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.
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 devices 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

Control memories

STP memory
- General memory: MB, MD, MW, ML
- I/O memory: INB, IND, OUTB, OUBD
- Position memory: P [PX, PY, PZ, PW, PR, PC, ARM]

Robot memory
- Position data:
  - Position: Pm
  - M data: Mm
  - Speed data: Fm
  - S code: Sm
- General memory: MRB, MRD
- I/O memory: ORB, ORD, IRB, IRD
- Status: STATUS
- Current position: HERE

Timer
- Global variables
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

Windows PC

HDBE

HNC

WinSTP

I/O devices

Other devices

STP

HAC-8XX series
Robot controller

I/O devices

Other devices

I/O for robot hands

(1) HBDE --- HrBasic Developing Environment

HBDE provides the environment to develop HrBasic program and to maintain 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.

HR Editor can manage the robot setting data.
2.3 Specifications

2.3.1 STP Hardware Specifications

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

2.3.2 STP Execution Time

<table>
<thead>
<tr>
<th>Item</th>
<th>HAC-8XX/STP (*1)</th>
<th>WinSTP (CPU:533MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>One step execution interval of HrBasic</td>
<td>Average 0.050msec</td>
<td>Average 0.050msec</td>
</tr>
<tr>
<td>Max. interruption time by operating system</td>
<td>About 1msec</td>
<td>About 6 to 10msec</td>
</tr>
</tbody>
</table>

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

Note: The value changes according to the running environment.
2.3.3 HrBasic Specifications

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

2.3.4 HrBasic Statements and Functions

<table>
<thead>
<tr>
<th>Kind</th>
<th>Usage</th>
<th>Description</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-Processor</td>
<td>Definition</td>
<td>Define the specified name as a constant.</td>
<td>Define</td>
</tr>
<tr>
<td></td>
<td>Macro</td>
<td>Define a format of macro call.</td>
<td>Macro</td>
</tr>
<tr>
<td></td>
<td>Header file</td>
<td>Read the specified header file.</td>
<td>Include</td>
</tr>
<tr>
<td>Definable instruction</td>
<td>Definition</td>
<td>Define as array variable.</td>
<td>Dim</td>
</tr>
<tr>
<td></td>
<td>DimNet</td>
<td>Define as network global variable</td>
<td>DimNet</td>
</tr>
<tr>
<td></td>
<td>Global</td>
<td>Define as global variable.</td>
<td>Global</td>
</tr>
<tr>
<td></td>
<td>DimPos</td>
<td>Define the number of position memory.</td>
<td>DimPos</td>
</tr>
<tr>
<td></td>
<td>Rem</td>
<td>Define the comment line.</td>
<td>Rem</td>
</tr>
<tr>
<td>Flow control</td>
<td>GoTo</td>
<td>Jump to a specified line, then execute.</td>
<td>GoTo</td>
</tr>
<tr>
<td></td>
<td>GoSub</td>
<td>Call subroutine.</td>
<td>GoSub</td>
</tr>
<tr>
<td></td>
<td>Return</td>
<td>Terminate subroutine, then resume the former process.</td>
<td>Return</td>
</tr>
<tr>
<td></td>
<td>For - Next</td>
<td>Repeat the instruction between For and Next.</td>
<td>For - Next</td>
</tr>
<tr>
<td></td>
<td>If Then Else</td>
<td>Decide the condition of logical expression.</td>
<td>If Then Else</td>
</tr>
<tr>
<td></td>
<td>Delay</td>
<td>Break temporarily the execution of job.</td>
<td>Delay</td>
</tr>
<tr>
<td></td>
<td>Wait</td>
<td>Wait until conditions are satisfied.</td>
<td>Wait</td>
</tr>
<tr>
<td></td>
<td>TimeOut</td>
<td>Get the result of timeout by Wait command.</td>
<td>TimeOut</td>
</tr>
<tr>
<td></td>
<td>On GoTo</td>
<td>Jump one of specified step.</td>
<td>On GoTo</td>
</tr>
<tr>
<td></td>
<td>On GoSub</td>
<td>Call one of specified subroutines.</td>
<td>On GoSub</td>
</tr>
<tr>
<td></td>
<td>Select Case</td>
<td>Evaluate an expression and execute the processing block.</td>
<td>Select Case</td>
</tr>
<tr>
<td></td>
<td>InitGoSub</td>
<td>Initialize the subroutine-call stack.</td>
<td>InitGoSub</td>
</tr>
<tr>
<td>Interrupt control instruction</td>
<td>Error control</td>
<td>Specify the destination at error.</td>
<td>On Error GoTo</td>
</tr>
<tr>
<td></td>
<td>Resume</td>
<td>Terminate error process, then resume the former process.</td>
<td>Resume</td>
</tr>
<tr>
<td></td>
<td>Err</td>
<td>Hold error code.</td>
<td>Err</td>
</tr>
<tr>
<td>Control instruction</td>
<td>Job control</td>
<td>Control job execution.</td>
<td>Job Start</td>
</tr>
<tr>
<td></td>
<td>Job On</td>
<td>Get the running priority of the current job.</td>
<td>Job On</td>
</tr>
<tr>
<td></td>
<td>Job Off</td>
<td>Set the running priority of the current job.</td>
<td>Job Off</td>
</tr>
<tr>
<td></td>
<td>GetPriority</td>
<td>Move a robot to specified coordinates.</td>
<td>Move</td>
</tr>
<tr>
<td></td>
<td>SetPriority</td>
<td>Set operating characteristic data of a robot.</td>
<td>Set</td>
</tr>
<tr>
<td></td>
<td>Ref</td>
<td>Deal data inside of a robot.</td>
<td>Ref</td>
</tr>
<tr>
<td></td>
<td>Seq - SeqEnd</td>
<td>Set or terminate robot sequence mode.</td>
<td>Seq - SeqEnd</td>
</tr>
<tr>
<td></td>
<td>Finish</td>
<td>Complete MOVE in sequence mode.</td>
<td>Finish</td>
</tr>
<tr>
<td></td>
<td>Hold</td>
<td>Specify or cancel the servo lock of the robot.</td>
<td>Hold</td>
</tr>
<tr>
<td></td>
<td>Disable</td>
<td>Inhibit robot movement.</td>
<td>Disable</td>
</tr>
<tr>
<td></td>
<td>Enable</td>
<td>Allow robot movement</td>
<td>Enable</td>
</tr>
<tr>
<td>Kind</td>
<td>Usage</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Function</strong></td>
<td></td>
</tr>
<tr>
<td>Calib</td>
<td></td>
<td>Execute automatic origin calibration.</td>
<td></td>
</tr>
<tr>
<td>SetRobNo</td>
<td></td>
<td>Set a robot number for the current job.</td>
<td></td>
</tr>
<tr>
<td>ClearRobNo</td>
<td></td>
<td>Clear the robot number for the current job.</td>
<td></td>
</tr>
<tr>
<td>GetRobNo</td>
<td></td>
<td>Get the robot number for the current job.</td>
<td></td>
</tr>
<tr>
<td>EnableOnlineErr</td>
<td></td>
<td>Enable robot ONLINE mode check.</td>
<td></td>
</tr>
<tr>
<td>DisableOnlineErr</td>
<td></td>
<td>Disable robot ONLINE mode check.</td>
<td></td>
</tr>
<tr>
<td>RobCheckBpZone</td>
<td></td>
<td>Check robot position within BP/ZONE.</td>
<td></td>
</tr>
<tr>
<td>RobCheckCurPos</td>
<td></td>
<td>Check robot position nearby teaching data.</td>
<td></td>
</tr>
<tr>
<td>RobCheckStop</td>
<td></td>
<td>Check robot stopped.</td>
<td></td>
</tr>
<tr>
<td>RobClearErr</td>
<td></td>
<td>Clear robot errors.</td>
<td></td>
</tr>
<tr>
<td>RobSetPosRange</td>
<td></td>
<td>Define allowable margin of position.</td>
<td></td>
</tr>
<tr>
<td>Inch</td>
<td></td>
<td>Execute inching motion.</td>
<td></td>
</tr>
<tr>
<td>AxesPara</td>
<td></td>
<td>Make axes parameter.</td>
<td></td>
</tr>
<tr>
<td>PosRec</td>
<td></td>
<td>Make one robot position record.</td>
<td></td>
</tr>
<tr>
<td>CollisionCheck</td>
<td></td>
<td>Enable or disable collision check between robots.</td>
<td></td>
</tr>
<tr>
<td>RobWorldPos</td>
<td></td>
<td>Get current robot position in the world coordinates system.</td>
<td></td>
</tr>
<tr>
<td>RobDistance</td>
<td></td>
<td>Get the distance between two robots.</td>
<td></td>
</tr>
<tr>
<td>RobGetCurSpeed</td>
<td></td>
<td>Get the current robot speed.</td>
<td></td>
</tr>
<tr>
<td>RobGetCurTorq</td>
<td></td>
<td>Get the current robot torque.</td>
<td></td>
</tr>
<tr>
<td>RobGetCurAveTorq</td>
<td></td>
<td>Get the current effective torque of a robot.</td>
<td></td>
</tr>
<tr>
<td>RobGetCurPos</td>
<td></td>
<td>Get the current encoder position of a robot.</td>
<td></td>
</tr>
<tr>
<td>RobReadSvoPara</td>
<td></td>
<td>Read servo parameter of a robot.</td>
<td></td>
</tr>
<tr>
<td>RobWriteSvoPara</td>
<td></td>
<td>Write servo parameter of a robot.</td>
<td></td>
</tr>
<tr>
<td>RobReadSG</td>
<td></td>
<td>Read system generation data of a robot.</td>
<td></td>
</tr>
<tr>
<td>RobWriteSG</td>
<td></td>
<td>Write system generation data of a robot.</td>
<td></td>
</tr>
<tr>
<td>Open</td>
<td></td>
<td>Open a communication file.</td>
<td></td>
</tr>
<tr>
<td>Close</td>
<td></td>
<td>Close a file.</td>
<td></td>
</tr>
<tr>
<td>Input$</td>
<td></td>
<td>Read the specified length of the string from a file.</td>
<td></td>
</tr>
<tr>
<td>Input #</td>
<td></td>
<td>Substitute data of a sequential file to a variable.</td>
<td></td>
</tr>
<tr>
<td>Line Input #</td>
<td></td>
<td>Read one line from a sequential file.</td>
<td></td>
</tr>
<tr>
<td>Print #</td>
<td></td>
<td>Output data to a file.</td>
<td></td>
</tr>
<tr>
<td>Eof</td>
<td></td>
<td>Examine the termination code of a file.</td>
<td></td>
</tr>
<tr>
<td>FreeFile</td>
<td></td>
<td>Get unused file number.</td>
<td></td>
</tr>
<tr>
<td>RchkHrcs</td>
<td></td>
<td>Check a HRCS protocol frame received.</td>
<td></td>
</tr>
<tr>
<td>ReadHrcs</td>
<td></td>
<td>Read a HRCS protocol frame.</td>
<td></td>
</tr>
<tr>
<td>WriteHrcs</td>
<td></td>
<td>Write a HRCS protocol frame.</td>
<td></td>
</tr>
<tr>
<td>EnableDSRCheck</td>
<td></td>
<td>Enable DSR signal check of RS232C.</td>
<td></td>
</tr>
<tr>
<td>DisableDSRCheck</td>
<td></td>
<td>Disable DSR signal check of RS232C.</td>
<td></td>
</tr>
<tr>
<td>EnableRTSAuto</td>
<td></td>
<td>Enable automatic RTS signal control of RS232C.</td>
<td></td>
</tr>
<tr>
<td>DisableRTSAuto</td>
<td></td>
<td>Disable automatic RTS signal control of RS232C.</td>
<td></td>
</tr>
<tr>
<td>ComFunction</td>
<td></td>
<td>Control RS232C signal.</td>
<td></td>
</tr>
<tr>
<td>GetComStatus</td>
<td></td>
<td>Get signal status of RS232C.</td>
<td></td>
</tr>
<tr>
<td>Pulse</td>
<td></td>
<td>Generate pulse. (Substitute a value for the specified period.)</td>
<td></td>
</tr>
<tr>
<td>Time$</td>
<td></td>
<td>Get or set the current system time.</td>
<td></td>
</tr>
<tr>
<td>Date$</td>
<td></td>
<td>Get or set the current system date.</td>
<td></td>
</tr>
<tr>
<td>Open</td>
<td></td>
<td>Open a network communication.</td>
<td></td>
</tr>
<tr>
<td>Close</td>
<td></td>
<td>Close a network communication.</td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td></td>
<td>Read data from a network communication.</td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td></td>
<td>Write data from a network communication.</td>
<td></td>
</tr>
<tr>
<td>Sin</td>
<td></td>
<td>Get sine.</td>
<td></td>
</tr>
<tr>
<td>Cos</td>
<td></td>
<td>Get cosine.</td>
<td></td>
</tr>
<tr>
<td>Tan</td>
<td></td>
<td>Get tangent.</td>
<td></td>
</tr>
<tr>
<td>Atn</td>
<td></td>
<td>Get arctangent.</td>
<td></td>
</tr>
<tr>
<td>Sgn</td>
<td></td>
<td>Get the sign of value.</td>
<td></td>
</tr>
<tr>
<td>Abs</td>
<td></td>
<td>Get absolute value.</td>
<td></td>
</tr>
<tr>
<td>Int</td>
<td></td>
<td>Remove decimals</td>
<td></td>
</tr>
<tr>
<td>Fix</td>
<td></td>
<td>Remove decimals</td>
<td></td>
</tr>
<tr>
<td>Log</td>
<td></td>
<td>Get natural logarithms.</td>
<td></td>
</tr>
<tr>
<td>Kind</td>
<td>Usage</td>
<td>Description</td>
<td>Function</td>
</tr>
<tr>
<td>--------------</td>
<td>---------</td>
<td>------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>Exp</td>
<td></td>
<td>Get e raised to a power.</td>
<td></td>
</tr>
<tr>
<td>Sqr</td>
<td></td>
<td>Get square root.</td>
<td></td>
</tr>
<tr>
<td>Operator</td>
<td>Mod</td>
<td>Execute arithmetic division and get the remainder.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Not</td>
<td>Execute negation.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>And</td>
<td>Execute logical multiplication.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Or</td>
<td>Execute logical addition.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Xor</td>
<td>Execute exclusive logical addition.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Eqv</td>
<td>Execute logical equivalence.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Imp</td>
<td>Execute logical implication.</td>
<td></td>
</tr>
<tr>
<td>Arithmetic Constant</td>
<td>Pai</td>
<td>Get the value of pi.</td>
<td></td>
</tr>
<tr>
<td>Character</td>
<td>Left$</td>
<td>Pick out arbitrary length from the left of a string.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mid$</td>
<td>Specify one part of a string.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Right$</td>
<td>Pick out arbitrary length from the right of a string.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Space$</td>
<td>Get a string with the arbitrary length blank.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Chr$</td>
<td>Get the character of specified character code.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>String$</td>
<td>Get the character string connected one arbitrary character.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hex$</td>
<td>Get the character string converted decimal into hexadecimal.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Str$</td>
<td>Convert numerical value into a string.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Val</td>
<td>Convert the number of a character string into actual value.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Asc</td>
<td>Get the character codes of characters.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Len</td>
<td>Get the length of a string.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>InStr</td>
<td>Get the first position of the string in another string.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ScanStr</td>
<td>Scan string data according to specified format. And get the value as parameter from string by operator in the format.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PrintStr</td>
<td>Print string data according to specified format. And put the data string of specified parameter by operator in the format.</td>
<td></td>
</tr>
<tr>
<td>Initialization</td>
<td>InitPos</td>
<td>Initialize position memory in STP.</td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td>ConsoleMsgOn</td>
<td>Enable to print message to STP console.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ConsoleMsgOff</td>
<td>Disable to print message to STP console.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ConsoleMsg</td>
<td>Print specified message to STP console.</td>
<td></td>
</tr>
</tbody>
</table>
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 job 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.

---

1 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, an 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.

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

“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.

<table>
<thead>
<tr>
<th>Byte No</th>
<th>Bit No</th>
<th>Name</th>
<th>Explanation</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>I.READY</td>
<td>Ready button</td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>I.START</td>
<td>Start button</td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>I.MANUAL</td>
<td>Manual button</td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>I.ERROR</td>
<td>Error</td>
<td>Not used</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td></td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td></td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td></td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td></td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td></td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td></td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td></td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td>Byte No</td>
<td>Bit No</td>
<td>Name</td>
<td>Explanation</td>
<td>Note</td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
<td>--------------</td>
<td>------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>O.BLUE</td>
<td>Signal tower: blue lamp</td>
<td>Not used</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>O.RED</td>
<td>Signal tower: red lamp</td>
<td>Not used</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>O.BUZZER</td>
<td>Buzzer</td>
<td>Not used</td>
</tr>
<tr>
<td>6-15</td>
<td></td>
<td></td>
<td></td>
<td>Not used</td>
</tr>
</tbody>
</table>

“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.

<table>
<thead>
<tr>
<th>No</th>
<th>Name</th>
<th>Explanation</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td>1</td>
<td>TIM.STOP</td>
<td>Timer for stopping time</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TIM.CYCLE</td>
<td>Timer for cycle time and running time</td>
<td></td>
</tr>
</tbody>
</table>

“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.

<table>
<thead>
<tr>
<th>No</th>
<th>Name</th>
<th>Explanation</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td>1</td>
<td>PM.ORIGIN</td>
<td>Origin position</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>PM.PICK</td>
<td>Point A to pick</td>
<td>Not used</td>
</tr>
<tr>
<td>101</td>
<td>PM.PLACE</td>
<td>Point B to place</td>
<td></td>
</tr>
</tbody>
</table>

“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 systems 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

<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 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 "I.O.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.

<table>
<thead>
<tr>
<th>Contents</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ML memory assignments</td>
<td>ML.hed</td>
</tr>
<tr>
<td>I/O assignments</td>
<td>I.O.hed</td>
</tr>
<tr>
<td>Timer assignments</td>
<td>TIM.hed</td>
</tr>
<tr>
<td>Robot position assignments</td>
<td>PM.hed</td>
</tr>
<tr>
<td>System constants</td>
<td>System.hed</td>
</tr>
</tbody>
</table>

<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.

![Diagram of job 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.

<table>
<thead>
<tr>
<th>Job Name</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td>Init.bas</td>
</tr>
<tr>
<td>Mode</td>
<td>Mode.bas</td>
</tr>
<tr>
<td>Main</td>
<td>Main.bas</td>
</tr>
<tr>
<td>Robot</td>
<td>Robot.bas</td>
</tr>
<tr>
<td>Tower</td>
<td>Tower.bas</td>
</tr>
<tr>
<td>Buzzer</td>
<td>Buzzer.bas</td>
</tr>
</tbody>
</table>

The explanation of each job is described below.

<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 variables\(^1\) 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 file\(^i\).

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

\(^1\) Global variable: See “6.2.3 Local variable and global variable” about details.

