot #1 putting position
define RB2AD.WAIT       100   'Robot #2 waiting position
define RB2AD.GET        200   'Robot #2 getting position
define RB2AD.PUT        300   'Robot #2 putting position

*****

' Input Signals
*****

define I.START          1     'Request to start
define I.ERR.RESET      2     'Request to reset error
define I.ST1.PART       5     'Presence on station #1
define I.TMP.PART       6     'Presence on temporary table
define I.ST2.PART       7     'Presence on station #2
define I.RB1.PART       11    'Presence on robot #1
define I.RB1.CHK.OPEN   12    'Robot #1 chuck opened
define I.RB1.CHK.CLOSE  13    'Robot #1 chuck closed
define I.RB2.PART       21    'Presence on robot #2
define I.RB2.VACUUM     22    'Robot #2 vacuum enabled

*****

' Output Signals
```

```
*****
define O.SYS.ERR          1    'System error
define O.RB1.CHK.CLOSE   10    'Close robot #1 chuck
define O.RB2.VACUUM      20    'Vacuum robot #2
define O.RB2.BLOW        21    'Blow robot #2

*****

'      Error Codes
*****

define ERR.PRESENCE      100
define ERR.ABSENCE       101
define ERR.CHK.OPEN      102
define ERR.CHK.CLOSE     103
define ERR.VACUUM        104
define ERR.TMP.TABLE     105
```

8.4.5 Job Programs

(1) "init" Job

```
*****

'      INIT JOB
*****

job name "init"
include "robot.hed"

'Globals
global   Rbt.Cmd%(2)
global   Get.Pt%(2)
global   Err.No%(2)
global   Init.End%

'Initialize variables
Init.End%=0
for i%=0 to ROB.NUM
    Rbt.Cmd%(i%)=&HFF
    Err.No%(i%)=0
    Get.Pt%(i%)=&HFF
next i%

'Initialize robot port
close #1
open "COM0" As #1 robtype=580 robnolist=RB1.NO,RB2.NO

'Initialize output
OUTB(O.SYS.ERR)=0
OUTB(O.RB1.CHK.CLOSE)=0
OUTB(O.RB2.VACUUM)=0
OUTB(O.RB2.BLOW)=0
```

```
'Get parts presence on robots
Get.Pt%(RB1.NO)=INB(I.RB1.PART)
Get.Pt%(RB2.NO)=INB(I.RB2.PART)
Init.End%=1
```

```
job "init" off
```

(2) "main" Job

```
*****
'   MAIN JOB
*****

job name "main"
include "robot.hed"

'Globals
global   Rbt.Cmd%(2)
global   Err.No%(2)
global   Get.Pt%(2)
global   Init.End%

'Initial
on error goto *system.error    'Register error handler
step.cnt%=0    'Clear step counter
delay 1    'Safety delay for Init.End%=0 in "init" job
wait Init.End%=1    'Wait for initialization

*Main.Loop
'Error recovery
if Err.No%(RB1.NO)<>0 or Err.No%(RB2.NO)<>0 then
    wait INB(I.ERR.RESET)=1    'Wait for request to reset error
    OUTB(O.SYS.ERR)=0    'Clear system error
    for i=RB1.NO to RB2.NO
        Err.No%(i)=0
        Rbt.Cmd%(i)=&HFF
    next i
    step.cnt%=0
    job "robot1" off
    job "robot2" off
    job "robot1" start
    job "robot2" start
    goto *Main.Loop
endif

'Request to start
if step.cnt%=0 then
    wait INB(I.START)=0, 3
    if timeout then goto *Main.Loop 'Not started
    step.cnt%=step.cnt%+1
    goto *Main.Loop
endif
```



```
select case step.cnt%
'Check parts presence on robots
case 1
    if INB(I.RB1.PART)=1 then
        Err.No%(RB1.NO)=ERR.PRESENCE
    endif
    if INB(I.RB2.PART)=1 then
        Err.No%(RB2.NO)=ERR.PRESENCE
    endif
'Move robot #1, #2 to waiting position
case 2
    set #1[rno:RB1.NO], speed=100
    set #1[rno:RB2.NO], speed=100
    Rbt.Cmd%(RB1.NO)=CMD.MOVE.WAIT
    Rbt.Cmd%(RB2.NO)=CMD.MOVE.WAIT
'Move robot #1 to get
case 3
    Rbt.Cmd%(RB1.NO)=CMD.MOVE.GET
'Close robot #1 chuck
case 4
    Rbt.Cmd%(RB1.NO)=CMD.CHK.CLOSE
'Check presence on robot #1
case 5
    if INB(I.RB1.PART)=0 then
        Err.No%(RB1.NO)=ERR.ABSENCE
    endif
'Move Robot #1 to put
case 6
    set #1[rno:RB1.NO], speed=50
    Rbt.Cmd%(RB1.NO)=CMD.MOVE.PUT
'Open robot #1 chuck
case 7
    Rbt.Cmd%(RB1.NO)=CMD.CHK.OPEN
'Check presence on temporary table
case 8
    if INB(I.TMP.PART)=0 then
        Err.No%(RB1.NO)=ERR.TMP.TABLE
    endif
'Move robot #1 to waiting position
'Move robot #2 to getting position
case 9
    set #1[rno:RB1.NO], speed=100
    Rbt.Cmd%(RB1.NO)=CMD.MOVE.WAIT
    Rbt.Cmd%(RB2.NO)=CMD.MOVE.GET
'Robot #2 gets part
case 10
    Rbt.Cmd%(RB2.NO)=CMD.PART.GET
'Move robot #2 to putting position
case 11
    set #1[rno:RB2.NO], speed=50
```

```

        Rbt.Cmd%(RB2.NO)=CMD.MOVE.PUT
'Robot #2 puts part
case 12
        Rbt.Cmd%(RB2.NO)=CMD.PART.PUT
'Move robot #2 to waiting position
case 13
        set #1[rno:RB2.NO], speed=100
        Rbt.Cmd%(RB2.NO)=CMD.MOVE.WAIT
'dummy
case else
        dummy%=0
end select

'Wait for procedure completion
wait (Rbt.Cmd%(RB1.NO)=&HFF and Rbt.Cmd%(RB2.NO)=&HFF) or
Err.No%(RB1.NO)<>0 or Err.No%(RB2.NO)<>0

'Next step
if Err.No%(RB1.NO)=0 and Err.No%(RB2.NO)=0 then
        if step.cnt% < 13 then
                step.cnt%=step.cnt%+1
        else
                step.cnt%=0
        endif
else
        OUTB(O.SYS.ERR)=1
endif
goto *Main.Loop

*system.error
        Err.No%(RB1.NO)=Err
        Err.No%(RB2.NO)=Err
        resume      *sys.err.resume      'Terminate error handler
*sys.err.resume
        goto *Main.Loop

```

(3) "robot1" Job

```

*****
'   ROBOT1 JOB
*****

job name "robot1"
include "robot.hed"

'Globals
global  Rbt.Cmd%(2)
global  Err.No%(2)
global  Get.Pt%(2)
global  Init.End%

'Initial

```

```

on error goto *sys.error    'Register error handler
delay 1    'Safety delay for Init.End%=0 in "init" job
wait Init.End%=1    'Wait for initialization
rbtno%=RB1.NO    'Robot number of this job
sys.error%=0    'Clear error code
err.flag%=0    'Clear error flag
rob.status%=0    'Clear robot status
setrobno(rbtno%) 'Set implicit robot number

*Loop
err.flag%=0
select case Rbt.Cmd%(rbtno%)
'Move to waiting position
case    CMD.MOVE.WAIT
    move #1, pm(RB1AD.WAIT)
    Get.Pt%(rbtno%)=0    'Out of collision area
'Move to get
case    CMD.MOVE.GET
    'Wait for part presence on station #1
    wait INB(I.ST1.PART)=1
    'Open chuck before z-axis down
    seq #1
        move #1, pm(RB1AD.GET)
        OUTB(O.RB1.CHK.CLOSE)=0    'Open chuck
        wait INB(I.RB1.CHK.OPEN)=1, 3    'Confirm chuck opened
        if timeout then
            disable #1
            err.flag%=ERR.CHK.OPEN
        else
            finish #1
            wait not robcheckstop(#1)    'Wait for completion of
positioning
        endif
    seqend #1
'Move to put
case    CMD.MOVE.PUT
    seq #1
        move #1, pm(RB1AD.PUT)
        'Confirm part absence and robot #2 is not near
        wait INB(I.TMP.PART)=0 and Get.Pt%(RB2.NO)=0
        finish #1
        'Set collision
        Get.Pt%(rbtno%)=1
*WORK.CHECK
    'Confirm part presence on robot #1 while moving
    if INB(I.RB1.PART)=0 then
        disable #1
        err.flag%=ERR.ABSENCE
    else
        'Check robot moving
        if not robcheckstop(#1) Then goto *WORK.CHECK

```

```
        endif
    seqend #1
'Open chuck
case    CMD.CHK.OPEN
    OUTB(O.RB1.CHK.CLOSE)=0
    wait INB(I.RB1.CHK.OPEN)=1, 3    'Confirm chuck opened
    if timeout then
        err.flag%=ERR.CHK.OPEN
    endif
'Close chuck
case    CMD.CHK.CLOSE
    OUTB(O.RB1.CHK.CLOSE)=1
    wait INB(I.RB1.CHK.CLOSE)=1, 3    'Confirm chuck closed
    if timeout then
        err.flag%=ERR.CHK.CLOSE
    endif
'Dummy
case else
    dummy%=0
end select

'Check error
if err.flag%<>0 then goto *error.routine

'End of procedure
Rbt.Cmd%(rbtno%)=&HFF

goto *Loop

'Error handler
*sys.error
    sys.err%=err    'Job error code
    robclearerr #1    'Clear robot error
    enable #1    'Enable robot moving
    resume *sys.err.resume 'Terminate error handler

*sys.err.resume
    select case sys.err%
    case 39
        err.flag%=2    'COM receiving error
    case 43
        err.flag%=3    'COM sending error
    case 80
        err.flag%=4    'COM Timeout
    case 81
        if (ref(#1,status8) and &H40) <> 0 then
            err.flag%=11    'Emergency stop
        else
            if (ref(#1,status8) and &H01 ) <> &H01 then
                err.flag%=10    'Online mode error
            else
```

```

        err.flag%=5      'Other robot error
    endif
endif
case 82
    err.flag%=6      'COM response error
case 83
    err.flag%=7      'Robot memory error
case else
    if sys.err% <= 14 then
        err.flag%=12 'STP system error
    else
        err.flag%=08 'STP application error
    endif
end select
*error.routine
    Err.No%(rbtno%)=err.flag% 'Set error code
*error.loop
    goto *error.loop 'Wait for job off

```

(4) "robot2" Job

```

*****
'    ROBOT2 JOB
*****
job name "robot2"
include "robot.hed"

'Globals
global    Rbt.Cmd%(2)
global    Err.no%(2)
global    Get.Pt%(2)
global    Init.End%

'Initial
on error goto *sys.error    'Register error handler
delay 1    'Safety delay for Init.End%=0 in "init" job
wait Init.End%=1    'Wait for initialization
rbtno%=RB2.NO    'Robot number of this job
sys.error%=0    'Clear error code
err.flag%=0    'Clear error flag
rob.status%=0    'Clear robot status
setrobno(rbtno%) 'Set implicit robot number

*Loop
    err.flag%=0
    select case Rbt.Cmd%(rbtno%)
    'Move to waiting position
    case    CMD.MOVE.WAIT
        move #1, pm(RB2AD.WAIT)
        Get.Pt%(rbtno%)=0    'Out of collision area
    'Move to getting position

```

```
case    CMD.MOVE.GET
    'Wait for part presence on table and collision safety
    wait INB(I.TMP.PART)=1 and Get.Pt%(RB1.NO)=0
    Get.Pt%(rbtno%)=1    'Set collision
    move #1, pm(RB2AD.GET)
'Move to putting position
case    CMD.MOVE.PUT
    'Wait for part absence on station #2
    wait INB(I.ST2.PART)=0
    'Move by no wait
    move #1, pm(RB2AD.PUT), NoWait
    'Confirm part presence on robot #2 while moving
*WORK.CHECK
    if INB(I.RB2.PART)=0 then
        disable #1
        err.flag%=ERR.ABSENCE
    else
        'Check robot moving
        if not robcheckstop(#1) Then goto *WORK.CHECK
    endif
    Get.Pt%(rbtno%)=0    'Set collision safety
'Get a part
case    CMD.PART.GET
    OUTB(O.RB2.VACUUM)=1    'Vacuum ON
    'Confirm vacuum enabled and part presence on robot #2
    wait INB(I.RB2.VACUUM)=1 and INB(I.RB2.PART)=1, 3
    if timeout then
        if INB(I.RB2.VACUUM)=0 then
            err.flag%=ERR.VACUUM
        endif
        if INB(I.RB2.PART)=0 then
            err.flag%=ERR.ABSENCE
        endif
    endif
'Put a part
case    CMD.PART.PUT
    OUTB(O.RB2.VACUUM)=0    'Vacuum OFF
    OUTB(O.RB2.BLOW)=1    'Blow ON
    delay 0.5
    OUTB(O.RB2.BLOW)=0    'Blow OFF
'Dummy
case else
    dummy%=0
end select

'Check error
if err.flag%<>0 then goto *error.routine

'End of procedure
Rbt.Cmd%(rbtno%)=&HFF
```

```
goto *Loop

'Error handler
*sys.error
    sys.err%=err      'Job error code
    robclearerr #1    'Clear robot error
    enable #1         'Enable robot moving
    resume *sys.err.resume 'Terminate error handler

*sys.err.resume
    select case sys.err%
    case 39
        err.flag%=2    'COM receiving error
    case 43
        err.flag%=3    'COM sending error
    case 80
        err.flag%=4    'COM Timeout
    case 81
        if (ref(#1,status8) and &H40) <> 0 then
            err.flag%=11 'Emergency stop
        else
            if (ref(#1,status8) and &H01) <> &H01 then
                err.flag%=10 'Online mode error
            else
                err.flag%=5 'Other robot error
            endif
        endif
    case 82
        err.flag%=6    'COM response error
    case 83
        err.flag%=7    'Robot memory error
    case else
        if sys.err% <= 14 then
            err.flag%=12 'STP system error
        else
            err.flag%=08 'STP application error
        endif
    end select
end select
*error.routine
    Err.No%(rbtno%)=err.flag% 'Set error code
*error.loop
    goto *error.loop 'Wait for job off
```

9. Commands

9.1 List of Commands

Kind	Usage	Description	Function
Pre-Processor	Definition	Define	Define the specified name as a constant.
	Macro	Macro	Define a format of macro call.
	Header file	Include	Read the specified header file.
Pseudo-instruction	Definition	Job Name	Define the entry of a job and job name.
Definable instruction	Definition	Dim	Define as array variable.
		DimNet	Define as network global variable
		Global	Define as global variable.
		DimPos	Define the number of position memory.
		Rem	Define the comment line.
	Flow control	GoTo	Jump to a specified line, then execute.
		GoSub	Call subroutine.
		Return	Terminate subroutine, then resume the former process.
		For - Next	Repeat the instruction between For and Next.
		If Then Else	Decide the condition of logical expression.
		Delay	Break temporarily the execution of job.
		Wait	Wait until conditions are satisfied.
		TimeOut	Get the result of timeout by Wait command.
		On GoTo	Jump one of specified step.
		On GoSub	Call one of specified subroutines.
		Select Case	Evaluate an expression and execute the processing block.
		InitGoSub	Initialize the subroutine-call stack.
Interrupt control instruction	Error control	On Error GoTo	Specify the destination at error.
		Resume	Terminate error process, then resume the former process.
		Err	Hold error code.
Control instruction	Job control	Job Start	Control job execution.
		Job On	
		Job Off	
	Robot control	GetPriority	Get the running priority of the current job.
		SetPriority	Set the running priority of the current job.
		Move	Move a robot to specified coordinates.
		Set	Set operating characteristic data of a robot.
		Ref	Deal data inside of a robot.
		Seq - SeqEnd	Set or terminate robot sequence mode.
		Finish	Complete MOVE in sequence mode.
		Hold	Specify or cancel the servo lock of the robot.
		Disable	Inhibit robot movement.
		Enable	Allow robot movement
		Calib	Execute automatic origin calibration.
		SetRobNo	Set a robot number for the current job.
		ClearRobNo	Clear the robot number for the current job.
		GetRobNo	Get the robot number for the current job.
		EnableOnlineErr	Enable robot ONLINE mode check.
		DisableOnlineErr	Disable robot ONLINE mode check.
		RobCheckBpZone	Check robot position within BP/ZONE.
		RobCheckCurPos	Check robot position nearby teaching data.
		RobCheckStop	Check robot stopped.
		RobClearErr	Clear robot errors.
		RobSetPosRange	Define allowable margin of position.
		Inching	Execute inching motion.
		AxesPara	Make axes parameter.

Kind	Usage	Description	Function
		PosRec	Make one robot position record.
		CollisionCheck	Enable or disable collision check between robots.
		RobWorldPos	Get current robot position in the world coordinates system.
		RobDistance	Get the distance between two robots.
		RobGetCurSpeed	Get the current robot speed.
		RobGetCurTorq	Get the current robot torque.
		RobGetCurAveTorq	Get the current effective torque of a robot.
		RobGetCurPos	Get the current encoder position of a robot.
		RobReadSvoPara	Read servo parameter of a robot.
		RobWriteSvoPara	Write servo parameter of a robot.
		RobReadSG	Read system generation data of a robot.
		RobWriteSG	Write system generation data of a robot.
	File control	Open	Open a communication file.
		Close	Close a file.
		Input\$	Read the specified length of the string from a file.
		Input #	Substitute data of a sequential file to a variable.
		Line Input #	Read one line from a sequential file.
		Print #	Output data to a file.
		Eof	Examine the termination code of a file.
		FreeFile	Get unused file number.
		RchkHrcs	Check a HRCS protocol frame received.
		ReadHrcs	Read a HRCS protocol frame.
		WriteHrcs	Write a HRCS protocol frame.
		EnableDSRCheck	Enable DSR signal check of RS232C.
		DisableDSRCheck	Disable DSR signal check of RS232C.
		EnableRTSAuto	Enable automatic RTS signal control of RS232C.
		DisableRTSAuto	Disable automatic RTS signal control of RS232C.
		ComFunction	Control RS232C signal.
		GetComStatus	Get signal status of RS232C.
	Pulse generation	Pulse	Generate pulse. (Substitute a value for the specified period.)
	Clock control	Time\$	Get or set the current system time.
		Date\$	Get or set the current system date.
Network instruction	Network communication	NetOpen	Open a network communication.
		NetClose	Close a network communication.
		NetRead	Read data from a network communication.
		NetWrite	Write data from a network communication.
Conversion instruction	Arithmetic function	Sin	Get sine.
		Cos	Get cosine.
		Tan	Get tangent.
		Atn	Get arctangent.
		Sgn	Get the sign of value.
		Abs	Get absolute value.
		Int	Remove decimals
		Fix	Remove decimals
		Log	Get natural logarithms.
		Exp	Get e raised to a power.
		Sqr	Get square root.
	Operator	Mod	Execute arithmetic division and get the remainder.
		Not	Execute negation.
		And	Execute logical multiplication.
		Or	Execute logical addition.
		Xor	Execute exclusive logical addition.
		Eqv	Execute logical equivalence.
		Imp	Execute logical implication.
	Arithmetic Constant	Pai	Get the value of pi.

Kind	Usage	Description	Function
	Character	Left\$	Pick out arbitrary length from the left of a string.
		Mid\$	Specify one part of a string.
		Right\$	Pick out arbitrary length from the right of a string.
		Space\$	Get a string with the arbitrary length blank.
		Chr\$	Get the character of specified character code.
		String\$	Get the character string connected one arbitrary character.
		Hex\$	Get the character string converted decimal into hexadecimal.
		Str\$	Convert numerical value into a string.
		Val	Convert the number of a character string into actual value.
		Asc	Get the character codes of characters.
		Len	Get the length of a string.
		InStr	Get the first position of the string in another string.
		ScanStr	Scan string data according to specified format. And get the value as parameter from string by operator in the format.
		PrintStr	Print string data according to specified format. And put the data string of specified parameter by operator in the format.
Initialization	Operation of position memory in STP	InitPos	Initialize position memory in STP.
Message	Print to STP console	ConsoleMsgOn	Enable to print message to STP console.
		ConsoleMsgOff	Disable to print message to STP console.
		ConsoleMsg	Print specified message to STP console.

9.2 How to Read Command Explanation

After the next section, all commands of HrBasic language are explained.
Each explanation has the following structure.

Comamnd-name

(Type)

Note) There are the following command types.

Statement	Command without return value. Declaration included.
Function	Command with return value.
Operator	Executes calculation of two values.

- Function

Function of the command is described.

- Format

This shows how to describe the command. Actual writing the code needs the following rules.

- 1) There is the case that the explanation uses two lines for the command, but actual programming has to be written in one line.
- 2) When typing in the commands, there are no difference between the uppercase and lowercase letters. However, in case of the character enclosed by double quotation marks (") except for the file name, distinguish between the uppercase and lowercase letters of alphabet.
- 3) When the space (␣) is specified, enter a blank as one character.
- 4) *Item of Italics* has to be specified by a user.
- 5) The items enclosed by square brackets "[]" except [rno:robot-number] are optional and can be omitted. When omitting the bracket, the default value (the value which has already set in HrBasic) or the value specified before is applied.
- 6) The symbols, except for "[]", parentheses "()", comma (","), semicolon (";"), minus symbol and equal symbol ("=") etc must be typed in the specified place.
- 7) The item which has ellipsis (...) can be repeated within the allowable length of one line. (255 characters at the maximum)

Example)

Constant [, *Constant*...] In this case 0, 10, 15

- Example

This shows a simple example for the usage of the command.

- Explanation

This explains the details of function, notice and usage of command.

9.3 Explanation of Each Command

Abs

(Function)

- Function
Gets the absolute value.
- Format
Abs(Numeric-expression)
- Argument and Return value

Parameter		Explanation
Argument	<i>Numeric-expression</i>	Value of a number
Return value		Absolute value
- Example
b% = Abs(-2) ‘ 2 is substituted for b%
- Explanation
The absolute value of *Numeric-expression* is returned.

And

(Operator)

- Function

Executes a logical multiplication of two numbers.

- Format

Numeric-expression#1 And *Numeric-expression #2*

- Arguments

Parameter	Explanation
<i>Numeric-expression#1</i>	A numeric expression.
<i>Numeric-expression#2</i>	A numeric expression.

- Example

a% = &H000F%

b% = &H0FFF%

c% = a% And b% ‘ &H000F% substituted for c%

- Explanation

- The following calculation is performed.

X	Y	X and Y
1	1	1
1	0	0
0	1	0
0	0	0

- See “6.4.3 Logical Operator”.

Asc

(Function)

- Function
Gets the ASCII character code.

- Format
Asc(String)

- Argument and Return value

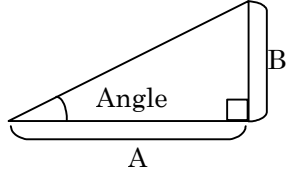
Parameter		Explanation
Argument	<i>String</i>	Character string
Return value		ASCII code (&H00 to &HFF)

- Example
a%=Asc("A") '65(&H41) for "A" is substituted for b%.
- Explanation
The ASCII code for the first character of *String* is returned.
- See also Chr\$.

Atn

(Function)

- Function
Gets the value of arctangent.
- Format
 $\text{Atn}(\text{Numeric-expression})$
- Argument and Return value

Parameter		Explanation
Argument	<i>Numeric-expression</i>	<p>Value of the ratio of B to A in the following right triangle.</p>  <p>$\text{Atn}\left(\frac{B}{A}\right) = \text{Angle}$</p>
Return value		Arctangent value by radian.

- Example
 $a! = \text{Atn}(y!/x!)$ 'Arctangent of $y!/x!$ is substituted for $a!$.
- Explanation
The Atn function returns the angle whose tangent is *Numeric-expression*. The range of the value is returned from $-\pi/2$ through $\pi/2$ in radians.
- See also Cos, Sin, Tan.

AxesPara

(Function)

● Function

Converts the specified parameters of each axis to the axes parameter in long integer.

● Format

AxesPara(*X-axis*, *Y-axis*, *Z-axis*, *W-axis*, *R-axis*, *C-axis*)

● Arguments and Return value

Parameter		Explanation
Arguments	<i>X-axis</i>	X axis parameter as a numeric expression. Valid number is 0 through 9.
	<i>Y-axis</i>	Y axis parameter as a numeric expression. Valid number is 0 through 9.
	<i>Z-axis</i>	Z axis parameter as a numeric expression. Valid number is 0 through 9.
	<i>W-axis</i>	W axis parameter as a numeric expression. Valid number is 0 through 9.
	<i>R-axis</i>	R axis parameter as a numeric expression. Valid number is 0 through 9.
	<i>C-axis</i>	C axis parameter as a numeric expression. Valid number is 0 through 9.
Return value		Axes parameter by long integer type

● Example

axes& = AxesPara(1, 2, 3, 4, 5, 6) 'Converted to 123456 (decimal).

● Explanation

- ◆ AxesPara function returns the long integer value calculated by the following formula.

$$\begin{aligned} \text{Axes parameter} = & \text{X axis parameter} * 100000 + \\ & \text{Y axis parameter} * 10000 + \\ & \text{Z axis parameter} * 1000 + \\ & \text{W axis parameter} * 100 + \\ & \text{R axis parameter} * 10 + \\ & \text{C axis parameter} \end{aligned}$$

- ◆ If each axis parameter is out of range, an error occurs at execution.
- ◆ If the return value is substituted for 16 bits variable with type declaration "%", there is a possibility that overflow error occurs.

● See also Inching.

Calib

(Statement)

● Function

Executes the automatic origin calibration (A-CAL) of robot.

● Format

Calib \square #*File-number*[[rno:*Robot-number*]][, *Axes-bits*][, NoWait]

Note) “AxesBits=” is supported by HNC-580 series and HAC-8XX controller.

● Arguments

Parameter	Explanation														
<i>File-number</i>	<p>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47.</p> <p>In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.</p>														
<i>Robot-number</i>	<p>A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.</p> <p>Calibrating robot axes can be specified by bits. Axes bit assignment is shown below.</p> <table><tr><td></td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td></td><td>C</td><td>R</td><td>W</td><td>Z</td><td>Y</td><td>X</td></tr></table>		5	4	3	2	1	0		C	R	W	Z	Y	X
	5	4	3	2	1	0									
	C	R	W	Z	Y	X									
<i>Axes-bits</i>	<p>Example)</p> <p>&H1B --- Calibrate X, Y, W and R axis.</p> <p>If omitted, all implemented axes are calibrated simultaneously. It can be specified as a number or variable.</p>														
NoWait	<p>If keyword "NoWait" is specified, the program goes to the next step, not waiting for the completion of moving. In this case, robot motion completed has to be confirmed by a program.</p>														

● Example

‘Calibrate all axes of robot #1.

Calib #1[rno:1]

‘Calibrate X axis of robot #2.

robno% = 2

Calib #1[rno:robno%], AxesBits=&H1

‘Calibrate X, Y axis of robot #3 without waiting to complete motion.

robno% = 3

axes% = &H3

Calib #1[rno:robno%], AxesBits=axes%, NoWait

● Explanation

- ◆ To move the robot to the position exactly, it is necessary that the memorized origin position in a robot controller equals the mechanical position of a robot. A-CAL automatically executes the

equalization of both origin positions to move a robot to the origin position.

- ◆ A robot with incremental encoder needs A-CAL when power on because the memorized origin position was cleared when power off.
- ◆ Generally, a robot with absolute encoder needs A-CAL only when installed because the memorized origin position always held without regard to power on or off.
- ◆ A-CAL can be executed by a teaching pendant connected with a robot controller.
- ◆ A program can check A-CAL completion to refer to A-CAL flag of robot status as follows. (See “4.2.8 STATUS”.)

< Checking A-CAL completion >

A-CAL flag is assigned in STATUS9.

STATUS9

7	6	5	4	3	2	1	0	Bit no.
&H80	&H40	&H20	&H10	&H8	&H4	&H2	&H1	Bit value

1: A-CAL completed
0: not completed

The following example program executes A-CAL if a robot has not been calibrated.

If (Ref(#1,STATUS9) and &H4) =0 Then Calib #1

- ◆ Without execution of A-CAL, the moving command such as Move statement results in a job error.



Note

As above example, “Ref(#1,STATUS9) and &H4” has to be enclosed by parentheses. Without parentheses, the program is compiled as “Ref(...) and (&H4=0)” and it does not work correctly.

Chr\$

(Function)

- Function
Gets the character string of the specified ASCII character code.

- Format
Chr\$(Numeric-expression)

- Argument and Return value

Parameter		Explanation
Argument	<i>Numeric-expression</i>	ASCII code of a character.
Return value		A string with the specified ASCII code.

- Example
a\$ = Chr\$(65) “A” with ASCII code 65 is substituted for a%
- Explanation
If the value of *Numeric-expression* is not from 0 through 255, a job error occurs.
- See also Asc.

ClearRobNo

(Function)

● Function

Clears a default robot number defined in the current job and the current job becomes state without a robot number.

● Format

ClearRobNo()

● Argument and Return value

Parameter	Explanation
Argument	Nothing
Return value	Nothing

● Example

Open "COM0" As #1 RobType=580 RobNoList=1,2,3 'for HNC-580

Open "COM1:9600,E,7,1" as #2 'for HNC-3XX/544

SetRobNo(2) 'Set default robot no. #2

Move #1,PTP,PM101 'Move COM0 robot #2 to PM101

ClearRobNo() 'Clear robot no.

Move #2,PTP,PM110 'Move COM1 robot to PM110

● Explanation

- ◆ SetRobNo function sets a default robot number for the current job.
- ◆ ClearRobNo clears a default robot number for the current job to set the value -1. Then a robot control without a robot number is enabled.
- ◆ After ClearRobNo function executed, GetRobNo function returns the value -1.
- ◆ See chapter 8 about usage of a robot number.
- ◆ Just after power on or program downloaded, all job starts without a robot number.

● See also GetRobNo, SetRobNo.

Close

(Statement)

- Function
Closes a file.
- Format
Close[\sqcup #*File-number*]

- Argument

Parameter	Explanation
<i>File-number</i>	A file number assigned by Open statement. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.

- Example
Close
Close #1
- Explanation
 - ◆ Close statement closes the file opened by Open statement. After a file is closed, the file cannot be accessed till it is opened.
 - ◆ If *File-number* is omitted, all opened files in the system are closed.
 - ◆ After a file is closed, the file number assigned for the closed file can be reused to open a file. And the closed file can be reopened using any file number.
- See also Open.

CollisionCheck

(Statement)

● Function

Enables or disables the collision check between the robots.

● Format

CollisionCheck □ On

CollisionCheck □ Off

● Argument

Parameter	Explanation
On	To specify “On”, the collision check is enabled.
Off	To specify “Off”, the collision check is disabled.

● Example

CollisionCheck On

CollisionCheck Off

● Explanation

- ◆ Now, only HAC-8XX supports the collision check. CollisionCheck statement can be executed on other controller or STP, but it returns to do nothing.
- ◆ After power on or program downloaded, the system starts to disable the collision check.
- ◆ The correct execution of the collision check needs to set the definition of the collision check to HAC. Refer to the document about the collision check.
- ◆ When the robot collision is detected, “Robot collision detected” error occurs in the job that has executed Move statement.

ComFunction

(Statement)

- Function
Controls a RS232C signal.
- Format
ComFunction \sqcup #*File-number*, *Signal-name*, *Value*

- Arguments

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Signal-name</i>	To specify "RTS", RTS signal of RS232C is controlled. To specify "DTR", DTR signal of RS232C is controlled. To specify the expression except above, a compiling error occurs.
<i>Value</i>	Variable or numeral constant can be specified. The value has to be 0 or 1.

- Example
‘Output 1 to RTS signal of COM1 port
fno% = 1
data% = 1
Open “COM1:38400,E,8,1” As #fno%
ComFunction #fno%, RTS, data%
- Explanation
 - ◆ RTS (Request To Send) signal is implemented at the 7th pin in D-SUB 9 pins. DTR (Data Terminal Ready) signal is implemented at the 4th pin in D-SUB 9 pins. These are the output signals.
 - ◆ In HAC, 1 is set to RTS and DTR signal when power on of HAC.
 - ◆ In WinSTP, 0 is set to RTS and DTR signal when WinSTP starts.
 - ◆ Generally, these signal need not to be controlled by a program. But, in case that a program needs to control the signals, ComFunction statement is useful for it.
- See also EnableRTSAuto, DisableRTSAuto, GetComStatus.

ConsoleMsg

(Function)

- **Function**
Prints a message to STP console.

- **Format**
ConsoleMsg(*String*)

- **Argument and Return value**

Parameter		Explanation
Argument	<i>String</i>	A character sting to print
Return value		Nothing

- **Example**

ConsoleMsgOn ‘ Enable to print to console.
 ConsoleMsg(“TEST”) ‘ Print “TEST” to console.
 ConsoleMsgOff ‘ Disable to print to console.

- **Explanation**

- ◆ ConsoleMsg function prints the specified string to STP console.
- ◆ Codes of carriage return and line feed are added to the specified string automatically.
- ◆ Console means the following equipment according to STP type.

STP type	Console
HAC-8XX	VGA monitor
WinSTP	Console window

**Note**

In HAC-8XX, console output takes several milliseconds to print ten characters and it stops the job execution. Therefore, it is recommended that console output is used for debugging and disabled when actual working.

- ◆ After STP system starts and a program is downloaded, console output is disabled automatically.
- ◆ ConsoleMsgOn statement enables to print to console and ConsoleMsgOff statement disables to print to it.
- ◆ During console output disabled, ConsoleMsg function returns immediately without output operation.
- See also ConsoleMsgOn, ConsoleMsgOff.

ConsoleMsgOff

(Statement)

● Function

Disables to print a message to STP console.

● Format

ConsoleMsgOff

● Argument

Parameter	Explanation
Argument	Nothing

● Example

ConsoleMsgOn ‘ Enable to print to console.
 ConsoleMsg(“TEST”) ‘ Print “TEST” to console.
 ConsoleMsgOff ‘ Disable to print to console.

● Explanation

- ◆ After ConsoleMsgOn statement is executed, ConsoleMsg function returns immediately without output operation.
- ◆ Console means the following equipment according to STP type.

STP type	Console
HAC-8XX	VGA monitor
WinSTP	Console window

- ◆ ConsoleMsgOn statement enables to print a message to STP console.

● See also ConsoleMsg, ConsoleMsgOn.

ConsoleMsgOn

(Statement)

● Function

Enables to print a message to STP console.

● 書式

ConsoleMsgOn

● Argument

Parameter	Explanation
Argument	Nothing

● Example

ConsoleMsgOn ‘ Enable to print to console.

ConsoleMsg(“TEST”) ‘ Print “TEST” to console.

ConsoleMsgOff ‘ Disable to print to console.

● Explanation

- ◆ After ConsoleMsgOn statement is executed, ConsoleMsg function prints a message to STP console.
- ◆ Console means the following equipment according to STP type.

STP type	Console
HAC-8XX	VGA monitor
WinSTP	Console window

- ◆ ConsoleMsgOff statement disables to print a message to STP console.

● See also ConsoleMsg, ConsoleMsgOff.

Cos

(Function)

- Function
Gets the value of cosine.

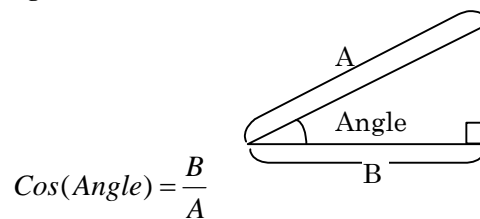
- Format
 $\text{Cos}(\text{Numeric-expression})$

- Argument and Return value

Parameter		Explanation
Argument	<i>Numeric-expression</i>	Angle by radian.
Return value		Cosine of the specified value is returned. The value is from -1.0 through +1.0.

- Example
 $c! = \text{Cos}(3.1415! / 2!)$ ‘Cosine of (3.1415! / 2!) is substituted for c!’

- Explanation
The ratio of “B” to “A” is returned specifying the angle “*Angle*” in the figure.



- See also Atn, Sin, Tan.

Date\$

(Statement)

● Function

Sets date to the system calendar.

● Format

Date\$ = *Date-string*

● Argument

Parameter	Explanation
Argument	Nothing

● Example

Date\$="03/03/10" ' Set 10.March 2004 to system calendar.

● Explanation

- ◆ This statement is used at the left side of substitution.
- ◆ The substituted string of a constant or variable has to be the following format.

"yy/mm/dd"

Year (00 - 99) ↑

Month (01 - 12) ↑

Day (01 - End) ↑

● See also Date\$ (function), Time\$.

Date\$

(Function)

- Function
Gets current date of the system calendar.

- Format
Date\$

- Argument and Return value

Parameter	Explanation
Argument	Nothing
Return value	A sting that contains current date with the following format. "yy/mm/dd"

- Example
b\$ = Date\$ ' Current date is substituted for b\$.

- Explanation
If the system time becomes "00:00:00", the system date changes to the next day.
The value of Date\$ is a string data, but it is not available in the string expression combined with string operators. For example, a\$=Date\$+b\$ is not available. In this case, a program has to be described as follows.

```
d$ = Date$
a$ = d$ +b$
```

- See also Date\$ (statement), Time\$.

Define

(Statement)

● Function

Defines the specified name as the specified constant.

The statement can be described only in a header file (suffix .hed).

The defined constant name can be used in a program after a program reads a include file by Include statement.

● Format

Define \hookrightarrow *Definition-name* \hookrightarrow *Constant*

● Arguments

Parameter	Explanation
<i>Definition-name</i>	Specify an arbitrary name within sixteen alphabetic or numeric characters or period (.).
<i>Constant</i>	A number or string by literal.

● Example

Define I.START 10 'Input#10: Start button

Define ML.COUNT 102 'ML(102): Running count

Define Z.UP 5.12! 'Z-axis up-motion length

Define ROB.COM "COM1:19200,E,7,1"

'Robot communication parameter

● Explanation



Guideline for
Programming

Definition-name does not care which case the character is. However, in general programming rules, it is recommended to describe all of *Definition-name* by upper case.

- ◆ A program can use *Definition-name* by means of reading a header file by Include statement that is described at the starting part of a job program where any executable sentence has not been described yet.
- ◆ *Definition-name* in a source program (.bas) is replaced with *Constant* during compilation. Both *Definition-name* and *Constant* are outputted to a list file (.lst) created after compilation.
- ◆ If a constant is frequently used in a program, the program modification is very easier to define the equivalent constant name by Define statement and to use it in a program, because only the Define sentence has to be modified and then the program has to be re-compiled it.



Guideline for
Programming

It is recommended that a constant, with a possibility of modification in the future, has to be defined by Define statement, even if the constant is used only one time in the program.

- ◆ If the definition name is spelled wrongly in a source program, a compiler treats it as a variable and then links. Ordinarily, this program cannot work well. In this case, the warning "Type declaration character is not found. A single precision real type is assumed for the variable." and "Value is not assigned to this variable." are displayed on compilation. To be careful to the message on compilation, check the program, remove all warnings, and then run the program.

- See also Include, “Chapter 3 Program Development Guideline”.

Delay

(Statement)

- **Function**
Breaks the program execution of the current job temporarily.
- **Format**
Delay \sqsubset *Numeric-expression*

- **Argument**

Parameter	Explanation
<i>Numeric-expression</i>	The time to break the job execution by seconds. Valid time is from 0.000 sec through 2147483.647 sec. Minus value cannot be specified.

- **Example**
Delay 1.5 ‘ Break this job for 1.5 sec.
- **Explanation**
 - ◆ The current job execution is broken for the specified period. After the specified time passes, the next step is executed.
 - ◆ Other job is not influenced for running by the execution of this statement.
 - ◆ During Delay breaking, if Job Off statement is executed, the timer for Delay is suspended. After job restarts by Job Off statement, the timer is restarted.

Dim

(Statement)

● Function

Declares an array variable and assign memory area to the variable.

● Format

Dim \sqsubset *Variable-name*(*Upper1* [, *Upper2* [, *Upper3*]])
 [, *Variable-name*(*Upper1* [, *Upper2* [, *Upper3*]]), ...]

● Argument

Parameter	Explanation
<i>Variable-Name</i>	Variable name to use as an array.
<i>Upper1</i>	A maximum subscript number of an array.
<i>Upper2</i>	A minimum subscript number is always zero. Therefore, number of array elements is <i>Upper</i> +1 for one dimension.
<i>Upper3</i>	The dimensions of array are available up to three.

● Example

Dim a!(12, 2), b\$(3), c%(1,2,4)

● Explanation

- ◆ Number of array elements or dimensions has the limitation of volume of the system memory.
- ◆ Greater subscript than the declared is specified in a program, a compiling error “Subscript out of range” or a job error “Array accessed out of range” occurs.



Note

Naming rule of variable is shown below.

- Variable name has to consist of alphabets or numerals.
- Length of variable name has to be 16 bytes or less including a type declaration character.
- Variable name cannot be the reserved nameⁱ, but a part of variable name can be the reserved name. It is not cared which case of alphabets variable name has.
- In case that the two variable names equal, if type declaration characters differ, the compiler distinguishes the two variables. Type declaration character is added to the end of variable name.
- If type declaration character is omitted, the compiler decides that the variable is single precision real-number type as if “!” is added.

- See “6.2.2 Array Variable”.

ⁱ Reserved name is keyword of HrBasic language, such as name of statement (e.g. Mid, If), name of function (e.g. Len, Abs), and operator (e.g. Or, Mod).

DimNet

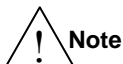
(Statement)

- Function
Declars a network global variable.
- Format
DimNet \sqsubset *Variable-name*[(*Upper1* [, *Upper2* [, *Upper3*]])]
[, *Variable-name*[(*Upper1* [, *Upper2* [, *Upper3*]])], ...]

- Argument

Parameter	Explanation
<i>Variable-Name</i>	Variable name to use as a network global variable.
<i>Upper1</i>	To declare a network global variable as array, specify a maximum subscript number of an array.
<i>Upper2</i>	A minimum subscript number is always zero.
<i>Upper3</i>	Therefore, number of array elements is <i>Upper</i> +1 for one dimension.
	The dimensions of array are available up to three.

- Example
DimNet ng.Name\$(10), ng.Port%(3,4), ng.Mode%
- Explanation
 - ◆ To use network global variable needs to create the network definition and download it to STP. Refer to HBDE operation manual or help about details.
 - ◆ A network global variable shares the value of a variable used in STPs connected in the network. If each program in STP declares the network global variable, all programs in the network can read or write it.



Note

Naming rule of variable is shown below.

- Variable name has to consist of alphabets or numerals.
- Length of variable name has to be 16 bytes or less including a type declaration character.
- Variable name cannot be the reserved nameⁱ, but a part of variable name can be the reserved name. It is not cared which case of alphabets variable name has.
- In case that the two variable names equal, if type declaration characters differ, the compiler distinguishes the two variables. Type declaration character is added to the end of variable name.
- If type declaration character is omitted, the compiler decides that the variable is single precision real-number type as if “!” is added.

- See “6.2.3 Local Variable, Global Variable and Network Global Variable”.

ⁱ Reserved name is keyword of HrBasic language, such as name of statement (e.g. Mid, If), name of function (e.g. Len, Abs), and operator (e.g. Or, Mod).

DimPos

(Statement)

● Function

Declares usage and size of P memory that the current job uses.

● Format

DimPos \sqcup *Number-of-Ps*

● Argument

Parameter	Explanation
<i>Number-of-Ps</i>	Number of P records that the current job uses. Invalid value is from 1 through 8000.

● Example

DimPos 1000

InitPos 0 to 999

● Explanation

- ◆ P memory is common for all jobs. But the number of P memory records is declared to use in each job. The above example declares that a job uses 1000 records of P memory from P0 through P999.
- ◆ Even if other job declares greater number of P memory, a job can used up to the number which a job has declared.
- ◆ Generally, P records hav to be accessed after InitPos statement has initialized them.

● See also InitPos, “4.2.5 P and Its Structure”.

Disable

(Statement)

- **Function**
Inhibits robot movement.
- **Format**
Disable \square #*File-number*[[rno:*Robot-number*]]

- **Arguments**

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.

- **Example**
Disable #1[rno:1]
- **Explanation**
 - ◆ Disable statement stops a robot immediately and inhibits it to move.
 - ◆ While a robot is moving by Move statement, Disable statement stops a robot motion. Executing Move statement is interrupted and returns immediately to go to the next step.
 - ◆ A program can confirm that a robot is disabled or not by reference to STATUS8. STOP flag becomes ON during disabled state.

7	6	5	4	3	2	1	0	Bit no.
&H80	&H40	&H20	&H10	&H8	&H4	&H2	&H1	Bit value
								1: ON-LINE mode
								1: MANUAL mode
								1: AUTO mode
								0: Reserved
								1: Sequence mode
								1: STOP ON / 0: STOP OFF
								1: ES (Emergency Stop)
								0: Reserved

- ◆ After a robot disabled by Disable statement, the execution of Enable statement has be required before a robot restarts to move.
- ◆ A job error occurs when a robot restarts to move without the execution of Enable statement.

DisableDSRCheck

(Statement)

● Function

Disables DSR signal check of RS232C.

● Format

DisableDSRCheck \sqcup *COM-number*

● Argument

Parameter	Explanation
<i>COM-number</i>	COM port number by numeric expression. Valid number is from 0 through 9.

● Example

DisableDSRCheck 2 ‘Disable DSP check of COM2

comno% = 3

DisableDSRCheck comno% ‘Disable DSP check of COM3

● Explanation

- ◆ If the specified COM is a RS232C port, DSR signal check of RS232C is disabled. If it is not RS232C, the statement returns without operation.
- ◆ After STP system restarts or a program is downloaded, DSR check is automatically enabled for all RS232C ports.
- ◆ DSR (Data Set Ready) signal (the 6th pin of D-SUB 9 pins) is the input signal for STP to check the disconnection of RS232C.
- ◆ If DSR check is enabled, STP checks DSR ON when data is transmitted to RS232C. If it is OFF, STP raises a job error “COM line not connected”.

● See also EnabledDSRCheck.

DisableOnlineErr

(Statement)

● Function

Disables robot ONLINE check. Even if ONLINE mode is off during robot motion, a job error does not occur.

● Format

DisableOnlineErr

● Argument

Parameter	Explanation
Argument	Nothing

● Example

DisableOnlineErr 'Disable ONLINE check

'Run without job error if ONLINE is off during robot motion.

robno% = 1 'Robot #1

Seq #1

Move #1[rno:robno%], PM1

Finish #1[rno:robno%]

*LOOP

'Check ONLINE

If (Ref(#1[rno: robno%], STATUS8) and &H1) = 0 Then

GoTo *ONLINE.ERR

EndIf

'Check positioning completion

If (Ref(#1[rno:robno%], STATUS9) and &H2) <> &H2 Then

GoTo *LOOP

EndIf

SeqEnd #1[rno:1]

EnableOnlineErr 'Enable ONLINE check

● Explanation

When STP system starts or a program is downloaded, the system starts to enable robot ONLINE check. In this state, a job error occurs if robot ONLINE becomes off during the execution of Move statement.

DisableOnlineErr disables robot ONLINE check during the execution of normal Move or sequence mode Move. A job error never occurs even if ONLINE mode is OFF. DisableOnlineErr is useful for the case such as a program needs to check ONLINE during the motion in sequence mode.

EnableOnlineErr statement enables robot ONLINE check.

DiableOnlineErr statement is effective to the current job.

● See also EnableOnlineErr.

DisableRTSAuto

(Statement)

● Function

Disables automatic RS232C RTS signal control.

● Format

DisableRTSAuto \sqcup *COM-number*

● 引数

Parameter	Explanation
<i>COM-number</i>	COM port number by numeric expression. Valid number is from 0 through 9.

● Example

‘ Disable automatic RTS signal control of COM2

DisableRTSAuto 2

‘ Disable automatic RTS signal control of COM3

comno% = 3

DisableRTSAuto comno%

● Explanation

- ◆ If the specified COM is a RS232C port, automatic RTS signal control is disabled. If it is not RS232C, the statement returns without operation.
- ◆ After STP system starts or a program is downloaded, automatic RTS signal control is disabled for all RS232C ports.
- ◆ RTS (Request To Send) signal (the 7th pin of D-SUB 9 pins) is the output signal for STP. About detail of automatic RTS signal control, refer to EnableRTSAuto statement.
- ◆ If DSR check is enabled, STP checks DSR ON when data is transmitted to RS232C. If it is OFF, STP raises a job error “COM line not connected”.

● See also EnableRTSAuto.

Enable

(Statement)

- **Function**
Allows robot movement.
- **Format**
Enable└─#*File-number*[[rno:*Robot-number*]]

- **Arguments**

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.

- **Example**
Enable #1[rno:1]
- **Explanation**
 - ◆ Enable statement enables a robot to move and allows robot movement.
 - ◆ If a robot has received Disable command, it never starts to move by a motion command. It is required that Enable command is transmitted to a robot before motion.
 - ◆ A program can confirm that a robot is enabled or not by reference to STATUS8. STOP flag becomes OFF during enabled state.

7	6	5	4	3	2	1	0	Bit no.
&H80	&H40	&H20	&H10	&H8	&H4	&H2	&H1	Bit value
								1: ON-LINE mode
								1: MANUAL mode
								1: AUTO mode
								0: Reserved
								1: Sequence mode
								1: STOP ON / 0: STOP OFF
								1: ES (Emergency Stop)
								0: Reserved

- See also Enable.

EnableDSRCheck

(Statement)

- **Function**
Enables DSR signal check of RS232C.

- **Format**
EnableDSRCheck □ *COM-number*

- **Argument**

Parameter	Explanation
<i>COM-number</i>	COM port number by numeric expression. Valid number is from 0 through 9.

- **Example**
EnableDSRCheck 2 ‘ Enable DSP check of COM2
comno% = 3
EnableDSRCheck comno% ‘ Enable DSP check of COM3

- **Explanation**
 - ◆ If the specified COM is a RS232C port, DSR signal check of RS232C is enabled. If it is not RS232C, the statement returns without operation.
 - ◆ After STP system restarts or a program is downloaded, DSR check is automatically enabled for all RS232C ports.
 - ◆ DSR (Data Set Ready) signal (6th pin of D-SUB 9 pins) is the input signal for STP to check the disconnection of RS232C.
 - ◆ If DSR check is enabled, STP checks DSR ON when data is transmitted to RS232C. If it is OFF, STP raises a job error “COM line not connected”.
- See also DisableDSRCheck.

EnableOnlineErr

(Statement)

- **Function**
Enables robot ONLINE check. If ONLINE mode is off during robot motion, a job error occurs.
- **Format**
EnableOnlineErr
- **Argument**

Parameter	Explanation
Argument	Nothing
- **Example**
 EnableOnlineErr ' Enable ONLINE check
 ' Job error occurs if robot ONLINE becomes off.
 Move #1, PTP, PM1
- **Explanation**
 EnableOnlineErr statement enables robot ONLINE check during the execution of normal Move or sequence mode Move. A job error occurs if robot ONLINE mode becomes OFF during robot motion.
 When STP system starts or a program is downloaded, the system starts to enable robot ONLINE check.
 DisableOnlineErr statement disables robot ONLINE check. After DisableOnlineErr is executed, EnableOnlineErr statement can enable ONLINE check.
 EnableOnlineErr statement is effective to the current job.
- See also DisableOnlineErr.

EnableRTSAuto

(Statement)

- Function
Enables automatic RS232C RTS signal control.

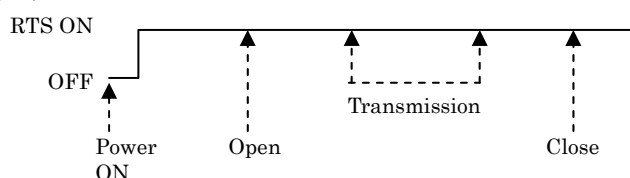
- Format
EnableRTSAuto \hookrightarrow *COM-number*, *Control-type*

- 引数

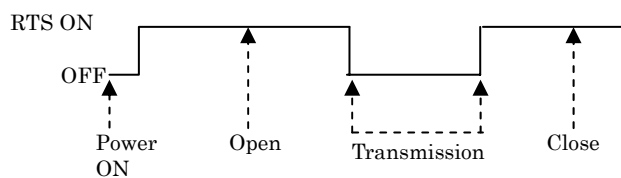
Parameter	Explanation
<i>COM-number</i>	COM port number by numeric expression. Valid number is from 0 through 9.
<i>Control-type</i>	RTS control type by numeric expression. The following numbers of types are supported now. 0: RTS signal ON during transmission 1: RTS signal OFF during transmission

- Example
‘COM2 RTS signal OFF during transmission
EnableRTSAuto 2, 1
‘COM3 RTS signal ON during transmission
comno% = 3
type% = 0
EnableRTSAuto comno%, type%

- Explanation
 - ◆ If the specified COM is a RS232C port, the specified type of automatic RTS signal control is enabled. If it is not RS232C, the statement returns without operation.
 - ◆ After STP system starts or a program is downloaded, automatic RTS signal control is disabled for all RS232C ports.
 - ◆ RTS (Request To Send) signal (the 7th pin of D-SUB 9 pins) is the output signal for STP, and it becomes ON after STP system starts. For an interface conversion from RS232C to RS485 or RS422, for example, there is a case to control the RTS signal to OFF during data transmission. For such case, EnableRTSAuto statement can be available.
 - ◆ RTS control type #0
RTS signal becomes automatically ON during data transmission. Even if the transmission is completed, the RTS signal continues to be ON.



- ◆ RTS control type #1
RTS signal becomes automatically OFF during data transmission. After the transmission is completed, the RTS signal is resumed to ON automatically.



- ◆ While automatic RTS control is disabled, RTS signal is controlled as the type #0.

- See also DisableRTSAuto.

Eof

(Function)

- Function
Examines the termination code of a file. This code is called “End of file”.

- Format
`Eof(File-number)`

- Argument and Return value

Parameter		Explanation
Argument	<i>File-number</i>	A file number assigned by Open statement. Valid range is from 0 through 47.
Return value		If end of file is detected, true value (-1) is returned. If not, false value (0) is returned.

- Example
`a$=Input$(1, #1)`
`If Eof(1) Then GoTo *ENDFILE`

- Explanation
 - ◆ If the specified file number indicates a communication port, Eof function returns the true value when the received buffer is empty or any data is received.
 - ◆ If the specified file number indicates a normal file, Eof function returns the true value when all data in the file is read and there is no data to read.

Eqv

(Operator)

- Function

Executes a logical equivalence of two numbers.

- Format

Numeric-expression#1 \sqcup Eqv \sqcup *Numeric-expression #2*

- Arguments

Parameter	Explanation
<i>Numeric-expression#1</i>	A numeric expression.
<i>Numeric-expression#2</i>	A numeric expression.

- Example

a% = &H000F%

b% = &H0FFF%

c% = a% Eqv b% ‘ &HF00F% substituted for c%.

- Explanation

- The following calculation is performed.

X	Y	X eqv Y
1	1	1
1	0	0
0	1	0
0	0	1

- See “6.4.3 Logical Operator”.

Err

(Function)

- **Function**
Gets the last job error code.

- **Format**
Err

- **Argument and Return value**

Parameter	Explanation
Argument	Nothing
Return value	Job error code

- **Example**
On Error GoTo *ERROR.ROUTINE
:
*ERROR.ROUTINE
 err.no%=Err
 If err.no%=7 Then GoTo *ERR7

- **Explanation**
 - ◆ Err function returns the last job error code which a job has memorized.
 - ◆ A job stops running at the step where an error occurs. But, if an error handler has been defined by On Error GoTo statement, a job jumps to the error handler and then executes it.
 - ◆ See “Appendix List of Job Error Code” about job error codes.
 - ◆ Resume statement or Job Start statement clears the last error memorized in a job.
- See also On Error GoTo, Resume.

Exp

(Function)

- Function
Returns the value specifying e (the base of natural logarithms - 2.718282...) raised to a power.
- Format
 $\text{Exp}(\text{Numeric-expression})$
- Argument and Return value

Parameter		Explanation
Argument	<i>Numeric-expression</i>	Power number by numeric expression.
Return value		Value of exponential function.
- Example
 $x\# = \text{Exp}(2.0)$ 'e to the 2.0 is substituted for x#
- Explanation
Exp function is the inverse function of Log function and is sometimes referred to as the antilogarithm.
- See also Log.

Finish

(Statement)

- Function
Completes robot motion in sequence mode to move z axis down.

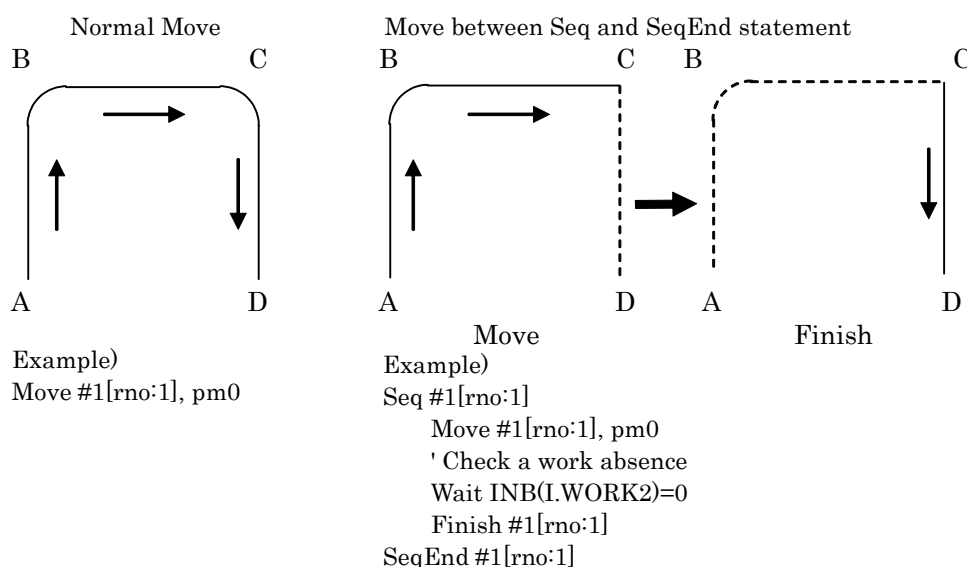
- Format
Finish \sqcup #*File-number*[[*rno:Robot-number*]]

- Arguments

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.

- Example
Finish #1[rno:1]

- Explanation
 - ◆ Between Seq and SeqEnd statement, robot motion by Move statement differs from the normal motion of Move statement. (See "Seq - SeqEnd statement".) As the following figure, when a robot moves to A-B-C-D, a robot stops at C position without down-motion of Z axis and waits for Finish command.



- ◆ Between Seq and SeqEnd statements, pair usage of Move statement and Finish statement is required. Even if z axis is not moved down, Finish command is necessary.
- See also Seq - SeqEnd.

Fix

(Function)

- Function
Removes the fractional part of number and returns the resulting integer value.
- Format
 $\text{Fix}(\text{Numeric-expression})$

- Argument and Return value

Parameter		Explanation
Argument	<i>Numeric-expression</i>	Any value by numeric expression.
Return value		Integer value.

- Example

$a\% = \text{Fix}(4.12)$ ' 4 is substituted for a%

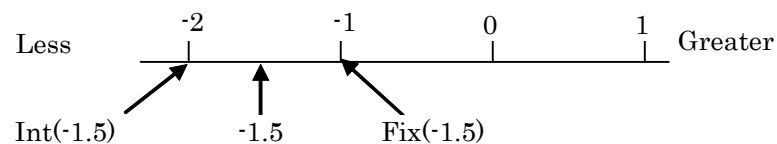
$b\% = \text{Fix}(-4.12)$ ' -4 is substituted for b%

- Explanation

If the specified value is positive, Fix function returns the integer value that is the same result of Int function.

If the specified value is negative, Fix removes the fractional part of number, and then returns the integer value that is greater than and nearest to the specified value.

On the other hand, Int function returns the integer value that is less than and nearest to the specified value.



- See also Int.

For...To...Step - Next

(Statement)

● Function

Repeats the execution of a program between For and Next statement.

● Format

For \sqcup *Variable* \Leftarrow *Start-number* \sqcup To \sqcup *End-number* [\sqcup Step \sqcup *Increment*]
:
Next [\sqcup *Variable*]

● Arguments

Parameter	Explanation
<i>Variable</i>	A variable for the condition of repetition.
<i>Start-number</i>	A number of a variable to start repetition.
<i>End-number</i>	A number of a variable to terminate repetition.
<i>Increment</i>	Increment number of a variable.

● Example

‘ i%=0, 2, 4,,96, 100 --- Repeats 51 times
For i%=0 to 100 Step 2
:
Next i%

● Explanation

If “Step” is omitted, *Increment* is regarded as +1 implicitly.

Negative number can be specified to *Increment*.

The conditions to terminate repetition are shown below. If the condition is satisfied, a program exits For-Next block to jump the next step of Next statement.

- 8) *Increment* is positive and value of *Variable* is greater than *End-number*.
- 9) *Increment* is negative and value of *Variable* is less than *End-number*.

A For - Next block can be described in a For - Next block. This structure is called nesting. There are the following notes for nesting.

- ◆ For statement and Next statement has to be used as a pair.
- ◆ Variables for repetition have to differ from each other.
- ◆ A For - Next block is completely included by another.
- ◆ Maximum number of nesting is 16.


**Guideline for
Programming**

- Generally, a variable for repetition should be 16 bits integer type (%) or 32 bits long integer type. In case of real type, there is a case that the infinite repetition of For - Next happens because the precision of real values is limited and then the condition to terminate is not satisfied.
- In coding rules of general programming languages including HrBasic, a repetition variable is usually named as i%, j%, k%... However, this need not apply if a variable name is necessary to be meaningful for programming.
Exaple)
 ‘ Standard programming
 For i%=1 To i.max%
 For j%=10 To j.max%
 For k%=0 To k.max%
 :
 Next k%
 Next j%
 Next i%
 ‘ A repetition variable is the equipment number
 For equip.no%=1 To 10
 If MB(equip.no%+100) = 1 Then
 :
 Next equip.no%

◆ See also “7.1.3 Iteration Structure”

FreeFile

(Function)

- Function
Gets a unused file number.

- Format
FreeFile()

- Argument and Return value

Parameter	Explanation
Argument	Nothing
Return value	The least unused file number from 0 through 47.

- Example
fno%=FreeFile()
Open "COM1:115200,N,8,1" As #fno% 'Open COM1
Move #fno%, PTP, PM1 'Move a robot to address #1
- Explanation
If all file numbers used already, a job error occurs.
- See also Open.

GetPriority

(Function)

- Function
Get the current priority of the specified job.

- Format
`GetPriority(Job-name)`

- Argument and return value

Parameter		Explanation
Argument	<i>Job-name</i>	String type expression of a job name.
Return value		Integer value of the job priority.

- Example
`p1%=GetPriority("robot1")`
- Explanation
 - ◆ GetPriority function returns the current priority of the specified job.
 - ◆ See SetPriority function about the meaning of the priority.
 - ◆ If the specified job is not found, job error occurs.
- See also SetPriority.

GetRobNo

(Function)

- **Function**
Gets a robot number for the robot communication of a current job.

- **Format**
GetRobNo()

- **Argument and Return value**

Parameter	Explanation
Argument	Nothing
Return value	A robot number of a current job from 1 through 999. After STP starts, a program is downloaded or ClearRobNo function is executed, GetRobNo function returns -1.

- **Example**
no%= GetRobNo ()
- **Explanation**
SetRobNo function sets a robot number of a current job. After a robot number is set, it is not necessary to specify a robot number to robot control commands such as Move, Calib or Seq and so on.
GetRobNo function returns a robot number set by SetRobNo function.
- See also SetRobNo, ClearRobNo.

Global

(Statement)

● Function

Declares global variables.

● Format

Global \sqcup *Variable-name* [, *Variable-name* [, ...]] \sqcup *Array* [, *Array* [, ...]]*Array*: *Variable-name* (*Upper1* [, *Upper2* [, *Upper3*]])

● Arguments

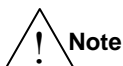
Parameter	Explanation
<i>Variable-name</i>	Variable name to use as a global variable.
<i>Upper1</i> <i>Upper2</i> <i>Upper3</i>	A maximum subscript number of an array. A minimum subscript number is always zero. Therefore, number of array elements is <i>Upper</i> +1 for one dimension. The dimensions of array are available up to three.

● Example

Global g.Mode%, g.Name\$, g.Table%(10, 20)

● Explanation

- ◆ A global variable in a job shares the memory area with other jobs that declare the same global variable name. The jobs that declare a global variable can read or write it at any time.
- ◆ If an array variable is used as global, the array does not need to be declared by Dim statement, but has to be declared by Global statement.
Global g.Array%(10, 20, 30)
- ◆ When a global variable is declared in some jobs, if the same name variable is not declared as global in a job, the variable is treated as local.

**Note**

Naming rule of variable is shown below.

- Variable name has to consist of alphabets or numerals.
- Length of variable name has to be 16 bytes or less including a type declaration character.
- Variable name cannot be the reserved nameⁱ, but a part of variable name can be the reserved name. It is not cared which case of alphabets variable name has.
- In case that the two variable names equal, if type declaration characters differ, the compiler distinguishes the two variables. Type declaration character is added to the end of variable name.
- If type declaration character is omitted, the compiler decides that the variable is single precision real-number type as if “!” is added.

- See “6.2.3 Local Variable, Global Variable and Network Global Variable”.

ⁱ Reserved name is keyword of HrBasic language, such as name of statement (e.g. Mid, If), name of function (e.g. Len, Abs), and operator (e.g. Or, Mod).

GoSub

(Statement)

- Function
Calls a subroutine.

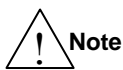
- Format
`GoSub \sqcup Label`

- Argument

Parameter	Explanation
<i>Label</i>	A label of a subroutine to call.

- Example
`GoSub *SUB1`

- Explanation
 - ◆ A subroutine is a block of program that starts at an entry label and exits by Return statement.
 - ◆ GoSub statement calls a subroutine to jump to the specified label. It cannot call a subroutine programmed in another job.
 - ◆ Subroutine program can call another subroutine. Maximum number of subroutine-call nests is 16. If the nests exceed the maximum number, a job error occurs.



Note

There are the following rules for using a label.

- The top of label name has to be an asterisk “*”.
- Except asterisk, the first character of label name has to be alphabetic.
- Except asterisk, available characters in label name are alphabetic, numerical or period “.”, regardless of upper or lower case.
- Label name after asterisk cannot be the reserved name (e.g. *MOVE). But, a part of label name after asterisk can be the reserved name (e.g. *MOVE.LOOP).
- The length of label name is maximum 16 characters except asterisk.
- Label name definition has to be written at the top of one line.

- See also Return, “7.2 Subroutine as Program Module”.

GoTo

(Statement)

- Function
Jumps to the specified label.

- Format
`GoTo \sqcup Label`

- Argument

Parameter	Explanation
<i>Label</i>	A label to jump.

- Example