\(^i\) Make file: Configuration file of compilation. Refer to “HBDE operation manual” or HBDE help about details.
3 Program Development Guideline

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
EndSelect

EndIf

*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
EndSelect

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

***********************************************************************
'Procedure: CHECK.ERROR
'Summary:   Check error input
'Return:    [OUT] ret% =1:Mode chenged
'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
  Endf
  Return

***********************************************************************
'Procedure: CHECK.MANUAL
'Summary:   Check manual button
'Return:    [OUT] ret% =1:Mode chenged
'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
  Endf
  Return

***********************************************************************
'Procedure: CHECK.READY
3. Program Development Guideline

3.16.3 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>

`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
  Endf
  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
  Endf
  Return

`
Job Name "Main"  

*** 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
  End If
End If


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
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>

```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 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”.

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 HNC reserved memory. They are explained in detail below.

(1) STP reserved memory

The list of STP reserved memory is shown below.

<table>
<thead>
<tr>
<th>Memory Type</th>
<th>Format</th>
<th>Index Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit memory</td>
<td>MBn</td>
<td>n=0 to 1023</td>
<td>General purpose bit memory</td>
</tr>
<tr>
<td>Byte memory</td>
<td>MDn</td>
<td>n=0 to 1023</td>
<td>General purpose byte memory</td>
</tr>
<tr>
<td>Word memory</td>
<td>MWn</td>
<td>n=0 to 16383</td>
<td>General purpose word (2 bytes) memory</td>
</tr>
<tr>
<td>Long word memory</td>
<td>MLn</td>
<td>n=0 to 1023</td>
<td>General purpose long word (4 bytes) memory</td>
</tr>
<tr>
<td>Input bit</td>
<td>INBn</td>
<td>n=0 to 255</td>
<td>Remote input bit</td>
</tr>
<tr>
<td>Input byte</td>
<td>INDn</td>
<td>n=0 to 31</td>
<td>Remote input byte</td>
</tr>
<tr>
<td>Output bit</td>
<td>OUTBn</td>
<td>n=0 to 255</td>
<td>Remote output bit</td>
</tr>
<tr>
<td>Output byte</td>
<td>OUTDn</td>
<td>n=0 to 31</td>
<td>Remote output byte</td>
</tr>
<tr>
<td>Position memory</td>
<td>Pn</td>
<td>n=0~7999</td>
<td>Position data memory</td>
</tr>
<tr>
<td>X axis data</td>
<td>PXn</td>
<td></td>
<td>X axis data of Pn</td>
</tr>
<tr>
<td>Y axis data</td>
<td>PYn</td>
<td></td>
<td>Y axis data of Pn</td>
</tr>
<tr>
<td>Z axis data</td>
<td>PZn</td>
<td></td>
<td>Z axis data of Pn</td>
</tr>
<tr>
<td>W axis data</td>
<td>PWN</td>
<td></td>
<td>W axis data of Pn</td>
</tr>
<tr>
<td>R axis data</td>
<td>PRn</td>
<td></td>
<td>R axis data of Pn</td>
</tr>
<tr>
<td>C axis data</td>
<td>PCn</td>
<td></td>
<td>C axis data of Pn</td>
</tr>
<tr>
<td>ARM data</td>
<td>PARMn</td>
<td></td>
<td>ARM data of Pn</td>
</tr>
<tr>
<td>M data</td>
<td>PDMn</td>
<td></td>
<td>M data of Pn</td>
</tr>
<tr>
<td>F code</td>
<td>PDFn</td>
<td></td>
<td>F code of Pn</td>
</tr>
<tr>
<td>S code</td>
<td>PDSn</td>
<td></td>
<td>S code of Pn</td>
</tr>
</tbody>
</table>

---

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

---
4 Individual Functions

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.

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

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.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD0</td>
<td>MB7</td>
<td>MB6</td>
<td>MB5</td>
<td>MB4</td>
<td>MB3</td>
<td>MB2</td>
<td>MB1</td>
<td>MB0</td>
</tr>
<tr>
<td>Exponential value</td>
<td>$2^7$</td>
<td>$2^6$</td>
<td>$2^5$</td>
<td>$2^4$</td>
<td>$2^3$</td>
<td>$2^2$</td>
<td>$2^1$</td>
<td>$2^0$</td>
</tr>
</tbody>
</table>

In case that the value of MD0 is 0, 150 or 255, the bits of MB0-MB7 are shown below.
In case that MD0 is 150, the following equation shows how MD0 value is derived from MB0-MB7.

\[ 150 = 2^7 \times 1 + 2^6 \times 0 + 2^5 \times 0 + 2^4 \times 1 + 2^3 \times 0 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 0 \]

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)

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

<table>
<thead>
<tr>
<th>MD</th>
<th>MB7</th>
<th>MB6</th>
<th>MB5</th>
<th>MB4</th>
<th>MB3</th>
<th>MB2</th>
<th>MB1</th>
<th>MB0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>150</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>255</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- 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.

- MW holds the last value after the power reset.

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

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

- MW(n) is equivalent to MWNn.
  - e.g.) MW(3)
- “n” of MWNn 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.

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.

- 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.

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.
<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>IND0</td>
<td>INB7</td>
<td>INB6</td>
<td>INB5</td>
<td>INB4</td>
<td>INB3</td>
<td>INB2</td>
<td>INB1</td>
<td>INB0</td>
</tr>
<tr>
<td>OUTD0</td>
<td>OUTB7</td>
<td>OUTB6</td>
<td>OUTB5</td>
<td>OUTB4</td>
<td>OUTB3</td>
<td>OUTB2</td>
<td>OUTB1</td>
<td>OUTB0</td>
</tr>
</tbody>
</table>

| 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.

<table>
<thead>
<tr>
<th>INB IND0 value</th>
<th>INB7</th>
<th>INB6</th>
<th>INB5</th>
<th>INB4</th>
<th>INB3</th>
<th>INB2</th>
<th>INB1</th>
<th>INB0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>150</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>255</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

| 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.

$$150 = 2^7 \times 1 + 2^6 \times 0 + 2^5 \times 0 + 2^4 \times 1 + 2^3 \times 0 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 0$$

$$= 128 \times 1 + 64 \times 0 + 32 \times 0 + 16 \times 1 + 8 \times 0 + 4 \times 1 + 2 \times 1 + 1 \times 0$$

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.

<table>
<thead>
<tr>
<th>IND/OUTD</th>
<th>INB/OUTB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>30</td>
<td>247</td>
</tr>
<tr>
<td>31</td>
<td>255</td>
</tr>
<tr>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>510</td>
<td>4087</td>
</tr>
<tr>
<td>511</td>
<td>4095</td>
</tr>
</tbody>
</table>

⚠️ **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 potion data form 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.

Job Name "Init"
Include "Sample.hed"
DisPos 8000
InitPos 0 to 7999

Job Name "Mode"
Include "Sample.hed"
DisPos 2000

Job Name "Main"
Include "Sample.hed"
DisPos 6000

Init job uses 8000 Ps from P0 to P7999 and initializes it.
Mode job uses 2000 Ps from P0 to P1999.
Main job uses 6000 Ps from P0 to P5999.

P memory has the following data structure.

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

- 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.
The examples of P memory usage are shown below.

- Read the robot position data (robot #1, address 110) to P10.
  
  \[ P10 = \text{Ref}(#\text{fno}\%[rno:1], \text{PM110}) \]

  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.

  \[ \text{P0} = \text{Ref}(\#\text{fno}\%[rno:1], \text{HERE}) \]
  \[ 'M,F,S' \text{ code cleared} \]
  \[ \text{PDM}0=1; \text{PDF}0=99; \text{PDS}=1 \]
  \[ '!! Set M,F,S code !!' \]
  \[ \text{PZ}0 = \text{PZ}0 - 10 \]
  \[ 'Z axis: 10mm up' \]
  \[ \text{PW}0=200 \]
  \[ 'W axis: 200 degree' \]
  \[ \text{Move } \#\text{fno}\%[rno:1], \text{PTP}, \text{P}0 \]

  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.

  \[ \text{P0} = \text{Ref}(\#\text{fno}\%[rno:1], \text{HERE}) \]
  \[ 'M,F,S' \text{ code cleared} \]
  \[ \text{PDM}0=1; \text{PDF}0=99; \text{PDS}=1 \]
  \[ '!! Set M,F,S code !!' \]
  \[ 'Write P0 to robot position PM100' \]
  \[ \text{Ref}(\#\text{fno}\%[rno:1], \text{PM100}) = \text{P}0 \]

  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.

  \[ \text{Ref}([1][rno:1], \text{MM100})=50 \]
  \[ 'Set 50 to M data of PM100' \]

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

  \[ \text{P0} = \text{Ref}(\#\text{fno}\%[rno:1], \text{PM}0) \]
  \[ 'Read address 0 to P0' \]
  \[ \text{PX}0 = \text{PX}0 + 100.0 \]
  \[ 'X axis +100mm' \]
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(#fno%[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.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRD0</td>
<td>IRB7</td>
<td>IRB6</td>
<td>IRB5</td>
<td>IRB4</td>
<td>IRB3</td>
<td>IRB2</td>
<td>IRB1</td>
<td>IRB0</td>
</tr>
<tr>
<td>ORD0</td>
<td>ORB7</td>
<td>ORB6</td>
<td>ORB5</td>
<td>ORB4</td>
<td>ORB3</td>
<td>ORB2</td>
<td>ORB1</td>
<td>ORB0</td>
</tr>
<tr>
<td>Exponential value</td>
<td>$2^7$</td>
<td>$2^6$</td>
<td>$2^5$</td>
<td>$2^4$</td>
<td>$2^3$</td>
<td>$2^2$</td>
<td>$2^1$</td>
<td>$2^0$</td>
</tr>
<tr>
<td></td>
<td>(128)</td>
<td>(64)</td>
<td>(32)</td>
<td>(16)</td>
<td>(8)</td>
<td>(4)</td>
<td>(2)</td>
<td>(1)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>IRD0 value</th>
<th>IRB7</th>
<th>IRB6</th>
<th>IRB5</th>
<th>IRB4</th>
<th>IRB3</th>
<th>IRB2</th>
<th>IRB1</th>
<th>IRB0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>150</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>255</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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.

$150 = 2^7 \times 1 + 2^6 \times 0 + 2^5 \times 0 + 2^4 \times 1 + 2^3 \times 0 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 0$

$= 128 \times 1 + 64 \times 0 + 32 \times 0 + 16 \times 1 + 8 \times 0 + 4 \times 1 + 2 \times 1 + 1 \times 0$

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.

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

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)
  ```plaintext
  Open "com1:4800,E,7,1" as #no%
  : dat% = Ref( #no%[rno:1], IRD(5) )
  Ref( #no%[rno:1], ORD(3) ) = 10
  ```
- IRB(n), IRD(n), ORB(n), ORD(n) are equivalent to IRBn, IRDn, ORBn, ORDn.
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.

PMn

<table>
<thead>
<tr>
<th>Address n</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>W</th>
<th>R</th>
<th>C</th>
<th>Arm</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMn</td>
<td>FMn</td>
<td>SMn</td>
<td>M data</td>
<td>F code</td>
<td>S code</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 #n0% :

P0 = Ref( #n0%[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(#n0%[rno:1], PM100 )  ‘Read robot #1 PM100 to STP P0
Ref( #n0%[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 dataaising F code PDFn
S code PDStype Cannot access

Note

![Note icon]
4 Individual Functions

- **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.
  \[ \text{MD1} = \text{Ref} (#\text{fno}[rno:1], \text{MM1}) \]  
  \[ \text{Ref}(\ #\text{fno}[rno:2], \text{MM200}) = 99 \]  

- **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.
  \[ \text{MD1} = \text{Ref}(\ #\text{fno}[rno:1], \text{FM1}) \]  

- **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.
  \[ \text{MD1} = \text{Ref}(\ #\text{fno}[rno:1], \text{SM1}) \]  

- **PM**, **MM**, **SM**, **FM**
  PM(n), MM(n), SM(n), FM(n) are equivalent to PMn, MMn, SMn, FMn.  
  \[ \text{e.g.) PM}(5) = \text{MM}(3) \]  
  
  - “n” of PM(n), MM(n), SM(n), FM(n) can be specified by the indirect expression such as a formula, variable or reserved memory. But in this case, parentheses are necessary for “n”.
  - \[ \text{e.g.) MM}(\text{MD5}) \]  --- 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.
- Open “com1:4800,E,7,1” as #fno%  
  \[ \text{ecode%} = \text{Ref}(\ #\text{fno}[rno:1], \text{STATUS0}) \]  

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

**Note**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS0</td>
<td>Robot error code</td>
</tr>
<tr>
<td>STATUS1</td>
<td>X axis error information</td>
</tr>
</tbody>
</table>
### 4 Individual Functions

#### 4-14

<table>
<thead>
<tr>
<th>Name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS2</td>
<td>Y axis error information</td>
</tr>
<tr>
<td>STATUS3</td>
<td>Z axis error information</td>
</tr>
<tr>
<td>STATUS4</td>
<td>W axis error information</td>
</tr>
<tr>
<td>STATUS5</td>
<td>R axis error information</td>
</tr>
<tr>
<td>STATUS6</td>
<td>C axis error information</td>
</tr>
<tr>
<td>STATUS7</td>
<td>未使用</td>
</tr>
<tr>
<td>STATUS8</td>
<td>Robot status #1</td>
</tr>
<tr>
<td>STATUS9</td>
<td>Robot status #2</td>
</tr>
</tbody>
</table>

Detail of each STATUS is described below.

(1) **STATUS0**

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

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

(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.
An origin sensor does not become ON while the axis is moving in the direction of origin.

An origin sensor does not become OFF or the axis cannot move back to the working area.

A limit sensor becomes ON while the axis is moving in the direction of origin.

Counter is under the regulation when reset.

Counter is over the regulation when reset.

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.

<table>
<thead>
<tr>
<th>Bit number</th>
<th>Value when bit ON</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>&amp;H80</td>
</tr>
<tr>
<td>6</td>
<td>&amp;H40</td>
</tr>
<tr>
<td>5</td>
<td>&amp;H20</td>
</tr>
<tr>
<td>4</td>
<td>&amp;H10</td>
</tr>
<tr>
<td>3</td>
<td>&amp;H8</td>
</tr>
<tr>
<td>2</td>
<td>&amp;H4</td>
</tr>
<tr>
<td>1</td>
<td>&amp;H2</td>
</tr>
<tr>
<td>0</td>
<td>&amp;H1</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Bit number</th>
<th>Value when bit ON</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>H80</td>
</tr>
<tr>
<td>6</td>
<td>H40</td>
</tr>
<tr>
<td>5</td>
<td>H20</td>
</tr>
<tr>
<td>4</td>
<td>H10</td>
</tr>
<tr>
<td>3</td>
<td>H8</td>
</tr>
<tr>
<td>2</td>
<td>H4</td>
</tr>
<tr>
<td>1</td>
<td>H2</td>
</tr>
<tr>
<td>0</td>
<td>H1</td>
</tr>
</tbody>
</table>

- 1: Inside of Z axis zone (SAFTY ZONE)
- 1: Positioning completed
- 1: A-CAL completed
- 0: Not used
- 0: Not used
- 0: Not used
- 1: Command execution
- 0: Not used

- 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.


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

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.
```
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 below.

(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
4 Individual Functions

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

- HRCS protocol
  This protocol has the following communication format.
  - STX: Start of Text (&H02)
  - ETX: End of Text (&H03)
  - LRC: Longitudinal Redundancy Check
    LRC 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

See “Appendix B LRC Calculation”.

---

\(^i\) 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.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>➢ The same COM port is not opened multiply at the same time even if different file number is used.</td>
</tr>
<tr>
<td>➢ 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.</td>
</tr>
</tbody>
</table>

(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.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>➢ If the file number is omitted in Close statement, all opened files are closed automatically.</td>
</tr>
<tr>
<td>➢ 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.</td>
</tr>
</tbody>
</table>

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

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

### WinSTP COM Port Functions

<table>
<thead>
<tr>
<th>COM No.</th>
<th>Interface</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM0</td>
<td>Inter-process communication</td>
<td>Only for HBDE debugging. HrBasic cannot access this port.</td>
</tr>
<tr>
<td>COM1</td>
<td>RS232C</td>
<td>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</td>
</tr>
<tr>
<td>COM2</td>
<td>RS232C</td>
<td></td>
</tr>
<tr>
<td>COM3</td>
<td>RS232C</td>
<td></td>
</tr>
<tr>
<td>COM4</td>
<td>RS232C</td>
<td></td>
</tr>
<tr>
<td>COM5</td>
<td>RS232C</td>
<td></td>
</tr>
<tr>
<td>COM6</td>
<td>RS232C</td>
<td></td>
</tr>
<tr>
<td>COM7</td>
<td>RS232C</td>
<td></td>
</tr>
<tr>
<td>COM8</td>
<td>RS232C</td>
<td></td>
</tr>
<tr>
<td>COM9</td>
<td>RS232C</td>
<td></td>
</tr>
<tr>
<td>COM10</td>
<td>Ethernet</td>
<td>To be available or not depends on the hardware implementation. 10BASE-T.</td>
</tr>
<tr>
<td>COM11</td>
<td>Inter-process communication</td>
<td>Windows application port. HrBasic in WinSTP can communicates with other application in Windows.</td>
</tr>
<tr>
<td>COM12</td>
<td>Inter-process communication</td>
<td></td>
</tr>
<tr>
<td>COM13</td>
<td>Inter-process communication</td>
<td></td>
</tr>
<tr>
<td>COM14</td>
<td>Inter-process communication</td>
<td></td>
</tr>
</tbody>
</table>
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” 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 occur 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.

```plaintext
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**

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.

---

**Note**

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

---

1. See “Appendix D Running Job Errors”.
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.

```
Sentence        Statement
              |   Substitution
```

<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.
5 Syntax Rules

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 subroutine

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”.
5 Syntax Rules

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

⚠️ Note
- “Syntax error” occurs when compiled if a program neglects the above-mentioned rules.
- In other job program, the same label name can be used. But if multiple usage of the label name is founded in one job, “Duplicate label” error occurs when compiled.

5.7 Header File

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”.

To define various constants in a header file with centralization reduces the cost of program modification and prevents the decrement of the program quality after modification.

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

Include "Header-file-name"

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

<Include statement example>
Job Name "Test"
Include "Sample.hed"

5.8 Define Statement

HrBasic compiler replaces the string in a source program with the defined string in a header file.
Define statement has the following format and has to be written in a header file.

Define String-A String-B

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

⚠️ Note
Define statement is written only in a header file.

ℹ️ See “3. Syntax Rules”.
Sample program is shown below.

- **Header file**
  ```plaintext
  '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**
  ```plaintext
  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.
   ```plaintext
   <Example>
   A=B+C : X=A
   ```

2. **Comma “,”**
   It is used to separate parameters.
   ```plaintext
   <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.

- **Character literal**
- **Numerical literal**
  - **Integer type**
    - Decimal type
    - Octal type
    - Hexadecimal type
  - **Real number type**
  - **Single precision type**
  - **Double precision type**

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.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Octal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>32767</td>
<td>&amp;O77777</td>
<td>0111 1111 1111 1111</td>
</tr>
<tr>
<td>32766</td>
<td>&amp;O77776</td>
<td>0111 1111 1111 1110</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>&amp;O00002</td>
<td>0000 0000 0000 0010</td>
</tr>
<tr>
<td>1</td>
<td>&amp;O00001</td>
<td>0000 0000 0000 0001</td>
</tr>
<tr>
<td>0</td>
<td>&amp;O00000</td>
<td>0000 0000 0000 0000</td>
</tr>
<tr>
<td>-1</td>
<td>&amp;O177777</td>
<td>1111 1111 1111 1111</td>
</tr>
<tr>
<td>-2</td>
<td>&amp;O177776</td>
<td>1111 1111 1111 1110</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-32767</td>
<td>&amp;O100001</td>
<td>1000 0000 0000 0001</td>
</tr>
<tr>
<td>-32768</td>
<td>&amp;O100000</td>
<td>1000 0000 0000 0000</td>
</tr>
</tbody>
</table>

- Long integer type value
  An ampersand mark “&” can be added at the end of a literal.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Octal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>2147483647</td>
<td>&amp;O17777777777</td>
<td>0111 1111 1111 1111 1111 1111 1111 1111</td>
</tr>
<tr>
<td>2147483646</td>
<td>&amp;O17777777776</td>
<td>0111 1111 1111 1111 1111 1111 1111 1110</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>&amp;O10000000002</td>
<td>0000 0000 0000 0000 0000 0000 0000 0001</td>
</tr>
<tr>
<td>1</td>
<td>&amp;O10000000001</td>
<td>0000 0000 0000 0000 0000 0000 0000 0000</td>
</tr>
<tr>
<td>0</td>
<td>&amp;O10000000000</td>
<td>0000 0000 0000 0000 0000 0000 0000 0000</td>
</tr>
<tr>
<td>-1</td>
<td>&amp;O37777777777</td>
<td>1111 1111 1111 1111 1111 1111 1111 1111</td>
</tr>
<tr>
<td>-2</td>
<td>&amp;O37777777776</td>
<td>1111 1111 1111 1111 1111 1111 1111 1110</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-2147483647</td>
<td>&amp;O30000000001</td>
<td>1000 0000 0000 0000 0000 0000 0000 0001</td>
</tr>
<tr>
<td>-2147483648</td>
<td>&amp;O30000000000</td>
<td>1000 0000 0000 0000 0000 0000 0000 0000</td>
</tr>
</tbody>
</table>

<Example>
- Short integer type
  &12345
  &O7777%
- Long integer type
  &O177777777
  &12345&
  &177777777&

(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.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Octal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>32767</td>
<td>&amp;H7FFF</td>
<td>0111 1111 1111 1111</td>
</tr>
<tr>
<td>32766</td>
<td>&amp;H7FFE</td>
<td>0111 1111 1111 1110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>&amp;H0002</td>
<td>0000 0000 0000 0010</td>
</tr>
<tr>
<td>1</td>
<td>&amp;H0001</td>
<td>0000 0000 0000 0001</td>
</tr>
<tr>
<td>0</td>
<td>&amp;H0000</td>
<td>0000 0000 0000 0000</td>
</tr>
<tr>
<td>-1</td>
<td>&amp;HFFFF</td>
<td>1111 1111 1111 1111</td>
</tr>
<tr>
<td>-2</td>
<td>&amp;HFFFE</td>
<td>1111 1111 1111 1110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>-32767</td>
<td>&amp;H8001</td>
<td>1000 0000 0000 0001</td>
</tr>
<tr>
<td>-32768</td>
<td>&amp;H8000</td>
<td>1000 0000 0000 0000</td>
</tr>
</tbody>
</table>

- **Long integer type value**
  
  An ampersand mark “&” can be added at the end of a literal.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Octal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>2147483647</td>
<td>&amp;H7FFFFFFF</td>
<td>0111 1111 1111 1111 1111 1111 1111 1111</td>
</tr>
<tr>
<td>2147483646</td>
<td>&amp;H7FFFFFFE</td>
<td>0111 1111 1111 1111 1111 1111 1111 1110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>&amp;H00000002</td>
<td>0000 0000 0000 0000 0000 0000 0000 0010</td>
</tr>
<tr>
<td>1</td>
<td>&amp;H00000001</td>
<td>0000 0000 0000 0000 0000 0000 0000 0001</td>
</tr>
<tr>
<td>0</td>
<td>&amp;H00000000</td>
<td>0000 0000 0000 0000 0000 0000 0000 0000</td>
</tr>
<tr>
<td>-1</td>
<td>&amp;HFFFFFFFF</td>
<td>1111 1111 1111 1111 1111 1111 1111 1111</td>
</tr>
<tr>
<td>-2</td>
<td>&amp;HFFFE</td>
<td>1111 1111 1111 1111 1111 1111 1111 1110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>-2147483647</td>
<td>&amp;H80000001</td>
<td>1000 0000 0000 0000 0000 0000 0000 0001</td>
</tr>
<tr>
<td>-2147483648</td>
<td>&amp;H80000000</td>
<td>1000 0000 0000 0000 0000 0000 0000 0000</td>
</tr>
</tbody>
</table>

<Example>

- **Short integer type**
  &H12345
  &HFFF%

- **Long integer type**
  &H7FFFFFFE
  &H12345&
  &H7FFFFE &

⚠️ **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.

\[
\begin{align*}
A1234560 \\
A1234568
\end{align*}
\]

Variable name cannot be the reserved name, 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.