```
*MAIN.LOOP
:
:
GoTo *MAIN.LOOP
```

- Explanation
 - ◆ GoTo statement jumps to the specified label unconditionally. It cannot jump to a label in another job.
 - ◆ In structured programming, GoTo statement is not used generally. But some case of repetition structure has to use GoTo statement. And in some case, a program is difficult to understand without GoTo statement. See “7.1.4 Usage of GoTo Statement” about this.



Note

There are the following rules for using a label.

- The top of label name has to be an asterisk “*”.
- Except asterisk, the first character of label name has to be alphabetic.
- Except asterisk, available characters in label name are alphabetic, numerical or period “.”, regardless of upper or lower case.
- Label name after asterisk cannot be the reserved name (e.g. *MOVE). But, a part of label name after asterisk can be the reserved name (e.g. *MOVE.LOOP).
- The length of label name is maximum 16 characters except asterisk.
- Label name definition has to be written at the top of one line.

- See “7.1.4 Usage of GoTo Statement”.

Hex\$

(Function)

- Function
Gets a string by hexadecimal expression converted from decimal value.

- Format
Hex\$(*Numeric-expression*)

- Argument and Return value

Parameter		Explanation
Argument	<i>Numeric-expression</i>	Value of a number
Return value		A string of the specified value by hexadecimal expression

- Example
a\$=Hex\$(30) “1E” is substituted for a\$.

- Explanation
Integer (16-bits, 32-bits) or real type expression can be specified to *Numeric-expression*.

There are two types of the returned string according to the specified value type, 16-bits integer (%) or 32-bits integer (&). In case of real type, the fractional part of the value is removed and the value is converted to long integer type.

- 1) 16-bits integer type

Numeric-expression has the value from -32768 through 32767 and the returned string has maximum four characters.

Relation between *Numeric-expression* and the returned string is shown below.

Value of <i>Numeric-expression</i>			Returned string		
-32768	...	-1	8000	...	FFFF
0	...	32767	0	...	7FFF

- 2) 32 bits long integer or real type

Numeric-expression has the value from -2147483648 through 2147483647 and the returned string has maximum eight characters. In case of real type, the fractional part of the value is removed.

Relation between *Numeric-expression* and the returned string is shown below.

Value of <i>Numeric-expression</i>			Returned string		
-2147483648	...	-32769	80000000	...	FFFF7FFF
-32768	...	-1	FFFF8000	...	FFFFFFFF
0	...	32767	0	...	7FFF
32768	...	65535	8000	...	FFFF
65536	...	2147483647	10000	...	7FFFFFFF

- See also Str\$, Val.

Hold On / Off

(Statement)

● Function

Specifies whether the robot holds (servo-locks) the position after the completion of positioning.

● Format

Hold On \sqcup #*File-number*[[rno:*Robot-number*]] [, *Axis*][, *Axis*]...

Hold Off \sqcup #*File-number*[[rno:*Robot-number*]] [, *Axis*][, *Axis*]...

● Arguments

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.
<i>Axis</i>	HNC-580 series or HAC-8XX controller does not support this parameter. Specify the following axis number to servo-lock or servo-unlock. X-axis:1, Y-axis:2, Z-axis:3, W-axis:4, R-axis:5, C-axis:6 If any axis is not specified, all implemented axes are controlled.

● Example

Hold On #1[rno:2] ' Servo-lock all axes

Delay 3.0 ' Timer for completion

Hold Off #1[rno:2] ' Servo-unlock all axes

Delay 3.0 ' Timer for completion

' Job error occurs for HNC-580 series or HAC-8XX

' when the following runs.

Hold On #2, 1, 2, 4 ' Servo-lock X, Y, W axis

Delay 3.0 ' Timer for completion

Hold Off #2, 1 ' Servo-unlock X axis

Delay 3.0 ' Timer for completion

● Explanation

To servo-lock or servo-unlock takes about one second or two seconds in the controller. About 3 seconds Delay is necessary as the previous example.

If ... Then - Else - EndIf

(Statement)

● Function

Branches into a procedure according to a condition.

● Format

- 1) If \neg *Condition* \neg Then \neg *Sentence#1* [\neg Else \neg *Sentence#2*]
- 2) If \neg *Condition* \neg Then
 Block#1
 [Else]
 [*Block#2*]
 EndIf

● Arguments

Format 1)

Parameter	Explanation
<i>Condition</i>	A condition to branch. The condition has true value (-1) or false value (0) as the result.
<i>Sentence#1</i>	A sentence to execute when <i>Condition</i> is true. Multi-statement is not available.
<i>Sentence#2</i>	A sentence to execute when <i>Condition</i> is false. Multi-statement is not available.

Format 2)

Parameter	Explanation
<i>Condition</i>	A condition to branch. The condition has true value (-1) or false value (0) as the result.
<i>Block#1</i>	A sentence or a program block to execute when <i>Condition</i> is true. Multi-statement is available.
<i>Block#2</i>	A sentence or a program block to execute when <i>Condition</i> is false. Multi-statement is available.

● Example

- 1) If a\$="y" Then GoSub *YES.SUB Else GoSub *NO.SUB
- 2) If TIM(5) Then ' If TIM(5) timeout,
 a!=b! + c! ' Execute a!=b! + c!
 Else ' If not so,
 GoTo *EXIT ' Jump to *EXIT.
 EndIf

● Explanation

- ◆ If-Then-Else-EndIf statement is used for two-branch structure.
- ◆ For structured programming, Format 2) is recommended.
- ◆ Maximum number of If-Then-Else-EndIf nests is 16.

● See "7.1.2 Selection Structure".

Imp

(Operator)

- Function

Executes a logical implication of two numbers.

- Format

Numeric-expression#1 \sqcup Imp \sqcup *Numeric-expression #2*

- Arguments

Parameter	Explanation
<i>Numeric-expression#1</i>	A numeric expression.
<i>Numeric-expression#2</i>	A numeric expression.

- Example

a% = &H00FF%

b% = &H0F0F%

c% = a% Imp b% ‘ &HFF0F% is substituted for c%.

- Explanation

- The following calculation is performed.

X	Y	X imp Y
1	1	1
1	0	0
0	1	1
0	0	1

- See “6.4.3 Logical Operator”.

Inching

(Statement)

● Function

Moves a robot by inching. Inching means 0.5-second jog-motion with low speed.

Note) Inching function returns immediately after a robot starts to move.
If the next Inching function is executed within 0.5 second, the robot continues to move by inching.

● Format

(Linear motion)

Inching ▯ Linear ▯ #*File-number*[[rno:*Robot-number*]], *Axes*, *Speed*

(Rotary motion)

Inching ▯ Rotary ▯ #*File-number*[[rno:*Robot-number*]], *Axes*, *Speed*

● 引数

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.
<i>Axes</i>	Specify axes and direction of inching motion by a constant or a variable that contains the decimal value with 6 digits. Multiple axes can be specified at once. Each digit indicates X, Y, Z, W, R and C axis from left to right. One digit represents the following actions. 0: Stop axis or unused axis. 1: Plus direction inching 2: Minus direction inching Example#1) 120001 --- X axis: plus, Y axis: minus, C axis: plus Example#2) 002200 --- Z axis: minus, W axis: minus
<i>Speed</i>	A constant or variable of number specifying inching speed 0: Low speed 1: High speed

● Example

1) Linear

Inching Linear #1[rno:1], 120001, 1 'Linear, +X, -Y, +C, High speed

2) Rotary

axes& = 002200

Inching Rotary #1[rno:1], axes&, 0 'Rotary, -Z, -W, Low speed

● Explanation

- ◆ Inching function inches the specified axis in the specified direction with the specified speed for 0.5 second.

- ◆ If a variable is used for *Axes*, a long integer type (type declaration character &) is necessary for the variable. If 16-bits integer type (type declaration character %) is used, A job error, overflow, may occur because 6-digits decimal value cannot be contained in the variable.
- ◆ If the value except 0, 1, 2 is specified for each axis of *Axes* parameter, an job error occurs.
- ◆ If the value except 0, 1 is specified for *Speed* parameter, an job error occurs.
- ◆ For *Axes* parameter, the following example shows how to convert the each axis setting into one long integer variable.

‘ Set each axis parameter

x.axis& = 1 '+direction of X-axis

y.axis& = 2 '-direction of Y-axis

z.axis& = 0 'Stop Z-axis

w.axis& = 0 'Stop W-axis

r.axis& = 2 '-direction of R-axis

c.axis& = 0 'Stop C-axis

‘ Covert into axes parameter

axes& = x.axis& * 100000 + y.axis& * 10000 + z.axis& * 1000
 + w.axis& * 100 + r.axis& * 10 + c.axis&

‘ Or uses AxesPara function.

axes& = AxesPara(x.axis&, y.axis&, z.axis&, w.axis&, r.axis&
 , c.axis&)

‘ Inching

Inching Linear #1[rno:1], axes&, 0

- ◆ Specify zero value to *Axes* parameter to stop inching motion of all axes.

Inching Linear #1[rno:1], 0, 1 'Stop all axes

- ◆ The following example stops the inching motion completely, after the 0.5-second inching is executed only one time.

Inching Linear #1[rno:1], 111111, 1 '+direction of all axes

Delay 0.5

Inching Linear #1[rno:1], 0, 1 'Stop all axes

- ◆ Repeat the execution of Inching function to continue inching for more than 0.5 second. The following example continues inching during INB#1 ON.

Wait INB(1) = 1 'Wait for INB#1 ON

*LOOP

 If INB(1) = 0 Then

 Inching Linear #1[rno:1], 0, 1 'Stop all axes

 Return 'Exit

 EndIf

 Inching Linear #1[rno:1], 111111, 1 '+direction of all axes

 GoTo *LOOP

- See also AxesPara.

Include

(Statement)

- **Function**
Reads the specified header file (suffix .hed) into a source program file (suffix .bas).
- **Format**
Include □ *Header-filename*
- **Argument**

Parameter	Explanation
<i>Header-filename</i>	A header file name to read.
- **Example**
Include "ml.hed"
Include "io.hed"
- **Explanation**
 - ◆ The specified header file has to be located at the directory defined in "Setup" -> "Directories" -> "Header files" in HBDE.
 - ◆ Reading a header file is executed during compilation.
 - ◆ When reading, Define statement described in the header file is analyzed and the definition name is registered on compilation.
 - ◆ Include statement has to be described before the defined name is used in a source program. Generally, Include statement is described at the starting part of a job program before an executable sentence is described.



**Guideline for
Programming**

It is recommended that Include statement is programmed after Job Name statement.

Example)

Job Name "robot"

Include "io.hed"

- See also Define, "Chapter 3 Program Development Guideline".

InitGoSub

(Statement)

- **Function**
Initialize the subroutine-call stack.
- **Format**
InitGoSub
- **Argument**
Nothing
- **Example**

```

On Error GoTo *ERR.HANDLER
*MAIN
:
*ERR.HANDLER
:
  InitGoSub
  Resume *MAIN

```
- **Explanation**
 - ◆ The subroutine-call stack is the management data of the return address of subroutines and the subroutine-call state, which is added by GoSub statement and removed by Return statement.
 - ◆ InitGoSub statement makes the initial state of the subroutine-call, which indicates a subroutine has not been called, the same after STP system starts or a program is downloaded.
 - ◆ A program may jump to an error handler from the middle of subroutine program without Return execution in case that the error handle has been defined by On Error GoTo statement. After that, the correspondence of the subroutine-call stack may not be satisfied if a program exits the error handler and then jumps to resume a main program by Resume statement or GoTo statement. In such case, InitGoSub statement is useful for resuming a main program to initialize the subroutine-call stack as the above example.
 - ◆ As the above-mentioned explanation, after InitGoSub execution, a program has to jumps into the head part of a job program where any subroutine has not been called.

InitPos

(Statement)

● Function

Initializes position memory in STP.

● Format

InitPos \leftarrow *Start-index* \leftarrow to \leftarrow *End-index*

● Arguments

Parameter	Explanation
<i>Start-index</i>	Start index number of STP position memory to initialize. Valid range is from 0 through 7999.
<i>End-index</i>	End index number of STP position memory to initialize. Valid range is from 0 through 7999.

● Example

DimPos 1000

InitPos 0 to 999 ‘ Initialize P(0) through P(999)

● Explanation

- ◆ InitPos statement initializes STP position memory with the specified range.
- ◆ Members of a position record are initialized by the following values.

Member	Expression	Value
X-axis data	PXn	0.0
Y-axis data	PYn	0.0
Z-axis data	PZn	0.0
W-axis data	PWn	0.0
R-axis data	PRn	0.0
C-axis data	PCn	0.0
Arm	PARMn	0
M data	PDMn	255
F code	PDFn	0
S code	PDSn	0
Coordinate type	—	0

- ◆ If position memory to initialize is out of the range declared by DimPos statement, a compiling error or a job error occurs.
- See also DimPos, “4.2.5 P and Its Structure”.

Input

(Statement)

- **Function**
Reads data from a file and put the value to the specified variables.

- **Format**
Input \sqsubset #*File-number*, *Variable* [, *Variable*...]

- **Arguments**

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Variable</i>	A variable to set the value of data read from a file.

- **Example**
Input #1, a!, b\$

- **Explanation**

- ◆ Input# statement reads data from a file and the value of data is set into the specified variable. If the data type does not match the variable type, a job error occurs.
- ◆ In case of reading multiple data, a delimiter varies according to reading data type.

Data type	Delimiter	
	Name	ASCII code
Number	Space (SP)	32 (&H20)
	Comma (,)	44 (&H2C)
	Carriage return (CR)	13 (&H0D)
Character string	Comma (,)	44 (&H2C)
	Carriage return (CR)	13 (&H0D)

- ◆ A Linefeed (LF, ASCII code 10) after a carriage return is disregarded.
- See also Input\$.

Input\$

(Function)

- Function
Reads data from a file with the specified length.
- Format
Input\$(*Length*, #*File-number*)

- Arguments and Return value

Parameter		Explanation
Arguments	<i>Length</i>	Length to read Valid range is from 1 through 255.
	<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
Return value		A character string read from a file

- Example

```

mozi$=""
*LOOP
a$=Input$(1, #1) ' Read 1 byte from a file
If Eof(1) Then ' End of file
    GoTo *EXIT
Else
    mozi$=mozi$+a$ ' Add reading data
    GoTo *LOOP
EndIf
*EXIT

```
- Explanation
 - ◆ Input\$ function does not return until the data with the specified length is put into a file. If there is already the data with the specified length in a file, it returns immediately after reading the data.
 - ◆ Input\$ function reads any byte-code including space, comma, carriage return, linefeed and control codesⁱ.
- See also Input #, Line Input #.

ⁱ Control code: Byte-code to control a display or peripheral equipment. This code is not printable and not able to display.

InStr

(Function)

- **Function**
Searches the specified string and returns the found position.

- **Format**
InStr([*Start*,] *String#1*, *String#2*)

- **Arguments and Return value**

Parameter		Explanation
Arguments	<i>Start</i>	Start position of <i>String#1</i> to search by integer value. Valid range is from 1 through the length of <i>String#1</i> . If omitted, the parameter is regarded as 1.
	<i>String#1</i>	A string to search. (The function searches this string for <i>String#2</i> .)
	<i>String#2</i>	A string to search for. (The function searches <i>String#1</i> for this string.)
Return value		If <i>String#2</i> is found in <i>String#1</i> , the found position at the top of <i>String#2</i> in <i>String#1</i> is returned. If not found, zero value is returned.

- **Example**
a\$="Hirata Corporation"
b%=InStr(a\$, "ra") ' 3 is substituted for b%.
c%=InStr(7, a\$, "ra") ' 13 is substituted for c%.
- **Explanation**
 - ◆ Searching is executed by the comparison of each byte.
 - ◆ If *String#2* is a null string, InStr function returns the value of *Start*.

Int

(Function)

- **Function**
Removes the fractional part of number and returns the resulting integer value.

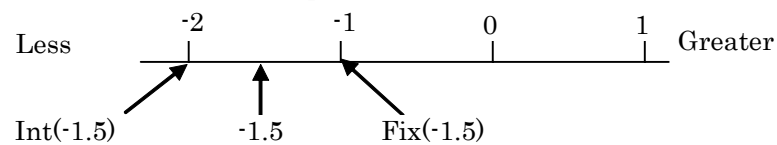
- **Format**
`Int(Numeric-expression)`

- **Argument and Return value**

Parameter		Explanation
Argument	<i>Numeric-expression</i>	Any value by numeric expression.
Return value		Integer value.

- **Example**
`a%=Int(4.12)` ' 4 is substituted for a%
`b%=Int(-4.12)` ' -5 is substituted for b%

- **Explanation**
 If the specified value is positive, Int returns the integer value that is the same result of Fix function.
 If the specified value is negative, Int function returns the integer value that is less than and nearest to the specified value.
 On the other hand, Fix function returns the integer value that is greater than and nearest to the specified value.



- See also Fix.

Job Name

(Statement)

- Function
Defines the entry of a job and declares job name.

- Format
`Job Name "Job-name"`

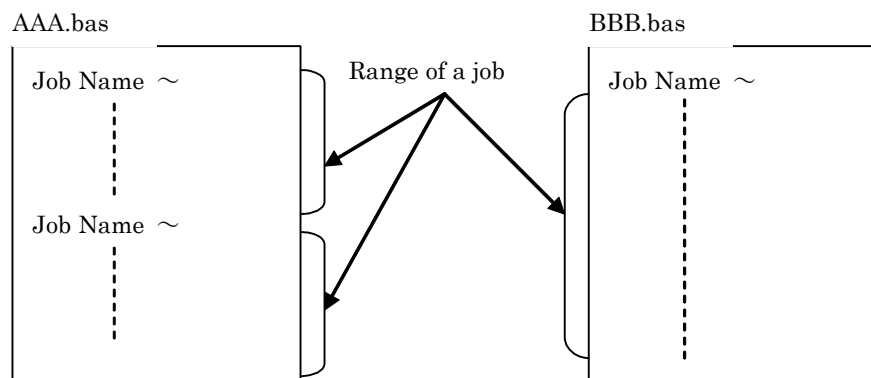
- Argument

Parameter	Explanation
<i>Job-name</i>	Job name of a program.

- Example
Job Name "Init"

- Explanation

The range of one job program is coded program steps from the "Job Name" statement to the next one or to the end step of a source file. Therefore, both only one job and two or over jobs can be coded in a source file.



Job name has the following rules.

- ◆ The length of a job name has to be 16 bytes or less.
- ◆ The first character of a job name has to be alphabetic or numeral.
- ◆ Job name may contain alphabet, numeral, period (.), hyphen (-), and underline (_).
- ◆ Job name has to be enclosed by double quotation marks.
- ◆ Job name cannot be the reserved nameⁱ, but a part of job name can be the reserved name. It is not cared which case of alphabets job name has.

The job specified in Job Start, Job On, Job Off statement has to be defined by Job Name statement, and it is required that the job has been linked into the downloaded program.

- See also Job Off, Job On, Job Start.

ⁱ Reserved name is keyword of HrBasic language, such as name of statement (e.g. Mid, If), name of function (e.g. Len, Abs), and operator (e.g. Or, Mod).

Job Off

(Statement)

- Function
Stops the execution of a job.

- Format
Job \sqsubset "*Job-name*" \sqsubset Off

- Argument

Parameter	Explanation
<i>Job-name</i>	Job name to stop.

- Example
Job "Init" Off

- Explanation

Job Off statement stops the execution of the specified job.

Job Off statement can stop not only the other job but also the current job. However, the stopped job has to be started or restarted by the other job using Job Start or Job On statement.

The job environment of execution, such as local variables or the executing step, is kept after the job stopped. Job On statement restarts the stopped job at the next step of stopped program.

When a program to stop is executing Move statement, Job Off statement stops a robot motion immediately and then stops a program execution. Job On statement restarts the job at the next step of Move. In the following example, if Job Off stops the job before the completion to move to PM(100), a robot stops immediately before a robot reaches PM(100). And then, if Job On statement restarts the job, a robot restarts to move to PM(200).

Example)

Move #1[rno:1], PM(100) \leftarrow If Job Off, robot stops immediately.

Move #1[rno:1], PM(200) \leftarrow If Job On, robot moves to PM(200).

- See also Job Name, Job On, Job Start.

Job On

(Statement)

- Function
Restarts a job execution.

- Format
Job \sqsubset "*Job-name*" \sqsubset On

- Argument

Parameter	Explanation
<i>Job-name</i>	Job name to restart.

- Example
Job "Robot1" On

- Explanation

Job On statement restarts a job stopped by Job Off statement.

All jobs starts automatically after STP system starts or a program is downloaded. The started job is stopped by the following operations.

- 3) Job Off statement by its own job
- 4) Job Off statement by other job

The stopped job can be restarted by other job. Job On statement restarts the stopped job at the next step of a stopped program.

In the following example, Buzzer job restarts at the step "*LOOP" after Main job executes Job On statement.

Job Name "Buzzer" Include "test.hed" *POWER.ON OUTB(O.BUZZER)=0 'Buzzer OFF Job "Buzzer" Off 'Job Off by myself 'After Job On by Main job, 'repeat the following procedure. *LOOP OUTB(O.BUZZER)=1 Delay 0.2 OUTB(O.BUZZER)=0 Delay 0.2 GoTo *LOOP	Job Name "Main" Include "test.hed" *MAIN.LOOP Wait INB(I.BUZZER)=1 Job "Buzzer" On 'Restart Buzzer job *RED.BLINK.LOOP 'Blink lamp Delay 1 : OUTB(O.RED)=1 Delay 1 : OUTB(O.RED)=0 If INB(I.BUZZER)=1 Then GoTo *RED.BLINK.LOOP EndIf Job "Buzzer" Off 'Stop Buzzer job 'Initilize and start Buzzer job Job "Buzzer" Start GoTo *MAIN.LOOP
--	--

If Job On is executed to the running job, Job On statement returns immediately without operation.

- See also Job Name, Job On, Job Start.

Job Start

(Statement)

● Function

Initializes a job and starts it from the first step.

● 書式

Job \sqsubset "*Job-name*" \sqsubset Start

● Argument

Parameter	Explanation
<i>Job-name</i>	Job name to start.

● Example

Job "Robot1" Start

● Explanation

Job Start statement initializes a job and starts it from the first step of a program.

All jobs starts automatically after STP system starts or a program is downloaded. The started job is stopped by the following operations.

- 1) Job Off statement by its own job
- 2) Job Off statement by other job

The stopped job can be started by other job. Job Start statement starts the stopped job from the first step after initialization of the job.

In the following example, Buzzer job starts from the step "**POWER.ON*" after Main job executes Job Start statement.

Job Name "Buzzer" Include "test.hed" *POWER.ON OUTB(O.BUZZER)=0 'Buzzer OFF Job "Buzzer" Off 'Job Off by myself 'After Job On by Main job, 'repeat the following procedure. *LOOP OUTB(O.BUZZER)=1 Delay 0.2 OUTB(O.BUZZER)=0 Delay 0.2 GoTo *LOOP	Job Name "Main" Include "test.hed" *MAIN.LOOP Wait INB(I.BUZZER)=1 Job "Buzzer" On 'Restart Buzzer job *RED.BLINK.LOOP 'Blink lamp Delay 1 : OUTB(O.RED)=1 Delay 1 : OUTB(O.RED)=0 If INB(I.BUZZER)=1 Then GoTo *RED.BLINK.LOOP EndIf Job "Buzzer" Off 'Stop Buzzer job 'Initilize and start Buzzer job Job "Buzzer" Start GoTo *MAIN.LOOP
--	--

If Job Satrt is executed to the running job, a job error occurs.

After the execution of Job Start statement, the target job is transferred into the following state.

- ◆ Nesting information of For-Next, subroutine and so on is cleared.
- ◆ Files that are opened by the job are closed.
- ◆ Error information of the job is cleared.

● See also Job Name, Job On, Job Off.

Left\$

(Function)

- **Function**
Gets the string containing a specified number of characters from the left side of a string.
- **Format**
Left\$(*String*, *Length*)
- **Arguments and Return value**

Parameter		Explanation
Arguments	<i>String</i>	String expression from which the leftmost characters are returned.
	<i>Length</i>	Numeric expression indicating number of characters to return. Valid range is from 0 through 255.
Return value		Leftmost string.
- **Example**
a\$="Hirata Corporation"
b\$=Left\$(a\$, 6) "Hirata" is substituted for b\$.
- **Explanation**
If *Length* exceeds the length of *String*, the function returns the same as the specified *String*.
If *Length* is zero, the function returns a null string.
- See also Right\$, Mid\$.

Len

(Function)

- Function
Gets length of a string.

- Format
Len(*String*)

- Argument and Return value

Parameter		Explanation
Argument	<i>String</i>	A string to get the length.
Return value		Byte size of the string length.

- Example
a\$="Hirata Corporation"
length%=Len(a\$) ' 18 is substituted for length%.

- Explanation
Any character codes such as control codesⁱ, space and so on are counted.

ⁱ Control code: Byte-code to control a display or peripheral equipment. This code is not printable and not able to display.

Line Input

(Statement)

● Function

Reads one line of characters from a file and set data to a string variable.

● Format

Line Input # *File-number, Variable*

● Arguments

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Variable</i>	A string variable to set the read data.

● Example

Line Input #1, a\$

● Explanation

Regardless of string delimiters such as comma, double quotations and so on, Line Input statement reads one line of characters from a file. A line has to be terminated by carriage return (CR) + linefeed (LF).

Name	ASCII code
Carriage return (CR)	13 (&HD)
Linefeed (LF)	10(&HA)

Log

(Function)

- Function
Gets the natural logarithm of a number.

- Format
 $\text{Log}(\text{Numeric-expression})$

- Argument and Return value

Parameter		Explanation
Argument	<i>Numeric-expression</i>	Any valid numeric expression greater than zero.
Return value		The natural logarithm of the specified value.

- Example
 $n! = 35 / 9$
 $a\# = \text{Log}(n!)$
- Explanation
 The natural logarithm is the logarithm to the base e. The constant e is approximately 2.718282.
 You can calculate base-n logarithms for any number x by dividing the natural logarithm of x by the natural logarithm of n as follows.

$$\text{Log}_n(x) = \text{Log}(x) / \text{Log}(n)$$
- See also Exp.

Macro

(Statement)

- Function
Defines a format of a macro-call in a macro file (suffix .bas).

- Format
Macro[\sqcup *Argument#1* [, *Argument#2* [, *Argument#3*...]]]

- Arguments

Parameter	Explanation
<i>Argument#n</i>	A variable used as a parameter that is specified by a macro-call. The number of arguments is allowed up to 10.

- Example

< Proc1.bas > --- Macro file (macro name "Proc1")

Macro para1&, para2&, para3&

 If para1& > 100 Then

 ML(ML.DATA) = para2& + para3&

 Else

 ML(ML.DATA) = para2& - para3&

 EndIf

< Main.bas >

data1&=10: data2&=1000: data3&=90

Proc1(data1&, data2&, data3&) ' Proc1 macro-call

data1&=101: data2&=100: data3&=90

Proc1(data1&, data2&, data3&) ' Proc1 macro-call

- 解説

- ◆ A macro file has to be located at the directory defined in "Setup" -> "Directories" -> "Macro files" in HBDE.
- ◆ Only one Macro statement has to be described at the top of a macro file. Macro name, which is used at the macro-call in the main program, is the name except filename suffix.
- ◆ See "7.5 Macro File" about details of macro usage.

Mid\$

(Statement)

● Function

Replaces a part of a string.

● Format

Mid\$(*String*, *Start* [, *Length*]) = *String-expression*

● Arguments

Parameter	Explanation
<i>String</i>	A string variable to modify. If a string constant, a compiling error occurs.
<i>Start</i>	Character position in <i>String</i> where the replacement of text begins.
<i>Length</i>	Number of characters to replace. If omitted, all of string is used.
<i>String-expression</i>	String expression that replaces part of <i>String</i> .

● Example

a\$ = "HrBasic Version 1.00"

Mid\$(a\$, 9, 7) = "##Ver##" ' "HrBasic ##Ver## 1.00" substituted for a\$

● Explanation

- ◆ The result of the example program is shown below.

Byte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
String	H	r	B	a	s	i	c		V	e	r	s	i	o	n		1	.	0	0
String	H	r	B	a	s	i	c		#	#	V	e	r	#	#		1	.	0	0

Start 9, Length 7

- ◆ If *Start* or *Length* is zero, or *Start* are greater than the length of *String*, a job error occurs.
- ◆ If *Length* is omitted or greater than the length of *String-expression*, replaced length is regarded as the length of *String-expression*.
- ◆ For example, the result of replacement from "harl-3" to "HrBasic" is shown below in the two cases of varying *Start* and *Length*.

Varying *Start*

```

For i%=1 to 6
  a$="harl-3"
  Mid$(a$,i%)="HrBasic"
Next i%
```

a\$ byte i%	1	2	3	4	5	6
0	Job error					
1	H	r	B	a	s	i
2	h	H	r	B	a	s
3	h	a	H	r	B	a
4	h	a	r	H	r	B
5	h	a	r	l	H	r
6	h	a	r	l	-	H
7	Job error					

Varying *Length*

```

For i%=1 to 6
  a$="harl-3"
  Md$(a$,1,i%)
Next i%
```

a\$ byte i%	1	2	3	4	5	6
0	Job error					
1	H	a	r	l	-	3
2	H	r	r	l	-	3
3	H	r	B	l	-	3
4	H	r	B	a	-	3
5	H	r	B	a	s	3
6	H	r	B	a	s	i
7	H	r	B	a	s	i

Mid\$

(Function)

- **Function**
Gets the string containing a specified number of characters from a string.

- **Format**
Mid\$(*String*, *Start* [, *Length*])

- **Arguments and Return value**

Parameter		Explanation
Arguments	<i>String</i>	String expression from which characters are returned.
	<i>Start</i>	Character position in string at which the part to be taken begins. Valid range is from 0 through 255.
	<i>Length</i>	Numeric expression indicating number of characters to return. Valid range is from 0 through 255.
Return value		Extracted string.

- **Example**
a\$ = "HrBasic Version 1.00"
b\$ = Mid\$(a\$, 9, 7) ' "Version" is substituted for b\$.

- **Explanation**
 - ◆ If *Length* is omitted, or the size from *Start* position through the end of *String*, Mid\$ statement returns the string of characters from *Start* position through the end.

Example)

a\$="1234567890"

b\$=Mid\$(a\$, 3) ' "34567890" substituted for b\$

c\$=Mid\$(a\$, 7, 5) ' "7890" substituted for c\$

- ◆ If *Start* exceeds the length of *String*, or *Length* is zero, Mid\$ statement returns a null string.

Example)

a\$="Hirata"

b\$=Mid\$(a\$, 8, 5) ' Null string substituted for b\$

c\$=Mid\$(a\$, 7, 0) ' Null string substituted for c\$

- See also Right\$, Left\$.

Mod

(Operator)

● Function

Divides two numbers and returns only the remainder.

● Format

Numeric-expression#1 Mod *Numeric-expression #2*

● Arguments

Parameter	Explanation
<i>Numeric-expression#1</i>	A numeric expression.
<i>Numeric-expression#2</i>	A numeric expression.

● Example

a% = 19

b% = 6

c% = a% Mod b% ‘ 1 substituted for c%.

● Explanation

- ◆ Mod operator divides *Numeric-expression#1* by *Numeric-expression #2* and returns only the remainder.

Move

(Statement)

● Function

Moves a robot to the specified position.

● Format

a) Standard format

Move #*File-number*[[rno:*Robot-number*]] [, *Motion*─ [*Sub-motion*]]
 , *Position* [, NoWait]

b) Pass PTP motion for discrete positions

Move # *File-number*[[rno:*Robot-number*]], PASS
 , *Position#1* [, *Position#2*, *Position#3*...] [, NoWait]

Note) Only supported by HNC-580 series and HAC-8XX.

● Arguments

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.
<i>Motion</i>	Motion pattern of a robot. PTP: A robot moves by PTP motion. If Sub-motion is omitted, a robot moves by the pattern specified by S code at each position. Omitted: A robot moves by the pattern specified by M data and S code at each position.
<i>Sub-motion</i>	In case of PTP motion, the following sub-motions can be specified. SlowDown: Insert motion SlowUp: Slow-up motion Slow: Insert and slow-up motion Omitted: A robot moves by the pattern specified by S code at each position.
<i>Position</i>	Expression to designate the position where the robot stops or passes. The expressions are categorized into the following six types. See "Explanation" about details. (1) Position data PMn stored in robot controller (2) Position data PMn stored in robot controller + Relative distance of axis (3) Position memory in STP (4) Position memory in STP + Relative distance of axis (5) Direct components of position (6) Current robot position + Relative distance of axis
NoWait	NoWait: Move statement returns without waiting for the positioning completion of a robot. Omitted: Move statement returns after the positioning completion of a robot.

● Example

(1) Position data PMn stored in robot controller

- Move #1[rno:1], PM(120)
- (2) Position data PMn stored in robot controller + Relative distance of axis
Move #1[rno:1], PM(120)+(, 10.3, , ,)
- (3) Position memory in STP
Move #1[rno:1], P0
- (4) Position memory in STP + Relative distance of axis
Move #1[rno:1], P0-(20, 10, 100)
- (5) Direct components of position
Move #1[rno:1], (100, , , 100, 0, 0, RIGHTY, 0, 1, 99, 1)
- (6) Current robot position + Relative distance of axis
Move #1[rno:1], HERE+(, , -100)

● Explanation

- (1) Position data PMn stored in robot controller

Format)

PMn

Explanation)

- Specify the stored position data that has been already taught in the robot controller.
- "n" is the number that indicates the address of position data to move.
- For HNC-3XX or HNC-544 controller, "n" is available as 0 to 999. For HNC-580 series or HAC-8XX controller, "n" is available as 0 to 3999.
- "n" is usually specified as the number. But, "n" can be specified as array expression as Example 2) and 3) using a variable.

Example 1)

PM100 --- stored position data of address 100

Example 2)

PM(100) --- stored position data of address 100

Example 3)

PM(addr%) --- stored position data of address set in addr%

- (2) Position data PMn stored in robot controller + Relative distance of axis

Format)

PMn + (X, Y, Z, W, R, C)

Explanation)

- Specify the stored position data that has been already taught in the robot controller and the relative distance of each axis from the stored position.
- Only "+" is available for a calculation operator. If "-" is specified, a compiling error occurs.
- How to specify PMn is the same as (1).
- Number or variable is available for relative distance of axis.
- Distance unit is millimeter or degree.
- Omitted axis means that the zero value is specified.

Example 1)

PM100 + (, 10.3, , ,)

--- position where 10.3 mm is added to Z axis to PM100.

Example 2)

PM(50) + (-3.6, , 10.3, , ,)

--- position where -3.6 mm, 10.3 mm are added

--- to X and Y axis to PM(50).

Example 3)

PM(10) + (-3.6, y.dis!, 10.3)

--- position where -3.6 mm, y.dis! mm, 10.3 mm are

--- added to X, Y and Z axis to PM(10).

(3) Position memory in STP

Format)

Pn

Explanation)

- Specify the position memory in STP.
- "n" is the number that indicates the index of position memory and available value is from 0 to 7999.
- "n" is usually specified as the number. But, "n" can be specified as array expression as Example 2) and 3) using a variable.

Example 1)

P100 --- position memory with index 100

Example 2)

P(100) --- position memory with index 100

Example 3)

P(addr%) --- position memory with index set in addr%

- Before using Pn memory, the number of Pn memories must be declared by DimPos statement. (See DimPos.)
- Pn memory is cleared by zero when STP starts. If the cleared Pn memory is specified to MOVE statement, a job error occurs. Before the reference of Pn memory when Move execution, the Pn memory must be set with the accurate data of the target position and motion. The accurate data of the target position and motion means that the data must contains the all axes position, arm component, dimension code and M,F,S code. There is the following ways that set the accurate data to a Pn memory.

- a) Copy the position data stored in robot controller to Pn memory

Example)

P(10) = Ref(#1, PM100)

PX(10) = 12.3

This sample copies the PM100 in the robot controller to P10 and it modifies only X axis data to 12.3 mm.

- b) PosRec function can set the position data to Pn memory in one step.

Example)

P(10) = PosRec (10, 20, 30, 40, 0, 0, LEFTY, 0, 1, 99, 0)

X axis --- 10.0 mm

Y axis --- 20.0 mm

Z axis --- 30.0 mm

W axis --- 40.0 degree

ARM component --- lefty

Dimension code --- zero

M code --- 1

F code --- 99

S code --- 0

Note)

Zero must be specified to dimension code.

Zero must be specified to unused axis.

- c) Copy the current position of the robot and then set M,F,S code. Axis data may be modified if necessary.

Example)

‘ Copy the current position but M,F,S code

‘ is cleared by zero.

P(10) = REF(#1, HERE)

‘ M,F,S code must be set.

PDM(10) = 1: PDF(10) = 99: PDS(10) = 0

‘ Decrease 20.0 mm in Z axis.

PZ(10) = PZ(10) - 20.0

Note)

Zero must be specified to dimension code.

Ref(#x, HERE) can get only the current axis position, arm component and dimension code and M,F,S data cannot be got and the its value is set to zero because the robot controller cannot decide M,F,S code for the current position.

So, the above sample program sets valid M,F,S code to Pn memory after REF(#1, HERE) is executed.

(4) Position memory in STP + Relative distance of axis

Format)

Pn + (X, Y, Z, W, R, C)

Pn - (X, Y, Z, W, R, C)

Explanation)

- Specify the position memory in STP and the relative distance of each axis from the position memory.
- How to specify and notice of Pn is the same as 3).
- Number or variable is available for relative distance of axis.
- Distance unit is millimeter or degree.
- Omitted axis means that the zero value is specified.

Example 1)

P100 + (, , 10.3, , ,)

--- position where 10.3 mm is added to Z axis to P100.

Example 2)

P(50) + (-3.6, , 10.3, , ,)

--- position where -3.6 mm, 10.3 mm are added

--- to X and Y axis to P(50).

Example 3)

P(10) + (-3.6, y.dis!, 10.3)

--- position where -3.6 mm, y.dis! mm, 10.3 mm are

--- added to X, Y and Z axis to P(10).

- "-" is available for calculation operator.

Example 4)

P(90) - (1.0, 2.0, 3.0, 4.0)

(5) Direct components of position

Format)

(*X, Y, Z, W, R, C, Arm, Dimension, M-data, F-code, S-code*)

Explanation)

- Specify the axis position, arm component, dimension code and M,F,S code directly.
- X, Y, Z, W, R, C can be omitted
- Distance unit of axes is millimeter or degree.
- In case of unused or not equipped axis, set zero or omit for the axis.
- If used or equipped axis is omitted, it means that the current position of the axis is specified.
- In case that the target is HNC-3XX or HNC-544 controller, *ARM, Dimension, M-data, F-code, S-code* can be omitted.
- In case that the target is HNC-580 series or HAC-8XX controller, *ARM, Dimension, M-data, F-code, S-code* cannot be omitted. If omitted, a job error occurs when Move statement is executed.
- Standard values are as follows.

Dimension --- 0

M-data --- 1

F-code --- 99

S-code --- 0

Specify zero to <Dimension code> in any case.

Example 1)

(-12.3, 2.3, 52.1, -184.3, 0, 0, LEFTY, 0, 1, 99, 0)
 --- X:-12.3mm Y:2.3mm Z:52.1mm W:-184.3deg.
 --- R:unused C:unused

Example 2)

(, , z.dis!, , , LEFTY, 0, 1, 99, 0)
 --- position where Z axis is added to the value z.dis!
 --- from the current position

(6) Current robot position + Relative distance of axis

Format)

HERE + (*X, Y, Z, W, R, C*)

Explanation)

- Specify the current robot position and the relative distance of each axis from it.
- Only "+" is available for a calculation operator.
- The keyword "HERE" represents the current position of the robot.
- If only HERE is specified without relative distance, a error occurs when compiling.
- Number or variable is available for relative distance of axis.
- Distance unit is millimeter or degree.
- Omitted axis means that the zero value

Example 1)

HERE + (, , 10.3, , ,)

--- position where 10.3 mm is added to Z axis
 --- to the current position.

Example 2)

HERE + (-3.6, , 10.3, , ,)
 --- position where -3.6 mm, 10.3 mm are added
 --- to X and Y axis to the current position.

Example 3)

HERE + (-3.6, y.dis!, 10.3)
 --- position where -3.6 mm, y.dis! mm, 10.3 mm are
 --- added to X, Y and Z axis to the current position.

◆ NoWait

If NoWait is specified, the program goes to the next step of Move statement not waiting for the completion of moving.

Then, while robot is moving, the program can check and control I/O for example. But, the program has to check the completion of moving.

The following is the sample program.

Example)

```
' Move to address 100 position stored in robot
Move #1[rno:2], PTP, PM(100), NoWait
*INB.CHECK
If INB(10) = 1 Then    ' Watch remote input bit #10
  Disable #1[rno:2]    ' Stop robot
  GoTo *ROB.STOP      ' Stopping procedure
EndIf
'If Robot is not stopped, check remote input.
If ((Ref(#1[rno:2], STATUS9) And &H2) <> &H2) Then
  GoTo *INB.CHECK
EndIf
```

Similarly, within Seq-SeqEnd block, the next step of Move statement is executed not waiting for the completion of moving without NoWait specified.

The difference of each case is shown below.

< Move execution in Seq-SeqEnd block >

The robot halts to move without moving the Z axis to pull down.

If Finish statement is executed, the Z axis is pulled down immediately.

Then the robot completes to move to the target position.

< Move execution with NoWait >

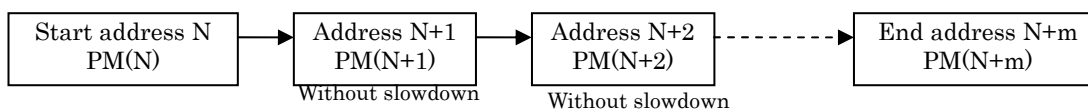
The robot does not halt and moves to the target position directly.

If Finish statement is executed, it has no effect of motion.

◆ Pass PTP motion

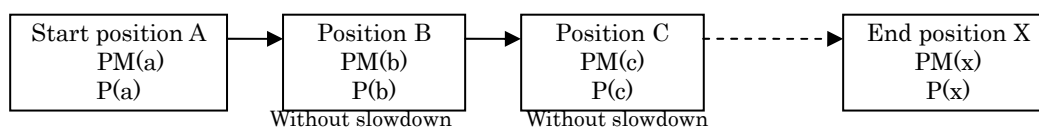
Pass PTP motion is the function of a robot controller. In pass PTP motion, a robot moves to pass sequentially addressed multiple positions which have been taught or programmed in a robot without slowdown, and finally stops at the end position. The positions have

to contain M=30-39 data. (Refer to robot operation manual about details.)



Since “a) Standard format” can operate only one position, after sequentially addressed multiple PMs with M=30-39 is set by teaching or program, the starting PM(N) has to be specified to Move statement.

In case of pass PTP motion using discrete positions, Move statement has to be described by “b) Pass PTP motion for discrete positions”.



In this case, available position data expression is “(1) Position data PMn stored in robot controller” or “(3) Position memory in STP”.

Maximum number of specified positions is 16.

It is not necessary that M data of each position is 30-39.

If the specified PMn is the top or middle of sequentially addressed multiple positions in a robot, only one specified PMn is used for the position.

This function is supported by only HNC-580 series and HAC-8XX.

Example)

Move #1[rno:1], PASS, PM(10), P(i%), P(i%+10), NoWait

NetClose

(Function)

- Function
Closes a network communication.

- Format
`NetClose(Network-ID)`

- Argument and Return value

Parameter		Explanation
Argument	<i>Network-ID</i>	Variable specifying a network identifier returned by NetOpen function.
Return value		Nothing

- Example

```
station%=2 'Station#2
nid%=NetOpen(station%) 'Open network for Station#2
:
NetClose(nid%) 'Close network
```

- Explanation

- ◆ NetClose function closes the network communication with the network identifier assigned by NetOpen function.
- ◆ Compiling error
 - NetClose does not have a returned value. If NetClose is substituted for a variable, the error "Type mismatch" occurs.
 - If an expression such as a numerical constant instead of a variable is specified to *Network-ID*, the error "Bad argument type of function" occurs.
- ◆ Job error
 - If the specified network identifier is invalid, the error "Incorrect usage of command or function" occurs.

- See also NetOpen.

NetOpen

(Function)

- Function
Opens a network communication.
- Format
NetOpen(*Station-Number*)

- Argument and Return value

Parameter		Explanation
Argument	<i>Station-Number</i>	A numeric expression specifying a station number assigned in the network definition. Valid station number is from 0 through 127.
Return value		Network identifier for the communication that is opened normally.

- Example

Station #1

```

Dim a%(10)
:
staion%=2
nid%=NetOpen(station%)
:
wlen%=NetWrite(nid%,a%(0),10) 'send a%(0)-a%(4)
:
NetClose(nid%)
:

```

Station #2

```

Dim a%(10)
:
staion%=1
nid%=NetOpen(station%)
:
rsize%=NetRead(nid%,a%(0),0) 'Receive a%(0)-a%(4)
:
NetClose(nid%)
:

```

- Explanation

- ◆ When you use NetOpen in HrBasic program running in STP, it is necessary that you have to create the network definition and download it to STP. Refer to operation manual of HBDE about details.
- ◆ NetOpen function opens the network communication for the station with the number specified to *Station-Number*.
- ◆ NetOpen returns the network identifier. NetRaed, NetWrite needs this network identifier to read from or write to the network. NetClose also needs this network identifier to close the network communication.
- ◆ When communicating with the network, NetOpen has been executed only one time for all jobs. If NetOpen is executed twice without closing, an execution error occurs.

- ◆ The maximum number of opened stations at the same time is restricted to 16.
- ◆ Compiling error
 - If the specified station number is the numerical constant with the value out of 0 through 127, the error “Illegal value of argument” occurs.
- ◆ Job error
 - If the specified station number has the value out of 0 through 127, the error “Incorrect usage of command or function” occurs.
 - If the specified station number is the own station number, the error “Own station number specified” occurs.
 - If the network definition of the specified station is not found, the error “Network CR(Communication Reference) undefined” occurs.
 - If NetOpen has been already executed for the specified station, the error “Network already opened” occurs.
 - If more than 16 stations are opened simultaneously, the error “Network open overflow” occurs.
- See also NetRead, NetWrite, NetOpen.

NetRead

(Function)

- Function
Reads data from the network communication opened by NetOpen.

- Format
NetRead(*Network-ID*, *Data-buffer*, *Option*)

- Arguments and Return value

Parameter		Explanation
Arguments	<i>Network-ID</i>	Variable specifying a network identifier returned by NetOpen function.
	<i>Data-buffer</i>	Variable for reading data buffer. String variable is available.
	<i>Option</i>	Optional flags. &H0000 Waiting for receiving data without execution of next step. &H0001 If received data not found, the next step is executed immediately
Return value		Data size actually received.

- Example

Station #1

```

Dim a%(10)
:
staion%=2
nid%=NetOpen(station%)
:
wlen%=NetWrite(nid%,a%(0),10) 'send a%(0)-a%(4)
:
NetClose(nid%)
:

```

Station #2

```

Dim a%(10)
:
staion%=1
nid%=NetOpen(station%)
:
rsize%=NetRead(nid%,a%(0),0) 'Receive a%(0)-a%(4)
:
NetClose(nid%)
:

```

- Explanation

- ◆ When you use NetRead in HrBasic program running in STP, it is necessary that you have to create the network definition and download it to STP. Refer to operation manual of HBDE about details.
- ◆ Maximum size of received data is 50 bytes.
- ◆ If the element of array like a%(0) is specified to *Data-buffer*, the received data is set to the sequential area in which the first element

is the specified element of array. You cannot specify the name of array.

Example)

```
Dim x%(10)
size%=NetRead(nid%,x%(1),0) 'Set to x%(1),x%(2),...
size%=NetRead(nid%,x%,0) 'Compiling error
```

- ◆ If the size of received data is bigger than the size of setting variable, area of other variable may be destroyed.
- ◆ Zero of return value means that data is not received. In case that the value with the bit #0 ON is specified to *Option*, the size of received data indicates that data has been received or not.

Example)

```
*LOOP
' Next step even if data not received
size% = Netread(nid%,data%,1)
' Read again when not received
If size% = 0 Then GoTo *LOOP
```

- ◆ Compiling error
 - If an expression such as a numerical constant instead of a variable is specified to *Network-ID*, the error “Bad argument type of function” occurs.
 - If an expression such as a numerical constant instead of a variable is specified to *Data-buffer*, the error “Illegal function call” occurs.
- ◆ Job error
 - If the specified network identifier is invalid, the error “Incorrect usage of command or function” occurs.
 - If the specified network identifier is not opened, the error “Network not opened” occurs.
 - If the network definition of the specified network identifier is not found, the error “Network CR(Communication Reference) undefined” occurs.
- See also NetOpen, NetWrite, NetClose.

NetWrite

(Function)

- **Function**
Writes data to the network communication opened by NetOpen.
- **Format**
`NetWrite(Network-ID, Data-buffer, Data-size)`

- **Arguments and Return value**

Parameter		Explanation
Arguments	<i>Network-ID</i>	Variable specifying a network identifier returned by NetOpen function.
	<i>Data-buffer</i>	Variable for writing data buffer. String variable is available.
	<i>Data-size</i>	Byte size to write.
Return value		Data size actually sent. Normally, this size is the same as <i>Data-size</i> .

- **Example**

Station #1

```

Dim a%(10)
:
staion%=2
nid%=NetOpen(station%)
:
wlen%=NetWrite(nid%,a%(0),10) 'send a%(0)-a%(4)
:
NetClose(nid%)
:

```



Station #2

```

Dim a%(10)
:
staion%=1
nid%=NetOpen(station%)
:
rsize%=NetRead(nid%,a%(0),0) 'Receive a%(0)-a%(4)
:
NetClose(nid%)
:

```

- **Explanation**

- ◆ When you use NetWrite in HrBasic program running in STP, it is necessary that you have to create the network definition and download it to STP. Refer to operation manual of HBDE about details.
- ◆ Maximum size of sending data is 50 bytes.
- ◆ If the element of array like a%(0) is specified to *Data-buffer*, the data of the sequential area in which the first element is the specified element of array is sent. You cannot specify the name of array.

Example)

Dim x%(10)

size% = NetWrite(nid%,x%(1), 6) 'Send x%(1),x%(2),x%(3)

size% = NetWrite(nid%,x%,6) 'Compiling error

◆ Compiling error

- If an expression such as a numerical constant instead of a variable is specified to *Network-ID*, the error “Bad argument type of function” occurs.
- If an expression such as a numerical constant instead of a variable is specified to *Data-buffer*, the error “Illegal function call” occurs.
- If a numerical constant out of 0 to 234 is specified to *Data-size*, the error “Illegal value of argument” occurs.

◆ Job error

- If the specified network identifier is invalid, the error “Incorrect usage of command or function” occurs.
- If the specified network identifier is not opened, the error “Network not opened” occurs.
- If the network definition of the specified network identifier is not found, the error “Network CR(Communication Reference) undefined” occurs.
- If the specified data size is out of 0 to 50 bytes, the error “Network writing size error” occurs.

- See also NetOpen, NetRead, NetClose.

Not

(Operator)

- Function
Executes a logical negation of a number.

- Format
Not \sqcup *Numeric-expression*

- Arguments

Parameter	Explanation
<i>Numeric-expression</i>	A numeric expression.

- Example
a% = &H00FF%
b% = Not a% ‘&HFF00% substituted for b%.

- Explanation
 - The following calculation is performed.

X	not X
1	0
0	1
 - See “6.4.3 Logical Operator”.

On Error GoTo

(Statement)

● Function

Defines a destination line to jump when a job error occurs.

● Foemat

- a) Registration of error routine

On Error GoTo *Label*

- b) Clearing registration of error routine

On Error GoTo 0

● Argument

Parameter	Explanation
<i>Label</i>	A label to jump. The label represents the entry of an error routine.

● Example

On Error GoTo *ERROR.HANDLER

:

‘ Error routine (sometimes called “error handler”)

*ERROR.HANDLER

err.no%=ERR

Resume *ERROR.RESUME ‘Exit error procedure

*ERROR.RESUME

Select Case err.no%

:

● Explanation

- ◆ On Error GoTo statement defines a destination step or line where a program jumps when a job error occurs. The destination routine has to be programmed as an “error routine” that executes the procedure, for example, that recovers and informs the error. The destination must be located in the same job.
- ◆ In the error routine, generally, a program has to recover and inform the error to check a type of the error using Err function. Then a program can exit the error routine by Resume statement.
- ◆ Error routine is frequently called “error handler”.
- ◆ On Error GoTo statement is not a declaration but a executable sentence. Therefore, the following example can be programmed to select an error routine.

Example)

If mode%=RUNNING Then ‘System mode is RUNNING.

On Error GoTo *ERROR.1

Else

On Error GoTo *ERROR.2

EndIf

- ◆ “On Error GoTo 0” clears the current registration of an error routine. After this execution, a job program stops at error step when a job error occurs.
- ◆ After STP system starts or a program downloaded, the registrations of an error routine for all jobs are cleared.

- ◆ Generally, “On Error GoTo *Label*” is described at the beginning of a job program.



There are the following rules for using a label.

- The top of label name has to be an asterisk “* “.
- Except asterisk, the first character of label name has to be alphabetic.
- Except asterisk, available characters in label name are alphabetic, numerical or period “.”, regardless of upper or lower case.
- Label name after asterisk cannot be the reserved name (e.g. *MOVE). But, a part of label name after asterisk can be the reserved name (e.g. *MOVE.LOOP).
- The length of label name is maximum 16 characters except asterisk.
- Label name definition has to be written at the top of one line.

- See “4.5 Error Handling”.

On...GoSub / On...GoTo

(Statement)

- Function
Branches to one of labels evaluating a numeric expression.

- Format
On \sqcup *Numeric-expression* \sqcup GoTo \sqcup *Label1* [, *Label2*...]
On \sqcup *Numeric-expression* \sqcup GoSub \sqcup *Label1* [, *Label2*...]

- Arguments

Parameter	Explanation
<i>Numeric-expression</i>	A numeric expression containing the value from 0 through 255.
<i>Label</i>	A label to branch.

- Example #1