<table>
<thead>
<tr>
<th>Type declaration character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>Integer or 16bits integer type (2bytes)</td>
</tr>
<tr>
<td>&amp;</td>
<td>Long integer or 32bits integer type (4bytes)</td>
</tr>
<tr>
<td>!</td>
<td>Single precision real-number type (4bytes)</td>
</tr>
<tr>
<td>#</td>
<td>Double precision real-number type (8bytes)</td>
</tr>
<tr>
<td>$</td>
<td>String type (max 255bytes)</td>
</tr>
</tbody>
</table>

A
A%
A&
A#
A$

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.

\[
\begin{align*}
<\text{OK}> \\
A\% &= 32767 \\
A\% &= 222\% \\
A& &= 123
\end{align*}
\]

\[
\begin{align*}
<\text{Compiling error}> \\
A\% &= 32768 \quad \text{Overflow for variable} \\
A\% &= 32768\% \quad \text{Overflow of literal}
\end{align*}
\]

\[\text{Note:} \quad \text{Reserved name is keyword of HrBasic language, such as name of statement (e.g. Mid, If), name of function (e.g. Len, Abs), and operator (e.g. Or, Mod).}\]
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)

<table>
<thead>
<tr>
<th>Element</th>
<th>Memory area</th>
</tr>
</thead>
<tbody>
<tr>
<td>a%(0)</td>
<td>2 bytes</td>
</tr>
<tr>
<td>a%(1)</td>
<td>2 bytes</td>
</tr>
<tr>
<td>a%(2)</td>
<td>2 bytes</td>
</tr>
<tr>
<td>a%(3)</td>
<td>2 bytes</td>
</tr>
<tr>
<td>a%(4)</td>
<td>2 bytes</td>
</tr>
<tr>
<td>a%(5)</td>
<td>2 bytes</td>
</tr>
<tr>
<td>A%(6)</td>
<td>2 bytes</td>
</tr>
</tbody>
</table>

Maximum three dimensions of array are available in HrBasic.

The following example shows the structure of two dimensions array.

Dim x&(2,2)

<table>
<thead>
<tr>
<th>Element</th>
<th>Memory area</th>
</tr>
</thead>
<tbody>
<tr>
<td>x&amp;(0, 0)</td>
<td>4 bytes</td>
</tr>
<tr>
<td>x&amp;(0, 1)</td>
<td>4 bytes</td>
</tr>
<tr>
<td>x&amp;(0, 2)</td>
<td>4 bytes</td>
</tr>
<tr>
<td>x&amp;(1, 0)</td>
<td>4 bytes</td>
</tr>
<tr>
<td>x&amp;(1, 1)</td>
<td>4 bytes</td>
</tr>
<tr>
<td>x&amp;(1, 2)</td>
<td>4 bytes</td>
</tr>
<tr>
<td>x&amp;(2, 0)</td>
<td>4 bytes</td>
</tr>
<tr>
<td>x&amp;(2, 1)</td>
<td>4 bytes</td>
</tr>
<tr>
<td>x&amp;(2, 2)</td>
<td>4 bytes</td>
</tr>
</tbody>
</table>

The following example shows the structure of three dimensions array.

Dim y#(2,2,2)

<table>
<thead>
<tr>
<th>Element</th>
<th>Memory area</th>
</tr>
</thead>
<tbody>
<tr>
<td>y#(0, 0, 0)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>y#(0, 0, 1)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>y#(0, 0, 2)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>y#(0, 1, 0)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>y#(0, 1, 1)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>y#(0, 1, 2)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>y#(0, 2, 0)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>y#(0, 2, 1)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>y#(0, 2, 2)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>y#(1, 0, 0)</td>
<td>8 bytes</td>
</tr>
</tbody>
</table>
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.

```
<table>
<thead>
<tr>
<th>Job Name “Port1”</th>
<th>Include “loader.hed”</th>
<th>Job Name “Port2”</th>
<th>Include “loader.hed”</th>
</tr>
</thead>
<tbody>
<tr>
<td>portno%=1</td>
<td>:</td>
<td>portno%=2</td>
<td>:</td>
</tr>
<tr>
<td></td>
<td>:</td>
<td></td>
<td>:</td>
</tr>
</tbody>
</table>
```

(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.

```
<table>
<thead>
<tr>
<th>Job Name “Port1”</th>
<th>Include “loader.hed”</th>
<th>Job Name “Port2”</th>
<th>Include “loader.hed”</th>
<th>Job Name “Mode”</th>
<th>Include “loader.hed”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global g.Mode%(2)</td>
<td>:</td>
<td>Global g.Mode%(2)</td>
<td>:</td>
<td>Global g.Mode%(2)</td>
<td>:</td>
</tr>
<tr>
<td>Global Err.no%</td>
<td>:</td>
<td>Global Err.no%</td>
<td>:</td>
<td>If g.Mode%(2)=3 Then</td>
<td>Err.no%=1 EndIf</td>
</tr>
<tr>
<td>portno%=1</td>
<td>:</td>
<td>portno%=2</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>g.Mode%(portno%)=1</td>
<td>:</td>
<td>g.Mode%(portno%)=1</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>
```

(3) Network global variable

STP supports one of the following three fieldbus.

a) InterBus
b) PROFIBUS
c) 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.

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.
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  '3.333333333333333 is substituted for a#
   b# = 10#/3#  '3.333333333333333 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 7 digits 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.

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

In case of changing priority, 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.

<table>
<thead>
<tr>
<th>Arithmetic expression</th>
<th>HrBasic expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) (2X + Y)</td>
<td>2 * X + Y</td>
</tr>
<tr>
<td>2) (\frac{X}{Y} + 2)</td>
<td>X / Y + 2</td>
</tr>
<tr>
<td>3) (\frac{X + Y}{2})</td>
<td>(X+Y) / 2</td>
</tr>
<tr>
<td>4) (X^2 + 2X + 1)</td>
<td>X^2 + 2*X + 1</td>
</tr>
<tr>
<td>5) (X^{Y2})</td>
<td>X^(Y^2)</td>
</tr>
<tr>
<td>6) ((X^Y)^2)</td>
<td>X^(Y^2)</td>
</tr>
<tr>
<td>7) (Y(-X))</td>
<td>Y* -X</td>
</tr>
</tbody>
</table>

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.

---

\[\text{Note} \]

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

---

\[\text{Note} \]

The remarks of operation are described below.

---

\[\text{Note} \]

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.

---

\[\text{Note} \]

See “6.5.1 Priority of Operations”.
Example)
\[ a\% = 2 / 0 \]
\[ b! = 0 ^ \text{-.3} \]

(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)
\[ a\% = 32000 + 10000 \]
\[ b! = 3 ^ {1000} \]

(3) Exponent
Exponent operation can not be calculated by negative real number. (positive number or negative integer is possible.)
Example)
\[ a! = b! ^ \text{-.23} \]

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.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;=</td>
<td>Smaller or equal</td>
<td>A &lt;= B</td>
<td>If A is smaller than B or equal, the result is true. If A is larger than B, it is false.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Larger or equal</td>
<td>A &gt;= B</td>
<td>If A is larger than B or equal, the result is true. If A is smaller than B, it is false.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Smaller</td>
<td>A &lt; B</td>
<td>If A is smaller than B, the result is true. If A is larger than B or equal, it is false.</td>
</tr>
<tr>
<td>&gt;</td>
<td>Larger</td>
<td>A &gt; B</td>
<td>If A is larger than B, the result is true. If A is smaller than B or equal, it is false.</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal</td>
<td>A &lt;&gt; B</td>
<td>If A is not equal to B, the result is true. If A is equal to B, it is false.</td>
</tr>
<tr>
<td>=</td>
<td>Equal</td>
<td>A = B</td>
<td>If A is equal to B, the result is true. If A is not equal to B, it is false.</td>
</tr>
</tbody>
</table>

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)
\[ a\% = 1 \]
\[ \text{If } a\% < 1.21! \text{ Then ‘Comparison of 1.00! and 1.21!} \]
Example#2)
\[ a! = 1.23! \]
\[ b\# = 2.3345# \]
\[ \text{If } a! < b\# \text{ Then ‘Comparison of 1.2300# and 2.3345#} \]
Relation of numerical type and precision is shown below.

<table>
<thead>
<tr>
<th>Type declaration</th>
<th>Type name</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>Double precision real</td>
<td>Most precise</td>
</tr>
<tr>
<td>!</td>
<td>Single precision real</td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>Long integer</td>
<td>Least precise</td>
</tr>
<tr>
<td>%</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

(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 then 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.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>Not</td>
<td>not (A=B)</td>
<td>If not (A=B), the result is true. If (A=B), the result is false.</td>
</tr>
<tr>
<td>and</td>
<td>And</td>
<td>(A=B) and (C=D)</td>
<td>If (A=B) and (C=D), the result is true. If not (A=B) or not (C=D), the result is false.</td>
</tr>
<tr>
<td>or</td>
<td>Inclusive or</td>
<td>(A=B) or (C=D)</td>
<td>If (A=B) or (C=D), the result is true. If not (A=B) and not (C=B), the result is false.</td>
</tr>
<tr>
<td>xor</td>
<td>Exclusive or</td>
<td>(A=B) xor (C=D)</td>
<td>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.</td>
</tr>
<tr>
<td>eqv</td>
<td>Logical equivalence</td>
<td>(A=B) eqv (C=D)</td>
<td>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.</td>
</tr>
</tbody>
</table>
### 6 Elements of Language

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>imp</td>
<td>Logical implication</td>
<td>(A=B) imp (C=D)</td>
<td>If (A=B) and not (C=D), the result is false. In other case, the result is true.</td>
</tr>
</tbody>
</table>

#### (2) Bit operation

In bit operation, the following operators are available.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
<th>Truth table</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>Not (Negation)</td>
<td>X not X</td>
</tr>
<tr>
<td>and</td>
<td>And (Logical multiplication)</td>
<td>X Y X and Y</td>
</tr>
<tr>
<td>or</td>
<td>Inclusive or (Logical addition)</td>
<td>X Y X or Y</td>
</tr>
<tr>
<td>xor</td>
<td>Exclusive or (Logical exclusion)</td>
<td>X Y X xor Y</td>
</tr>
<tr>
<td>eqv</td>
<td>Logical equivalence</td>
<td>X Y X eqv Y</td>
</tr>
<tr>
<td>imp</td>
<td>Logical implication</td>
<td>X Y X imp Y</td>
</tr>
</tbody>
</table>

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>

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>32767</td>
<td>&amp;H7FFF</td>
<td>0111 1111 1111 1111</td>
</tr>
<tr>
<td>32766</td>
<td>&amp;H7FFE</td>
<td>0111 1111 1111 1110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>&amp;H0002</td>
<td>0000 0000 0000 0010</td>
</tr>
<tr>
<td>1</td>
<td>&amp;H0001</td>
<td>0000 0000 0000 0001</td>
</tr>
<tr>
<td>0</td>
<td>&amp;H0000</td>
<td>0000 0000 0000 0000</td>
</tr>
<tr>
<td>-1</td>
<td>&amp;HFFFF</td>
<td>1111 1111 1111 1111</td>
</tr>
<tr>
<td>-2</td>
<td>&amp;HFFFE</td>
<td>1111 1111 1111 1110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>-32767</td>
<td>&amp;H8001</td>
<td>1000 0000 0000 0001</td>
</tr>
<tr>
<td>-32768</td>
<td>&amp;H8000</td>
<td>1000 0000 0000 0000</td>
</tr>
</tbody>
</table>

#### <Long integer>
### 6.4.4 Character string operator

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

**Example:**

\[ \text{a$ = “HIRATA” : b$ = “INDUSTRIAL” : c$ = “ROBOT”} \]
\[ \text{d$ = a$ + “#” + b$ + “#” + c$} \]

--- “HIRATA#INDUSTRIAL#ROBOT” is substituted for d$.
### 6.4.5 Priority of Operations

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

<table>
<thead>
<tr>
<th>Priority</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>( )</td>
</tr>
<tr>
<td></td>
<td>Exponent (^)</td>
</tr>
<tr>
<td></td>
<td>Plus sign (+), Minus sign (-)</td>
</tr>
<tr>
<td></td>
<td>*, /</td>
</tr>
<tr>
<td></td>
<td>mod</td>
</tr>
<tr>
<td></td>
<td>+, -</td>
</tr>
<tr>
<td></td>
<td>Relational operators (&lt;=, &gt;=, &lt;, &gt;, &lt;&gt;, =)</td>
</tr>
<tr>
<td></td>
<td>not</td>
</tr>
<tr>
<td></td>
<td>and</td>
</tr>
<tr>
<td></td>
<td>or</td>
</tr>
<tr>
<td>Low</td>
<td>xor, eqv, imp</td>
</tr>
</tbody>
</table>
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!
Tan(x#)
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.

<table>
<thead>
<tr>
<th>Type</th>
<th>Structure</th>
<th>Program</th>
</tr>
</thead>
</table>
| Two branches #1     | ![Diagram](image1.png)                         | If Condition
|                     | Procedure                                      | Procedure
|                     | True                                           | EndIf   |
|                     | False                                          |         |
| Two branches #2     | ![Diagram](image2.png)                         | If Condition
|                     | Procedure #1                                   | Procedure
|                     | Else                                           | Procedure
|                     | True                                           |          |
|                     | False                                          |          |
| Multiple selected  | ![Diagram](image3.png)                         | Select Case Condition
| branches #1          | Condition                                      | Case #1
|                     | #1                                             | Procedure
|                     | #2                                             |          |
|                     | ...                                            |          |
|                     | #n                                             | Case #n
|                     | Proc.#1                                        | Procedure
|                     | Proc.#2                                        |          |
|                     | ....                                           |          |
|                     | Proc.#n                                        | Case Else
|                     | Proc.#n+1                                      | Procedure
|                     | End Select                                     |         |

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 nests i 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.

---

1 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% = 13  'Error flag
      EndIf
End Select
7 Structured Programming

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.

⚠️ Note

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
   End If

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.
   Example)
   For i = 0 to pate.no%
   If plate.mode(i%) = BUSY Then
      equip.mode% = BUSY
      GoTo *NEXT.CHECK
   End If
   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
'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 same 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 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
*SUB1
: GoSub *SUB2
: Return
*SUB2
: Return

*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.

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

(4) Independence of subroutine
A subroutine has to be functioned simply and it is necessary to have high independence from other program.
In the following example, the left program has to be modified as the right program.

<table>
<thead>
<tr>
<th>Job Name “sample3”</th>
</tr>
</thead>
</table>
| *MAINLOOP  
  GoSub *AAA  
  :  
  GoSub *BBB  
  :  
  goto *MAINLOOP  
  **Subroutines***  
  *AAA  
  :  
  *BBB  
  :  
  Return | This procedure has to be divided to another subroutine. |

<table>
<thead>
<tr>
<th>Job Name “sample3”</th>
</tr>
</thead>
</table>
| *MAINLOOP  
  GoSub *AAA  
  :  
  GoSub *BBB  
  :  
  GoTo *MAINLOOP  
  **Subroutines***  
  *AAA  
  :  
  GoSub *BBB  
  :  
  Return  
  *BBB  
  :  
  Return |

(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.

---

1 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!: hight! = 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

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

Guideline for Programming

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**

<table>
<thead>
<tr>
<th>Bad example</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>If addr% &lt; 10 Then</td>
<td>If addr% &lt;= 10 Then</td>
</tr>
<tr>
<td>GoTo *RB.MOVE</td>
<td>If addr% = 10 Then</td>
</tr>
<tr>
<td>Endf</td>
<td>OUTB(addr% + 100) = 1</td>
</tr>
<tr>
<td>If addr% = 10 Then</td>
<td>Endf</td>
</tr>
<tr>
<td>OUTB(addr% + 100) = 1</td>
<td>Move #1,PTP,PM(addr%)</td>
</tr>
<tr>
<td>*RB.MOVE</td>
<td>Endf</td>
</tr>
<tr>
<td>Move #1,PTP,PM(addr%)</td>
<td></td>
</tr>
</tbody>
</table>
### Example #2) Exit from the middle of selection block

<table>
<thead>
<tr>
<th>Bad example</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>If addr% = 20 Then</td>
<td>flag% = 0</td>
</tr>
<tr>
<td>OUTB(addr% + 100) = 1</td>
<td>If addr% = 20 Then</td>
</tr>
<tr>
<td>Wait INB(addr% + 200), 10</td>
<td>OUTB(addr% + 100) = 1</td>
</tr>
<tr>
<td>OUTB(addr% + 100) = 0</td>
<td>Wait INB(addr% + 200), 10</td>
</tr>
<tr>
<td>If TimeOut Then</td>
<td>If TimeOut Then flag% = 1</td>
</tr>
<tr>
<td>GoTo *NEXT.STEP</td>
<td>OUTB(addr% + 100) = 0</td>
</tr>
<tr>
<td>Endif</td>
<td>Endf</td>
</tr>
<tr>
<td>Move #1,PTP,PM(addr%)</td>
<td>If flag% = 0 Then</td>
</tr>
<tr>
<td>*NEXT.STEP</td>
<td>Move #1,PTP,PM(addr%)</td>
</tr>
<tr>
<td></td>
<td>Endf</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Bad example</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>If mode% = AUTO Then</td>
<td>flag% = 0</td>
</tr>
<tr>
<td>arm.rbt% = 1 : table.rbt% = 0</td>
<td>If mode% = AUTO Then</td>
</tr>
<tr>
<td>If (Ref(#3,SATTUS9) and &amp;h01)=0 Then</td>
<td>arm.rbt% = 1 : table.rbt% = 0</td>
</tr>
<tr>
<td>GoTo *SKIP.SEQ</td>
<td>If (Ref(#3,STATUS9) and &amp;h01)=0 Then</td>
</tr>
<tr>
<td>Endif</td>
<td>flag% = 1</td>
</tr>
<tr>
<td>arm.rbt% = 0 : table.rbt% = 0</td>
<td>Else</td>
</tr>
<tr>
<td>Else</td>
<td>arm.rbt% = 0 : table.rbt% = 0</td>
</tr>
<tr>
<td>*SKIP.SEQ</td>
<td>Endf</td>
</tr>
<tr>
<td>Disable #2</td>
<td>If mode%&lt;&gt;AUTO or flag% = 1 Then</td>
</tr>
<tr>
<td>SeqEnd #2</td>
<td>arm.rbt% = 0 : table.rbt% = 1</td>
</tr>
<tr>
<td>Endif</td>
<td>Disable #2</td>
</tr>
<tr>
<td></td>
<td>SeqEnd #2</td>
</tr>
<tr>
<td></td>
<td>Endf</td>
</tr>
</tbody>
</table>
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(O.LIFT.UP)=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%

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 an 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.