```

' On...GoSub
:
  On a% GoSub *Sub1, *Sub2, *Sub3
:
' Subroutine executed when a%=1
*Sub1
:
  Return
' Subroutine executed when a%=2
*Sub1
:
  Return
' Subroutine executed when a%=3
*Sub3
:
  Return

```

- Example #2

```

' On...GoTo
:
  On a% GoTo *PROC1, *PROC2, *PROC3
:
' Jumped when a%=1
*PROC1
:
  GoTo *NEXT.PROC
' Jumped when a%=2
*PROC2
:
  GoTo * NEXT.PROC
' Jumped when a%=3
*PROC3
:
  GoTo * NEXT.PROC
:

```

```

‘ Next procedure
*NEXT.PROC
:

```

● Explanation

- ◆ *Numeric-expression* is evaluated into integer value.
- ◆ A program jumps to Nth *Label* according to the integer value N of *Numeric-expression*. For example, the value of *Numeric-expression* is three, a program jumps to third *Label*.
- ◆ If the value of *Numeric-expression* is negative, a job error occurs.
- ◆ If the value of *Numeric-expression* is zero or more than the number of labels, the next step is executed without branch.



Note

There are the following rules for using a label.

- The top of label name has to be an asterisk “ * “.
- Except asterisk, the first character of label name has to be alphabetic.
- Except asterisk, available characters in label name are alphabetic, numerical or period “ . “, regardless of upper or lower case.
- Label name after asterisk cannot be the reserved name (e.g. *MOVE). But, a part of label name after asterisk can be the reserved name (e.g. *MOVE.LOOP).
- The length of label name is maximum 16 characters except asterisk.
- Label name definition has to be written at the top of one line.

- See also GoSub, GoTo.

Open

(Statement)

● Function

Opens a file in the disk (such as hard disk, memory card) to access.
This function is supported by only WinSTP.

● Format

Open \sqcup *File-name* [\sqcup For \sqcup *File-mode*]
[\sqcup Access \sqcup *Access-type*] \sqcup As \sqcup [#] *File-number*

● Arguments

Parameter	Explanation
<i>File-name</i>	String constant or variable specifying a file name to open.
<i>File-mode</i>	Specify the following file modes. APPEND: A file is opened as a sequential file and data will be appended to it. BINARY: A file is opened as a binary file. INPUT: A file is opened as a sequential file with data input mode. OUTPUT: A file is opened as a sequential file with data output mode. RANDOM: A file is opened as a random access file. If omitted, RANDOM is selected implicitly. When the mode except INPUT specified, a file is created automatically if a file is not found.
<i>Access-type</i>	Specify the following access type. READ: Allows only reading. WRITE: Allows only writing. READ \sqcup WRITE: Allows both reading and writing. If omitted, READ \sqcup WRITE is selected implicitly.
<i>File-number</i>	Constant or variable specifying the number assigned for the opened file. After the file is opened, this number has to be used to access the file. Available number is 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.

● Example

● Constant (literal) used

Open "C:\temp\temp.txt" For OUTPUT Access WRITE as #1

● Variable used

fname\$="C:\temp\temp.txt"

fno%=1

Open fname\$ For INPUT Access READ as fno%

● Explanation

The file number already opened cannot be used to open.

● See "4.4.1 How to Access Data File".

Open “COM...”

(Statement)

- **Function**
Opens a communication port.
- **Format**
 - a) One argument of COM port and communication settings
Open `⌊` “COM n :Settings” `⌋` As [#] *File-number*
`⌊` RobType=*Robot-type* `⌊` RobNoList=*Robot-list* `⌋`
Note) Only constant (literal) is available for underline.
 - b) Two arguments of COM port and communication settings
Open `⌊` “COM n ” `⌊` “Settings” `⌋` As [#] *File-number*
`⌊` RobType=*Robot-type* `⌊` RobNoList=*Robot-list* `⌋`
Note) Constant (literal) or variable is available for underline.

● Arguments

Parameter	Explanation
n (Port number)	Numerical literal of a COM port number to communicate. See “4.4.2 How to Communicate with Peripheral Device” about details.
<i>Settings</i>	Specify RS232C communication parameters as the following format. <i>Speed</i> , <i>Parity</i> , <i>Data-length</i> , <i>Stop-bits</i> After STP system starts or a program is downloaded, parameters are initialized to default settings. <i>Speed</i> : Communication speed by bit-per-second (bps). Following values are available. 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 [default:115200] <i>Parity</i> : Parity bit of communication. O: Odd parity E: Even parity N: None parity [default:N] <i>Data-length</i> : Bit length of one character data. 7, 8 [default:8] <i>Stop-bits</i> : Stop bits of communication. 1, 2 [default:1] One or all of items can be omitted. If omitted, the last setting is used for communication.
<i>File-number</i>	Constant or variable specifying the number assigned for the opened file. After the file is opened, this number has to be used to access the file. Available number is 0 through 47. In case of variable, “#” can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Robot-type</i>	Specify a number of a robot controller type when a robot communication port is opened. A variable is not available. HNC-580 series : 580 HAC-8XX(COM0) : 580 Other : 0 or omit “RobType=”.

Parameter	Explanation
<i>Robot-list</i>	<p>This must be specified when <i>Robot-type=580</i>. A robot controller of type “580” can control the maximum four virtual robots. A robot number is assigned to distinguish each virtual robot. The robot number is defined as “MAINTENANCE” - “MAINTENANCE DATA” - “STATION NO.” of system generation data in a controller. Standard robot numbers are 1, 2, 3 and 4 for each virtual robot. Specify the list of the robot numbers of virtual robots that a program attempts to control as the following format up to maximum four robots. <i>Robot-No</i>[, <i>Robot-No</i>][, <i>Robot-No</i>][, <i>Robot-No</i>] Valid range of a robot number is from 1 through 999.</p>

● Example

- a) One argument of COM port and communication settings
‘ COM0---HAC-8XX, Robot numbers 1,2,3
Open “COM0” As #1 RobType=580 RobNoList=1,2,3
‘ COM1---HNC580 series, Robot numbers 1,5,7
Open “COM1:19200,E,7,1” As #1 RobType=580 RobNoList=1,5,7
- b) Two arguments of COM port and communication settings
‘ COM2---HNC-3XX
Open “COM2” ”9600,E,7,1” As #1

● Explanation

In case of two arguments of COM port and communication settings, a variable is available for an argument.

Example)

```
‘ COM1---HNC580 series, Robot numbers 1,5,7
file$ = “COM1”
para$ = “19200,E,7,1”
fno% = 1
Open file$ para$ As fno% RobNo=580 RobNoList=1,5,7
```

The communication port that has been already opened cannot be opened multiply. Moreover, the file number that has been already opened cannot be used multiply.

- See “4.4.2 How to Communicate with Peripheral Device”, “Chapter 8 Robot Control Programming”.

Or

(Operator)

- Function

Executes a logical addition of two numbers.

- Format

Numeric-expression#1 \sqcup Or \sqcup *Numeric-expression #2*

- Arguments

Parameter	Explanation
<i>Numeric-expression#1</i>	A numeric expression.
<i>Numeric-expression#2</i>	A numeric expression.

- Example

a% = &H000F%

b% = &H0FFF%

c% = a% Or b% ‘&H0FFF% substituted for c%

- Explanation

- The following calculation is performed.

X	Y	X or Y
1	1	1
1	0	1
0	1	1
0	0	0

- See “6.4.3 Logical Operator”.

Pai

(Function)

- Function
Gets the value of pi (π).

- Format
Pai

- Argument and Return value

Parameter	Explanation
Argument	Nothing
Return value	The value of pi.

- Example
`a# = Pai` ' the value of pi is substituted for a#.
- Explanation
Pi (π) is the circular constant, 3.1415927.

PosRec

(Function)

● Function

Makes a position data record that contains the specified elements.

● Format

PosRec(*X-axis*, *Y-axis*, *Z-axis*, *W-axis*, *R-axis*, *C-axis*
, *Arm-data*, *Coordinate-type*, *M-data*, *F-code*, *S-code*)

● Arguments and Return value

Parameter		Explanation
Arguments	<i>X-axis</i> <i>Y-axis</i> <i>Z-axis</i> <i>W-axis</i> <i>R-axis</i> <i>C-axis</i>	A numeric expression specifying the value of axis coordinate.
	<i>Arm-data</i>	The following words of the robot arm position have to be specified. LEFTY: Lefty position RIGHTY: Righty position
	<i>Coordinate-type</i>	A numeric expression specifying a coordinate type. Only zero value is available now.
	<i>M-data</i> ,	A numeric expression specifying M data. Valid range is from 0 through 255.
	<i>F-code</i> ,	A numeric expression specifying F code. Valid range is from 0 through 255.
	<i>S-code</i> ,	A numeric expression specifying S code. Valid range is from 0 through 255.
Return value		A position data record that contains the specified elements.

● Example

```
x.axis! = 1.1 ' X-axis
y.axis! = 2.2 ' Y-axis
z.axis! = 3.3 ' Z-axis
w.axis! = 4.4 ' W-axis
m.data% = 1 ' M-data
f.code% = 99 ' F-code
s.code% = 0 ' S-code
P(10) = PosRec(x.axis!, y.axis!, z.axis!, w.axis!, 0, 0, LEFTY, 0
, m.data%, f.code%, s.code%)
Move #1, P(10)
```

● Explanation

- ◆ Zero value has to be specified to the axis that is not equipped in the system.

(Statement)

- Function

Writes character strings or numerals specified in expressions to a file.

- Format

```
Print#File-number, Expression [|,| Expression...][|,|  
                                |;|                [;|]
```

- Arguments

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Expression</i>	A string or numeric expression of data to write.

- Example

Print #1, “ABC”

- Explanation

- ◆ Basically, Print# statement writes data with the following format adding CR/LF code to the end.

TEXT	C_R	L_F
------	-------	-------

CR (Carriage Return) &H0D

LF (Linefeed) &H0A

- ◆ Writing data format differs according to whether a string or number.

- Writing a string
"ABC" is written.

Print #1, “ABC”

ABC	C_R	L_F
-----	-------	-------

- Writing a number.

The number 123 is written. The sign character (space for positive value, minus for negative value) is added at the top of the number and the space is added at the end of the number.

Print #1, 123

$\sqcup 123 \sqcup$	C_R	L_F
---------------------	-------	-------

- ◆ When several expressions specified, writing data format differs according to a specified delimiter.

- a) A delimiter is semicolon and the end of sentence is expression.

Format) Print \sqcup #*File-number*, *Expression*[; *Expression*...]

Example) Print #1, 123; "ABC"; ", "; "VWXYZ"; -9876

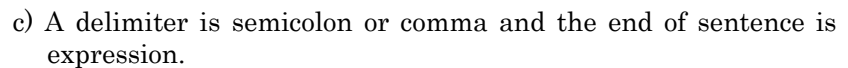
		(1)	(2)	(3)	(4)	(5)
$\sqsubset 123 \sqsubset$	ABC	,	VWXYZ	-9876	$\sqsubset C_R$	L_F

(1) (2) (3) (4) (5)

- b) A delimiter is comma and the end of sentence is expression.

Format) Print \sqcup #*File-number*, *Expression*[, *Expression*...]

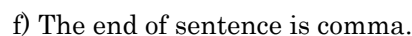
Example) Print #1, "ABC", -9876, "XYZ"


$$Expression[|, Expression[|; | Expression]...] \\ |; | \quad |, |$$

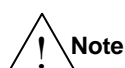
Example) Print #1, 123; "ABC", -9876


$$Expression[|, | Expression [|; | Expression]...];$$

Example) Print #1, 123; "ABC", -9876;


$$Expression[|, | Expression [|; | Expression]...],$$

Example) Print #1, 123;"ABC",



9-103

PrintStr

(Statement)

● Function

Makes a string using the specified format and then output a string to the specified variable. When a parameter descriptor is specified in the format, the value of the corresponded variable is converted to a string according to the descriptor.

● Format

PrintStr \sqsubset *String-variable*, *Format* [, *Parameter#1* [, *Parameter#2*, ...]]

● Arguments

Parameter	Explanation
<i>String-variable</i>	A string variable to output.
<i>Format</i>	A string expression specifying format to output.
<i>Parameter</i>	A variable containing the value that is converted to a string according to a parameter descriptor.

● Example

para1%=2003 : para2%=4 : para3%=2

PrintStr w\$, "year=%d month=%d day=%d", para1%, para2%, para3%
 "year=2003 month=4 day=2" is substituted for w\$

● Explanation

- ◆ In *Format*, a string started by "%" and terminated by an alphabet is regarded as a parameter descriptor. If a parameter descriptor is found in the format, the value of corresponded parameter variable is converted to a string according to the specified descriptor. The following descriptors are supported.

Following descriptors are supported:																		
Descriptor	Function	Example																
%[N]d	A parameter is converted to decimal integer expression. N specifies the number of output characters. If the number of the actual converted string is less than N, “0” is filled to the top of a string. If the number of the actual converted string is greater than N, the converted string is outputted as it is. If N omitted, the converted string is outputted as it is.	<ul style="list-style-type: none">• “12345” is contained in a parameter variable.<table><tr><td>Descriptor</td><td>Converted</td></tr><tr><td>"%2d"</td><td>"12345"</td></tr><tr><td>"%7d"</td><td>"0012345"</td></tr><tr><td>"%d"</td><td>"12345"</td></tr></table>• “-12345” is contained in a parameter variable.<table><tr><td>Descriptor</td><td>Converted</td></tr><tr><td>"%2d"</td><td>"-1"</td></tr><tr><td>"%7d"</td><td>"-012345"</td></tr><tr><td>"%d"</td><td>"-12345"</td></tr></table>	Descriptor	Converted	"%2d"	"12345"	"%7d"	"0012345"	"%d"	"12345"	Descriptor	Converted	"%2d"	"-1"	"%7d"	"-012345"	"%d"	"-12345"
Descriptor	Converted																	
"%2d"	"12345"																	
"%7d"	"0012345"																	
"%d"	"12345"																	
Descriptor	Converted																	
"%2d"	"-1"																	
"%7d"	"-012345"																	
"%d"	"-12345"																	
%[N]x	A parameter is converted to hexadecimal integer expression by lowercase. N specifies the number of output characters. If the number of the actual converted string is less than N, “0” is filled to the top of a string. If the number of the actual converted string is greater than N, the converted string is outputted as it is. If N omitted, the converted string is outputted as it is.	<ul style="list-style-type: none">• “&H0ABC” is contained in a parameter variable.<table><tr><td>Descriptor</td><td>Converted</td></tr><tr><td>"%2x"</td><td>"abc"</td></tr><tr><td>"%4x"</td><td>"0abc"</td></tr><tr><td>"%x"</td><td>"abc"</td></tr></table>	Descriptor	Converted	"%2x"	"abc"	"%4x"	"0abc"	"%x"	"abc"								
Descriptor	Converted																	
"%2x"	"abc"																	
"%4x"	"0abc"																	
"%x"	"abc"																	

Descriptor	Function	Example																
%[N]X	<p>A parameter is converted to hexadecimal integer expression by uppercase.</p> <p>N specifies the number of output characters. If the number of the actual converted string is less than N, “0” is filled to the top of a string. If the number of the actual converted string is greater than N, the converted string is outputted as it is. If N omitted, the converted string is outputted as it is.</p>	<ul style="list-style-type: none">“&H0ABC” is contained in a parameter variable. <table><tr><td>Descriptor</td><td>Converted</td></tr><tr><td>"%2x"</td><td>"ABC"</td></tr><tr><td>"%4x"</td><td>"0ABC"</td></tr><tr><td>"%x"</td><td>"ABC"</td></tr></table>	Descriptor	Converted	"%2x"	"ABC"	"%4x"	"0ABC"	"%x"	"ABC"								
Descriptor	Converted																	
"%2x"	"ABC"																	
"%4x"	"0ABC"																	
"%x"	"ABC"																	
%[N.M]f	<p>A parameter is converted to floating expression as real value.</p> <p>N specifies the number of output characters. If the number of the actual converted string is less than N, “0” is filled to the top of a string. If the number of the actual converted string is greater than N, the converted string is outputted as it is.</p> <p>M specifies the number of output characters in the fractional part. If the number of the actual converted string is less than M, “0” is filled to the end of a string. If the number of the actual converted string is greater than M, exceeded digits are removed.</p> <p>If N.M omitted, the converted string is outputted as it is.</p>	<ul style="list-style-type: none">“123.456” is contained in a parameter variable. <table><tr><td>Descriptor</td><td>Converted</td></tr><tr><td>"%4.2f"</td><td>"123.45"</td></tr><tr><td>"%7.2f"</td><td>"0123.45"</td></tr><tr><td>"%f"</td><td>"123.456"</td></tr></table> <ul style="list-style-type: none">“-123.456” is contained in a parameter variable. <table><tr><td>Descriptor</td><td>Converted</td></tr><tr><td>"%4.2f"</td><td>"-123.45"</td></tr><tr><td>"%8.2f"</td><td>"-0123.45"</td></tr><tr><td>"%f"</td><td>"-123.456"</td></tr></table>	Descriptor	Converted	"%4.2f"	"123.45"	"%7.2f"	"0123.45"	"%f"	"123.456"	Descriptor	Converted	"%4.2f"	"-123.45"	"%8.2f"	"-0123.45"	"%f"	"-123.456"
Descriptor	Converted																	
"%4.2f"	"123.45"																	
"%7.2f"	"0123.45"																	
"%f"	"123.456"																	
Descriptor	Converted																	
"%4.2f"	"-123.45"																	
"%8.2f"	"-0123.45"																	
"%f"	"-123.456"																	
%[N]s	<p>A parameter is converted to string expression.</p> <p>N specifies the number of output characters. If the number of the actual converted string is less than N, space is filled to the end of a string. If the number of the actual converted string is greater than N, right side of a string is removed. If N omitted, the converted string is outputted as it is.</p>	<ul style="list-style-type: none">“abcdefgh” is contained in a parameter variable. <table><tr><td>Descriptor</td><td>Converted</td></tr><tr><td>"%4s"</td><td>"abcd"</td></tr><tr><td>"%10s"</td><td>"abcdefgh␣␣␣"</td></tr><tr><td>"%s"</td><td>"abcdefgh"</td></tr></table>	Descriptor	Converted	"%4s"	"abcd"	"%10s"	"abcdefgh␣␣␣"	"%s"	"abcdefgh"								
Descriptor	Converted																	
"%4s"	"abcd"																	
"%10s"	"abcdefgh␣␣␣"																	
"%s"	"abcdefgh"																	

- ◆ Maximum number of parameter descriptors and parameter variables is 62.
- ◆ "%%" has to be described in a parameter descriptor to output "%" as a character.
- ◆ If the value type of a parameter descriptor differs from the type of the corresponded variable, it will be converted by the following rules.
 - %[N]d

If the specified variable is single-precision real type (!), area of the variable is regarded as 32-bits (4 bytes) integer. If the specified variable is double-precision real type (#), a job error

occurs. If the specified variable is string type (\$), the first byte is converted as integer.

- %[N]x or %[N]X

If the specified variable is single-precision real type (!), 4-bytes binary area of the variable is converted to a hexadecimal expression. If the specified variable is double-precision real type (#), 8-bytes binary area of the variable is converted to a hexadecimal expression. If the specified variable is string type (\$), the first byte is converted to a hexadecimal expression.

- %[N.M]f

If the specified variable is 2-bytes integer (%), 4-bytes integer (&) or string type (\$), a job error occurs.

- %[N]s

If the specified variable is not string type, a job error occurs.

- ◆ Total size of an output string is limited to 255 bytes. If the size exceeds 255 bytes, the tail of an output string is removed.
- ◆ It has to be careful to use “f” or “F” descriptor specifying a real type variable. For example, (1) dividing the value by almost zero. (2) calculating the exponential value. (3) dividing the value by huge value. These calculation causes that the result of (1)(2) may be huge, the result of (3) may be tiny. Generally, if the real value expressed as “a.aaaE+nnn” or “a.aaaE-nnn” is outputted by “f” or “F” descriptor, the size of output string becomes almost “nnn”. The following example checks the huge or tiny value before PrintStr execution.

Example)

```
If x# >= 1.0E-10 and x# <= 1.0E+10 Then
```

```
    PrintStr text$, "%f", x# 'about 10 bytes for output
```

```
Endif
```

- ◆ If the number of parameter descriptors differs from the number of parameter variables, the statement is executed as follows.
 - the number of parameter descriptors < the number of parameter variables
No error occurs when a program compiled or runs. Unused variables are never referred.
 - the number of parameter descriptors > the number of parameter variables
No error occurs when a program compiled, but a job error occurs when a program runs.

- See also ScanStr.

Pulse

(Statement)

● Function

Substitutes a value for a variable for the specified period as pulse output.

● Format

Pluse \sqsubset *Variable*=*Expresion#1*, *Expresion#2*

● Arguments

Parameter	Explanation
<i>Variable</i>	A variable to be pulsed. String variable is not available.
<i>Expresion#1</i>	A number substituted for a variable.
<i>Expresion#2</i>	The period of pulsing substitution by second. Available range is form 0.000 through 2147483.647 second

● Example

Pluse OUTB(0)=1, 2.0 ‘ OUTB(0)=1 for 2.0 sec.

Pluse OUTD(1)=&HFF, 3 ‘ OUTD(1)=&HFF for 3 sec.

Pluse a%=10, 5.1 ‘ a%=10 for 5.1 sec.

● Explanation

Pulse statement sets the value of *Expresion#1* to *Variable* for the period of *Expresion#2*.

After the time of *Expresion#2* passes, *Variable* resumes the previous value.

RchkHrcs

(Function)

- **Function**
Checks a HRCS protocol frame received.

- **Format**
RchkHrcs (*File-number*)

- **Argument and Return value**

Parameter		Explanation
Argument	<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. A constant or variable can be specified.
Return value		If a HRCS frame has been received, the function returns true value (-1). If not, it returns false value (0).

- **Example**

```
Open "COM1:115200,N,8,1" As #1 'Open COM1
*LOOP
'Loop if HRCS data not received.
If Not RchkHrcs(1) Then GoTo *LOOP
ReadHrcs #1, rcv$ 'Set received data to rcv$.
```

- **Explanation**
It is checked whether the following HRCS protocol frame has been received or not.

S _{TX}	TEXT	E _{TX}	L _{RC}
-----------------	------	-----------------	-----------------

- **STX (Start of Text)**
The head of HRCS protocol frame. (&H02)
- **ETX (End of Text)**
The end of HRCS protocol frame. (&H03)
- **LRC (Longitudinal Redundancy Check)**
LRCⁱ is a check code for communication data calculated by exclusive OR of bytes in TEXT to ETX.

If STP has already received a HRCS protocol frame, the function returns true value (-1). If not received, the function returns false value (0).

- See also ReadHrcs.

ⁱ See “Appendix B LRC Calculation”.

ReadHrcs

(Statement)

- **Function**
Reads a HRCS protocol frame.
- **Format**
ReadHrcs \leftarrow #*File-number*, *Variable*

- **Arguments**

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Variable</i>	A variable to set a received HRCS protocol frame.

- **Example**