<table>
<thead>
<tr>
<th>STP type</th>
<th>Using COM port</th>
<th>OPEN statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>STP on HAC</td>
<td>COM0</td>
<td>open &quot;COM0&quot;</td>
</tr>
<tr>
<td>WinSTP</td>
<td>COMn (n&gt;=1)</td>
<td>open &quot;COMn: ...&quot;</td>
</tr>
</tbody>
</table>

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 [MAINTENANCE]-[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.

<table>
<thead>
<tr>
<th>Virtual Robot</th>
<th>Robot number [STATION NO]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROBOT 1</td>
<td>1</td>
</tr>
<tr>
<td>ROBOT 2</td>
<td>2</td>
</tr>
<tr>
<td>ROBOT 3</td>
<td>3</td>
</tr>
<tr>
<td>ROBOT 4</td>
<td>4</td>
</tr>
</tbody>
</table>

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 all communications of virtual robots are closed.
Example)
close #1 'Close port of file number #1

![Note]
If CLOSE statement is executed without the filenumber, 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

<table>
<thead>
<tr>
<th>Job Name</th>
<th>Program Name</th>
<th>Function</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>init</td>
<td>init.bas</td>
<td>System initialization</td>
<td>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.</td>
</tr>
<tr>
<td>main</td>
<td>main.bas</td>
<td>Process management</td>
<td>After &quot;init&quot; 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</td>
</tr>
<tr>
<td>robot1</td>
<td>robot1.bas</td>
<td>Robot #1 control</td>
<td>After &quot;init&quot; job completes initialization, the job controls robot #1 by indication from &quot;main&quot; job.</td>
</tr>
</tbody>
</table>
8.4.3 Motion of Robots

The position of robots and stations is shown below.

Robot #1 and #2 moves as follows.

1) Robot #1

2) Robot #2

3)
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
'******************************************************************************
8 Robot Control Programming

#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

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
endif
   goto *Main.Loop
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
'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
er.err%=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
seq end #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
end if
''Open chuck
  case CMD.CHK.OPEN
    OUTB(0.RB1.CHK.CLOSE)=0
    wait INB(1.RB1.CHK.OPEN)=1, 3  'Confirm chuck opened
    if timeout then
      err.flag%=ERR.CHK.OPEN
    endif

  'Close chuck
  case CMD.CHK.CLOSE
    OUTB(1.RB1.CHK.CLOSE)=1
    wait INB(1.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
   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.err%=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
*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

<table>
<thead>
<tr>
<th>Kind</th>
<th>Usage</th>
<th>Description</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pre-Processor</strong></td>
<td>Definition</td>
<td>Define</td>
<td>Define the specified name as a constant.</td>
</tr>
<tr>
<td></td>
<td>Macro</td>
<td>Macro</td>
<td>Define a format of macro call.</td>
</tr>
<tr>
<td></td>
<td>Header file</td>
<td>Include</td>
<td>Read the specified header file.</td>
</tr>
<tr>
<td><strong>Psuedo-instruction</strong></td>
<td>Definition</td>
<td>Job Name</td>
<td>Define the entry of a job and job name.</td>
</tr>
<tr>
<td><strong>Definable instruction</strong></td>
<td>Definition</td>
<td>Dim</td>
<td>Define as array variable.</td>
</tr>
<tr>
<td></td>
<td>DimNet</td>
<td>Define as network global variable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Global</td>
<td>Define as global variable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DimPos</td>
<td>Define the number of position memory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rem</td>
<td>Define the comment line</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GoTo</td>
<td>Jump to a specified line, then execute.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GoSub</td>
<td>Call subroutine.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Return</td>
<td>Terminate subroutine, then resume the former process.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For · Next</td>
<td>Repeat the instruction between For and Next.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If Then Else</td>
<td>Decide the condition of logical expression.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delay</td>
<td>Break temporarily the execution of job.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wait</td>
<td>Wait until conditions are satisfied.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TimeOut</td>
<td>Get the result of timeout by Wait command.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>On GoTo</td>
<td>Jump one of specified step.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>On GoSub</td>
<td>Call one of specified subroutines.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Select Case</td>
<td>Evaluate an expression and execute the processing block.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>InitGoSub</td>
<td>Initialize the subroutine-call stack.</td>
<td></td>
</tr>
<tr>
<td><strong>Interrupt control instruction</strong></td>
<td>Error control</td>
<td>On Error GoTo</td>
<td>Specify the destination at error.</td>
</tr>
<tr>
<td></td>
<td>Resume</td>
<td>Terminate error process, then resume the former process.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Err</td>
<td>Hold error code.</td>
<td></td>
</tr>
<tr>
<td><strong>Control instruction</strong></td>
<td>Job control</td>
<td>Job Start</td>
<td>Control job execution.</td>
</tr>
<tr>
<td></td>
<td>Job On</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Job Off</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetPriority</td>
<td>Get the running priority of the current job.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SetPriority</td>
<td>Set the running priority of the current job.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Move</td>
<td>Move a robot to specified coordinates.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Set</td>
<td>Set operating characteristic data of a robot.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ref</td>
<td>Deal data inside of a robot.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Seq · SeqEnd</td>
<td>Set or terminate robot sequence mode.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Finish</td>
<td>Complete MOVE in sequence mode.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hold</td>
<td>Specify or cancel the servo lock of the robot.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Disable</td>
<td>Inhibit robot movement.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Enable</td>
<td>Allow robot movement.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Calib</td>
<td>Execute automatic origin calibration.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SetRobNo</td>
<td>Set a robot number for the current job.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ClearRobNo</td>
<td>Clear the robot number for the current job.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetRobNo</td>
<td>Get the robot number for the current job.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EnableOnlineErr</td>
<td>Enable robot ONLINE mode check.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DisableOnlineErr</td>
<td>Disable robot ONLINE mode check.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RobCheckBpZone</td>
<td>Check robot position within BP/ZONE.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RobCheckCurPos</td>
<td>Check robot position nearby teaching data.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RobCheckStop</td>
<td>Check robot stopped.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RobClearErr</td>
<td>Clear robot errors.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RobSetPosRange</td>
<td>Define allowable margin of position.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Inching</td>
<td>Execute inching motion.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AxesPara</td>
<td>Make axes parameter.</td>
<td></td>
</tr>
<tr>
<td>Kind</td>
<td>Usage</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>File control</td>
<td>PosRec</td>
<td>Make one robot position record.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CollisionCheck</td>
<td>Enable or disable collision check between robots.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RobWorldPos</td>
<td>Get current robot position in the world coordinates system.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RobDistance</td>
<td>Get the distance between two robots.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RobGetCurSpeed</td>
<td>Get the current robot speed.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RobGetCurTorq</td>
<td>Get the current robot torque.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RobGetCurAveTorq</td>
<td>Get the current effective torque of a robot.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RobGetCurPos</td>
<td>Get the current encoder position of a robot.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RobReadSvoPara</td>
<td>Read servo parameter of a robot.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RobWriteSvoPara</td>
<td>Write servo parameter of a robot.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RobReadSG</td>
<td>Read system generation data of a robot.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RobWriteSG</td>
<td>Write system generation data of a robot.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Open</td>
<td>Open a communication file.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Close</td>
<td>Close a file.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Input$</td>
<td>Read the specified length of the string from a file.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Input #</td>
<td>Substitute data of a sequential file to a variable.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Line Input #</td>
<td>Read one line from a sequential file.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Print #</td>
<td>Output data to a file.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Eof</td>
<td>Examine the termination code of a file.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FreeFile</td>
<td>Get unused file number.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RchkHrcs</td>
<td>Check a HRCS protocol frame received.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ReadHrcs</td>
<td>Read a HRCS protocol frame.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WriteHrcs</td>
<td>Write a HRCS protocol frame.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EnableDSRCheck</td>
<td>Enable DSR signal check of RS232C.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DisableDSRCheck</td>
<td>Disable DSR signal check of RS232C.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EnableRTSAuto</td>
<td>Enable automatic RTS signal control of RS232C.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DisableRTSAuto</td>
<td>Disable automatic RTS signal control of RS232C.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ComFunction</td>
<td>Control RS232C signal.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetComStatus</td>
<td>Get signal status of RS232C.</td>
<td></td>
</tr>
<tr>
<td>Pulse generation</td>
<td>Pulse</td>
<td>Generate pulse. (Substitute a value for the specified period.)</td>
<td></td>
</tr>
<tr>
<td>Clock control</td>
<td>Time$</td>
<td>Get or set the current system time.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Date$</td>
<td>Get or set the current system date.</td>
<td></td>
</tr>
<tr>
<td>Network instruction</td>
<td>NetOpen</td>
<td>Open a network communication.</td>
<td></td>
</tr>
<tr>
<td>Network communication</td>
<td>NetClose</td>
<td>Close a network communication.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NetRead</td>
<td>Read data from a network communication.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NetWrite</td>
<td>Write data from a network communication.</td>
<td></td>
</tr>
<tr>
<td>Conversion instruction</td>
<td>Sin</td>
<td>Get sine.</td>
<td></td>
</tr>
<tr>
<td>Arithmetic function</td>
<td>Cos</td>
<td>Get cosine.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tan</td>
<td>Get tangent.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Atn</td>
<td>Get arctangent.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sgn</td>
<td>Get the sign of value.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Abs</td>
<td>Get absolute value.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Int</td>
<td>Remove decimals</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fix</td>
<td>Remove decimals</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Log</td>
<td>Get natural logarithms.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exp</td>
<td>Get e raised to a power.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sqr</td>
<td>Get square root.</td>
<td></td>
</tr>
<tr>
<td>Operator</td>
<td>Mod</td>
<td>Execute arithmetic division and get the remainder.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Not</td>
<td>Execute negation.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>And</td>
<td>Execute logical multiplication.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Or</td>
<td>Execute logical addition.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Xor</td>
<td>Execute exclusive logical addition.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Eqv</td>
<td>Execute logical equivalence.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Imp</td>
<td>Execute logical implication.</td>
<td></td>
</tr>
<tr>
<td>Arithmetic Constant</td>
<td>Pai</td>
<td>Get the value of pi.</td>
<td></td>
</tr>
<tr>
<td>Constant</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 9 Commands

<table>
<thead>
<tr>
<th>Kind</th>
<th>Usage</th>
<th>Description</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>Left$</td>
<td>Pick out arbitrary length from the left of a string.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mid$</td>
<td>Specify one part of a string.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Right$</td>
<td>Pick out arbitrary length from the right of a string.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Space$</td>
<td>Get a string with the arbitrary length blank.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Chr$</td>
<td>Get the character of specified character code.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>String$</td>
<td>Get the character string connected one arbitrary character.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hex$</td>
<td>Get the character string converted decimal into hexadecimal.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Str$</td>
<td>Convert numerical value into a string.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Val</td>
<td>Convert the number of a character string into actual value.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Asc</td>
<td>Get the character codes of characters.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Len</td>
<td>Get the length of a string.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>InStr</td>
<td>Get the first position of the string in another string.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ScanStr</td>
<td>Scan string data according to specified format.</td>
<td>And get the value as parameter from string by operator in the format.</td>
</tr>
<tr>
<td></td>
<td>PrintStr</td>
<td>Print string data according to specified format.</td>
<td>And put the data string of specified parameter by operator in the format.</td>
</tr>
<tr>
<td>Initialization</td>
<td>InitPos</td>
<td>Initialize position memory in STP.</td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td>ConsoleMsgOn</td>
<td>Enable to print message to STP console.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ConsoleMsgOff</td>
<td>Disable to print message to STP console.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ConsoleMsg</td>
<td>Print specified message to STP console.</td>
<td></td>
</tr>
</tbody>
</table>
9.2 How to Read Command Explanation

After the next section, all commands of HrBasic language are explained. Each explanation has the following structure.

<table>
<thead>
<tr>
<th>Command-name</th>
<th>(Type)</th>
</tr>
</thead>
</table>

Note) There are the following command types.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement</td>
<td>Command without return value. Declaration included.</td>
</tr>
<tr>
<td>Function</td>
<td>Command with return value.</td>
</tr>
<tr>
<td>Operator</td>
<td>Executes calculation of two values.</td>
</tr>
</tbody>
</table>

- 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 (_whitespace) 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

<table>
<thead>
<tr>
<th>Abs (Function)</th>
</tr>
</thead>
</table>

- **Function**
  Gets the absolute value.

- **Format**
  \texttt{Abs(Numeric-expression)}

- **Argument and Return value**
  
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{Numeric-expression}</td>
<td>Value of a number</td>
</tr>
<tr>
<td>Return value</td>
<td>Absolute value</td>
</tr>
</tbody>
</table>

- **Example**
  \texttt{b\% = Abs(-2)} \quad \text{2 is substituted for b\%}

- **Explanation**
  The absolute value of \texttt{Numeric-expression} is returned.
And (Operator)

- Function
  Executes a logical multiplication of two numbers.

- Format
  \( \text{Numeric-expression}#1 \text{ And } \text{Numeric-expression} #2 \)

- Arguments
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{Numeric-expression}#1</td>
<td>A numeric expression.</td>
</tr>
<tr>
<td>\text{Numeric-expression}#2</td>
<td>A numeric expression.</td>
</tr>
</tbody>
</table>

- Example
  a\% = &H000F\%
  b\% = &H0FFF%
  c\% = a\% And b\%  ' &H000F% substituted for c\%

- Explanation
  - The following calculation is performed.
    \[
    \begin{array}{ccc}
    X & Y & X \text{ and } Y \\
    1 & 1 & 1 \\
    1 & 0 & 0 \\
    0 & 1 & 0 \\
    0 & 0 & 0 \\
    \end{array}
    \]

  - See “6.4.3 Logical Operator”.
Asc (Function)

- **Function**
  Gets the ASCII character code.

- **Format**
  Asc(String)

- **Argument and Return value**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>String</td>
</tr>
<tr>
<td>Return</td>
<td>ASCII code (&amp;H00 to &amp;HFF)</td>
</tr>
</tbody>
</table>

- **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
  Atn(Numeric-expression)

- Argument and Return value

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Value of the ratio of B to A in the following right triangle.</td>
</tr>
<tr>
<td>Numeric-expression</td>
<td></td>
</tr>
<tr>
<td>Return value</td>
<td>Arctangent value by radian.</td>
</tr>
</tbody>
</table>

- Example
  a! = 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**
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X)-axis</td>
<td>X axis parameter as a numeric expression. Valid number is 0 through 9.</td>
</tr>
<tr>
<td>(Y)-axis</td>
<td>Y axis parameter as a numeric expression. Valid number is 0 through 9.</td>
</tr>
<tr>
<td>(Z)-axis</td>
<td>Z axis parameter as a numeric expression. Valid number is 0 through 9.</td>
</tr>
<tr>
<td>(W)-axis</td>
<td>W axis parameter as a numeric expression. Valid number is 0 through 9.</td>
</tr>
<tr>
<td>(R)-axis</td>
<td>R axis parameter as a numeric expression. Valid number is 0 through 9.</td>
</tr>
<tr>
<td>(C)-axis</td>
<td>C axis parameter as a numeric expression. Valid number is 0 through 9.</td>
</tr>
</tbody>
</table>

**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.
  \[
  \text{Axes parameter} = X \text{ axis parameter} \times 100000 + \\
  Y \text{ axis parameter} \times 10000 + \\
  Z \text{ axis parameter} \times 1000 + \\
  W \text{ axis parameter} \times 100 + \\
  R \text{ axis parameter} \times 10 + \\
  C \text{ axis parameter}
  \]

- 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**

**Function**
Executes the automatic origin calibration (A-CAL) of robot.

**Format**
Calib=#File-number[rno:Robot-number][, Axes-bits][, NoWait]

Note) “AxesBits=” is supported by HNC-580 series and HAC-8XX controller.

**Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File-number</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Robot-number</strong></td>
<td>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. Calibrating robot axes can be specified by bits. Axes bit assignment is shown below.</td>
</tr>
<tr>
<td><strong>Axes-bits</strong></td>
<td>Example) &amp;H1B --- Calibrate X, Y, W and R axis. If omitted, all implemented axes are calibrated simultaneously. It can be specified as a number or variable.</td>
</tr>
<tr>
<td><strong>NoWait</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>STATUS9</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;H80</td>
<td>&amp;H40</td>
<td>&amp;H20</td>
<td>&amp;H10</td>
<td>&amp;H8</td>
<td>&amp;H4</td>
<td>&amp;H2</td>
<td>&amp;H1</td>
<td></td>
</tr>
</tbody>
</table>

Bit no. | 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.
9 Commands

### Chr$ (Function)

- **Function**
  Gets the character string of the specified ASCII character code.

- **Format**
  
  `Chr$(Numeric-expression)`

- **Argument and Return value**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td><em>Numeric-expression</em></td>
</tr>
<tr>
<td>Return value</td>
<td></td>
</tr>
</tbody>
</table>

- **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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Nothing</td>
</tr>
<tr>
<td>Return value</td>
<td>Nothing</td>
</tr>
</tbody>
</table>

- **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

Function
Closes a file.

Format
Close[¬#File-number]

Argument

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File-number</td>
<td>A file number assigned by Open statement. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
</tbody>
</table>

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

**Function**
Enables or disables the collision check between the robots.

**Format**
- `CollisionCheck On`
- `CollisionCheck Off`

**Argument**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>On</td>
<td>To specify “On”, the collision check is enabled.</td>
</tr>
<tr>
<td>Off</td>
<td>To specify “Off”, the collision check is disabled.</td>
</tr>
</tbody>
</table>

**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 #File-number, Signal-name, Value

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File-number</strong></td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td><strong>Signal-name</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>Variable or numeral constant can be specified. The value has to be 0 or 1.</td>
</tr>
</tbody>
</table>

- **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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td><code>String</code></td>
</tr>
<tr>
<td></td>
<td>A character sting to print</td>
</tr>
<tr>
<td>Return value</td>
<td>Nothing</td>
</tr>
</tbody>
</table>

- 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.

<table>
<thead>
<tr>
<th>STP type</th>
<th>Console</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAC-8XX</td>
<td>VGA monitor</td>
</tr>
<tr>
<td>WinSTP</td>
<td>Console window</td>
</tr>
</tbody>
</table>

- 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`. 
9 Commands

ConsoleMsgOff (Statement)

- **Function**
  Disables to print a message to STP console.

- **Format**
  ConsoleMsgOff

- **Argument**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Nothing</td>
</tr>
</tbody>
</table>

- **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.

<table>
<thead>
<tr>
<th>STP type</th>
<th>Console</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAC-8XX</td>
<td>VGA monitor</td>
</tr>
<tr>
<td>WinSTP</td>
<td>Console window</td>
</tr>
</tbody>
</table>

- 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Nothing</td>
</tr>
</tbody>
</table>

- **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.

<table>
<thead>
<tr>
<th>STP type</th>
<th>Console</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAC-8XX</td>
<td>VGA monitor</td>
</tr>
<tr>
<td>WinSTP</td>
<td>Console window</td>
</tr>
</tbody>
</table>

- ConsoleMsgOff statement disables to print a message to STP console.

- See also ConsoleMsg, ConsoleMsgOff.
**Cos**

- **Function**
  Gets the value of cosine.

- **Format**
  \( \text{Cos}(\text{Numeric-expression}) \)

- **Argument and Return value**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Numeric-expression</td>
</tr>
<tr>
<td>Return value</td>
<td>Cosine of the specified value is returned. The value is from -1.0 through +1.0.</td>
</tr>
</tbody>
</table>

- **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.

\[
\cos(\text{Angle}) = \frac{B}{A}
\]

- **See also** Atn, Sin, Tan.
**Date$**

**Function**
Sets date to the system calendar.

**Format**
Date$ = *Date*string

**Argument**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Nothing</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Nothing</td>
</tr>
<tr>
<td>Return value</td>
<td>A string that contains current date with the following format.</td>
</tr>
<tr>
<td></td>
<td>&quot;yy/mm/dd&quot;</td>
</tr>
</tbody>
</table>

- **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 → `Definition-name` → `Constant`

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Definition-name</code></td>
<td>Specify an arbitrary name within sixteen alphabetic or numeric characters or period (.).</td>
</tr>
<tr>
<td><code>Constant</code></td>
<td>A number or string by literal.</td>
</tr>
</tbody>
</table>

- **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**
  - `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.