```

Open "COM1:115200,N,8,1" As #1 'Open COM1
*LOOP
'Loop if HRCS data not received.
If Not RchkHrcs(1) Then GoTo *LOOP
ReadHrcs #1, recv$ 'Set received data to recv$.

```

- **Explanation**

A HRCS protocol frame is shown below.

ReadHrcs statement receives a HRCS protocol frame and extracts TEXT part in the figure to set to a variable.

S _{TX}	TEXT	E _{TX}	L _{RC}
-----------------	------	-----------------	-----------------

- **STX (Start of Text)**
The head of HRCS protocol frame. (&H02)
- **ETX (End of Text)**
The end of HRCS protocol frame. (&H03)
- **LRC (Longitudinal Redundancy Check)**
LRCⁱ is a check code for communication data calculated by exclusive OR of bytes in TEXT to ETX.
- See also RchkHrcs, WriteHrcs.

ⁱ See "Appendix B LRC Calculation".

Ref

(Statement)

● Function

Sets data to reserved memory in a robot.

● Format

Ref(*#File-number*[*[rno:Robot-number]*], *Reserved-memory*)=*Data*

● Arguments

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.
<i>Reserved-memory</i>	A reserved memory in a robot to set. Available memories are shown below. IRBn: Robot input bit IRDn: Robot input byte ORBn: Robot output bit ORDn: Robot output byte PMn: Robot position memory MMn: M data FMn: F code SMn: S code EXPARAn: Extended parameter
<i>Data</i>	Setting data to a robot reserved memory. The valid ranges of values are shown below. IRBn: 0 / 1 IRDn: 0 to 255 (&H0 to &HFF) ORBn: 0 / 1 ORDn: 0 to 255 (&H0 to &HFF) PMn: STP position memory P MMn: 0 to 99, and 255 FMn: 0 to 99 SMn: 0 to 99 EXPARAn: n=0-500 &H0 - &HFFFFFFF n=500-1000 0.0 - 2147483.647 n=1001-1099 &H0-&HFFFFFFF

● Example

- ◆ Set 1 to robot output bit #10 in robot #1
Ref(#1[rno:1], ORB(10))=1
- ◆ Set STP P(100) to PM(2) in robot #2.
Ref(#1[rno:2], PM(2)) = P(100)
- ◆ Set 99 to M data of PM(200) in robot #3.
Ref(#1[rno:3], MM(200)) = 99
- ◆ Set 1000 to extended parameter #10 in robot #2.
Ref(#1[rno:2], ERXPRA(10))=1000

- Explanation
Only STP P memory can be substituted for robot position memory PM.
- See “4.2 Reserved Memory”.

Ref

(Function)

- Function
Gets data from reserved memory in a robot.
- Format
Ref(*#File-number*[*rno:Robot-number*], *Reserved-memory*)
- Arguments and Return value

Parameter		Explanation
Arguments	<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
	<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.
	<i>Reserved-memory</i>	A reserved memory in a robot to get. Available memories are shown below. IRBn: Robot input bit IRDn: Robot input byte ORBn: Robot output bit ORDn: Robot output byte PMn: Robot position memory MMn: M data FMn: F code SMn: S code STATUSn: Robot status HERE: Robot current position EXPARAn: Extended parameter
Return value		The following range of the value is returned. IRBn: 0 / 1 IRDn: 0 to 255 (&H0 to &HFF) ORBn: 0 / 1 ORDn: 0 to 255 (&H0 to &HFF) PMn: STP position memory P MMn: 0 to 99, and 255 FMn: 0 to 99 SMn: 0 to 99 STATUSn: 0 to 255 (&H0 to &HFF) HERE: STP position memory P EXPARAn: n=0-500 &H0 - &HFFFFFFF n=500-1000 0.0 - 2147483.647 n=1001-1099 &H0-&HFFFFFFF

● Example

- ◆ Robot input byte #5 in robot #1 is substituted for dat%.
dat% = Ref(#1[rno:1], IRD(5))
- ◆ PM(100) in robot #1 is substituted for STP P(0).
P(0) = Ref(#1[rno:1], PM(100))
- ◆ M data of PM(1) in robot #1 is substituted for MD(1).
MD(1) = Ref(#1[rno:1], MM(1))
- ◆ STATUS0 (robot error code) in robot #1 is substituted for ecode%.
ecode% = Ref(#1[rno:1], STATUS0)
- ◆ Current position of robot #1 is substituted for STP P(100).
P(100) = Ref(#1[rno:1], HERE)
- ◆ Extended parameter #10 in robot #1 is substituted for para%.
para% = Ref(#1[rno:1], EXPARA(10))

● Explanation

- ◆ Robot position memory PM or current position HERE can be substituted for only STP P memory.
- ◆ Multiple usage of Ref function in a sentence is not allowed.
Example)
‘!!This is not allowed!!
If Ref(#1[rno:1],IRD0)=&H10 or Ref(#2[rno:2],IRD0)=&H10 Then

● See “4.2 Reserved Memory”.

Rem

(Statement)

- Function

Defines a comment sentence.

- Format

Rem \sqcup [*Comment*]

- Argument

Parameter	Explanation
<i>Comment</i>	Write comment (annotation).

- Format

Rem --- Robot Control Program ---

- Explanation

- ◆ Rem statement is not an executable command. Using Rem statement, any sentence can be described without execution. Described Rem sentence is outputted to the source list as it is.

- ◆ Apostrophe (') can replace Rem statement.

Example)

' --- Robot Control Program ---

- ◆ After Rem sentence adding a colon, a next executable sentence (multi statements) cannot be described.

Example)

' !!This example cannot be available!!

Rem --- Robot Control Program --- : a\$=Date\$

' “: a\$=Date\$” is regarded as comment

- ◆ When Rem statement is described after an executable sentence, a colon is not necessary.

Example)

a\$=Date\$ Rem --- Robot Control Program ---

Resume

(Statement)

- **Function**
Exits an error process, then resumes executing the main program.

- **Format**
Resume \sqcup [Next | *Label*]

- **Argument**

Parameter	Explanation
(Omitted)	Resume executing the step where a job error occurs.
Next	Resume executing the next of the step where a job error occurs.
<i>Label</i>	Resume executing a program at the specified label.

- **Example**

- Resume
- Resume Next
- Resume *

- **Explanation**

When a job error has not occurred, an error (&H12) occurs.

**Note**

There are the following rules for using a label.

- The top of label name has to be an asterisk “* “.
- Except asterisk, the first character of label name has to be alphabetic.
- Except asterisk, available characters in label name are alphabetic, numerical or period “.”, regardless of upper or lower case.
- Label name after asterisk cannot be the reserved name (e.g. *MOVE). But, a part of label name after asterisk can be the reserved name (e.g. *MOVE.LOOP).
- The length of label name is maximum 16 characters except asterisk.
- Label name definition has to be written at the top of one line.

- See also “4.5 Error Handling”, Err, On Error GoTo.

Return

(Statement)

- Function
Exits a subroutine and returns to the main program.

- Format
Return \sqcup [*Label*]

- Arguments

Parameter	Explanation
(Omitted)	A program exits a subroutine and then returns to the next of the step calling the subroutine.
<i>Label</i>	A program exits a subroutine and then returns to the specified label.

- Example

- Return
- Return *EXIT

- Explanation

- ◆ Multiple Return statements can be described in a subroutine.
- ◆ After calling a subroutine by GoSub statement, the GoSub stack counter counts up. After returning from a subroutine by Return statement, the GoSub stack counter counts down. If Return statement is executed when the GoSub stack counter is zero, a job error “RETUNR without GOSUB” occurs.
- ◆ Return statement with the specified *Label* makes a program very complex and causes low maintenanceability of a program. From the viewpoint of structured programming, “Return *Label*” must not be used.



Note

There are the following rules for using a label.

- The top of label name has to be an asterisk “* “.
- Except asterisk, the first character of label name has to be alphabetic.
- Except asterisk, available characters in label name are alphabetic, numerical or period “.”, regardless of upper or lower case.
- Label name after asterisk cannot be the reserved name (e.g. *MOVE). But, a part of label name after asterisk can be the reserved name (e.g. *MOVE.LOOP).
- The length of label name is maximum 16 characters except asterisk.
- Label name definition has to be written at the top of one line.

- See also “Chapter 7 Structured Programming“, GoSub, On Error GoTo, On GoSub.

Right\$

(Function)

- **Function**
Gets the string containing a specified number of characters from the right side of a string.
- **Format**
Right\$(*String*, *Length*)

- **Arguments and Return value**

Parameter		Explanation
Arguments	<i>String</i>	String expression from which the rightmost characters are returned.
	<i>Length</i>	Numeric expression indicating number of characters to return. Valid range is from 0 through 255.
Return value		Rightmost string.

- **Example**
a\$="HrBasic"
b\$=Right\$(a\$, 5) "Basic" is substituted for b\$
- **Explanation**
If *Length* exceeds the length of *String*, the function returns the same as the specified *String*.
If *Length* is zero, the function returns a null string.
- See also Left\$, Mid\$.

RobCheckBpZone

(Function)

● Function

Checks BP/ZONE state of robot.

Supported by HNC-580 series and HAC-8XX controller.

● Format

RobCheckBpZone(*#File-number*[*rno:Robot-number*], *BP/ZONE-number*)

● Arguments and Return value

Parameter		Explanation
Arguments	<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
	<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.
	<i>BP/ZONE-number</i>	BP/ZONE number to check. Valid range is from 1 through 8.
Return value		Current BP/ZONE state of a robot. If the current position of a robot is the inside of BP/ZONE setting, the function returns true value (-1). If not, it returns false value (0).

● Example

Open "COM1:115200,N,8,1" As #1

:

If RobCheckBpZone(#1,[rno:2], 2) Then GoTo *RECOVER1

● Explanation

Eight BP/ZONE state can be detected per a robot in a controller.

BP/ZONE state means the following state of the current position.

a) BP (BASE POS)

Whether the current position of a robot is near the specified robot base position.

b) ZONE

Whether the current position of robot axes is in the range between the specified upper limit and lower limit.

Refer to "Robot Operation Manual" about BP/ZONE.

The function can be executed during motion.

RobCheckCurPos

(Function)

● Function

Checks whether the current position of robot axes is near the position of the specified address.

● Format

RobCheckCurPos(*#File-number*[[*rno:Robot-number*]]
, Position-address, Axes)

● Arguments and Return value

Parameter		Explanation
Arguments	<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
	<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.
	<i>Position-address</i>	Position address of a robot compared with the current position. Valid range is from 0 through 3999.
	<i>Axes</i>	Robot axes flags to check the current position.
Return value		If the current position of axes is near the specified position, the function returns true value (-1). If not, it returns false value (0).

● Example

RobSetPosRange #fno%[rno:2], 2.0, , , 3.0 ' Set checking range.

If RobCheckCurPos (#fno%[rno:2], 823, 9) <> 0 The

:

Else

:

EndIf

● Explanation

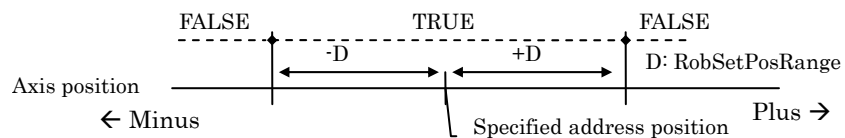
The function checks whether the current position of robot axes is near the position of the specified address.

Bits of *Axes* are assigned as follows.

Bit	7	6	5	4	3	2	1	0
Axis	-	-	C	R	W	Z	Y	X
Bit value	2 ⁷ (128)	2 ⁶ (64)	2 ⁵ (32)	2 ⁴ (16)	2 ³ (8)	2 ² (4)	2 ¹ (2)	2 ⁰ (1)

Specify checking axes to add each bit logically. For example, when X, Y and R axis are intended to check, 1+2+16=19 (&H1 or &H2 or &H10=&H13) has to be specified.

If the current position of the specified all axes is within the range specified by RobSetPosRange statement, the function returns true value.



If RobSetPosRange statement has not been executed, the default value 1.0 (mm or deg) is used for the range.

The function can be executed during motion.

- See also RobSetPosRange.

RobCheckStop

(Function)

● Function

Checks a robot is stopping now.

● Format

RobCheckStop(*#File-number*[[*rno:Robot-number*]])

● Arguments and Return value

Parameter		Explanation
Arguments	<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
	<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.
Return value		If a robot is stopping now, the function returns true value (-1). If not, it returns false value (0).

● Example

Seq #1[rno:1]

Move #1[rno:1], PM(PM.ADDR)

Finish #1[rno:1]

*CHECK

If Not RobCheckStop(#1[rno:1]) Then GoTo *CHECK ‘動作中

SeqEnd #1[rno:1]

● Explanation

RobCheckStop function examines STATUS9 in robot status. See “4.2.8 STATUS” about STATUS9.

RobClearErr

(Statement)

● Function

Clears error state of a robot.

● Format

RobClearErr \sqcup #*File-number*[[rno:*Robot-number*]]

● Arguments

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.

● Example

RobClearErr #1[rno:2]

● Explanation

RobClearErr statement clears error state held in a robot controller after a robot error has occurred. It is necessary that RobClearErr statement is executed before a program restarts to move a robot.

However, some kind of robot errors needs power-reset of the controller.

If RobClearErr is executed during motion, a job error occurs.

(Statement)

- Format
RobDistance \sqcup COM-port#1, Robot-number#1
COM-port#2, Robot-number#2, Variable

Parameter	Explanation
<i>COM-port#1</i>	A numeric expression specifying a COM port number to control the first robot. Only COM0 is available for HNC-8XX controller.
<i>Robot-number#1</i>	A numeric expression specifying the first robot station number. Valid range is from 1 through 999.
<i>COM-port#2</i>	A numeric expression specifying a COM port number to control the second robot. Only COM0 is available for HNC-8XX controller.
<i>Robot-number#2</i>	A numeric expression specifying the second robot station number. Valid range is from 1 through 999.
<i>Variable</i>	A variable to get the distance in the world coordinates system. Only single or double precision real type (!)(#) is available. In case of other type, a compiling error occurs.

- Explanation
 - ◆ “The world coordinate system” is the coordinate system that has the common space for the robots where the collision check of robots can be executed.
 - ◆ RobDistance statement is available only when the collision check data is defined in a HAC system. If it is not defined, the job error 131, “Local-World coordinates conversion data not defined”, occurs when running.
 - ◆ RobDistance statement can be executed during robot motion.
 - ◆ Refer to the document of collision check about the definition of collision check data.
- See also CollisionCheck, RobWorldPos.

RobGetCurAveTorq

(Statement)

● Function

Reads the current motor effective torque (current) of axes.

A unit of the read torque is percentage of rated torque (current).

Supported by HAX-8XX controller.

● Format

RobGetCurAveTorq \sqsubset #*File-number*[[rno:*Robot-number*]], *Motor-torque*

RobGetCurAveTorq \sqsubset #ファイル番号[[rno:ロボット番号]], 読出しトルク

● Arguments

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.
<i>Motor-torque</i>	One dimension array to set motor effective torque. Only long integer (32-bits) type variable (&) is available. Six and over elements are needed for array volume. If the array contains less than five elements, other variable area may be destroyed.

● Example

Dim avetorq&(6)

RobGetCurAveTorq #1[rno:2], avetorq&(1)

‘ X-axis motor effective torque set to avetorq&(1).

‘ Y-axis motor effective torque set to avetorq&(2).

‘ Z-axis motor effective torque set to avetorq&(3).

‘ W-axis motor effective torque set to avetorq&(4).

‘ R-axis motor effective torque set to avetorq&(5).

‘ C-axis motor effective torque set to avetorq&(6).

● Explanation

RobGetCurAveTorq statement can be executed whether a robot is ONLINE or not.

RobGetCurPos

(Statement)

● Function

Reads the current robot position indicated by the motor encoder.
Supported by HAX-8XX controller.

● Format

RobGetCurPos \sqsubset #*File-number*[[rno:*Robot-number*]], *Position-memory*

● Arguments

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.
<i>Position-memory</i>	STP P memory to set the current robot position. If P memory is not specified, a compiling error occurs.

● Example

DimPos 100 ‘ Use P(0) to P(99)

RobGetCurPos #1[rno:2], P(10)

- ‘ X-axis current position set to PX(10).
- ‘ Y-axis current position set to PX(10).
- ‘ Z-axis current position set to PX(10).
- ‘ W-axis current position set to PX(10).
- ‘ R-axis current position set to PX(10).
- ‘ C-axis current position set to PX(10).

● Explanation

- ◆ RobGetCurPos statement reads the robot position that the motor encoder of the axis indicates currently. If a robot moves with high speed, the position read by RobGetCurPos statement is more exact than Ref(#n, HERE) statement.
- ◆ A unit of read position value is “mm” or “degree”.
- ◆ The items except axis coordinates in P memory are not overwritten by RobGetCurPos statement.
- ◆ RobGetCurPos statement can be executed whether a robot is ONLINE or not.

RobGetCurSpeed

(Statement)

● Function

Reads the current motor speed of axes.

A unit of the read value is rpm (rotations per minute).

Supported by HAX-8XX controller.

● Format

RobGetCurSpeed \sqcup #*File-number*[[rno:*Robot-number*]], *Motor-speed*

● Arguments

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.
<i>Motor-speed</i>	One dimension array to set motor speed. Only long integer (32-bits) type variable (&) is available. Six and over elements are needed for array volume. If the array contains less than five elements, other variable area may be destroyed.

● Example

Dim speed&(6)

RobGetCurSpeed #1[rno:2], speed&(1)

‘ X-axis motor speed set to speed&(1).

‘ Y-axis motor speed set to speed&(2).

‘ Z-axis motor speed set to speed&(3).

‘ W-axis motor speed set to speed&(4).

‘ R-axis motor speed set to speed&(5).

‘ C-axis motor speed set to speed&(6).

● Explanation

RobGetCurSpeed statement can be executed whether a robot is ONLINE or not.

RobGetCurTorq

(Statement)

● Function

Reads the current motor torque (current) of axes.

A unit of the read torque is percentage of rated torque (current).

Supported by HAX-8XX controller.

● Format

RobGetCurTorq \sqcup #*File-number*[[rno:*Robot-number*]], *Motor-torque*

● Arguments

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.
<i>Motor-torque</i>	One dimension array to set motor torque. Only long integer (32-bits) type variable (&) is available. Six and over elements are needed for array volume. If the array contains less than five elements, other variable area may be destroyed.

● Example

Dim torque&(6)

RobGetCurTorq #1[rno:2], torque&(1)

‘ X-axis motor torque set to torque&(1).

‘ Y-axis motor torque set to torque&(2).

‘ Z-axis motor torque set to torque&(3).

‘ W-axis motor torque set to torque&(4).

‘ R-axis motor torque set to torque&(5).

‘ C-axis motor torque set to torque&(6).

● Explanation

RobGetCurTorq statement can be executed whether a robot is ONLINE or not.

(Statement)

- | Parameter | Explanation |
|-----------------------|--|
| <i>File-number</i> | A file number corresponded to the communication port must be specified. Valid range is from 0 through 47.
In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted. |
| <i>Robot-number</i> | A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable.
If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number. |
| <i>SG-group-name</i> | A string constant, variable specifying a SG group name of SG data to get. The format of a SG group name is;
"Main-group-name¥Sub-group-name"
See "Explanation". |
| <i>Array-variable</i> | A top elements of an array variable to which SG data is read. Long integer (&), single precision real (!), double precision real (#) and string (\$) type are available for an array. The reading size of SG data varies by the kind of SG group, but the capacity of an array must be defined sufficiently. |

- ◆ RobReadSG statement reads SG data of the specified SG group to the specified array variable.
- ◆ Available SG group name list is shown below.
 - HNC-580 series and HAC-8XX controller
 - “LIMIT¥ADDRESS MAX”
 - “LIMIT¥AREA LIMIT”
 - “MAINT¥EXPANSION A”

“MAINTENANCE EXPANSION B”
 “MAINTENANCE DATA”
 “ORIGIN SET-UP SYSTEM”
 “ORIGIN AXIS DIRECTION”
 “ORIGIN AXIS SELECT”
 “ADJUST AR TYPE ADJUST”
 “ADJUST MMB TYPE ADJUST”
 “CAPABILITY ROBOT CAPABILITY”
 “CAPABILITY EXPANSION A”

- ◆ Refer to “System Generation” and “System Generation List” in the robot controller operation manual about SG group.
- ◆ The elements of the specified SG group are set to the array variable sequentially in the order that is described at the “System Generation List” in the robot controller operation manual. After the example program is executed, SG data is set to the variable as follows.

Element of array	Data name	Meaning
sg.data!(1)	UPPER LMT A	Upper area limit of X(A) axis
sg.data!(2)	LOWER LMT A	Lower area limit of X(A) axis
sg.data!(3)	UPPER LMT B	Upper area limit of Y(B) axis
sg.data!(4)	LOWER LMT B	Lower area limit of Y(B) axis
sg.data!(5)	UPPER LMT Z	Upper area limit of Z axis
sg.data!(6)	LOWER LMT Z	Lower area limit of Z axis
sg.data!(7)	UPPER LMT W	Upper area limit of W axis
sg.data!(8)	LOWER LMT W	Lower area limit of W axis
sg.data!(9)	UPPER LMT R	Upper area limit of R axis
sg.data!(10)	LOWER LMT R	Lower area limit of R axis
sg.data!(11)	UPPER LMT C	Upper area limit of C axis
sg.data!(12)	LOWER LMT C	Lower area limit of C axis

- ◆ An element of some group may contain string data. In this case, a string variable has to be specified to RobReadSG statement. String data of the SG group is read to the corresponded element of a string array. Other number data is converted to the string and then set to the array. You can find a string type element of SG group that is described as “Selection” in “System Generation List”.
 - ◆ If a string element of SG data is read to numerical type array, a job error occurs when running.
- See also RobWriteSG.

RobReadSvoPara

(Statement)

● Function

Reads servo parameter of a robot.

Note) Only supported by HNC-580 series and HAC-8XX controller.

● Format

RobReadSvoPara \sqsubset #*File-number*[[*rno*:*Robot-number*]], *Axis-number*
, Array-variable

● Arguments

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.
<i>Axis-number</i>	A numeric expression specifying an axis number to get servo parameter. The axis numbers are; X-axis: 1 Y-axis: 2 Z-axis: 3 W-axis: 4 R-axis: 5 C-axis: 6
<i>Array-variable</i>	A top elements of an array variable to which servo parameter is read. Only long integer type (&) are available for an array. The reading size of servo parameter varies by the kind of a controller type, but the capacity of an array must be defined sufficiently.

● Example

‘ Define an array that has sufficient area for servo parameter.

Dim svo.para&(50)

:

‘ Reads servo parameter of X-axis to svo.para&(1), svo.para&(2),...

axis% = 1 ‘ X-axis

RobReadSvoPara #fno%[rno:1], axis%, svo.para&(1)

‘ Modify servo parameter

svo.para&(20) = svo.para&(20) - 2 ‘ Positive torque limit to -2%

svo.para&(21) = svo.para&(21) - 2 ‘ Negative torque limit to -2%

‘ Write servo parameter

RobWriteSvoPara #fno%[rno:1], axis%, svo.para&(1)

● Explanation

- ◆ RobReadSvoPara statement reads servo parameter of the specified axis to the specified array variable.
- ◆ Refer to “Automatic Creation of Robot Data” - “Servo Parameter” in the robot controller operation manual about servo parameter. The

elements of the specified servo parameter are set to the array variable sequentially in the order that is described in the manual.

- See also RobWriteSvoPara.

RobSetPosRange

(Statement)

● Function

Sets the position range that is used for checking a robot near the specific position.

● Format

RobSetPosRange \sqcup #*File-number*[[*rno*:*Robot-number*]]
[, [*X-axis*][, [*Y-axis*][, [*Z-axis*][, [*W-axis*][, [*R-axis*][, [*C-axis*]]]]]

● Arguments

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.
<i>X-axis</i>	The checking position range. Integer or real constant, and variable can be specified. A unit of value is "mm" or "degree". If omitted, the default value 1.0 mm(degree) is used.
<i>Y-axis</i>	
<i>Z-axis</i>	
<i>W-axis</i>	
<i>R-axis</i>	
<i>C-axis</i>	

● 文例

- X range=10.0mm, Y range=20.0mm, Z range=30.0mm, Other range=1.0mm for robot #2

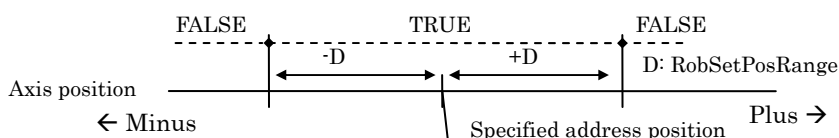
RobSetPosRange #1[rno:2], 10.0, , , 20.0, ,30.0

- Y range=5.0mm, Other range=1.0mm for robot #2

RobSetPosRange #1[rno:2], , 5.0

● Explanation

The value set by this statement is used by RobCheckCurPos function.
Specify the value "D" in the following figure for each axis.



● See also RobCheckCurPos.

RobWorldPos

(Statement)

● Function

Gets the robot current position in the world coordinates system used for the robot collision check.

Note) Only supported by HAC-8XX controller.

● Format

RobWorldPos ← *COM-port*, *Robot-number*, *P-memory*

● Arguments

Parameter	Explanation								
<i>COM-port</i>	A numeric expression specifying a COM port number to control a robot. Only COM0 is available for HNC-8XX controller.								
<i>Robot-number</i>	A numeric expression specifying the robot station number. Valid range is from 1 through 999.								
<i>P-memory</i>	STP P(n) memory to which the current position in the world coordinates system is set. A compiling error occurs except P memory. In P(n), axis coordinates are set to PX(n), PY(n), PZ(n), PW(n), PR(n), PC(n) by millimeter or degree. The following elements are initialized as follows: <table border="0"> <tr> <td>PARM(n)</td><td>0</td></tr> <tr> <td>PDM(n)</td><td>255</td></tr> <tr> <td>PDF(n)</td><td>0</td></tr> <tr> <td>PDS(n)</td><td>0</td></tr> </table>	PARM(n)	0	PDM(n)	255	PDF(n)	0	PDS(n)	0
PARM(n)	0								
PDM(n)	255								
PDF(n)	0								
PDS(n)	0								

● Example

comno% = 0 ' COM0

robno% = 1 ' Robot#1

RobWorldPos comno%, robno%, P(10) ' World coordinates set to P(10)

● Explanation

- ◆ "The world coordinate system" is the coordinate system that has the common space for the robots where the collision check of robots can be executed.
- ◆ RobWorldPos statement is available only when the collision check data is defined in a HAC system. If it is not defined, the job error 131, "Local-World coordinates conversion data not defined", occurs when running.
- ◆ RobDistance statement can be executed during robot motion.
- ◆ Refer to the document of collision check about the definition of collision check data.

● See also CollisionCheck, RobDistance.

(Statement)

- ◆ RobWriteSG statement writes SG data of the specified SG group from the specified array variable.
- ◆ Available SG group name list is shown in the explanation of RobReadSG statement.
- ◆ Refer to “System Generation” and “System Generation List” in the robot controller operation manual about SG group.
- ◆ The elements of the array are written to SG data in a robot sequentially in the order that is described at the “System

Generation List” in the robot controller operation manual. After the example program is executed, SG data is written as follows.

Element of array	Data name	Meaning
sg.data!(1)	UPPER LMT A	Upper area limit of X(A) axis
sg.data!(2)	LOWER LMT A	Lower area limit of X(A) axis
sg.data!(3)	UPPER LMT B	Upper area limit of Y(B) axis
sg.data!(4)	LOWER LMT B	Lower area limit of Y(B) axis
sg.data!(5)	UPPER LMT Z	Upper area limit of Z axis
sg.data!(6)	LOWER LMT Z	Lower area limit of Z axis
sg.data!(7)	UPPER LMT W	Upper area limit of W axis
sg.data!(8)	LOWER LMT W	Lower area limit of W axis
sg.data!(9)	UPPER LMT R	Upper area limit of R axis
sg.data!(10)	LOWER LMT R	Lower area limit of R axis
sg.data!(11)	UPPER LMT C	Upper area limit of C axis
sg.data!(12)	LOWER LMT C	Lower area limit of C axis

- ◆ An element of some group may contain string data. In this case, Set string data of the SG group to the string array, and set other number data as the numerical string after conversion to the string array. Then specify the string variable RobReadSG statement. You can find a string type element of SG group that is described as “Selection” in “System Generation List”.

- See also RobReadSG.

RobWriteSvoPara

(Statement)

● Function

Writes servo parameter of a robot.

Note) Only supported by HNC-580 series and HAC-8XX controller.

● Format

RobWriteSvoPara \sqsubset #*File-number*[[rno:*Robot-number*]], *Axis-number*
, Array-variable

● Arguments

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.
<i>Axis-number</i>	A numeric expression specifying an axis number to write servo parameter. The axis numbers are; X-axis: 1 Y-axis: 2 Z-axis: 3 W-axis: 4 R-axis: 5 C-axis: 6
<i>Array-variable</i>	A top elements of an array variable that contains servo parameter to write. Only long integer type (&) are available for an array. The writing size of servo parameter varies by the kind of a controller type.

● Example

‘ Define an array that has sufficient area for servo parameter.

Dim svo.para&(50)

:

‘ Reads servo parameter of X-axis to svo.para&(1), svo.para&(2),...

axis% = 1 ‘ X-axis

RobReadSvoPara #fno%[rno:1], axis%, svo.para&(1)

‘ Modify servo parameter

svo.para&(20) = svo.para&(20) - 2 ‘ Positive torque limit to -2%

svo.para&(21) = svo.para&(21) - 2 ‘ Negative torque limit to -2%

‘ Write servo parameter

RobWriteSvoPara #fno%[rno:1], axis%, svo.para&(1)

● Explanation

- ◆ RobWriteSvoPara statement writes servo parameter of the specified axis from the specified array variable.

- Refer to “Automatic Creation of Robot Data” - “Servo Parameter” in the robot controller operation manual about servo parameter. The elements of the array are written to servo parameter sequentially in the order that is described in the manual.

- See also RobReadSvoPara.

ScanStr

(Function)

● Function

Scans a string according to the specified format. If the string matches the format, the function returns true value (-1). If not, it returns false value. Moreover, the function gets the value as parameter from a string by a parameter operator in the format.

● Format

ScanStr(*String*, *Format*, *Parameter#1* [, *Parameter#2*...])

● Arguments and Return value

Parameter		Explanation
Arguments	<i>String</i>	A string expression to scan.
	<i>Format</i> ,	A string expression specifying format to scan.
	<i>Parameter</i>	A variable to which the converted value is set according to a parameter descriptor.
Return value		If the string matches the specified format, the function returns true value (-1). If not, it returns false value (0).

● Example

```
ws$="year:2003 month:4" 'String to scan
If ScanStr(ws$, "year:%d month:%d", para1%, para2%) Then
    If para1% <= 2000 Then
        century%=20
    Else
        century%=21
    EndIf
EndIf
```

● Explanation

- ◆ Scanning *String* is executed to compare with *Format*. When a parameter descriptor is found in *Format*, the value is extracted from *String* and set to a parameter variable.
- ◆ The character that cannot be converted by a parameter descriptor is detected, ScanStr function exits and returns false value (0) immediately. If *String* matches *Format*, the function returns true value (-1).

Example)

```
w$="para:xyz"
a%=ScanStr(w$, "para:%d", p1%) 'a% is false(0).
```

- ◆ Comparison between *String* and *Format* distinguishes uppercase and lowercase of a character.

Example)

```
w$ = "x:4 y:10"
a1% = ScanStr( w$, "x:%d y:%d" p1%, p2%) 'a% is false(0).
a2% = ScanStr( w$, "X:%d Y:%d" p1%, p2%) 'a2% is true(-1).
```

- ◆ If *String* is longer than *Format*, *String* is scanned with the length of *Format*.

Example)