  - 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**
  \[ \text{Delay} \quad \text{Numeric-expression} \]

- **Argument**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric-expression</td>
<td>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.</td>
</tr>
</tbody>
</table>

- **Example**
  \[ \text{Delay 1.5} \quad \text{Break this job for 1.5 sec.} \]

- **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 Variable-name(Upper1 [, Upper2 [, Upper3]])
  [, Variable-name(Upper1 [, Upper2 [, Upper3]])], ...]
  ```

- **Argument**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable-Name</td>
<td>Variable name to use as an array.</td>
</tr>
<tr>
<td>Upper1</td>
<td>A minimum subscript number is always zero. Therefore, number of array elements is Upper+1 for one dimension.</td>
</tr>
<tr>
<td>Upper2</td>
<td>The dimensions of array are available up to three.</td>
</tr>
<tr>
<td>Upper3</td>
<td></td>
</tr>
</tbody>
</table>

- **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**
  - Variable name 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 name, 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”.

---

1 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**
  Declares a network global variable.

- **Format**
  \[
  \text{DimNet } \text{Variable-name}[(\text{Upper1} [, \text{Upper2} [, \text{Upper3}]], \ldots [, \text{Variable-name}[(\text{Upper1} [, \text{Upper2} [, \text{Upper3}]], \ldots ]]
  \]

- **Argument**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable-Name</td>
<td>Variable name to use as a network global variable.</td>
</tr>
<tr>
<td>Upper1</td>
<td>To declare a network global variable as array, specify a maximum subscript number of an array.</td>
</tr>
<tr>
<td>Upper2</td>
<td>A minimum subscript number is always zero.</td>
</tr>
<tr>
<td>Upper3</td>
<td>Therefore, number of array elements is ( \text{Upper} + 1 ) for one dimension.</td>
</tr>
<tr>
<td></td>
<td>The dimensions of array are available up to three.</td>
</tr>
</tbody>
</table>

- **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 name, 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”.

---

1. 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 \(\text{Number-of-Ps}\)

- **Argument**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number-of-Ps</td>
<td>Number of P records that the current job uses. Invalid value is from 1 through 8000.</td>
</tr>
</tbody>
</table>

- **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”.

---

9-28
Disable (Statement)

- **Function**
  Inhibits robot movement.

- **Format**
  Disable #File-number[rno:Robot-number]

- **Arguments**
  
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File-number</td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td>Robot-number</td>
<td>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.</td>
</tr>
</tbody>
</table>

- **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.

<table>
<thead>
<tr>
<th>Bit no.</th>
<th>Bit value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6 5 4 3 2 1 0</td>
<td>&amp;H80 &amp;H40 &amp;H20 &amp;H10 &amp;H8 &amp;H4 &amp;H2 &amp;H1</td>
</tr>
</tbody>
</table>

  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<COM-number>

- Argument

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM-number</td>
<td>COM port number by numeric expression.</td>
</tr>
<tr>
<td></td>
<td>Valid number is from 0 through 9.</td>
</tr>
</tbody>
</table>

- 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 EnableDSRCheck.
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Nothing</td>
</tr>
</tbody>
</table>

- 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.
  DisableOnlineErr statement is effective to the current job.

- See also EnableOnlineErr.
DisableRTSAuto (Statement)

- **Function**
  Disables automatic RS232C RTS signal control.

- **Format**
  DisableRTSAuto ← COM-number

- **Parameter**
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM-number</td>
<td>COM port number by numeric expression. Valid number is from 0 through 9.</td>
</tr>
</tbody>
</table>

- **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**

**Function**
Allows robot movement.

**Format**
Enable #File-number[#rno:Robot-number]

**Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File-number</strong></td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td><strong>Robot-number</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Bit no.</th>
<th>Bit value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>&amp;H0</td>
</tr>
<tr>
<td>6</td>
<td>&amp;H40</td>
</tr>
<tr>
<td>5</td>
<td>&amp;H20</td>
</tr>
<tr>
<td>4</td>
<td>&amp;H10</td>
</tr>
<tr>
<td>3</td>
<td>&amp;H8</td>
</tr>
<tr>
<td>2</td>
<td>&amp;H4</td>
</tr>
<tr>
<td>1</td>
<td>&amp;H2</td>
</tr>
<tr>
<td>0</td>
<td>&amp;H1</td>
</tr>
</tbody>
</table>

- 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM-number</td>
<td>COM port number by numeric expression. Valid number is from 0 through 9.</td>
</tr>
</tbody>
</table>

- **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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Nothing</td>
</tr>
</tbody>
</table>

- **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

- Function
  Enables automatic RS232C RTS signal control.

- Format
  EnableRTSAuto COM-number, Control-type

- Parameter
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM-number</td>
<td>COM port number by numeric expression. Valid number is from 0 through 9.</td>
</tr>
<tr>
<td>Control-type</td>
<td>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</td>
</tr>
</tbody>
</table>

- 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td><em>File-number</em> A file number assigned by Open statement. Valid range is from 0 through 47.</td>
</tr>
<tr>
<td>Return value</td>
<td>If end of file is detected, true value (-1) is returned. If not, false value (0) is returned.</td>
</tr>
</tbody>
</table>

- **Example**
  ```vbscript
  a$=Input$(1, #1)
  If Eof(1) Then GoTo *ENDFILE
  ```

- **Explanation**
  - If the specified file number indecates 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 indecates 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**
  
  \[ \text{Numeric-expression}\#1 \text{ Eqv } \text{Numeric-expression}\#2 \]

- **Arguments**
  
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{Numeric-expression}#1</td>
<td>A numeric expression.</td>
</tr>
<tr>
<td>\text{Numeric-expression}#2</td>
<td>A numeric expression.</td>
</tr>
</tbody>
</table>

- **Example**
  
  ```
  a\% = &H000F\%
  b\% = &H0FFF\%
  c\% = a\% \text{ Eqv } b\% \quad \text{"&HF00F\% substituted for } c\%."
  ```

- **Explanation**
  
  - The following calculation is performed.
    
    | X | Y | X \text{ 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Nothing</td>
</tr>
<tr>
<td>Return value</td>
<td>Job error code</td>
</tr>
</tbody>
</table>

- **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
  Exp(Numeric-expression)

- Argument and Return value

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Numeric-expression</td>
</tr>
<tr>
<td>Power number by numeric expression.</td>
<td></td>
</tr>
<tr>
<td>Return value</td>
<td>Value of exponential function.</td>
</tr>
</tbody>
</table>

- Example
  x# = 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 #File-number[rno:Robot-number]

- Arguments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File-number</td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td>Robot-number</td>
<td>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.</td>
</tr>
</tbody>
</table>

- 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 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**
  Fix(Numeric-expression)

- **Argument and Return value**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Numeric-expression</td>
</tr>
<tr>
<td>Return value</td>
<td>Integer value.</td>
</tr>
</tbody>
</table>

- **Example**
  a%=Fix(4.12)  ‘ 4 is substituted for a%  
  b%=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.

```
<table>
<thead>
<tr>
<th>Less</th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>Greater</th>
</tr>
</thead>
</table>

Int(-1.5) -1.5  Fix(-1.5) |
```

- **See also Int.**
For...To...Step - Next

- **Function**
  Repeats the execution of a program between For and Next statement.

- **Format**
  ```
  For Variable=Start-number To End-number [Step Increment] :
  Next [Variable]
  ```

- **Arguments**
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>A variable for the condition of repetition.</td>
</tr>
<tr>
<td>Start-number</td>
<td>A number of a variable to start repetition.</td>
</tr>
<tr>
<td>End-number</td>
<td>A number of a variable to terminate repetition.</td>
</tr>
<tr>
<td>Increment</td>
<td>Increment number of a variable.</td>
</tr>
</tbody>
</table>

- **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.
- 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.

Example)

' 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Nothing</td>
</tr>
<tr>
<td>Return value</td>
<td>The least unused file number from 0 through 47.</td>
</tr>
</tbody>
</table>

- 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Job-name</em></td>
<td>String type expression of a job name.</td>
</tr>
<tr>
<td>Return value</td>
<td>Integer value of the job priority.</td>
</tr>
</tbody>
</table>

- **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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Nothing</td>
</tr>
<tr>
<td>Return value</td>
<td>A robot number of a current job from 1 through 999.</td>
</tr>
<tr>
<td></td>
<td>After STP starts, a program is downloaded or ClearRobNo function is executed, GetRobNo function returns -1.</td>
</tr>
</tbody>
</table>

- **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**
  
  ```plaintext
  Global Variable-name[, Variable-name[, ...]]
  
  Array[, Array[, ...]]
  
  Array: Variable-name (Upper1[, Upper2[, Upper3]])
  ```

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable-name</td>
<td>Variable name to use as a global variable.</td>
</tr>
<tr>
<td>Upper1</td>
<td>A maximum subscript number of an array.</td>
</tr>
<tr>
<td>Upper2</td>
<td>A minimum subscript number is always zero. Therefore, number of array elements is Upper+1 for one dimension.</td>
</tr>
<tr>
<td>Upper3</td>
<td>The dimensions of array are available up to three.</td>
</tr>
</tbody>
</table>

- **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.
    
    ```plaintext
    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 name, 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”.

---

1 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 → Label

- **Argument**
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>A label of a subroutine to call.</td>
</tr>
</tbody>
</table>

- **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 calls 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.

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. *MOVELOOP).
- 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 *Label*

- **Argument**
  Parameter | Explanation |
  --- | --- |
  *Label* | 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**

- 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**
  \[
  \text{Hex$($ Numeric-expression $)}
  \]

- **Argument and Return value**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Numeric-expression</td>
</tr>
<tr>
<td>Return value</td>
<td>A string of the specified value by hexadecimal expression</td>
</tr>
</tbody>
</table>

- **Example**
  \[
  a$=\text{Hex}(30)\quad \text{“1E” is substituted for a$}.
  \]

- **Explanation**

  Integer (16-bits, 32-bits) or real type expression can be specified to \textit{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

  \textit{Numeric-expression} has the value from -32768 through 32767 and the returned string has maximum four characters.

  Relation between \textit{Numeric-expression} and the returned string is shown below.

<table>
<thead>
<tr>
<th>Value of \textit{Numeric-expression}</th>
<th>Returned string</th>
</tr>
</thead>
<tbody>
<tr>
<td>-32768 ... -1</td>
<td>8000 ... FFFF</td>
</tr>
<tr>
<td>0 ... 32767</td>
<td>0 ... 7FFF</td>
</tr>
</tbody>
</table>

  2) 32 bits long integer or real type

  \textit{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 \textit{Numeric-expression} and the returned string is shown below.

<table>
<thead>
<tr>
<th>Value of \textit{Numeric-expression}</th>
<th>Returned string</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2147483648 ... -32769</td>
<td>80000000 ... FFFF7FFF</td>
</tr>
<tr>
<td>-32768 ... -1</td>
<td>FFFF8000 ... FFFFFFFF</td>
</tr>
<tr>
<td>0 ... 32767</td>
<td>0 ... 7FFF</td>
</tr>
<tr>
<td>32768 ... 65535</td>
<td>8000 ... FFFF</td>
</tr>
<tr>
<td>65536 ... 2147483647</td>
<td>10000 ... 7FFFFFF</td>
</tr>
</tbody>
</table>

- 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 #File-number[rno:Robot-number] [ , Axis][ , Axis]…
  Hold Off #File-number[rno:Robot-number] [ , Axis][ , Axis]…

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File-number</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Robot-number</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Axis</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

- **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 Condition Then Sentence#1 [Else Sentence#2]
  2) If Condition Then Block#1
     [Else]
     [Block#2]
     EndIf

- Arguments
  Format 1)
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>A condition to branch. The condition has true value (-1) or false value (0) as the result.</td>
</tr>
<tr>
<td>Sentence#1</td>
<td>A sentence to execute when Condition is true. Multi-statement is not available.</td>
</tr>
<tr>
<td>Sentence#2</td>
<td>A sentence to execute when Condition is false. Multi-statement is not available.</td>
</tr>
</tbody>
</table>

  Format 2)
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>A condition to branch. The condition has true value (-1) or false value (0) as the result.</td>
</tr>
<tr>
<td>Block#1</td>
<td>A sentence or a program block to execute when Condition is true. Multi-statement is available.</td>
</tr>
<tr>
<td>Block#2</td>
<td>A sentence or a program block to execute when Condition is false. Multi-statement is available.</td>
</tr>
</tbody>
</table>

- 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**
  
  \[ \text{Numeric-expression}#1 \rightarrow \text{Imp} \rightarrow \text{Numeric-expression} \ #2 \]

- **Arguments**
  
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Numeric-expression}#1 )</td>
<td>A numeric expression.</td>
</tr>
<tr>
<td>( \text{Numeric-expression}#2 )</td>
<td>A numeric expression.</td>
</tr>
</tbody>
</table>

- **Example**
  
  \[ a\% = \&H00FF\% \]
  \[ b\% = \&H0F0F\% \]
  \[ c\% = a\% \text{ Imp } b\% \quad \&HFF0F\% \text{ is substituted for } c\%. \]

- **Explanation**
  
  - The following calculation is performed.
    
    | X | Y | X \text{ 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. Inchning means 0.5-second jog-motion with low speed.
  
  **Note:** Inchning function returns immediately after a robot starts to move. If the next Inchning function is executed within 0.5 second, the robot continues to move by inching.

- **Format**
  **(Linear motion)**
  Inchning Linear #File-number[rno:Robot-number], Axes, Speed
  
  **(Rotary motion)**
  Inchning Rotary #File-number[rno:Robot-number], Axes, Speed

- **Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File-number</td>
<td>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.</td>
</tr>
<tr>
<td>Robot-number</td>
<td>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.</td>
</tr>
<tr>
<td>Axes</td>
<td>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</td>
</tr>
<tr>
<td>Speed</td>
<td>A constant or variable of number specifying inching speed 0: Low speed 1: High speed</td>
</tr>
</tbody>
</table>

- **Example**
  1) Linear
     Inchning Linear #1[rno:1], 120001, 1  ‘Linear, +X, -Y, +C, High speed
  2) Rotary
     axes& = 002200
     Inchning Rotary #1[rno:1], axes&, 0  ‘Rotary, -Z, -W, Low speed

- **Explanation**
  - Inchning 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`

- Convert 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header-filename</td>
<td>A header file name to read.</td>
</tr>
</tbody>
</table>

- **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 Start-index to End-index`

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Start-index</strong></td>
<td>Start index number of STP position memory to initialize. Valid range is from 0 through 7999.</td>
</tr>
<tr>
<td><strong>End-index</strong></td>
<td>End index number of STP position memory to initialize. Valid range is from 0 through 7999.</td>
</tr>
</tbody>
</table>

- **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.

<table>
<thead>
<tr>
<th>Member</th>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-axis data</td>
<td>PXn</td>
<td>0.0</td>
</tr>
<tr>
<td>Y-axis data</td>
<td>PYn</td>
<td>0.0</td>
</tr>
<tr>
<td>Z-axis data</td>
<td>PZn</td>
<td>0.0</td>
</tr>
<tr>
<td>W-axis data</td>
<td>PWn</td>
<td>0.0</td>
</tr>
<tr>
<td>R-axis data</td>
<td>PRn</td>
<td>0.0</td>
</tr>
<tr>
<td>C-axis data</td>
<td>PCn</td>
<td>0.0</td>
</tr>
<tr>
<td>Arm</td>
<td>PARMn</td>
<td>0</td>
</tr>
<tr>
<td>M data</td>
<td>PDMn</td>
<td>255</td>
</tr>
<tr>
<td>F code</td>
<td>PDFn</td>
<td>0</td>
</tr>
<tr>
<td>S code</td>
<td>PDSn</td>
<td>0</td>
</tr>
<tr>
<td>Coordinate type</td>
<td>—</td>
<td>0</td>
</tr>
</tbody>
</table>

- 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”.
9 Commands

Input #

(Statement)

- **Function**
  Reads data from a file and put the value to the specified variables.

- **Format**
  Input #File-number, Variable [, Variable...]

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File-number</td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td>Variable</td>
<td>A variable to set the value of data read from a file.</td>
</tr>
</tbody>
</table>

- **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.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Delimiter</th>
<th>ASCII code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>Space (SP)</td>
<td>32 (&amp;H20)</td>
</tr>
<tr>
<td></td>
<td>Comma (,)</td>
<td>44 (&amp;H2C)</td>
</tr>
<tr>
<td></td>
<td>Carriage return (CR)</td>
<td>13 (&amp;H0D)</td>
</tr>
<tr>
<td>Character string</td>
<td>Comma (,)</td>
<td>44 (&amp;H2C)</td>
</tr>
<tr>
<td></td>
<td>Carriage return (CR)</td>
<td>13 (&amp;H0D)</td>
</tr>
</tbody>
</table>

  - A Linefeed (LF, ASCII code 10) after a carriage return is disregarded.

- **See also** Input$.
### Input$(Length, #File-number)

**Function**
Reads data from a file with the specified length.

**Format**
\texttt{Input$( Length, #File-number )}

**Arguments and Return value**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>Length to read</td>
</tr>
<tr>
<td>File-number</td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
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 codes\(^i\).

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Start</strong></td>
<td>Start position of String#1 to search by integer value. Valid range is from 1 through the length of String#1. If omitted, the parameter is regarded as 1.</td>
</tr>
<tr>
<td><strong>String#1</strong></td>
<td>A string to search. (The function searches this string for String#2)</td>
</tr>
<tr>
<td><strong>String#2</strong></td>
<td>A string to search for. (The function searches String#1 for this string.)</td>
</tr>
</tbody>
</table>

- **Return value**
  If String#2 is found in String#1, the found position at the top of String#2 in String#1 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Any value by numeric expression.</td>
</tr>
<tr>
<td>Return value</td>
<td>Integer value.</td>
</tr>
</tbody>
</table>

- **Example**
  
  a% = Int(4.12)  
  b% = Int(-4.12)

- **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.

  ![Number Line](image)

  Int(-1.5) = -1.5  
  Fix(-1.5) = -1

- **See also Fix.**
Job Name (Statement)

- **Function**
  Defines the entry of a job and declares job name.

- **Format**
  \[ \text{Job Name} = \textit{"Job-name"} \]

- **Argument**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job-name</td>
<td>Job name of a program.</td>
</tr>
</tbody>
</table>

- **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.

  ![Diagram](image)

  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 name\(^1\), 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.

---

\(^1\) 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

**Function**
Stops the execution of a job.

**Format**
Job “Job-name” Off

**Argument**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Job-name</em></td>
<td>Job name to stop.</td>
</tr>
</tbody>
</table>

**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) ← If Job Off, robot stops immediately.
Move #1[rno:1], PM(200) ← 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 "Job-name" On`

- **Argument**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Job-name</code></td>
<td>Job name to restart.</td>
</tr>
</tbody>
</table>

- **Example**
  Job "Robot1" On

- **Explanation**
  Job On statement restarts a job stopped by Job Off statement.
  All jobs start 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.

---

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 "Job-name" Start
  ```

- **Argument**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job-name</td>
<td>Job name to start.</td>
</tr>
</tbody>
</table>

- **Example**
  `Job "Robot1" Start`

- **Explanation**
  The Job Start statement initializes a job and starts it from the first step of a program.
  All jobs start automatically after the 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"

  *MAINLOOP
  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**
  Gets the string containing a specified number of characters from the left side of a string.

- **Format**
  Left$(String, Length)$

- **Arguments and Return value**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>String expression from which the leftmost characters are returned.</td>
</tr>
<tr>
<td>Length</td>
<td>Numeric expression indicating number of characters to return. Valid range is from 0 through 255.</td>
</tr>
</tbody>
</table>

- **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
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>String</td>
</tr>
<tr>
<td>Return value</td>
<td>Byte size of the string length.</td>
</tr>
</tbody>
</table>

- Example
  a$="Hirata Corporation"
  length%=Len(a$)  ' 18 is substituted for length%.

- Explanation
  Any character codes such as control codes¹, space and so on are counted.

¹ 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**
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File-number</strong></td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td><strong>Variable</strong></td>
<td>A string variable to set the read data.</td>
</tr>
</tbody>
</table>

- **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).

<table>
<thead>
<tr>
<th>Name</th>
<th>ASCII code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carriage return (CR)</td>
<td>13 (&amp;HD)</td>
</tr>
<tr>
<td>Linefeed (LF)</td>
<td>10 (&amp;HA)</td>
</tr>
</tbody>
</table>
Log

- **Function**
  Gets the natural logarithm of a number.

- **Format**
  Log(*Numeric-expression*)

- **Argument and Return value**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td><em>Numeric-expression</em> Any valid numeric expression greater than zero.</td>
</tr>
<tr>
<td>Return value</td>
<td>The natural logarithm of the specified value.</td>
</tr>
</tbody>
</table>

- **Example**
  
  n! = 35 / 9
  
  a# = 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.
  
  Log n (x) = Log(x) / Log(n)

- **See also Exp.**
Macro (Statement)

- **Function**
  Defines a format of a macro-call in a macro file (suffix .bas).

- **Format**
  
  Macro[Argument#1 [, Argument#2 [, Argument#3,...]]]

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument#n</td>
<td>A variable used as a parameter that is specified by a macro-call. The number of arguments is allowed up to 10.</td>
</tr>
</tbody>
</table>

- **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$**

- **Function**
  Replaces a part of a string.

- **Format**
  \[
  \text{Mid$}( String, \text{Start}[, \text{Length}]) = \text{String-expression}
  \]

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>A string variable to modify. If a string constant, a compiling error occurs.</td>
</tr>
<tr>
<td>Start</td>
<td>Character position in String where the replacement of text begins.</td>
</tr>
<tr>
<td>Length</td>
<td>Number of characters to replace. If omitted, all of string is used.</td>
</tr>
<tr>
<td>String-expression</td>
<td>String expression that replaces part of String.</td>
</tr>
</tbody>
</table>

- **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.
  - 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

<table>
<thead>
<tr>
<th>i%</th>
<th>a$ byte</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Job error</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>H r B a s i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>h H r B a s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>h a H r B a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>h a r H r B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>h a r 1 H r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>h a r 1 H a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Job error</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Varying Length

<table>
<thead>
<tr>
<th>i%</th>
<th>a$ byte</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Job error</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>H a r 1 - 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>H r r 1 - 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>H r B 1 - 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>H r B a - 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>H r B a s 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>H r B a s i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Job error</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
9 Commands

Mid$ (Function)

- Function
  Gets the string containing a specified number of characters from a string.

- Format
  Mid$ (String, Start [, Length])

- Arguments and Return value

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>String expression from which characters are returned.</td>
</tr>
<tr>
<td>Start</td>
<td>Character position in string at which the part to be taken begins. Valid range is from 0 through 255.</td>
</tr>
<tr>
<td>Length</td>
<td>Numeric expression indicating number of characters to return. Valid range is from 0 through 255.</td>
</tr>
</tbody>
</table>

- 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**
  \[ \text{Numeric-expression} \#1 \mod \text{Numeric-expression} \#2 \]

- **Arguments**
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Numeric-expression}#1 )</td>
<td>A numeric expression.</td>
</tr>
<tr>
<td>( \text{Numeric-expression}#2 )</td>
<td>A numeric expression.</td>
</tr>
</tbody>
</table>

- **Example**
  \[
  \begin{align*}
  \text{a}\% & = 19 \\
  \text{b}\% & = 6 \\
  \text{c}\% & = \text{a}\% \mod \text{b}\% \quad \text{‘1 substituted for c\%.}
  \end{align*}
  \]

- **Explanation**
  - Mod operator divides \( \text{Numeric-expression}\#1 \) by \( \text{Numeric-expression}\#2 \) 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File-number</strong></td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td><strong>Robot-number</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
| **Motion** | 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. |
| **Sub-motion** | 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. |
| **Position** | 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, , , )

---

9 Commands

9-78
--- 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.
  \begin{align*}
  \text{Dimension} & \cdots 0 \\
  \text{M-data} & \cdots 1 \\
  \text{F-code} & \cdots 99 \\
  \text{S-code} & \cdots 0 \\
  \end{align*}
  Specify zero to \(<\text{Dimension code}>\) in any case.

Example 1)
\((-12.3, 2.3, 52.1, -184.3, 0, 0, LEFTY, 0, 1, 99, 0)\)
\(\cdots X: -12.3\text{mm} Y: 2.3\text{mm} Z: 52.1\text{mm} W: -184.3\text{deg.} \)
\(\cdots R: \text{unused} C: \text{unused} \)

Example 2)
\((, , z.dis!, , , , LEFTY, 0, 1, 99, 0)\)
\(\cdots \text{position where Z axis is added to the value } z.dis! \)
\(\cdots \text{from the current position} \)

(6) Current robot position + Relative distance of axis
Format)
\[\text{HERE} + (X, Y, Z, W, R, C)\]
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)
\(\text{HERE} + (, , 10.3, , , )\)
--- 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 \( M=30\cdot39 \) data. (Refer to robot operation manual about details.)

Since “a) Standard format” can operate only one position, after sequentially addressed multiple PMs with \( M=30\cdot39 \) 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td><em>Network-ID</em></td>
</tr>
<tr>
<td>Returned</td>
<td>Variable specifying a network identifier returned by NetOpen function.</td>
</tr>
</tbody>
</table>

- **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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Station-Number</td>
</tr>
<tr>
<td></td>
<td>A numeric expression specifying a station number assigned in the network definition. Valid station number is from 0 through 127.</td>
</tr>
<tr>
<td>Return value</td>
<td>Network identifier for the communication that is opened normally.</td>
</tr>
</tbody>
</table>

- **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%)
:rszize%=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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network-ID</td>
<td>Variable specifying a network identifier returned by NetOpen function.</td>
</tr>
<tr>
<td>Data-buffer</td>
<td>Variable for reading data buffer. String variable is available.</td>
</tr>
</tbody>
</table>

- Option
  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)
  : station%=2
  nid%=NetOpen(station%)
  :
  wlen%=NetWrite(nid%,a%(0),10) 'send a%(0)-a%(4)
  :
  NetClose(nid%)
  :
  ``

  Station #2

  ```
  Dim a%(10)
  : station%=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)

```vba
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)

```vba
*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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arguments</strong></td>
<td></td>
</tr>
<tr>
<td>Network-ID</td>
<td>Variable specifying a network identifier returned by NetOpen function.</td>
</tr>
<tr>
<td>Data-buffer</td>
<td>Variable for writing data buffer. String variable is available.</td>
</tr>
<tr>
<td>Data-size</td>
<td>Byte size to write.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>Data size actually sent. Normally, this size is the same as Data-size.</td>
</tr>
</tbody>
</table>

- **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**

- Example)

  ```
  Dim a%(10)
  : staion%=2
  nid%=NetOpen(station%)
  :
  wlen%=NetWrite(nid%,a%(0),10)  'send a%(0)-a%(4)
  :
  NetClose(nid%)
  :
  ```
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**
  \texttt{Not \textit{Numeric-expression}}

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{Numeric-expression}</td>
<td>A numeric expression.</td>
</tr>
</tbody>
</table>

- **Example**
  
  
  \texttt{a\% = &H00FF\%}  
  \texttt{b\% = Not a\% \ ' &HFF00\% substituted for b\%.}

- **Explanation**
  
  - The following calculation is performed.
    
    \[
    \begin{array}{cc}
    X & \text{not} \ X \\
    1 & 0 \\
    0 & 1 \\
    \end{array}
    \]
  
  - See “6.4.3 Logical Operator”.
On Error GoTo (Statement)

- **Function**
  Defines a destination line to jump when a job error occurs.

- **Format**
  a) Registration of error routine
     
  ```v
  On Error GoTo Label
  ```
  b) Clearing registration of error routine
     
  ```v
  On Error GoTo 0
  ```

- **Argument**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>A label to jump. The label represents the entry of an error routine.</td>
</tr>
</tbody>
</table>

- **Example**

  ```v
  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**

  ```v
  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.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>There are the following rules for using a label.</td>
</tr>
<tr>
<td>➢ The top of label name has to be an asterisk “ * “.</td>
</tr>
<tr>
<td>➢ Except asterisk, the first character of label name has to be alphabetic.</td>
</tr>
<tr>
<td>➢ Except asterisk, available characters in label name are alphabetic, numerical or period “ . “, regardless of upper or lower case.</td>
</tr>
<tr>
<td>➢ 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).</td>
</tr>
<tr>
<td>➢ The length of label name is maximum 16 characters except asterisk.</td>
</tr>
<tr>
<td>➢ Label name definition has to be written at the top of one line.</td>
</tr>
</tbody>
</table>

- See “4.5 Error Handling”.

-
On...GoSub / On...GoTo (Statement)

- **Function**
  Branches to one of labels evaluating a numeric expression.

- **Format**
  
  $$\text{On} \quad \text{Numeric-expression} \quad \text{GoTo} \quad \text{Label1} \quad [\text{Label2...}]$$
  
  $$\text{On} \quad \text{Numeric-expression} \quad \text{GoSub} \quad \text{Label1} \quad [\text{Label2...}]$$

- **Arguments**
  
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric-expression</td>
<td>A numeric expression containing the value from 0 through 255.</td>
</tr>
<tr>
<td>Label</td>
<td>A label to branch.</td>
</tr>
</tbody>
</table>

- **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**

- 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.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File-name</td>
<td>String constant or variable specifying a file name to open.</td>
</tr>
<tr>
<td></td>
<td>Specify the following file modes.</td>
</tr>
<tr>
<td></td>
<td>APPEND: A file is opened as a sequential file and data will be appended to it.</td>
</tr>
<tr>
<td></td>
<td>BINARY: A file is opened as a binary file.</td>
</tr>
<tr>
<td></td>
<td>INPUT: A file is opened as a sequential file with data input mode.</td>
</tr>
<tr>
<td></td>
<td>OUTPUT: A file is opened as a sequential file with data output mode.</td>
</tr>
<tr>
<td></td>
<td>RANDOM: A file is opened as a random access file.</td>
</tr>
<tr>
<td></td>
<td>When the mode except INPUT specified, a file is created automatically if a file is not found.</td>
</tr>
<tr>
<td>File-mode</td>
<td>Specify the following access type.</td>
</tr>
<tr>
<td></td>
<td>READ: Allows only reading.</td>
</tr>
<tr>
<td></td>
<td>WRITE: Allows only writing.</td>
</tr>
<tr>
<td></td>
<td>READWRITE: Allows both reading and writing.</td>
</tr>
<tr>
<td></td>
<td>If omitted, READWRITE is selected implicitly.</td>
</tr>
<tr>
<td>Access-type</td>
<td>Constant or variable specifying the number assigned for the opened file.</td>
</tr>
<tr>
<td></td>
<td>After the file is opened, this number has to be used to access the file.</td>
</tr>
<tr>
<td></td>
<td>Available number is 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td>File-number</td>
<td>Constant or variable specifying the number assigned for the opened file.</td>
</tr>
</tbody>
</table>

- 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\textsubscript{\textit{n}}:Settings\ As \textnumero\ 
     \textsl{RobType}=	extit{Robot-type}\textsubscript{\textit{[\textnumero]Robot-list}}
     
     Note) Only constant (literal) is available for underline.
  
  b) Two arguments of COM port and communication settings
     
     Open COM\textsubscript{\textit{n}}\ Settings\ As \textnumero\ 
     \textsl{RobType}=	extit{Robot-type}\textsubscript{\textnumero}Robot-list\]
     
     Note) Constant (literal) or variable is available for underline.

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{n} \ (Port number)</td>
<td>Numerical literal of a COM port number to communicate. See “4.4.2 How to Communicate with Peripheral Device” about details.</td>
</tr>
<tr>
<td>\textit{Settings}</td>
<td>Specify RS232C communication parameters as the following format. \textit{Speed}, \textit{Parity}, \textit{Data-length}, \textit{Stop-bits} \br After STP system starts or a program is downloaded, parameters are initialized to default settings. \br \textit{Speed}: Communication speed by bit-per-second (bps). Following values are available. 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 \ [default:115200]\br \textit{Parity}: Parity bit of communication. \br O: Odd parity \br E: Even parity \br N: None parity \ [default:N]\br \textit{Data-length}: Bit length of one character data. \br 7, 8 \ [default:8]\br \textit{Stop-bits}: Stop bits of communication. \br 1, 2 \ [default:1]\br One or all of items can be omitted. If omitted, the last setting is used for communication.</td>
</tr>
<tr>
<td>\textit{File-number}</td>
<td>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, &quot;\textnumero&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td>\textit{Robot-type}</td>
<td>Specify a number of a robot controller type when a robot communication port is opened. A variable is not available. \br HNC-580 series: 580 \br HAC-8XX(COM0): 580 \br Other: 0 or omit “RobType=”.</td>
</tr>
</tbody>
</table>
### 9-Commands

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robot-list</td>
<td>This must be specified when Robot-type=580. 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. Robot-No[, Robot-No][, Robot-No][, Robot-No] Valid range of a robot number is from 1 through 999.</td>
</tr>
</tbody>
</table>

#### 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”. 

---

9-98
**Or (Operator)**

- **Function**
  
  Executes a logical addition of two numbers.

- **Format**

  \[
  \text{Numeric-expression}#1 \text{ Or} \text{ Numeric-expression }#2
  \]

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{Numeric-expression}#1</td>
<td>A numeric expression.</td>
</tr>
<tr>
<td>\text{Numeric-expression}#2</td>
<td>A numeric expression.</td>
</tr>
</tbody>
</table>

- **Example**

  a\% = &H000F\%
  b\% = &H0FFF\%
  c\% = a\% Or b\%   ' &H0FF% 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Nothing</td>
</tr>
<tr>
<td>Return value</td>
<td>The value of pi.</td>
</tr>
</tbody>
</table>

- **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**
  \[ \text{PosRec}(X\text{-axis}, Y\text{-axis}, Z\text{-axis}, W\text{-axis}, R\text{-axis}, C\text{-axis}, \text{Arm-data}, \text{Coordinate-type}, M\text{-data}, F\text{-code}, S\text{-code}) \]

- **Arguments and Return value**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>X\text{-axis}</td>
<td>A numeric expression specifying the value of axis coordinate.</td>
</tr>
<tr>
<td>Y\text{-axis}</td>
<td></td>
</tr>
<tr>
<td>Z\text{-axis}</td>
<td></td>
</tr>
<tr>
<td>W\text{-axis}</td>
<td></td>
</tr>
<tr>
<td>R\text{-axis}</td>
<td></td>
</tr>
<tr>
<td>C\text{-axis}</td>
<td></td>
</tr>
<tr>
<td>Arm-data</td>
<td>The following words of the robot arm position have to be specified. (\text{LEFTY: Lefty position}) (\text{RIGHTY: Righty position})</td>
</tr>
<tr>
<td>Coordinate-type</td>
<td>A numeric expression specifying a coordinate type. Only zero value is available now.</td>
</tr>
<tr>
<td>M-data,</td>
<td>A numeric expression specifying M data. Valid range is from 0 through 255.</td>
</tr>
<tr>
<td>F-code,</td>
<td>A numeric expression specifying F code. Valid range is from 0 through 255.</td>
</tr>
<tr>
<td>S-code,</td>
<td>A numeric expression specifying S code. Valid range is from 0 through 255.</td>
</tr>
</tbody>
</table>

- **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.
Print #
(Statement)

- **Function**
  Writes character strings or numerals specified in expressions to a file.

- **Format**
  ```
  Print# File-number, Expression [1,1 Expression...]1,1 ];1
  ```

- **Arguments**
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File-number</strong></td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td><strong>Expression</strong></td>
<td>A string or numeric expression of data to write.</td>
</tr>
</tbody>
</table>

- **Example**
  ```
  Print #1, “ABC”
  ```

- **Explanation**
  - Basically, Print# statement writes data with the following format adding CR/LF code to the end.
    ```
    TEXT CR LF
    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”
      ```
    - 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
      ```
  - 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.
      ```
      Print# File-number, Expression[; Expression...
      ```
      Example) Print #1, 123; “ABC”; ””; ”VWXYZ”; -9876
    ```
    (1) (2) (3) (4) (5)
    ```
    123 ABC C R LF
    ```
    - b) A delimiter is comma and the end of sentence is expression.
      ```
      Print# File-number, Expression[, Expression...]
      ```
When comma is used for delimiter, spaces are filled to the end of 14 characters.

Example) Print #1, "ABC", -9876, “XYZ”

```
ABC                          -9876
```

14 characters

```
XYZ                          CR    LF
```

14 characters

c) A delimiter is semicolon or comma and the end of sentence is expression.

Format) Print #File-number,

```
Expression [, | Expression [, | Expression ... ]]
```

When comma is used for delimiter, spaces are filled to the end of 14 characters.

Example) Print #1, 123: "ABC", –9876

```
123 ABC                          9876
```

14 characters

d) The end of sentence is semicolon.

Format) Print #File-number,

```
Expression [, | Expression [, | Expression ... ] ;]
```

When the sentence is terminated by a semicolon, CR/LF code is not added at the end.

Example) Print #1, 123: "ABC", -9876:

```
123 ABC                          -9876
```

14 characters

f) The end of sentence is comma.

Format) Print #File-number,

```
Expression [, | Expression [, | Expression ... ] ,]
```

When the sentence is terminated by a comma, CR/LF code is not added at the end.

Example) Print #1, 123: "ABC",

```
123 ABC                          -9876
```

14 characters

⚠️ Note

Total length of writing data has to be 255 bytes or less including space, minus or CR/LF.
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 = String-variable, Format [, Parameter#1 [, Parameter#2, ...]]`

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>String-variable</code></td>
<td>A string variable to output.</td>
</tr>
<tr>
<td><code>Format</code></td>
<td>A string expression specifying format to output.</td>
</tr>
<tr>
<td><code>Parameter</code></td>
<td>A variable containing the value that is converted to a string according to a parameter descriptor.</td>
</tr>
</tbody>
</table>

- **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.

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Function</th>
<th>Example</th>
</tr>
</thead>
</table>
| `%[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. | - “12345” is contained in a parameter variable.  
  Descriptor Converted  
  `%2d" "12345"  
  `%7d" "0012345"  
  `%d" "12345"  
  - “-12345” is contained in a parameter variable.  
  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. | - “&H0ABC” is contained in a parameter variable.  
  Descriptor Converted  
  `%2x" "abc"  
  `%4x" "0abc"  
  `%x" "abc" |
### %+[N]X

A parameter is converted to hexadecimal integer expression by uppercase.

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.

- **Example**
  - “&H0ABC” is contained in a parameter variable.
    
    | Descriptor | Converted |
    |------------|-----------|
    | "%2x"     | "ABC"    |
    | "%4x"     | "0ABC"   |
    | "%x"      | "ABC"    |

### %+[N.M]f

A parameter is converted to floating expression as real value.

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.

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.

If N,M omitted, the converted string is outputted as it is.

- **Example**
  - “123.456” is contained in a parameter variable.
    
    | Descriptor | Converted |
    |------------|-----------|
    | "%4.2f"   | "123.45" |
    | "%7.2f"   | "0123.45"|
    | "%f"      | "123.456"|
  - “-123.456” is contained in a parameter variable.
    
    | Descriptor | Converted |
    |------------|-----------|
    | "%4.2f"   | "-123.45"|
    | "%8.2f"   | "-0123.45"|
    | "%f"      | "-123.456"|

### %+[N]s

A parameter is converted to string expression.

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.

- **Example**
  - “abcdefgh” is contained in a parameter variable.
    
    | 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**
  \texttt{Pluse \Variable=Expression#1, Expression#2}

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>\Variable</td>
<td>A variable to be pulsed. String variable is not available.</td>
</tr>
<tr>
<td>Expression#1</td>
<td>A number substituted for a variable.</td>
</tr>
<tr>
<td>Expression#2</td>
<td>The period of pulsing substitution by second. Available range is form 0.000 through 2147483.647 second</td>
</tr>
</tbody>
</table>

- **Example**
  \texttt{Pluse OUTB(0)=1, 2.0} ' OUTB(0)=1 for 2.0 sec.
  \texttt{Pluse OUTD(1)=&HFF, 3} ' OUTD(1)=&HFF for 3 sec.
  \texttt{Pluse a%=10, 5.1} ' a%=10 for 5.1 sec.

- **Explanation**
  Pulse statement sets the value of \texttt{Expression#1} to \Variable for the period of \texttt{Expression#2}.
  After the time of \texttt{Expression#2} passes, \Variable resumes the previous value.
RchkHrcs (Function)

- **Function**
  Checks a HRCS protocol frame received.

- **Format**
  RchkHrcs (*File-number*)

- **Argument and Return value**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td><em>File-number</em></td>
</tr>
<tr>
<td></td>
<td>A file number corresponded to the communication port must be specified.</td>
</tr>
<tr>
<td></td>
<td>Valid range is from 0 through 47.</td>
</tr>
<tr>
<td></td>
<td>A constant or variable can be specified.</td>
</tr>
<tr>
<td>Return value</td>
<td>If a HRCS frame has been received, the function returns true value (-1).</td>
</tr>
<tr>
<td></td>
<td>If not, it returns false value (0).</td>
</tr>
</tbody>
</table>

- **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**

  It is checked whether the following HRCS protocol frame has been received or not.

<table>
<thead>
<tr>
<th>sTx</th>
<th>TEXT</th>
<th>eTx</th>
<th>LRC</th>
</tr>
</thead>
</table>

  - **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)**
    LRC 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.

---

1 See “Appendix B LRC Calculation”.
ReadHrcs (Statement)

● Function
Reads a HRCS protocol frame.

● Format
ReadHrcs #File-number, Variable

● Arguments
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File-number</td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td>Variable</td>
<td>A variable to set a received HRCS protocol frame.</td>
</tr>
</tbody>
</table>

● 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.

<table>
<thead>
<tr>
<th>STX</th>
<th>TEXT</th>
<th>ETX</th>
<th>LRC</th>
</tr>
</thead>
</table>

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

● See also RchkHrcs, WriteHrcs.

See “Appendix B LRC Calculation”.

---

1 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File-number</td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td>Robot-number</td>
<td>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.</td>
</tr>
<tr>
<td>Reserved-memory</td>
<td>A reserved memory in a robot to set. Available memories are shown below.</td>
</tr>
<tr>
<td>Data</td>
<td>Setting data to a robot reserved memory. The valid ranges of values are shown below.</td>
</tr>
</tbody>
</table>
|              | 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  |

**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], EXPARA(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