```
w$="year:2003 month:4"
```

`a%=ScanStr(w$, "year:%d" p1%) 'a% is true(-1).`

- ◆ If *String* is shorter than *Format*, the function always returns false value (0).

Example)

`w$="year:2003"`

`a%=ScanStr(w$, "year:%d month:%d" p1%) 'a% is false(0).`

- ◆ In *Format*, a string started by "%" and terminated by an alphabet is regarded as a parameter descriptor. If a parameter descriptor is found in the format, a string at the position of *String*, where the descriptor indicates, is converted to the value. Then the value is set to the corresponded parameter variable.

The following descriptors are supported.

Descriptor	Function	Example
%[N]d or %[N]D	Scanned string is converted to set as a decimal number. N specifies the number of input characters. If N omitted, the string is scanned until an invalid character for conversion appears. The sequential space or tab at the top of the string is passed to input, but the scanning count increases.	<ul style="list-style-type: none">• “01234” scanned Descriptor Converted "%2d" 1 "%4d" 123 "%d" 1234• “-01234” scanned Descriptor Converted "%2d" 0 "%4d" -12 "%d" -1234• “␣1234” scanned Descriptor Converted "%2d" 1 "%4d" 123 "%d" 1234• “+01234” scanned Descriptor Converted "%2d" 0 "%4d" 12 "%d" 1234
%[N]x or %[N]X	Scanned string is converted to set as a hexadecimal number without character cases. N specifies the number of input characters. If N omitted, the string is scanned until an invalid character for conversion appears. The sequential space or tab at the top of the string is passed to input, but the scanning count increases.	<ul style="list-style-type: none">• “FF012345” scanned Descriptor Converted "%2x" &HFF "%4x" &HFF01 "%x" &HFF012345• “abcd1234” scanned Descriptor Converted "%2x" &HAB "%4x" &HABCD "%x" &HABCD1234• “␣ABCD” scanned Descriptor Converted "%2x" &HA "%4x" &HABC "%x" &HABCD

Descriptor	Function	Example																																
<p>%[N]f or %[N]F</p>	<p>Scanned string is converted to set as a real number.</p> <p>N specifies the number of input characters. If N omitted, the string is scanned until an invalid character for conversion appears.</p> <p>The sequential space or tab at the top of the string is passed to input, but the scanning count increases.</p>	<ul style="list-style-type: none">• “1234.567” scanned<table><tr><td>Descriptor</td><td>Converted</td></tr><tr><td>"%2f"</td><td>12.0</td></tr><tr><td>"%5f"</td><td>1234.5</td></tr><tr><td>"%f"</td><td>1234.567</td></tr></table>• “-1234.567” scanned<table><tr><td>Descriptor</td><td>Converted</td></tr><tr><td>"%2f"</td><td>-1.0</td></tr><tr><td>"%5f"</td><td>-1234.0</td></tr><tr><td>"%f"</td><td>-1234.567</td></tr></table>• “␣1.234” scanned<table><tr><td>Descriptor</td><td>Converted</td></tr><tr><td>"%2f"</td><td>1.0</td></tr><tr><td>"%5f"</td><td>1.23</td></tr><tr><td>"%f"</td><td>1.234</td></tr></table>• “+1.234” scanned<table><tr><td>Descriptor</td><td>Converted</td></tr><tr><td>"%2f"</td><td>1.0</td></tr><tr><td>"%5f"</td><td>1.23</td></tr><tr><td>"%f"</td><td>1.234</td></tr></table>	Descriptor	Converted	"%2f"	12.0	"%5f"	1234.5	"%f"	1234.567	Descriptor	Converted	"%2f"	-1.0	"%5f"	-1234.0	"%f"	-1234.567	Descriptor	Converted	"%2f"	1.0	"%5f"	1.23	"%f"	1.234	Descriptor	Converted	"%2f"	1.0	"%5f"	1.23	"%f"	1.234
Descriptor	Converted																																	
"%2f"	12.0																																	
"%5f"	1234.5																																	
"%f"	1234.567																																	
Descriptor	Converted																																	
"%2f"	-1.0																																	
"%5f"	-1234.0																																	
"%f"	-1234.567																																	
Descriptor	Converted																																	
"%2f"	1.0																																	
"%5f"	1.23																																	
"%f"	1.234																																	
Descriptor	Converted																																	
"%2f"	1.0																																	
"%5f"	1.23																																	
"%f"	1.234																																	
<p>%[N]s or %[N]S</p>	<p>The string is scanned to set as a string. <u>A corresponded parameter variable has to be string type.</u></p> <p>The inputted string begins at the character except space, tab or carriage-return, and terminates at the character before pace, tab or carriage-return.</p> <p>N specifies the number of input characters. If N omitted, the all scanned string is inputted.</p> <p>When N specified, it is attempted to input N characters of the string. However, if the size of inputted string is less than N, spaces are added to the top of the inputted string.</p>	<ul style="list-style-type: none">• “123456 AAAAAA” scanned<table><tr><td>Descriptor</td><td>Converted</td></tr><tr><td>"%s%s"</td><td>"123456" "AAAAAA"</td></tr><tr><td>"%3s%3s"</td><td>"123" "456"</td></tr></table>• “ A B” scanned<table><tr><td>"%s%s"</td><td>"A" "B"</td></tr><tr><td>"%3s%3s"</td><td>" A" " B"</td></tr></table>	Descriptor	Converted	"%s%s"	"123456" "AAAAAA"	"%3s%3s"	"123" "456"	"%s%s"	"A" "B"	"%3s%3s"	" A" " B"																						
Descriptor	Converted																																	
"%s%s"	"123456" "AAAAAA"																																	
"%3s%3s"	"123" "456"																																	
"%s%s"	"A" "B"																																	
"%3s%3s"	" A" " B"																																	

- ◆ Maximum number of parameter descriptors and parameter variables is 62.
- ◆ Descriptor except “s”, “S”
 If the value type of a parameter descriptor differs from the type of the corresponded variable, scanned and inputted characters are converted properly matching the variable type. When the parameter variable is string type, inputted characters are set to the variable without conversion.
 Example)
 w\$ = “example:12.34”
 ScanStr(w\$, “example:%2f”, para#) ‘ para#=12.0
 ScanStr(w\$, “example:%2f”, para%) ‘ para%=12
 ScanStr(w\$, “example:%2f”, para\$) ‘ para\$=”12”
 ScanStr(w\$, “example:%3f”, para\$) ‘ para\$=”12.”
- ◆ Descriptor “s”, “S”

The corresponded parameter variable has to be a string type. If not string type, a job error occurs when executed.

- ◆ "%%" has to be described in a parameter descriptor to output "%" as a character.
- ◆ If the number of parameter descriptors differs from the number of parameter variables, the statement is executed as follows.
 - the number of parameter descriptors < the number of parameter variables
No error occurs when a program compiled or runs. Unused variables are never referred.
 - the number of parameter descriptors > the number of parameter variables
No error occurs when a program compiled, but a job error occurs when a program runs.
- See also PrintStr.

Select Case

(Statement)

- Function
Selects a procedure block evaluating the specified expression.

- Format
Select \sqcup Case \sqcup *Test-expression*
Case \sqcup *Expression-list#1*
 Statements-block#1
Case \sqcup *Expression-list#2*
 Statements-block#2
 :
Case \sqcup Else
 Statements-block#N
End \sqcup Select

- Arguments

Parameter	Explanation
<i>Test-expression</i>	An expression to test for selection.
	An expression of value that <i>Test-expression</i> may contains. Specifying format and example is shown below.
	<ul style="list-style-type: none"> ● <i>Number, Expression, String</i> Example) Case 1, 3, 5, 7 Case "ABC", "DEF" ● <i>Expression#1 To Expression#2</i> Specify the range from <i>Expression#1</i> to <i>Expression#2</i> by a number, numeric expression or string. Example) Case 1 To 5 Case "FMT" To "HIT" ● <i>Is Relational-perator Expression</i> "=", "<>", "<", "<=", ">", ">=" are available for <i>Relational-perator</i>. <i>Expression</i> is a number, numeric expression or string. Example) Case Is = 5 Case Is <> 5 Case Is < 5 Case Is <= 5 Case Is > 5 Case Is >= 5
<i>Expression-list</i> (1 to N-1)	<i>Expression-list</i> can contain the combination of above formats with a delimiter comma.
<i>Statements-block(1 to N-1)</i>	A block of statements that is executed when <i>Expression-list</i> becomes true.
<i>Statements-block(N)</i>	A block of statements that is executed when all <i>Expression-list</i> (1 to N-1) becomes false.

- Example

- ◆ Example#1

Select Case a%*2

Case 10, 12, 20 to 100, Is <=200 ‘ Case#1

‘ If (a%*2) is one of 10, 12, 20 to 100, 200 or less,

‘ <Statement block #1> is executed.

<Statement block #1>

Case 250, n% ‘ Case#2

‘ If (a%*2) is 250 or n%,

‘ <Statement block #2> is executed.

<Statement block #2>

Case Else

‘ If both Case#1 and Case#2 are false,

‘ <Statement block #3> is executed.

<Statement block #3>

End Select

- ◆ Example#2

Select Case a\$

Case “ABC”, “FG” to b\$, Is < “ZZZ”

‘ If a\$ is one of “ABC”, “FG” to b\$, less than “ZZZ”,

‘ <Statement block #1> is executed.

<Statement block #1>

Case c\$+b\$

‘ If a\$ is (c\$+b\$),

‘ <Statement block #2> is executed.

<Statement block #2>

Case Else

‘ If both Case#1 and Case#2 are false, do nothing.

End Select

- Explanation

- ◆ Select Case statement always needs Case Else and End Select.
- ◆ If one of conditions in *Expression-list* is true, the statement block is executed.
- ◆ Any sentences or statements including Select Case statement can be described in *Statements-block*. A program jumps to the next of End Select after *Statements-block* has been executed. *Statements-block* can be omitted.
- ◆ Maximum number of Select Case nestsⁱ is 8.
- ◆ Maximum number of Case and Case Else statements is 127 in one Select Case - End Select.
- ◆ Select Case, Case Else and End Select statement are not compiled to executable code.

ⁱ Nest : means a structure located in the same structure.

Seq - SeqEnd

(Statement)

- Function
Starts and terminates robot sequence mode.

- Foramt
Seq \hookleftarrow #*File-number* [[rno:*Robot-number*]]
:
SeqEnd \hookleftarrow #*File-number* [[rno:*Robot-number*]]

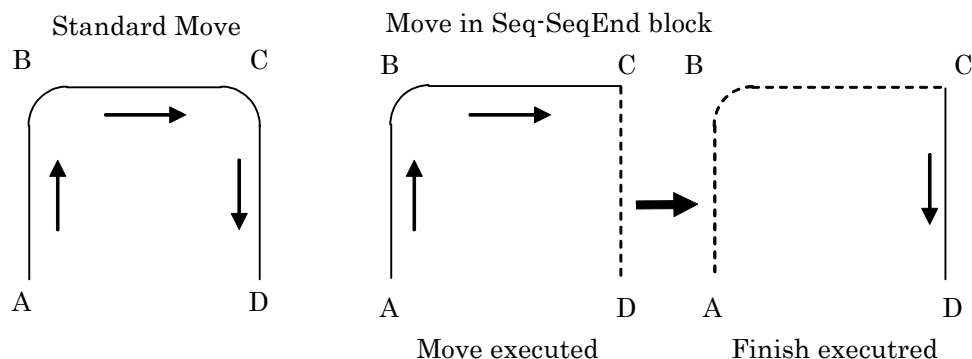
- Arguments

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.

- Example
Seq #1[rno:1]
Move #1[rno:1], PM(PM.PLACE)
' Confirm a work does not exist.
Wait INB(I.WORK)=0
Finish #1[rno:1]
SeqEnd #1[rno:1]

- Explanation

- ◆ When Move statement is executed in Seq-SeqEnd block, starting motion, Move returns immediately without positioning check.
- ◆ After the motion by Move statement, Z axis does not moves down in Seq-SeqEnd block. As the following figure, Move statement in Seq-SeqEnd block moves a robot to A->B->C, and then finally a robot stops at the point C. After Finish statement is executed, Z axis moves down toward the point D.



If Finish statement is executed before a robot reaches the point C, a robot moves to the point D without stopping at the point C.

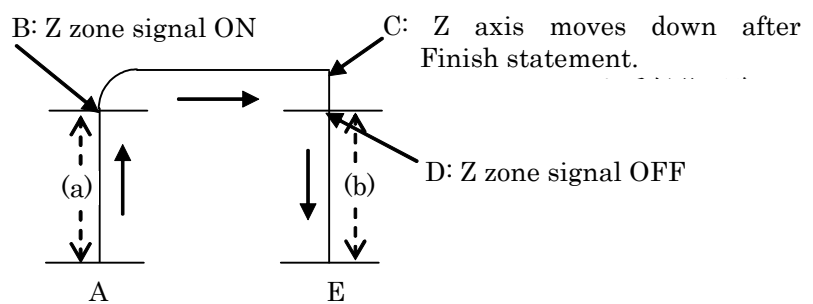
In the following example, Z zone signal triggers to execute Finish.
Example)