```plaintext
Ref(#File-number[[rno:Robot-number]], Reserved-memory)
```

#### Arguments and Return value

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File-number</strong></td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td><strong>Robot-number</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>A reserved memory in a robot to get. Available memories are shown below.</td>
</tr>
</tbody>
</table>
| **Reserved-memory** | 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 · &HFFFFFFFF  
n=500-1000 0.0 · 2147483.647  
n=1001-1099 &H0·&HFFFFFFFF |
9 Commands

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~-[Comment]

- Argument

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment</td>
<td>Write comment (annotation)</td>
</tr>
</tbody>
</table>

- 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 \[Next | Label\]

- **Argument**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Omitted)</td>
<td>Resume executing the step where a job error occurs.</td>
</tr>
<tr>
<td>Next</td>
<td>Resume executing the next of the step where a job error occurs.</td>
</tr>
<tr>
<td>Label</td>
<td>Resume executing a program at the specified label.</td>
</tr>
</tbody>
</table>

- **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. *MOVELOOP).
- 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[Label]

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Omitted)</td>
<td>A program exits a subroutine and then returns to the next of the step calling the subroutine.</td>
</tr>
<tr>
<td>Label</td>
<td>A program exits a subroutine and then returns to the specified label.</td>
</tr>
</tbody>
</table>

- **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 maintainability of a program. From the viewpoint of structured programming, “Return Label” must not be used.

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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>String</strong></td>
<td>String expression from which the rightmost characters are returned.</td>
</tr>
<tr>
<td><strong>Length</strong></td>
<td>Numeric expression indicating number of characters to return. Valid range is from 0 through 255.</td>
</tr>
</tbody>
</table>

- **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$, Mid$. 

---

9-117
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File-number</td>
<td>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.</td>
</tr>
<tr>
<td>Robot-number</td>
<td>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.</td>
</tr>
<tr>
<td>BP/ZONE-number</td>
<td>BP/ZONE number to check. Valid range is from 1 through 8.</td>
</tr>
</tbody>
</table>

- 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File-number</strong></td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td><strong>Robot-number</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Position-address</strong></td>
<td>Position address of a robot compared with the current position. Valid range is from 0 through 3999.</td>
</tr>
<tr>
<td><strong>Axes</strong></td>
<td>Robot axes flags to check the current position.</td>
</tr>
</tbody>
</table>

- **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.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axis</td>
<td>-</td>
<td>-</td>
<td>C</td>
<td>R</td>
<td>W</td>
<td>Z</td>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>Bit value</td>
<td>$2^7$ (128)</td>
<td>$2^6$ (64)</td>
<td>$2^5$ (32)</td>
<td>$2^4$ (16)</td>
<td>$2^3$ (8)</td>
<td>$2^2$ (4)</td>
<td>$2^1$ (2)</td>
<td>$2^0$ (1)</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File-number</strong></td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td><strong>Robot-number</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>If a robot is stopping now, the function returns true value (-1). If not, it returns false value (0).</td>
</tr>
</tbody>
</table>

- **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 #File\_number\{[rno:\_Robot\_number]\}

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File_number</strong></td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td><strong>Robot_number</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

- **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.
**RobDistance**

**Function**

Gets the distance between two robots in the world coordinates system used for the robot collision check.

Note) Only supported by HAC-8XX controller.

**Format**

```
RobDistance COM-port#1, Robot-number#1, COM-port#2, Robot-number#2, Variable
```

**Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM-port#1</td>
<td>A numeric expression specifying a COM port number to control the first robot. Only COM0 is available for HNC-8XX controller.</td>
</tr>
<tr>
<td>Robot-number#1</td>
<td>A numeric expression specifying the first robot station number. Valid range is from 1 through 999.</td>
</tr>
<tr>
<td>COM-port#2</td>
<td>A numeric expression specifying a COM port number to control the second robot. Only COM0 is available for HNC-8XX controller.</td>
</tr>
<tr>
<td>Robot-number#2</td>
<td>A numeric expression specifying the second robot station number. Valid range is from 1 through 999.</td>
</tr>
<tr>
<td>Variable</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Example**

```
' Get the distance between robot#1 and robot#2 of COM0.
comno% = 0  ' COM0
RobDistance comno%, 1, comno%, 2, distance#
```

**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.
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[#File-number[[rno:Robot-number]], Motor-torque
RobGetCurAveTorq[#ファイル番号[[rno:ロボット番号]], 読出しトルク

Arguments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File-number</td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td>Robot-number</td>
<td>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.</td>
</tr>
<tr>
<td>Motor-torque</td>
<td>One dimension array to set motor effective torque. Only long integer (32-bits) type variable (&amp;) 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.</td>
</tr>
</tbody>
</table>

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**

**Function**
Reads the current robot position indicated by the motor encoder.
Supported by HAX-8XX controller.

**Format**
`RobGetCurPos [#File-number][rno:Robot-number], Position-memory`

**Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File-number</strong></td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td><strong>Robot-number</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Position-memory</strong></td>
<td>STP P memory to set the current robot position. If P memory is not specified, a compiling error occurs.</td>
</tr>
</tbody>
</table>

**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 #File-number[rno:Robot-number], Motor-speed

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File-number</td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td>Robot-number</td>
<td>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.</td>
</tr>
<tr>
<td>Motor-speed</td>
<td>One dimension array to set motor speed. Only long integer (32-bits) type variable (&amp;) 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.</td>
</tr>
</tbody>
</table>

- **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=#File-number[[rno:Robot-number]], Motor-torque

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File-number</td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td>Robot-number</td>
<td>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.</td>
</tr>
<tr>
<td>Motor-torque</td>
<td>One dimension array to set motor torque. Only long integer (32-bits) type variable (&amp;) 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.</td>
</tr>
</tbody>
</table>

- **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.
RobReadSG

**Function**
Reads SG (System Generation) data of a robot.
Note) Only supported by HNC-580 series and HAC-8XX controller.

**Format**

```
RobReadSG File-number[rno:Robot-number], SG-group-name, Array-variable
```

**Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File-number</td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td>Robot-number</td>
<td>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.</td>
</tr>
<tr>
<td>SG-group-name</td>
<td>A string constant, variable specifying a SG group name of SG data to get. The format of a SG group name is: &quot;Main-group-name¥Sub-group-name&quot; See “Explanation”.</td>
</tr>
<tr>
<td>Array-variable</td>
<td>A top elements of an array variable to which SG data is read. Long integer (&amp;), 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.</td>
</tr>
</tbody>
</table>

**Example**

```
' Define an array that has sufficient area for SG data.
Dim sg.data!(50)
:
' Reads SG data of "LIMIT¥AREA LIMIT" group to
  sg.data!(1), sg.data!(2),...
RobReadSG #fno%[rno:1], "LIMIT¥AREA LIMIT", sg.data!(1)
' Modify SG data.
sg.data!(5) = sg.data!(5) - 5.0!  ' upper limit Z to -5.0mm
sg.data!(6) = sg.data!(6) + 2.0!  ' lower limit Z to +2.0mm
' Write SG data.
RobWriteSG #fno%[rno:1], "LIMIT¥AREA LIMIT", sg.data!(1)
```

**Explanation**

- 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"
    - "MAINTEN¥EXPANSION A"
“MAINTENANCE EXPANSION B”
“MAINTENANCE MAINTENANCE DATA”
“ORIGIN SET-UP SYSTEM”
“ORIGIN AXIS DIRECTION”
“ORIGIN AXIS SELECT”
“ADJUST AR TYPE ADJUST”
“ADJUST MB 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.

<table>
<thead>
<tr>
<th>Element of array</th>
<th>Data name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>sg.data!(1)</td>
<td>UPPER LMT A</td>
<td>Upper area limit of X(A) axis</td>
</tr>
<tr>
<td>sg.data!(2)</td>
<td>LOWER LMT A</td>
<td>Lower area limit of X(A) axis</td>
</tr>
<tr>
<td>sg.data!(3)</td>
<td>UPPER LMT B</td>
<td>Upper area limit of Y(B) axis</td>
</tr>
<tr>
<td>sg.data!(4)</td>
<td>LOWER LMT B</td>
<td>Lower area limit of Y(B) axis</td>
</tr>
<tr>
<td>sg.data!(5)</td>
<td>UPPER LMT Z</td>
<td>Upper area limit of Z axis</td>
</tr>
<tr>
<td>sg.data!(6)</td>
<td>LOWER LMT Z</td>
<td>Lower area limit of Z axis</td>
</tr>
<tr>
<td>sg.data!(7)</td>
<td>UPPER LMT W</td>
<td>Upper area limit of W axis</td>
</tr>
<tr>
<td>sg.data!(8)</td>
<td>LOWER LMT W</td>
<td>Lower area limit of W axis</td>
</tr>
<tr>
<td>sg.data!(9)</td>
<td>UPPER LMT R</td>
<td>Upper area limit of R axis</td>
</tr>
<tr>
<td>sg.data!(10)</td>
<td>LOWER LMT R</td>
<td>Lower area limit of R axis</td>
</tr>
<tr>
<td>sg.data!(11)</td>
<td>UPPER LMT C</td>
<td>Upper area limit of C axis</td>
</tr>
<tr>
<td>sg.data!(12)</td>
<td>LOWER LMT C</td>
<td>Lower area limit of C axis</td>
</tr>
</tbody>
</table>

♦ 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

### Function
Reads servo parameter of a robot.

Note: Only supported by HNC-580 series and HAC-8XX controller.

### Format
```
RobReadSvoPara #File-number[rno:Robot-number], Axis-number , Array-variable
```

### Arguments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File-number</strong></td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td><strong>Robot-number</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
| **Axis-number** | A numeric expression specifying an axis number to get servo parameter. The axis numbers are:

<table>
<thead>
<tr>
<th>Axis</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-axis</td>
<td>1</td>
</tr>
<tr>
<td>Y-axis</td>
<td>2</td>
</tr>
<tr>
<td>Z-axis</td>
<td>3</td>
</tr>
<tr>
<td>W-axis</td>
<td>4</td>
</tr>
<tr>
<td>R-axis</td>
<td>5</td>
</tr>
<tr>
<td>C-axis</td>
<td>6</td>
</tr>
<tr>
<td><strong>Array-variable</strong></td>
<td>A top elements of an array variable to which servo parameter is read. Only long integer type (&amp;) 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.</td>
</tr>
</tbody>
</table>

### Example

- Define an array that has sufficient area for servo parameter.
  ```vba```
  Dim svo.para&(50)
- : 
  ```vba```
  ' 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.
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 [#File-number][[rno:Robot-number]]
  [, [X-axis][, [Y-axis][, [Z-axis][, [W-axis][, [R-axis][, [C-axis]]]]]]]
  ```

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File-number</td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td>Robot-number</td>
<td>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.</td>
</tr>
<tr>
<td>X-axis</td>
<td>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.</td>
</tr>
<tr>
<td>Y-axis</td>
<td></td>
</tr>
<tr>
<td>Z-axis</td>
<td></td>
</tr>
<tr>
<td>W-axis</td>
<td></td>
</tr>
<tr>
<td>R-axis</td>
<td></td>
</tr>
<tr>
<td>C-axis</td>
<td></td>
</tr>
</tbody>
</table>

- **Example**
  - 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

- **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\(\text{COM-port, Robot-number, P-memory}\)

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
</table>
  | COM-port    | A numeric expression specifying a COM port number to control a robot.  
                 Only COM0 is available for HNC-8XX controller.                     |
  | Robot-number| A numeric expression specifying the robot station number. Valid range is from 1 through 999. |
  | P-memory    | 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:
                 PARM(n) = 0
                 PDM(n) = 255
                 PDF(n) = 0
                 PDS(n) = 0

- **Example**
  ```plaintext```
  ```-----
  comno% = 0  \text{‘} \text{COM0}
  robno% = 1  \text{‘} \text{Robot#1}
  RobWorldPos comno%, robno%, P(10) \text{‘} \text{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.
### RobWriteSG (Statement)

- **Function**
  Writes SG (System Generation) data of a robot.
  Note) Only supported by HNC-580 series and HAC-8XX controller.

- **Format**
  \[ \text{RobWriteSG} \#\text{File-number}(\text{[rno:Robot-number]}), \text{SG-group-name}, \text{Array-variable} \]

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File-number</strong></td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td><strong>Robot-number</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>SG-group-name</strong></td>
<td>A string constant, variable specifying a SG group name of SG data to set. The format of a SG group name is: &quot;Main-group-name¥Sub-group-name&quot; See &quot;Explanation&quot;.</td>
</tr>
<tr>
<td><strong>Array-variable</strong></td>
<td>A top elements of an array variable that contains SG data to write. Long integer (&amp;), single precision real (!), double precision real (#) and string ($) type are available for an array. The writing size of SG data varies by the kind of SG group.</td>
</tr>
</tbody>
</table>

- **Example**
  'Define an array that has sufficient area for SG data.
  Dim sg.data!(50)
  :
  'Reads SG data of "LIMIT¥AREA LIMIT" group to
  'sg.data!(1), sg.data!(2),...
  RobReadSG #fno%[rno:1], "LIMIT¥AREA LIMIT", sg.data!(1)
  'Modify SG data.
  sg.data!(5) = sg.data!(5) * 5.0! 'upper limit Z to -5.0mm
  sg.data!(6) = sg.data!(6) + 2.0! 'lower limit Z to +2.0mm
  'Write SG data.
  RobWriteSG #fno%[rno:1], "LIMIT¥AREA LIMIT", sg.data!(1)

- **Explanation**
  - 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.

<table>
<thead>
<tr>
<th>Element of array</th>
<th>Data name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>sg.data!(1)</td>
<td>UPPER LMT A</td>
<td>Upper area limit of X(A) axis</td>
</tr>
<tr>
<td>sg.data!(2)</td>
<td>LOWER LMT A</td>
<td>Lower area limit of X(A) axis</td>
</tr>
<tr>
<td>sg.data!(3)</td>
<td>UPPER LMT B</td>
<td>Upper area limit of Y(B) axis</td>
</tr>
<tr>
<td>sg.data!(4)</td>
<td>LOWER LMT B</td>
<td>Lower area limit of Y(B) axis</td>
</tr>
<tr>
<td>sg.data!(5)</td>
<td>UPPER LMT Z</td>
<td>Upper area limit of Z axis</td>
</tr>
<tr>
<td>sg.data!(6)</td>
<td>LOWER LMT Z</td>
<td>Lower area limit of Z axis</td>
</tr>
<tr>
<td>sg.data!(7)</td>
<td>UPPER LMT W</td>
<td>Upper area limit of W axis</td>
</tr>
<tr>
<td>sg.data!(8)</td>
<td>LOWER LMT W</td>
<td>Lower area limit of W axis</td>
</tr>
<tr>
<td>sg.data!(9)</td>
<td>UPPER LMT R</td>
<td>Upper area limit of R axis</td>
</tr>
<tr>
<td>sg.data!(10)</td>
<td>LOWER LMT R</td>
<td>Lower area limit of R axis</td>
</tr>
<tr>
<td>sg.data!(11)</td>
<td>UPPER LMT C</td>
<td>Upper area limit of C axis</td>
</tr>
<tr>
<td>sg.data!(12)</td>
<td>LOWER LMT C</td>
<td>Lower area limit of C axis</td>
</tr>
</tbody>
</table>

- 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.
Function
Writes servo parameter of a robot.
Note) Only supported by HNC-580 series and HAC-8XX controller.

Format
RobWriteSvoPara#File-number[rno:Robot-number], Axis-number, Array-variable

Arguments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File-number</strong></td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td><strong>Robot-number</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Axis-number</strong></td>
<td>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</td>
</tr>
<tr>
<td><strong>Array-variable</strong></td>
<td>A top elements of an array variable that contains servo parameter to write. Only long integer type (&amp;) are available for an array. The writing size of servo parameter varies by the kind of a controller type.</td>
</tr>
</tbody>
</table>

Example
1. Define an array that has sufficient area for servo parameter.
   Dim svo.para&(50)
   :   
   1. 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)
   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%
   1. 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**
  \[ \text{ScanStr(} \text{String, Format, Parameter}\#1 [, Parameter}\#2\ldots\text{)} \]

- **Arguments and Return value**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arguments</strong></td>
<td></td>
</tr>
<tr>
<td>String</td>
<td>A string expression to scan.</td>
</tr>
<tr>
<td>Format,</td>
<td>A string expression specifying format to scan.</td>
</tr>
<tr>
<td>Parameter</td>
<td>A variable to which the converted value is set according to a parameter descriptor.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>If the string matches the specified format, the function returns true value (-1). If not, it returns false value (0).</td>
</tr>
</tbody>
</table>

- **Example**
  \[ \text{ws$s$="year:2003 month:4" } \text{ 'String to scan} \]
  If ScanStr(ws$s, "year:%d month:%d", para1%, para2%) Then
    If para1% <= 2000 Then
      century% = 20
     Else
      century% = 21
     EndIf
  EndIf

- **Explanation**
  - Scanning \textit{String} is executed to compare with \textit{Format}. When a parameter descriptor is found in \textit{Format}, the value is extracted from \textit{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 \textit{String} matches \textit{Format}, the function returns true value (-1).
    Example)
    \[ \text{w$s$="para:xyz"} \]
    a% = ScanStr(w$s, "para:%d", p1%) \ 'a% is false(0).
  - Comparison between \textit{String} and \textit{Format} distinguishes uppercase and lowercase of a character.
    Example)
    \[ \text{w$s$ = "x:4 y:10"} \]
    a1% = ScanStr( w$s, "x:%d y:%d" p1%, p2%) \ 'a% is false(0).
    a2% = ScanStr( w$s, "X:%d Y:%d" p1%, p2%) \ 'a2% is true(-1).
  - If \textit{String} is longer than \textit{Format}, \textit{String} is scanned with the length of \textit{Format}.
    Example)
    \[ \text{w$s$="year:2003 month:4"} \]
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.