```
Seq #1[rno:1]
Move #1[rno:1], PM(PM.PLACE)
```

```

      : ' Procedure during the section (a)
      : ' before Z zone becomes ON.
Wait (Ref(#1[rno:1], STATUS9) and &H)=1
      : ' Procedure after Z zone becomes OFF.
      : ' If a robot reaches the posit C before this procedure,
      : ' a robot stops without moving Z axis down.
Finish #1[rno:1] ' Moves Z down
Wait (Ref(#1[rno:1], STATUS9) and &H1)=0
      : ' Procedure during the section (b)
      : ' after Z zone becomes OFF.
' Motion completed.
Wait (Ref(#1[rno:1], STATUS9) and &H2)=2
SeqEnd #1[rno:1]

```



- ◆ In case that Z axis does not move, Finish statement is needed in Seq-SeqEnd block after Move statement.
- ◆ It is not allowed that a program jumps into or out of Seq-SeqEnd block using branch statement such as GoTo.
- ◆ Nest of Seq-SeqEnd is not allowed.

Set

(Statement)

● Function

Sets motion parameters to a robot.

● Format

SET┐#*File-number*[[rno:*Robot-number*]] , *Motion-parameter*

● Arguments

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>Robot-number</i>	A station number of a virtual robot. Valid number is 1 through 999. See chapter 8 about the robot number. It can be specified as a number or variable. If the robot number is omitted, the number registered by SetRobNo function is used as the current robot number.
<i>Motion-parameter</i>	Sub-command to set a robot motion parameter. The following sub-commands are supported. See "Explanation" about each sub-command. (1) Speed for PTP motion (2) Speed for CPC motion (3) Acceleration and deceleration for PTP motion (4) Acceleration and deceleration for CPC motion (5) Pull-up of Z axis (6) Arch motion data (7) Slow-up motion data (8) Insert motion data

● Example

Set #1[rno:2], Speed=100

● Explanation

(1) Speed for PTP motion

- Format#1 : Speed=All-axes-*value*
- Format#2 : Speed=(XYW-axes-*value*, Z-axes-*value*)
- Range : 0 to 100
- Precision : 1 %
- Default : 100

Specify speed of all motion except CPC during ONLINE mode. The maximum speed varies according to the robot type. Specify the ratio of the maximum speed from zero (minimum speed) through 100 (maximum speed). Refer to "13.1.2 AXIS SPEED" in operation manual of a robot controller.

(2) Speed for CPC motion

- Format : Line Speed= *Value*
- Range : 0 to 999
- Precision : 1 mm/sec
- Default : 100

Specify CPC motion speed.

The head speed is related to "Line Speed" and F code of motion start.

$$\text{Head speed} = \text{"Line Speed"} * \frac{1+F_{\text{code}}}{100} (\text{mm/sec})$$

The maximum speed varies according to the robot type. If the value exceeding the maximum is set, a robot moves within the maximum speed.

Refer to "13.1.3 CPC CONSTANT" -> "CPC SPEED" in operation manual of a robot controller.

(3) Acceleration and deceleration for PTP motion

- Format#1 : Accel=All-axes-*value*
- Format#2 : Accel=(XYW-axes-*value*, Z-axes-*value*)
- Range : 0 to 100
- Precision : 1 %
- Default : 80

Specify ratio of acceleration and deceleration for PTP motion.

In standard usage, the value of 70 through 100 has to be specified. If the smaller value is set, acceleration and deceleration becomes slower.

Refer to "13.2.1 ACCEL" in operation manual of a robot controller.

(4) Acceleration and deceleration for CPC motion

- Format : CP Accel= *Value*
- Range : 0 to 100
- Precision : 1 %
- Default : 80

Specify ratio of acceleration and deceleration for CPC motion.

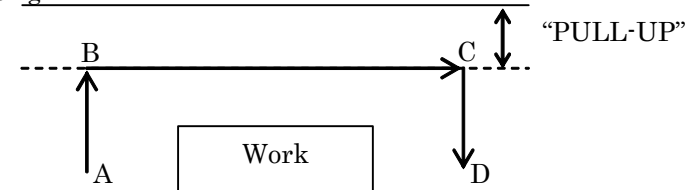
Refer to "13.1.3 CPC CONSTANT" -> "CPC ACCEL/DECEL" in operation manual of a robot controller.

(5) Pull-up of Z axis

- Format : Pull \perp Up= *Value*
- Range : 5.0 to Z area limit
- Precision : 0.001 mm
- Default : 10.0

Specify Z upper position of auto pull-up.

Z-axis origin

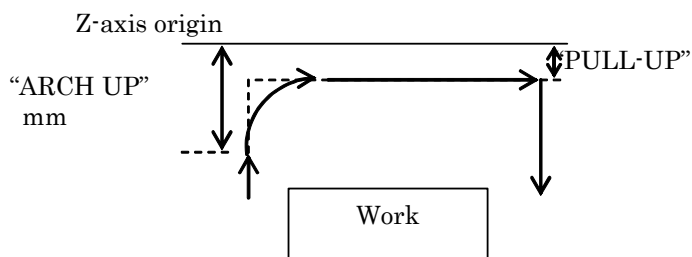


Refer to "13.1.1 MOTION" -> "PULL-UP" in operation manual of a robot controller.

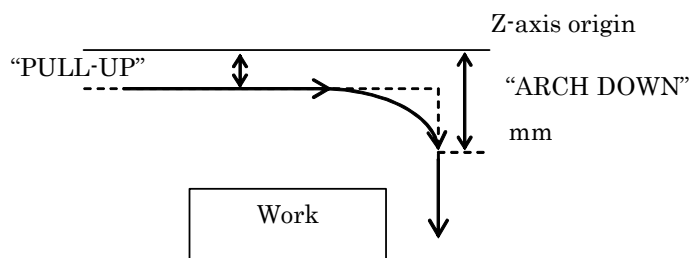
(6) Arch motion data

- Format : Arch=(*Arch-up-value*, *Arch-down-value*)
- Range : 0.0 to Z area limit
- Precision : 0.001 mm
- Default : 0.0

Arch-up-value is the distance of arch-up motion, which is the length from the position of Z-axis origin.



Arch-down-value is the distance of arch-down motion, which is the length from the position of Z-axis origin.



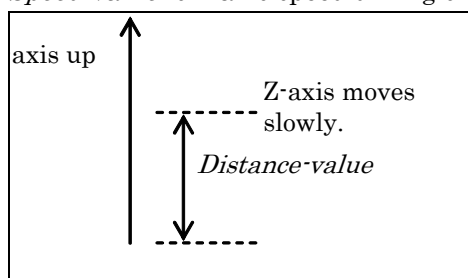
Refer to “13.1.1 MOTION” -> “ARCH UP”, “ARCH DOWN” in operation manual of a robot controller.

(7) Slow-up motion data

- Format : Slowup=(*Distance-value*, *Speed-value*)
- Range : *Distance-value* 0.0 to Z area limit
- *Speed-value* 0 to 99
- Precision : *Distance-value* 0.001 mm
- *Speed-value* 1 %
- Default : *Distance-value* 20.0
- *Speed-value* 20

Distance-value is the distance of slow-up motion, where Z-axis moves by low speed after Z-axis starts.

Speed-value is Z-axis speed during slow-up motion.



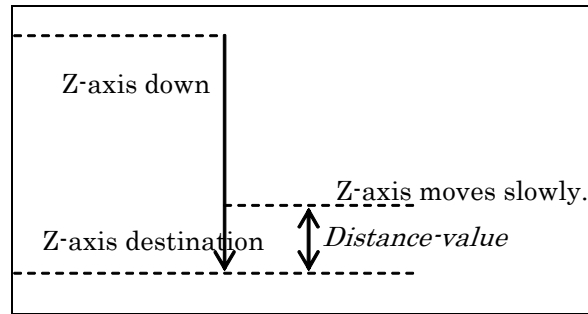
Refer to “13.1.1 MOTION” -> “UP DIS”, “UP SPEED” in operation manual of a robot controller.

(8) Insert motion data

- Format : Slowdown=(*Distance-value*, *Speed-value*)
- Range : *Distance-value* 0.0 to Z area limit
- *Speed-value* 0 to 99
- Precision : *Distance-value* 0.001 mm
- *Speed-value* 1 %
- Default : *Distance-value* 20.0
- *Speed-value* 20

Distance-value is the distance of slow-down motion. Z-axis decelerates from the position of the Z-axis destination minus *Distance-value*.

Speed-value is Z-axis speed during slow-down motion.



Refer to “13.1.1 MOTION” -> “INS DIS”, “INS SPEED” in operation manual of a robot controller.

- Refer to operation manual of a robot controller.

SetPriority

(Function)

- Function
Set and change the priority of a job and returns the old priority.

- Format
SetPriority(*Job-name*, *Priority*)

- Arguments and Return value

Parameter		Explanation
Arguments	<i>Job-name</i>	String type expression of a job name.
	<i>Priority</i>	Numeric expression of the new job priority.
Return value		The old job priority.

- Example
old.priority% = SetPriority("robot1", 10)

- Explanation
 - ◆ The priority of a job has the level as 1 (the lowest) to 10 (the highest). After STP system starts or a program is downloaded, the priority of each job is set as 1 and each job runs equally.
 - ◆ The priority of a job means as follows.
Among jobs with the priority 1, if you increase the priority of one job to 10, this job can run 10 steps during other jobs running 1 step. In the same way, if the priority ratio of 4 jobs has changed to 10:8:6:4, the executing step ratio changes to 5:4:3:2. If the all priorities have the same number (for example 10: 10:10), each job runs equally since the executing step ratio is flat as 1: 1:1:
 - ◆ If the specified job is not found, a job error occurs.
- See also GetPriority.

SetRobNo

(Function)

- Function
Sets a robot number for the robot communication of a current job.

- Format
SetRobNo(*Robot-number*)

- Argument and Return value

Parameter		Explanation
Argument	<i>Robot-number</i>	Numeric expression specifying a robot number of a current job from 1 through 999.
Return value		Nothing

- Example
SetRobNo(1)

- Explanation
 - ◆ SetRobNo function sets a robot number of a current job, which is used for the robot communication. After SetRobNo function is executed, a job program can omit specifying a robot number when a robot control command such as Move, Set, Ref and so on.
 - ◆ *Robot-number* must be specified by the value that is set to [MAINTENANCE]-[MAINTENANCE DATA]-[STATION NO.] in S.G. data of the controller. In HNC-590 series, HAX-8XX controller, default number 1, 2, 3, 4 is assigned to four virtual robots.
 - ◆ After STP system starts, a program is downloaded or ClearRobNo function is executed, robot number of a job is set by the value -1.
 - ◆ If *Robot-number* is specified by a constant, a compiling error occurs when the specified value is out of 1 through 999.
 - ◆ If *Robot-number* is specified by a variable, a job error occurs when the specified value is out of 1 through 999.
- See also GetRobNo, ClearRobNo.

Sgn

(Function)

- Function
Gets the sign of a number.

- Format
 $\text{Sgn}(\text{Numeric-expression})$

- Argument and Return value

Parameter		Explanation
Argument	<i>Numeric-expression</i>	A numeric expression specifying a number.
Return value		The function returns the following integer value to check the number.
		Number < 0 -1
		Number = 0 0
		Number > 0 1

- Example
 $\text{b\%} = \text{Sgn}(-10.34)$ -1 is substituted for b%.

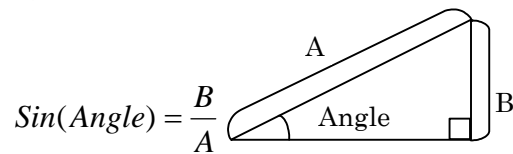
Sin

(Function)

- Function
Gets the value of sine.
- Format
 $\text{Sin}(\text{Numeric-expression})$
- Argument and Return value

Parameter		Explanation
Argument	<i>Numeric-expression</i>	Angle by radian.
Return value		Sine of the specified value is returned. The value is from -1.0 through +1.0.

- Example
 $\text{angle!} = \text{Pai} / 3$
 $\text{x!} = \text{Sin}(\text{angle!})$
- Explanation
The ratio of “B” to “A” is returned specifying the angle “*Angle*” in the figure.



- See also Atn, Cos, Tan.

Space\$

(Function)

- Function
Gets a string containing serial space characters.

- Format
Space\$(*Length*)

- Argument and Return value

Parameter		Explanation
Argument	<i>Length</i>	A numeric expression specifying the length of serial spaces. Valid range is from 0 through 255.
Return value		Serial space characters (code:&H20) with the specified length.

- Example
a\$="Manual"
b\$=a\$+Space\$(2)+"ABC" "Manual␣␣ABC" is substituted for b\$.
- Explanation
A null string is returned if *Length* is zero.

Sqr

(Function)

- Function
Gets the square root of a number.

- Format
Sqr(Numeric-expression)

- Argument and Return value

Parameter		Explanation
Argument	<i>Numeric-expression</i>	Numeric expression specifying a number.
Return value		The square root of the specified value.

- Example
`a! = Sqr(x!*x! + y!*y!)`

Str\$

(Function)

- Function
Converts a number to a string.

- Format
 Str(*Numeric-expression*)$

- Argument and Return value

Parameter		Explanation
Argument	<i>Numeric-expression</i>	Numeric expression specifying a number.
Return value		A converted string.

- Example
 $\text{num1!}=10.2$: $\text{num2\%}=-12$
 $\text{a\$}=\text{Str}$(\text{num1!})+\text{"ABC"}$ “ $\sqcup 10.2\text{ABC}$ ” is substituted for a\$.
 $\text{b\$}=\text{Str}$(\text{num2\%})+\text{"ABC"}$ “ -12ABC ” is substituted for b\$.

- Explanation
The first character of the returned string is a sign character of a number.
In case of minus number, it is a minus character (-). In case of plus number, it is a space character.

String\$

(Function)

- Function
Gets a repeating character string of the specified length.

- Format
String\$(*Length*, *String-expression*)
String\$(*Length*, *Character-code*)

- Arguments and Return value

Parameter		Explanation
Arguments	<i>Length</i>	A numeric expression specifying the length of the returned string.
	<i>String-expression</i>	A string expression in which the first character is repeated.
	<i>Character-code</i>	A numeric expression specifying a repeating character code. Valid value is from 0 through 255 (&HFF).
Return value		A string repeating the specified character.

- Example
a\$=String\$(3, “#”) “”###” is substituted for a\$.
b\$=String\$(4, &H40) “”@@@@” is substituted for b\$.
c\$=String\$(5, “HrBasic”) “”HHHHH” is substituted for c\$.
- Explanation
In case of *String-expression*, the first character of the specified string is repeated.

Tan

(Function)

- Function
Gets the value of tangent.

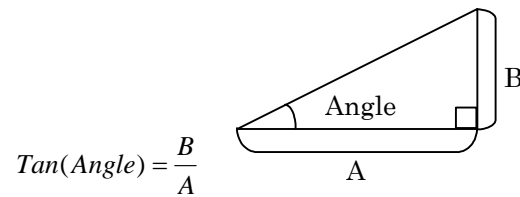
- Format
 $\text{Tan}(\text{Numeric-expression})$

- Argument and Return value

Parameter		Explanation
Argument	<i>Numeric-expression</i>	Angle by radian.
Return value		Tangent of the specified value is returned.

- Example
angle! = Pai / 4
a! = Tan(angle!)

- Explanation
The ratio of “B” to “A” is returned specifying the angle “*Angle*” in the figure.



$$\text{Tan}(\text{Angle}) = \frac{B}{A}$$

- See also Atn, Cos, Sin.

Time\$

(Statement)

● Function

Sets time to the system calendar.

● Format

Time\$ = *Time-String*

● Argument

Parameter	Explanation
Argument	Nothing

● Example

Time\$="13:10:00" ‘Set 13:00:00 to system calendar.

● Explanation

- ◆ This statement is used at the left side of substitution.
- ◆ The substituted string of a constant or variable has to be the following format.

“hh:mm:ss”

Hour (00 to 23) ————↑

Minute(00 to 59) ————↑

Second(00 to 59) ————↑

● See also Date\$, Time\$ (function).

Time\$

(Function)

- Function
Gets current time of the system calendar.

- Format
Time\$

- Argument and Return value

Parameter	Explanation
Argument	Nothing
Return value	A sting that contains current time with the following format. "hh:mm:ss"

- Example
b\$ = Time\$ ' Current time is substituted for b\$.

- Explanation
The value of Time\$ is a string data, but it is not available in the string expression combined with string operators. For example, a\$=Date\$+b\$ is not available. In this case, a program has to be described as follows.
t\$ = Time\$
a\$ = t\$ + b\$

- See also Date\$, Time\$ (statement).

TimeOut

(Function)

● Function

Gets the timeout state after the execution of Wait statement.

● Format

TimeOut

● Return value

Parameter	Explanation
Return value	<p>A logical value whether the last Wait statement has resulted in timeout.</p> <p>Timeout: true (-1)</p> <p>Not timeout: false (0)</p>

● Example

Wait INB(I.BUTTON)=1, 3.0 ‘ Wait for button ON for 3.0 seconds.

If TimeOut Then GoTo *TOUT.ERR ‘ Timeout

● Explanation

- ◆ The function returns the timeout state by a logical value after the last Wait statement has been executed. The timeout state is held until the next Wait statement is executed.
- ◆ The timeout state is kept for each job.
- ◆ Initial value of TimeOut function is zero (false value).

● See also Wait.

Val

(Function)

- Function
Converts a string to a number.

- Format
`Val(String-expression)`

- Argument and Return value

Parameter		Explanation
Argument	<i>String-expression</i>	A string expression specifying a number. A decimal, hexadecimal or real expression is available, but Octal expression is not available.
Return value		Value to which the specified string is converted.

- Example

`a% = Val("123")` ' 123 is substituted for a%.
`b# = Val("&HFF")` ' 255.0 is substituted for b#.
`c% = Val("&H15")` ' 15 is substituted for c%.
`d& = Val("AB")` ' Zero is substituted for d&.

- Explanation

- ◆ If the first character of *String-expression* is not '+', '-', '&' or numeral, the function returns zero.
- ◆ If it is detected that the character is not numeric during scanning the string, the rest of characters is neglected. In case of hexadecimal string, a character of 'A' to 'F' is regarded as numerical.
- ◆ Space (&H20) in the string is neglected to scan.

- See also Str\$.

Wait

(Statement)

● Function

Waits at its step until the specified condition is satisfied, or until the specified time passes.

● Format

Wait \sqsubset *Condition* [, *Time*]

● Argument

Parameter	Explanation
<i>Condition</i>	A conditional expression resulting in the logical value, true (-1) or false (0).
<i>Time</i>	The statement waits until this time passes. Specify a numeric expression containing time value with the precision of 0.001 sec. Available range is from 0 through 2147483.647 sec.

● Example

Wait INB(3)=0, 8 ‘ Wait for INB(3) OFF for 8 seconds.

● Explanation

- ◆ If the result of *Condition* is true (-1), Wait statement returns immediately and then the next program step is executed. If the result of *Condition* is false (0), Wait statement waits until it becomes true (-1).
- ◆ Wait statement exits after *Time* passes even if the result of *Condition* is false. After this case, TimeOut function returns a true value (-1). The following program check whether the last Wait statement has exited by timeout.
If TimeOut Then GoTo *T.OUT ‘ Wait timeout
- ◆ If *Time* is omitted, Wait statement waits infinitely until *Condition* is satisfied.
- ◆ Wait statement does not affect other job.
- ◆ After a job is stopped by Job Off during waiting by Wait statement, Job Start restarts a job to execute Wait statement still. However, countdown of Wait timer is executed during the state of Job Off.

● See also TimeOut.

WriteHrcs

(Statement)

- **Function**
Writes the specified string by HRCS protocol.

- **Format**
WriteHrcs \sqcup #*File-number*, *String*

- **Arguments**

Parameter	Explanation
<i>File-number</i>	A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, "#" can be omitted but in case of a numeral constant, it cannot be omitted.
<i>String</i>	A string of TEXT part in a HRCS protocol frame to send.

- **Example**
a\$="HrBasic Manual"
WriteHrcs #1, a\$

- **Explanation**
A HRCS protocol frame is shown below.
WriteHrcs statement sets the specified string to TEXT part in the figure and then sends a HRCS protocol frame.

S _{TX}	TEXT	E _{TX}	L _{RC}
-----------------	------	-----------------	-----------------

- **STX (Start of Text)**
The head of HRCS protocol frame. (&H02)
 - **ETX (End of Text)**
The end of HRCS protocol frame. (&H03)
 - **LRC (Longitudinal Redundancy Check)**
LRCⁱ is a check code for communication data calculated by exclusive OR of bytes in TEXT to ETX.
- ◆ See also RchkHrcs, WriteHrcs.

ⁱ See "Appendix B LRC Calculation".

Xor

(Operator)

- **Function**
Executes a logical exclusion of two numbers.
- **Format**
Numeric-expression#1⊔Xor⊔*Numeric-expression #2*

- **Arguments**

Parameter	Explanation
<i>Numeric-expression#1</i>	A numeric expression.
<i>Numeric-expression#2</i>	A numeric expression.

- **Example**
a% = &H00FF%
b% = &H0F0F%
c% = a% Xor b% ‘ &H0FF0% substituted for c%.

- **Explanation**
 - The following calculation is performed.

X	Y	X xor Y
1	1	0
1	0	1
0	1	1
0	0	0

 - See “6.4.3 Logical Operator”.

Appendix

Appendix-A ASCII Codes

Lower 4 bits	Upper 4 bits								
	Hex	0	1	2	3	4	5	6	7
	0	NUL	DEL	SP	0	@	P	`	p
	1	SOH	DC1	!	1	A	Q	a	q
	2	STX	DC2	"	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	\$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
	7	BEL	ETB	'	7	G	W	g	w
	8	BS	CAN	(8	H	X	h	x
	9	HT	EM)	9	I	Y	i	y
	A	LF	BUS	*	:	J	Z	j	z
	B	VT	ESC	+	;	K	[k	{
	C	FF	FS	,	<	L	¥	l	
	D	CR	GS	-	=	M]	m	}
	E	SO	RS	.	>	N	^	n	~
	F	SI	US	/	?	O	_	o	DEL

For example, the code of "A" is &H42 as hexadecimal expression.

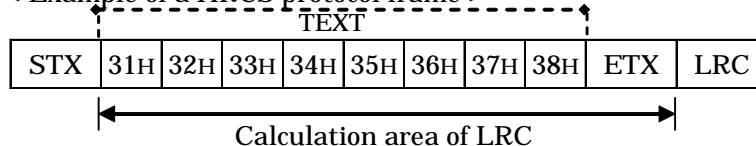
Appendix-B LRC Caluculation

How to calculate LRC(Longitudinal Redundancy Check) is described below.

(1) LRC calculation area

LRC is made by excusive-or operation of all bytes from the next byte of STX through ETX.

< Example of a HRCS protocol frame >



(2) LRC calculation

- A unit of calculation is each byte of communication data.
- Calculate exclusive-or (XOR)ⁱ of the first byte and the second byte in TEXT and the symbol, X1 represents the result.
- Calculate XOR of X1 and the third byte in TEXT and the X2 represents the result.
- Calculate XOR of all TEXT bytes similarly, and finally, LRC is calculated by XOR of the result and ETX.

< LRC calculation of HRCS protocol frame example >

ASCII TEXT	Hex	Binary	XOR
1	31H	00110001	
2	32H	00110010	00000011 X1
3	33H	00110011	00110000 X2
4	34H	00110100	00000100 X3
5	35H	00110101	00110001 X4
6	36H	00110110	00000111 X4
7	37H	00110111	00110000 X5
8	38H	00111000	00001000 X6
ETX	03H	00000011	00001011 LRC

ⁱ Truth table of XOR --- A XOR B = C

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Appendix-C Compiling Errors

Code (Dec)	Code (Hex)	Error message	Explanation
0	00	Syntax error	Invalid character or keyword is used, or sequence of command or parameter is illegal.
1	01	Division by Zero	A number is divided by the constant zero.
2	02	Duplicate label	The same label is defined two times or more.
3	03	FOR without NEXT	For statement is described, but Next statement is not found.
4	04	Illegal function call	Number of arguments is illegal, or the type of argument is invalid.
5	05	Line buffer overflow	Divide a program described in one line.
6	06	NEXT without FOR	Next statement is described, but For statement is not found.
7	07	Out of memory	Variables, labels or subroutines are too much. Make a program more structured, for example, using subroutines.
8	08	IF formula too complex	A condition expression of If statement is too complex. For example, calculate the condition in advance.
9	09	Overflow	Value of the specified number is out of valid range.
10	0A	RETURN without GOSUB	Return statement is not found in a subroutine.
11	0B	String formula too complex	Divide a calculating expression.
12	0C	String too long	Reduce the number of characters in a string.
13	0D	Subscript out of range	The specified subscript of array exceeds the limit defined by declaration.
14	0E	Type mismatch	Type combination of the left side and the right side are invalid.
15	0F	Undefined label	The specified label is not defined in a job.
16	10	Undefined line number	(Unused)
17	11	Duplicate variable	Reserved memory is specified to Dim, Global, DimNet statement, or the same variable name is specified to Global and DimNet statement.
18	12	Variable name too long	Length of a variable name exceeds the maximum.
19	13	Label name too long	Length of a label name exceeds the maximum.
20	14	FOR statement missing	(Unused)
21	15	IF statement missing	Description of a If...Then...Else...EndIf sentence is invalid.
22	16	Misplace ELSE	An Else sentence is described independently.
23	17	GOTO statement missing label	Unexpected error occurs in a GoTo sentence.
24	18	Too much code defined in file	(Unused)
25	19	Too much code defined in line	Compiled intermediate code of one line exceeds the maximum volume. Divide a program of the line.
26	1A	Duplicate Definition	The same name of array variable is defined two times or more.
27	1B	Too much dimension number	The number of array dimensions has to be one through three.
28	1C	Constant expression required	Array dimension has be specified by a constant.

Code (Dec)	Code (Hex)	Error message	Explanation
29	1D	Undefined JOBNAME statement	Job name statement is not found in a job.
30	1E	Illegal octal digit	An invalid character is found in an octal expression. See "6.1.3 Integer Type Literal".
31	1F	Illegal hex digit	An invalid character is found in a hexadecimal expression. See "6.1.3 Integer Type Literal".
32	20	Too many decimal points	(Unused)
33	21	DIM statement missing	The specified variable is not array.
34	22	Invalid indirection	A variable that is not defined as array is used in a program.
35	23	Invalid position number	The specified position address is invalid.
36	24	EOF number out of range	The specified file number for Eof function is out of valid range.
37	25	Bad file mode	(Unused)
38	26	File number out of range	The specified file number is out of valid range.
39	27	DELAY statement missing	The minus value is specified to an argument, or the specified variable has the invalid type.
40	28	Port number out of range	The specified COM port number is out of valid range.
41	29	ERROR statement missing	(Unused)
42	2A	LINE INPUT statement missing	An argument has to be a string variable.
43	2B	MACRO statement missing	The invalid syntax is found in a MACRO sentence.
44	2C	Not found INCLUDE file	The header file or the header directory set in HBDE does not exist.
45	2D	Specified axis missing	The number of specified axes is invalid.
46	2E	MOVE statement missing	The invalid syntax is found in a Move sentence.
47	2F	Position count mismatch	Two positions or more are needed for Move statement according to a motion type.
48	30	Illegal precious number	The invalid syntax is found in the specified precision in a Move sentence.
49	31	Option missing	Invalid variable or memory is specified.
50	32	Component expression missing	Invalid expression is found in axis data or arm direction.
51	33	Duplicate SEQ statement	(Unused)
52	34	SEQEND without SEQ	SeqEnd statement is described, but Seq statement is not found.
53	35	DRIVE statement missing	(Unused)
54	36	JOG statement missing	(Unused)
55	37	OPEN statement missing	The invalid format of the specified communication parameter or the invalid syntax is founded in an Open sentence.
56	38	Out of range in numeric constant	Value of the specified numeric constant is out of valid range.
57	39	SEQ without SEQEND	Seq statement is described, but SeqEnd statement is not found.
58	3A	WHILE without WEND	(Unused)
59	3B	WEND without WHILE	(Unused)
60	3C	Not enough memory	(Unused)

Code (Dec)	Code (Hex)	Error message	Explanation
61	3D	DEFINE statement missing	The invalid description of an argument is found in a Define sentence.
62	3E	Duplicate JOB NAME	The same job name is used in the different job programs.
63	3F	Undefined JOB NAME	A job with the specified job name is not found in the linked program.
64	40	Bad object file	Invalid contents of an object file (.obj) are found on compilation or linking. There is a possibility that the file is destroyed.
65	41	Bad label file	(Unused)
66	42	Bad local variables file	Invalid contents of a local variable file (.var) are found on compilation or linking. There is a possibility that the file is destroyed.
67	43	Bad global variables file	Invalid contents of a global variable file (.gbl) are found on compilation or linking. There is a possibility that the file is destroyed.
68	44	More than 32 jobs defined	The number of linked jobs exceeds the maximum.
69	45	Too many global variables	The number of all global variables exceeds the maximum.
70	46	Bad make file	Invalid contents are found in a make file. There is a possibility that the file is destroyed.
71	47	ENDIF without IF	If...EndIf combination is incorrect in a job program.
72	48	SELECT CASE nesting over	The number of Select Case nests exceeds the maximum. Extract the program in a Case block as a subroutine to be more structured.
73	49	SELECT CASE statement missing	Case, Case Else, End Select statement is described, but Select Case statement is not found.
74	4A	IS operator is not tail	Is operator in Case statement has to be described the tail of the sentence.
75	4B	CASE ELSE statement missing	Case Else statement has to be described even if there is no procedure for it.
76	4C	END SELECT statement missing	End Select statement is needed after a Case Else block to terminate a Select Case block.
77	4D	Sequence error in SELECT CASE block	Procedure in a Select Case block is too complex. It has to be simpler structure by means of structured programming using subroutines.
78	4E	CASE statements more than 127	Reduce the number of Case sentences to integrate several conditions to one Case sentence.
79	4F	Statement between SELECT CASE and CASE	There has not to be an executable sentence between Select Case and Case statement.
80	50	WAIT expression not logical	The condition expression in Wait sentence is not logical expression.
81	51	Illegal assignment	A value has not to be substituted for HERE, STATUS.
82	52	Illegal robot type	Invalid value is specified to RobType command.
83	53	Bad argument type of function or statement	The type of the specified constant or variable is not allowed for the function or statement.
84	54	Illegal value of argument	Value of the specified constant for the function or statement is out of valid range.

Code (Dec)	Code (Hex)	Error message	Explanation
85	55	Conditions of CASE statement too long	Condition expression is too long in a Case sentence. Calculate the condition in advance, or divide the condition to multiple Case sentences.
86	56	Path name not character string type	The argument specifying a file path name is not string type.
87	57	Parameter not character string type	The argument specifying the parameter is not string type.
88	58	Illegal type of file number	Number type of the specified file number is not integer.
89	59	Invalid COM port number	The specified COM port number is out of valid range.
90	5A	Arm,Local,F,M,S cannot be specified	Arm direction, coordinate type, M-data, F-code, S-code cannot be described to a component expression of position data.
91	5B	One of Local,F,M,S cannot be omitted	All of coordinate type, M-data, F-code, S-code have to be described to a component expression of position data.
92	5C	Axis data, Arm cannot be omitted	Axis value or arm direction cannot be omitted in a component expression of position data.
93	5D	Variable that is not a position is specified	Only position data can be specified to an argument of the function or statement.
94	5E	Too many robots in the list	The number of robots specified to RobNoList command exceeds the maximum.
95	5F	Invalid robot number	The specified robot number is out of valid range.
96	60	Illegal type of robot number	The type of the specified robot number is not integer.
97	61	Robot number duplicated	A robot number specified in RobNoList command is duplicated.
98	62	Real number specified in integer constant	In a constant, the value is real type though type declaration is integer.
99	63	The right side is out of range for the left side	In substitution, the substituted value is out of valid range for the left side.
100	64	Only available string variable or constant	Only a string constant or variable can be specified to a argument of the function or statement.
101	65	Timer value is out of range	The value set to a timer is out of valid range.
102	66	Too many arguments of function	The number of arguments for the function exceeds the maximum.
103	67	Only string variable available	Only a string variable can be specified to a argument of the function or statement.
104	68	Cannot use the function without return value	The function that returns nothing cannot be described here.
105	69	Only user variable can be specified for argument of function	Only a variable can be specified to a output argument in the function. A constant cannot be specified.
106	6A	Too many axes for arguments	The number of specified axes for the function or statement exceeds the maximum.
107	6B	Bad usage of operator	The described operator cannot be used here.
108	6C	Nests of FOR-NEXT overflow	The number of For-Next nests exceeds the maximum. Extract the program in a For-Next block as a subroutine to be more structured.

Code (Dec)	Code (Hex)	Error message	Explanation
109	6D	Nests of IF-ENDIF overflow	The number of If-EndIf nests exceeds the maximum. Extract the program in a If-EndIf block as a subroutine to be more structured.
110	6E	Nests of WHILE-WEND overflow	The number of While-Wend nests exceeds the maximum. Extract the program in a While-Wend block as a subroutine to be more structured.
111	6F	Variable in NEXT not matched to FOR	A variable in a Next sentence has be the same as the corresponded For sentence.
112	70	Invalid character code (including space or tab)	2-bytes code of a character is detected in a executable sentence.
113	71	Relative position missing	It is necessary to specify the relative position.
114	72	Only long integer variable available	Only 32-bits long integer (&) can be specified. A constant cannot be specified.
115	73	Invalid axis number	The specified axis number is out of valid range.

Appendix-D Running Job Errors

Code (Dec)	Code (Hex)	Error message	Explanation
1	01	Illegal program pointer	There may be OS (Operating System) trouble.
2	02	Bus error exception	STP: There may be OS trouble. WinSTP: Unused
3	03	Address error exception	STP: There may be OS trouble. WinSTP: Unused
4	04	Illegal instruction exception	STP: There may be OS trouble. WinSTP: Unused
5	05	Zero divide exception	STP: There may be OS trouble. WinSTP: Unused
6	06	CHK instruction exception	(Unused)
7	07	TRAPV instruction exception	(Unused)
8	08	Privilege violation	(Unused)
9	09	Format error	(Unused)
10	0A	Line 1010 emulator exception	(Unused)
11	0B	Line 1111 emulator exception	(Unused)
12	0C	(Unused)	(Unused)
13	0D	(Unused)	(Unused)
14	0E	Arithmetic co-processor exception	(Unused)
15	0F	Program not downloaded	Download a program to STP.
16	10	Calculation overflow	The result of calculation exceeds the maximum value.
17	11	Divided by Zero	Dividing a value by zero is executed.
18	12	RESUME without error	Resume statement is available only after a job error has occurred.
19	13	Written to out of area	Out of variable memory area is accessed when substitution. There is a possibility of OS trouble.
20	14	Invalid internal data	An unexpected internal error. For example, invalid data is contained in the memory that a program cannot access. There is a possibility of OS trouble.
21	15	Array accessed out of range	A subscript indicates out of the defined area of the array.
22	16	Nests of FOR-NEXT overflow	Extract the program in a For-Next block as a subroutine to be more structured.
23	17	FOR and NEXT not a pair	For or Next statement is not described.
24	18	Undefined command	The command (function or statement) is not supported at present.
25	19	RETURN without GOSUB	GoSub or Return statement is not executed as a pair. There is a possibility that a program has jumped to a subroutine by GoTo statement instead of GuSub.
26	1A	Incorrect usage of command or function	The followings cause the error. Type of argument is invalid. The parameter format specified to a function or statement is invalid. A sentence has incorrect syntax.
27	1B	OPEN already executed	Open statement is executed for the file or communication port that is already opened.

Code (Dec)	Code (Hex)	Error message	Explanation
28	1C	File accessed without OPEN	The file or communication port that is not opened is accessed.
29	1D	Data nothing to receive	(Unused)
30	1E	Calculation underflow	The result of calculation becomes less than the minimum value.
31	1F	Receiving buffer overflow	(Unused)
32	20	Stack error at FOR-NEXT execution	(Unused)
33	21	Character string expression too complex	Divide a string expression to the smaller ones.
34	22	Character string too long	The string has to be treated as the smaller ones.
35	23	Type of variable or data mismatched	Invalid type of a variable or data is specified.
36	24	Incorrect format command	(Unused)
37	25	Cannot convert data type	Automatic type conversion cannot be executed. Check the combination of two types of data.
38	26	Arithmetic co-processor error	(Unused)
39	27	Data receiving (parity, overrun, framing) error	The error occurs on RS232C communication. There are possibilities of communication speed inconsistency, noise, or disconnection.
40	28	OPENed file already used	Open statement is executed with the file number that is already used.
41	29	Stack control error	OS cannot manage the inner stack correctly. There is a possibility of OS trouble.
42	2A	Nests of GOSUB-RETURN overflow	GoSub or Return statement is not executed as a pair. There is a possibility that a program has returned to main program by GoTo statement instead of Return.
43	2B	COM line not connected	On RS232C transmission, DSR signal becomes OFF. Check the disconnection.
44	2C	Sending buffer overflow	(Unused)
45	2D	JOB START without JOB OFF	The specified job has to be Job Off state before Job Start statement is executed.
46	2E	Position memory access out of range	(Unused)
47	2F	Network already opened	The already opened network is opened without close.
48	30	Network open overflow	The number of opened networks exceeds the maximum.
49	31	Network not opened	A network is accessed without open.
50	32	Network writing size error	Writing size specified to NetWrite function is out of valid range.
51	33	Network CR (Communication Reference) undefined	Network definition is not created correctly.
52	34	Own station number specified	Own station number is specified for the network communication.
53	35	Specified COM port not implemented.	The specified COM port is not available, or not implemented.

Code (Dec)	Code (Hex)	Error message	Explanation
54	36	Communication buffer overflow	The sending buffer or the receiving buffer is full. In case of sending, sending cycle of a program is too fast. In case of receiving, receiving cycle of a program is too slow.
55	37	Invalid baud rate for serial COM	The specified value of RS232C communication speed is invalid.
56	38	Invalid parity for serial COM	The specified value of RS232C parity is invalid.
57	39	Invalid character length for serial COM	The specified value of RS232C character length is invalid.
58	3A	Invalid stop bits for serial COM	The specified value of RS232C stop bits is invalid.
59	3B	No available file number	All file numbers are used now. Reduce the number of concurrently used files.
60	3C	Specified COM port not available	The specified COM port number is out of valid range.
61	3D	File number out of range	The specified file number is out of valid range.
62	3E	Can not open COM port	Setup of the specified COM port fails. There is a possibility of hardware trouble. In WinSTP, the PC COM port is not available, so check the device.
63	3F	Serial COM parameter format error	The format of the specified RS232C communication parameter is invalid.
64	40	No such file or directory	The specified file or directory is not found.
65	41	File open error	The specified storage file cannot be opened.
66	42	File write error	The specified storage file cannot be written.
67	43	File read error	The specified storage file cannot be read.
68	44	Received HRCS data format error	The format of a received HRCS protocol frame is invalid.
69	45	Communication TxRDY off	RS232C TxRDY signal becomes OFF unexpectedly.
70	46	Communication RxRDY off	RS232C RxRDY signal becomes OFF unexpectedly.
71	47	System-call or API error	In WinSTP, a Windows API (System-call) error occurs.
72	48	Mode error in communication hardware	In WinSTP communication, a hardware mode error occurs.
73	49	General I/O error in communication hardware	In WinSTP communication, a general I/O error occurs.
74	4A	Break status detected in communication hardware	In WinSTP communication, the break signal is detected.
75	4B	Transmission timeout in communication hardware	In WinSTP communication, a transmission timeout occurs.
76	4C	Other error of communication	In communication, a miscellaneous error occurs.
77	4D	Re-open PROCON or HOST failed	When closing a communication port, re-setup of the hardware fails.
78	4E	Specified file cannot communicate	The specified file number is not for a communication port.
79	4F	Undefined error detected	Undefined error about communication. There is a possibility of OS trouble.

Code (Dec)	Code (Hex)	Error message	Explanation
80	50	Communication timeout with the robot	A robot does not respond.
81	51	Error received from the robot	A robot responds an error.
82	52	Illegal response format of the robot	The response format of the robot is illegal or not supported by STP.
83	53	Data Not registered in the robot	(Unused)
84	54	Not SEQ mode in the robot	(Unused)
85	55	Too many robots for communication	The number of robots that are concurrently communicated with STP exceeds the maximum.
86	56	Duplicated sending to robots	A next communication command is sent to a robot before the response.
87	57	Received position data invalid	Invalid position data is received from a robot.
88	58	Robot response without robot no.	A robot number is not found in the response from a robot that has to manage the robot number.
89	59	Different robot no. in response	The robot number received from a robot is not same as the sent one.
90	5A	Robot memory index out of range	The specified index (subscript) of robot memory is out of valid range.
91	5B	Robot is not ONLINE mode	Operation mode of the robot to communicate is not ONLINE.
92	5C	Robot motion stopped incompletely	A robot stopped before it completes the motion to the programmed position.
93	5D	Robot axis no. error	The specified axis number is out of valid range.
94	5E	Operands overflow	In robot function or statement, the number of specified arguments (operands) exceeds the maximum.
95	5F	Number of robot axes overflow	In robot function or statement, the number of specified axes exceeds the maximum.
96	60	OPEN in the through mode	A communication port in Through Mode is opened.
97	61	CLOSE in the through mode	A communication port in Through Mode is closed.
98	62	Sending data too long	The size of sending data at one time exceeds the maximum.
99	63	Minus value specified as a parameter	Minus value is specified to an argument of a robot function or statement.
100	64	Null string detected	Null string is specified to a function or statement.
101	65	Accessed out of string	A function or statement for string operation accesses the area out of the string.
102	66	Invalid code specified	A character code out of 0 to 255 is specified to a function or statement for string operation.
103	67	Cannot execute for current robot type	For the controller specified by RobType, the function or statement cannot be executed.
104	68	Not available to write time or date	In WinSTP, system date and time cannot be written.

Code (Dec)	Code (Hex)	Error message	Explanation
105	69	Format error of time or date	The format to write system date or time is invalid.
106	6A	Bad robot no. in response from robot	The format of the robot number received from a robot is invalid.
107	6B	Robot number out of range	The specified robot number is out of valid range.
108	6C	Robot number duplicated	The same robot number is registered two times and more.
109	6D	Robot number not found in OPEN list	The specified robot number is not defined by RobNoList of Open statement.
110	6E	Specified job not downloaded	The specified job has not been downloaded.
111	6F	LRC error of HRCS protocol	LRC error is detected onHRCS protocol communication.
112	70	Too many arguments	The number of the specified arguments exceeds the maximum.
113	71	Invalid parameter descriptor	The format of a parameter descriptor is invalid.
114	72	Invalid type of argument	Type of the specified argument of a function or statement is invalid.
115	73	Parameter value out of range	The value of parameter specified to a function or statement is out of valid range.
116	74	Cannot execute when robot is moving	The function or statement cannot be executed when a robot is moving.
117	75	Invalid position data	The content of position data is invalid.
118	76	Position data empty	All elements of position data are zero.
119	77	M,F,S data all zero	All M-data, S-code and F-code of position data are zero.
120	78	Robot controlled by other job	(Unused)
121	79	Number of FOR-NEXT overflow	The number of all For-Next statements exceeds the maximum.
122	7A	MOVE executed without ENABLE	Enable statement has to be executed before Move statement.
123	7B	Invalid job priority	The value of the specified job priority is out of valid range.
124	7C	Specified job not found	The specified job is not found in the system.
125	7D	Cannot execute on the current platform	In the current platform (type of a controller or STP), the function or statement is not available.
126	7E	Invalid data of robot collision check	The downloaded data of robot collision check is invalid.
127	7F	Robot collision check data overflow	Data overflow occurs on the setup of robot collision check when Open or Close statement is executed.
128	80	Safety length for robot collision not defined	In the downloaded data of robot collision check, safety length is not defined.
129	81	Internal function error of robot collision	An unexpected error occurs in the internal function of robot collision check.
130	82	Robot collision detected	Robot collision is detected by the internal function of robot collision check.
131	83	Local-World coordinates conversion data not defined	In the downloaded data of robot collision check, Local-World coordinates conversion data is not defined.

Code (Dec)	Code (Hex)	Error message	Explanation
132	84	Invalid specified robot number	The specified robot number is not defined in a robot controller.
133	85	SG/SP data group error	The specified group name of SG/SP data is incorrect.
134	86	SG/SP data invalid	The specified value of SG/SP data is invalid.

Appendix-E Standard Coding Rules

HrBasic can be programmed with a free coding style because it is based on general-purpose programming language such as BASIC.

However, some rules of coding style are necessary for the following purpose.

- Easy maintenance of a program
Easy to read and understand a program
Easy to modify a program
Easy to debug a program
- High efficiency to develop a program
High portability of a program module
- High quality of a program
High portability of a program module
Easy to debug a program

A program with a completely free style decreases these easy-maintenance, efficient-development and high-quality.

The HrBasic standard coding rules are shown below as the reference to your coding style that will be created to fit the various developing environment and the target system.

(1) Job

- One job has to be programmed in one source file.
- A job name has to be the same as the filename except suffix.
- A job name has to be simple and indicates the function of the job.
- Standard jobs programmed commonly are named as the followings.

System initialization: Init

Mode management: Mode

Manual operation: Manual

Error procedure: Error

Example)

Job Name "Init" → Init.bas

Job Name "Robot" → Robot.bas

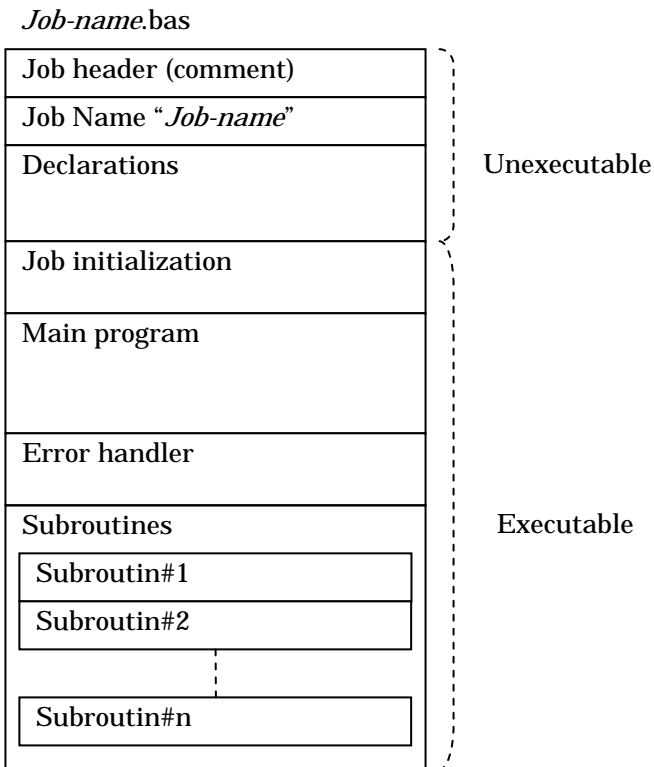
Job Name "Mode" → Mode.bas

(2) Job structure

- For the purpose of reusing a job program, the job structure has to be hierarchical. The hierarchical structure realizes the software packaging and the combination of the packaged programs can be applied to the various systems easily.
- See "3.4 Job Structure" about details.

(3) Program structure in a job

- The following figure shows the standard structure of a job program.



- Job header
The function of a job, created date, version, revision and so on are described by comments.
Example)

```
*****  
' XXXX System  
' Job Name: Init  
' Function: Job initialization  
' Author: XXXX  
' Crated: 2004.1.15  
' (R)(C) All rights reserved by HIRATA Corporation.  
' Revisions:  
' '04.08.24 XXXX YYYYYYYY Ver0.85  
*****
```
- Job Name
Job Name statement defines a job name and declares the top of a job program.
- Declarations
Including header files, definition of global variables, definition of arrays and so on are described.
Example)

```
'<<<< STP Position Memory >>>>  
    DimPos    8000  
'<<<< Include File >>>>  
Include "Io.hed"  
Include "Mb.hed"  
Include "Robot.hed"  
'<<<< Global Variables >>>>
```

-
- ```

Global g.Mode% ' System mode
'<<<< Arrays >>>>
Dim rob.err%(ROB.MAX) ' Robot error

```
- Job initialization  
Initialization of local variables, initialization of global variables that is managed by this job, definition of an error handler, opening a communication port and so on are described.  
Example)  

```

g.Mode%=MODE.INIT ' Initial mode
On Error GoTo *ERR.HANDLER ' Definition of error handler
' Open robot communication
Open "COM0" As #FNO.ROBOT RobType=580 RobNoList=1,2,3

```
  - Main program  
Main program has the loop structure generally. It selects the procedure according to the internal state and calls a subroutine. After the procedure, it goes to the top of the loop.  
In the system initialization job, a program may terminate a job without a loop structure.  
Example)  

```

*MAIN.LOOP
Select Case g.Mode% ' System mode
Case MODE.INIT ' Initial
GoSub *INIT
Case MODE.MANUAL ' Manual
GoSub *MANUAL
Case MODE.RUN ' Running
GoSub *RUN
Case Else ' No process
End Select
GoTo *MAIN.LOOP

```
  - Error handler  
An error handler jumped after a job error has occurred is described.  
Example)  

```

*ERR.HANDLER
' Error procedure
:
Resume *MAIN.LOOP

```
  - Subroutines  
Subroutines are described. A header description that contains the function of a subroutine is added to the head of a subroutine.  
Example)  

```

' Procedure: Subroutine-name
' Summary: Function
' Return: [OUT] Explanation-of-return-value
' Argument: [IN] Explanation-of-input-parameter
' [OUT] Explanation-of-output-parameter
' Caution: Remarks

*Subroutine-name
' Procedure of subroutine

```
-

## Return

## (4) Subroutine name

- The name has to contain only upper cases within 15 characters.
- The name has to be easy to understand using periods (.).

Example)

```
*ROB.MOVE
*ZAXIS.UP
```

## (5) Lable name

- The name has to contain only upper cases within 15 characters.
- The name has to be easy to understand using periods (.).

Example)

```
*ERR.HANDLER
*MAIN.LOOP
```

## (6) Variable name

- The name of a local variable has to contain only lower cases within 15 characters.
- The name of a global variable has to begin with "g." and the first character of a word in a variable has to be upper case.
- The name of a network global variable has to begin with "ng." and the top of a word in a variable has to be upper case.
- Only one variable has to be defined by Global, Dim, DimNet statement.
- The name has to be easy to understand using periods (.).
- A loop variable used for For-Next statement is named simply as i%, j%, k% if the variable does not have the special meaning. (This is a common description rule in all kinds of programming languages.)

Example)

```
Global g.Mode% ' System mode (global)
DimNet ng.St.Stat& ' Station status (network global)
Dim err.rob%(ROB.MAX) ' Robot error (local)
err.code% = 1 ' Error code (local)
For i%=1 To 10 ' Example of a loop variable
 count&(i%) = count&(i%) + 1
Next i%
```

## (7) Header file

- The constant name defined by Define statement has to contain only upper cases within 15 characters.
- The name has to be easy to understand using periods (.).
- I/O number, MB number, MW number, ML number, TIM number, PM number have to be defined in a different file respectively. The following table shows a file name and a prefix of a constant name.

| Definition type | Header file name | Constant name                         |
|-----------------|------------------|---------------------------------------|
| I/O number      | Io.hed           | Input: I.XX...XX<br>Output: O.XX...XX |
| MB number       | Mb.hed           | MB.XX...XX                            |
| MW number       | Mw.hed           | MW.XX...XX                            |
| ML number       | ML.hed           | ML.XX...XX                            |
| TIM number      | Tim.hed          | TIM.XX...XX                           |
| PM number       | Pm.hed           | PM.XX...XX                            |

## (8) Expression, statement, function

- The first character of a word in the name has to be upper case and the remains of it have to be lower cases.

Example)

```
If mode% = 1 Then Return
GoSub *SUB1
SetRobNo(1)
```

- A reserved memory name has to contain only upper cases. Index number of it must be described with parentheses.

Example)

```
PM(addr%)
MM(PM.ORIGIN)
HERE
P(10)
```