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Function</th>
<th>Example</th>
</tr>
</thead>
</table>
| %[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. | • “01234” scanned
Descriptor Converted
"%2d" 1
"%4d" 123
"%d" 1234
• “-01234” scanned
Descriptor Converted
"%2d" 0
"%4d" -12
"%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. | • “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 |
<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Function</th>
<th>Example</th>
</tr>
</thead>
</table>
| %[N]f | Scanned string is converted to set as a real 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. | "1234.567" scanned

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Converted</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;%2f&quot;</td>
<td>12.0</td>
</tr>
<tr>
<td>&quot;%5f&quot;</td>
<td>1234.5</td>
</tr>
<tr>
<td>&quot;%f&quot;</td>
<td>1234.567</td>
</tr>
</tbody>
</table>
| "-1234.567" scanned

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Converted</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;%2f&quot;</td>
<td>-1.0</td>
</tr>
<tr>
<td>&quot;%5f&quot;</td>
<td>-1234.0</td>
</tr>
<tr>
<td>&quot;%f&quot;</td>
<td>-1234.567</td>
</tr>
</tbody>
</table>
| "1.234" scanned

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Converted</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;%2f&quot;</td>
<td>1.0</td>
</tr>
<tr>
<td>&quot;%5f&quot;</td>
<td>1.23</td>
</tr>
<tr>
<td>&quot;%f&quot;</td>
<td>1.234</td>
</tr>
</tbody>
</table>
| "+1.234" scanned

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Converted</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;%2f&quot;</td>
<td>1.0</td>
</tr>
<tr>
<td>&quot;%5f&quot;</td>
<td>1.23</td>
</tr>
<tr>
<td>&quot;%f&quot;</td>
<td>1.234</td>
</tr>
<tr>
<td>&quot;%N%s&quot; or %[N]S</td>
<td>The string is scanned to set as a string. A corresponded parameter variable has to be string type. The inputted string begins at the character except space, tab or carriage-return, and terminates at the character before pace, tab or carriage-return. N specifies the number of input characters. If N omitted, the all scanned string is inputted. 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.</td>
</tr>
</tbody>
</table>

| "123456 AAAAAA" scanned

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Converted</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;%s%s&quot;</td>
<td>&quot;123456&quot; &quot;AAAAAA&quot;</td>
</tr>
<tr>
<td>&quot;%s%s&quot;</td>
<td>&quot;123&quot; &quot;456&quot;</td>
</tr>
</tbody>
</table>
| " A B" scanned

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Converted</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;%s%s&quot;</td>
<td>&quot;A&quot; &quot;B&quot;</td>
</tr>
<tr>
<td>&quot;%s%s&quot;</td>
<td>&quot; A&quot; &quot; B&quot;</td>
</tr>
<tr>
<td>&quot;%%s&quot; or %[N]s</td>
<td>%</td>
</tr>
</tbody>
</table>

- 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 Case Test-expression
  Case Expression-list#1
    Statements-block#1
  Case Expression-list#2
    Statements-block#2
  ...:
  Case Else
    Statements-block#N
  End Select
  ```

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test-expression</td>
<td>An expression to test for selection.</td>
</tr>
<tr>
<td>Expression-list</td>
<td>An expression of value that Test-expression may contains.</td>
</tr>
<tr>
<td>(1 to N-1)</td>
<td>Specifying format and example is shown below.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Number, Expression, String</strong></td>
</tr>
<tr>
<td></td>
<td>Example)</td>
</tr>
<tr>
<td></td>
<td>Case 1, 3, 5, 7</td>
</tr>
<tr>
<td></td>
<td>Case “ABC”, “DEF”</td>
</tr>
<tr>
<td></td>
<td>- <strong>Expression#1 To Expression#2</strong></td>
</tr>
<tr>
<td></td>
<td>Specify the range from Expression#1 to Expression#2 by a number, numeric</td>
</tr>
<tr>
<td></td>
<td>expression or string.</td>
</tr>
<tr>
<td></td>
<td>Example)</td>
</tr>
<tr>
<td></td>
<td>Case 1 To 5</td>
</tr>
<tr>
<td></td>
<td>Case “FMT” To “HIT”</td>
</tr>
<tr>
<td></td>
<td>- <strong>Is Relational-perator Expression</strong></td>
</tr>
<tr>
<td></td>
<td>Expression is a number, numeric expression or string.</td>
</tr>
<tr>
<td></td>
<td>Example)</td>
</tr>
<tr>
<td></td>
<td>Case Is = 5</td>
</tr>
<tr>
<td></td>
<td>Case Is &lt;&gt; 5</td>
</tr>
<tr>
<td></td>
<td>Case Is &lt; 5</td>
</tr>
<tr>
<td></td>
<td>Case Is &lt;= 5</td>
</tr>
<tr>
<td></td>
<td>Case Is &gt; 5</td>
</tr>
<tr>
<td></td>
<td>Case Is &gt;= 5</td>
</tr>
<tr>
<td></td>
<td>Expression-list can contain the combination of above formats with a</td>
</tr>
<tr>
<td></td>
<td>delimiter comma.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Statements-block(1 to N-1)</th>
<th>A block of statements that is executed when Expression-list becomes true.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statements-block(N)</td>
<td>A block of statements that is executed when all Expression-list (1 to N-1) becomes false.</td>
</tr>
</tbody>
</table>
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 nests 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.

Nest : means a structure located in the same structure.
Seq - SeqEnd

(Statement)

- **Function**
  Starts and terminates robot sequence mode.

- **Format**
  Seq
  
  ```
  File-number[[rno:Robot-number]]
  ```

  SeqEnd
  
  ```
  File-number [[rno:Robot-number]]
  ```

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File-number</td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td>Robot-number</td>
<td>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.</td>
</tr>
</tbody>
</table>

- **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.

![Diagram](image)

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

\[
\text{SET} = \text{#File-number}[\text{rno:Robot-number}], \text{Motion-parameter}
\]

#### Arguments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File-number</strong></td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td><strong>Robot-number</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Motion-parameter</strong></td>
<td>Sub-command to set a robot motion parameter. The following sub-commands are supported. See “Explanation” about each sub-command.</td>
</tr>
<tr>
<td></td>
<td>(1) Speed for PTP motion</td>
</tr>
<tr>
<td></td>
<td>(2) Speed for CPC motion</td>
</tr>
<tr>
<td></td>
<td>(3) Acceleration and deceleration for PTP motion</td>
</tr>
<tr>
<td></td>
<td>(4) Acceleration and deceleration for CPC motion</td>
</tr>
<tr>
<td></td>
<td>(5) Pull-up of Z axis</td>
</tr>
<tr>
<td></td>
<td>(6) Arch motion data</td>
</tr>
<tr>
<td></td>
<td>(7) Slow-up motion data</td>
</tr>
<tr>
<td></td>
<td>(8) Insert motion data</td>
</tr>
</tbody>
</table>

#### Example
Set #1[rno:2], Speed=100

#### Explanation

1. **Speed for PTP motion**
   - Format: \( \text{Format1} : \text{Speed} = \text{All-axes-value} \)
   - Format: \( \text{Format2} : \text{Speed} = \{\text{XYW-axes-value}, \text{Z-axes-value}\} \)
   - 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=\text{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.
Head speed = "Line Speed" * \( \frac{1+F\text{code}}{100} \) (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:\text{axes}\cdot\text{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= \text{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-Up=Value
- Range : 5.0 to Z area limit
- Precision : 0.001 mm
- Default : 10.0
Specify Z upper position of auto pull-up.

Refer to “13.1.1 MOTION” -> “PULL-UP” in operation manual of a robot controller.

(6) Arch motion data
- Format : Arch=(\text{Arch\text{-}up\text{-}value}, \text{Arch\text{-}down\text{-}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.

Distance-value is the distance of slow-up motion, where Z-axis moves by low speed after Z-axis starts.

Distance-value is the distance of slow-down motion. Z-axis decelerates from the position of the Z-axis destination minus Distance-value.
\textit{Speed-value} is Z-axis speed during slow-down motion.

- Refer to operation manual of a robot controller.

Refer to “13.1.1 MOTION” -> “INS DIS”, “INS SPEED” in 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job-name</td>
<td>String type expression of a job name.</td>
</tr>
<tr>
<td>Priority</td>
<td>Numeric expression of the new job priority.</td>
</tr>
<tr>
<td>Return value</td>
<td>The old job priority.</td>
</tr>
</tbody>
</table>

- 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**
  \[
  \text{SetRobNo}(\text{Robot-number})
  \]

- **Argument and Return value**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument Robot-number</td>
<td>Numeric expression specifying a robot number of a current job from 1 through 999.</td>
</tr>
<tr>
<td>Return value</td>
<td>Nothing</td>
</tr>
</tbody>
</table>

- **Example**
  \[
  \text{SetRobNo}(1)
  \]

- **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**
  Sgn(*Numeric-expression*)

- **Argument and Return value**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>A numeric expression specifying a number.</td>
</tr>
<tr>
<td>Return value</td>
<td>The function returns the following integer value to check the number.</td>
</tr>
<tr>
<td>Number &lt; 0</td>
<td>-1</td>
</tr>
<tr>
<td>Number = 0</td>
<td>0</td>
</tr>
<tr>
<td>Number &gt; 0</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Example**

b% = Sgn(-10.34)  ‘-1 is substituted for b%.'
Sin (Function)

- Function
  Gets the value of sine.

- Format
  Sin(Numeric-expression)

- Argument and Return value

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Numeric-expression Angle by radian.</td>
</tr>
<tr>
<td>Return value</td>
<td>Sine of the specified value is returned. The value is from -1.0 through +1.0.</td>
</tr>
</tbody>
</table>

- Example
  angle! = Pai / 3
  x! = Sin(angle!)

- Explanation
  The ratio of “B” to “A” is returned specifying the angle “Angle” in the figure.

\[
\sin(\text{Angle}) = \frac{B}{A}
\]

- See also Atn, Cos, Tan.
Space$ (Function)

- Function
  Gets a string containing serial space characters.

- Format
  Space$(Length)$

- Argument and Return value

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Length</td>
</tr>
<tr>
<td>Return value</td>
<td></td>
</tr>
</tbody>
</table>

- Example
  a$="Manual"
b$=a$+Space$(2)"ABC"  "ManualABC" is substituted for b$.

- Explanation
  A null string is returned if Length is zero.
Function
Gets the square root of a number.

Format
Sqr(Numeric-expression)

Argument and Return value

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Numeric expression specifying a number.</td>
</tr>
<tr>
<td>Return value</td>
<td>The square root of the specified value.</td>
</tr>
</tbody>
</table>

Example
a! = Sqr(x!*x! + y!*y!)
Str$(Function)

- Function
  Converts a number to a string.

- Format
  Str$(Numeric-expression)

- Argument and Return value

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Numeric-expression</td>
</tr>
<tr>
<td>Return value</td>
<td>A converted string.</td>
</tr>
</tbody>
</table>

- Example
  num1! = 10.2  : num2% = -12
  a$ = Str$(num1!)+"ABC"  ' "10.2ABC" is substituted for a$.
  b$ = Str$(num2%)+"ABC"  ' "-12ABC" is substituted for b$.

- 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**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>A numeric expression specifying the length of the returned string.</td>
</tr>
<tr>
<td>String-expression</td>
<td>A string expression in which the first character is repeated.</td>
</tr>
<tr>
<td>Character-code</td>
<td>A numeric expression specifying a repeating character code. Valid value is from 0 through 255 (&amp;HFF).</td>
</tr>
</tbody>
</table>

- **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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>\textit{Numeric-expression}</td>
</tr>
<tr>
<td>Return value</td>
<td>Tangent of the specified value is returned.</td>
</tr>
</tbody>
</table>

- **Example**

  \[
  \text{angle!} = \text{Pai} / 4 \\
  a! = \text{Tan}(\text{angle!})
  \]

- **Explanation**

  The ratio of “B” to “A” is returned specifying the angle “\textit{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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Nothing</td>
</tr>
</tbody>
</table>

- 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.

- See also Date$, Time$ (function).
Time$ (Function)

- Function
  Gets current time of the system calendar.

- Format
  Time$

- Argument and Return value

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Nothing</td>
</tr>
<tr>
<td>Return value</td>
<td>A string that contains current time with the following format. &quot;hh:mm:ss&quot;</td>
</tr>
</tbody>
</table>

- 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**
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return value</td>
<td>A logical value whether the last Wait statement has resulted in timeout.</td>
</tr>
<tr>
<td></td>
<td>Timeout: true (-1)</td>
</tr>
<tr>
<td></td>
<td>Not timeout: false (0)</td>
</tr>
</tbody>
</table>

- **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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>String-expression</td>
</tr>
<tr>
<td>Return value</td>
<td>Value to which the specified string is converted.</td>
</tr>
</tbody>
</table>

- Example
  
  a% = Val("123")  ' 123 is substituted for a%.
  b# = Val("&HFF")  ' 255.0 is substituted for b#.
  c% = Val("&H20F")  ' 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 Condition [, Time]

- Argument
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>A conditional expression resulting in the logical value, true (-1) or false (0).</td>
</tr>
<tr>
<td>Time</td>
<td>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.</td>
</tr>
</tbody>
</table>

- 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 #File-number, String

- **Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File-number</td>
<td>A file number corresponded to the communication port must be specified. Valid range is from 0 through 47. In case of variable, &quot;#&quot; can be omitted but in case of a numeral constant, it cannot be omitted.</td>
</tr>
<tr>
<td>String</td>
<td>A string of TEXT part in a HRCS protocol frame to send.</td>
</tr>
</tbody>
</table>

- **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.

  \[
  \begin{array}{c|c}
  \text{STX} & \text{TEXT} & \text{ETX} & \text{LRC} \\
  \end{array}
  \]

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

  ◆ See also RchkHrcs, WriteHrcs.

---

See “Appendix B LRC Calculation”.

---

9-164
**Xor**  (Operator)

- **Function**
  Executes a logical exclusion of two numbers.

- **Format**
  
  \[ \text{Numeric-expression#1} \text{Xor} \text{ Numeric-expression #2} \]

- **Arguments**
  
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric-expression#1</td>
<td>A numeric expression.</td>
</tr>
<tr>
<td>Numeric-expression#2</td>
<td>A numeric expression.</td>
</tr>
</tbody>
</table>

- **Example**
  
  \[ a\% = \&H00FF\% \]
  \[ b\% = \&H0F0F\% \]
  \[ c\% = a\% \text{Xor} b\% \quad \&H0FF0\% \text{substituted for } c\%. \]

- **Explanation**
  
  - The following calculation is performed.
    
    \[
    \begin{array}{ccc}
    X & Y & X \text{xor} Y \\
    1 & 1 & 0 \\
    1 & 0 & 1 \\
    0 & 1 & 1 \\
    0 & 0 & 0
    \end{array}
    \]
  
  - See “6.4.3 Logical Operator”.
# Appendix-A ASCII Codes

<table>
<thead>
<tr>
<th>Upper 4 bits</th>
<th>Hex</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Upper 4 bits</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Lower 4 bits</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>NUL</td>
<td>DEL</td>
<td>SP</td>
<td>0</td>
<td>@</td>
<td>P</td>
<td>`</td>
<td>p</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>DC1</td>
<td>!</td>
<td>1</td>
<td>A</td>
<td>Q</td>
<td>a</td>
<td>q</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>DC2</td>
<td>“</td>
<td>2</td>
<td>B</td>
<td>R</td>
<td>b</td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>DC3</td>
<td>#</td>
<td>3</td>
<td>C</td>
<td>S</td>
<td>c</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
<td>DC4</td>
<td>$</td>
<td>4</td>
<td>D</td>
<td>T</td>
<td>d</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>NAK</td>
<td>%</td>
<td>5</td>
<td>E</td>
<td>U</td>
<td>e</td>
<td>u</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>SYN</td>
<td>&amp;</td>
<td>6</td>
<td>F</td>
<td>V</td>
<td>f</td>
<td>v</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
<td>ETB</td>
<td>’</td>
<td>7</td>
<td>G</td>
<td>W</td>
<td>g</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>CAN</td>
<td>(</td>
<td>8</td>
<td>H</td>
<td>X</td>
<td>h</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>HT</td>
<td>EM</td>
<td>)</td>
<td>9</td>
<td>I</td>
<td>Y</td>
<td>i</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>LF</td>
<td>BUS</td>
<td>*</td>
<td>:</td>
<td>J</td>
<td>Z</td>
<td>j</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>VT</td>
<td>ESC</td>
<td>+</td>
<td>;</td>
<td>K</td>
<td>[</td>
<td>k</td>
<td>{</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>FF</td>
<td>FS</td>
<td>,</td>
<td>&lt;</td>
<td>L</td>
<td>¥</td>
<td>l</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>CR</td>
<td>GS</td>
<td>-</td>
<td>=</td>
<td>M</td>
<td>]</td>
<td>m</td>
<td>}</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>SO</td>
<td>RS</td>
<td>.</td>
<td>&gt;</td>
<td>N</td>
<td>^</td>
<td>n</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>SI</td>
<td>US</td>
<td>/</td>
<td>?</td>
<td>O</td>
<td>_</td>
<td>o</td>
<td>DEL</td>
<td></td>
</tr>
</tbody>
</table>

For example, the code of “A” is \&H42 as hexadecimal expression.
Appendix-B  LRC Calculation

How to calculate LRC (Longitudinal Redundancy Check) is described below.

(1) LRC calculation area
LRC is made by exclusive-or operation of all bytes from the next byte of STX through ETX.

<Example of a HRCS protocol frame>

<table>
<thead>
<tr>
<th>ASCII TEXT</th>
<th>Hex</th>
<th>Binary</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>31H</td>
<td>00110001</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>32H</td>
<td>00110010</td>
<td>X1</td>
</tr>
<tr>
<td>3</td>
<td>33H</td>
<td>00110011</td>
<td>X2</td>
</tr>
<tr>
<td>4</td>
<td>34H</td>
<td>00110100</td>
<td>X3</td>
</tr>
<tr>
<td>5</td>
<td>35H</td>
<td>00110101</td>
<td>X4</td>
</tr>
<tr>
<td>6</td>
<td>36H</td>
<td>00110110</td>
<td>X5</td>
</tr>
<tr>
<td>7</td>
<td>37H</td>
<td>00110111</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>38H</td>
<td>00111000</td>
<td>X6</td>
</tr>
<tr>
<td>ETX</td>
<td>03H</td>
<td>00000111</td>
<td>LRC</td>
</tr>
</tbody>
</table>

(2) LRC calculation
- A unit of calculation is each byte of communication data.
- Calculate exclusive-or (XOR) 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.

Truth table of XOR --- A XOR B = C

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
### Appendix-C Compiling Errors

<table>
<thead>
<tr>
<th>Code (Dec)</th>
<th>Code (Hex)</th>
<th>Error message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>Syntax error</td>
<td>Invalid character or keyword is used, or sequence of command or parameter is illegal.</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>Division by Zero</td>
<td>A number is divided by the constant zero.</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>Duplicate label</td>
<td>The same label is defined two times or more.</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>FOR without NEXT</td>
<td>For statement is described, but Next statement is not found.</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>Illegal function call</td>
<td>Number of arguments is illegal, or the type of argument is invalid.</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>Line buffer overflow</td>
<td>Divide a program described in one line.</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>NEXT without FOR</td>
<td>Next statement is described, but For statement is not found.</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>Out of memory</td>
<td>Variables, labels or subroutines are too much. Make a program more structured, for example, using subroutines.</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>IF formula too complex</td>
<td>A condition expression of If statement is too complex. For example, calculate the condition in advance.</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>Overflow</td>
<td>Value of the specified number is out of valid range.</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>RETURN without GOSUB</td>
<td>Return statement is not found in a subroutine.</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>String formula too complex</td>
<td>Divide a calculating expression.</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>String too long</td>
<td>Reduce the number of characters in a string.</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>Subscript out of range</td>
<td>The specified subscript of array exceeds the limit defined by declaration.</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>Type mismatch</td>
<td>Type combination of the left side and the right side are invalid.</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>Undefined label</td>
<td>The specified label is not defined in a job.</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>Undefined line number</td>
<td>(Unused)</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>Duplicate variable</td>
<td>Reserved memory is specified to Dim, Global, DimNet statement, or the same variable name is specified to Global and DimNet statement.</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>Variable name too long</td>
<td>Length of a variable name exceeds the maximum.</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>Label name too long</td>
<td>Length of a label name exceeds the maximum.</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>FOR statement missing</td>
<td>(Unused)</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>IF statement missing</td>
<td>Description of a If...Then...Else...EndIf sentence is invalid.</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>Misplace ELSE</td>
<td>An Else sentence is described independently.</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>GOTO statement missing label</td>
<td>Unexpected error occurs in a GoTo sentence.</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>Too much code defined in file</td>
<td>(Unused)</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>Too much code defined in line</td>
<td>Compiled intermediate code of one line exceeds the maximum volume. Divide a program of the line.</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>Duplicate Definition</td>
<td>The same name of array variable is defined two times or more.</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>Too much dimension number</td>
<td>The number of array dimensions has to be one through three.</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>Constant expression required</td>
<td>Array dimension has be specified by a constant.</td>
</tr>
<tr>
<td>Code (Dec)</td>
<td>Code (Hex)</td>
<td>Error message</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>---------------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>Undefined JOBNAME statement</td>
<td>Job name statement is not found in a job.</td>
</tr>
</tbody>
</table>
| 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”.
<p>| 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)                                                         |</p>
<table>
<thead>
<tr>
<th>Code (Dec)</th>
<th>Code (Hex)</th>
<th>Error message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>3D</td>
<td>DEFINE statement missing</td>
<td>The invalid description of an argument is found in a Define sentence.</td>
</tr>
<tr>
<td>62</td>
<td>3E</td>
<td>Duplicate JOB NAME</td>
<td>The same job name is used in the different job programs.</td>
</tr>
<tr>
<td>63</td>
<td>3F</td>
<td>Undefined JOB NAME</td>
<td>A job with the specified job name is not found in the linked program.</td>
</tr>
<tr>
<td>64</td>
<td>40</td>
<td>Bad object file</td>
<td>Invalid contents of an object file (.obj) are found on compilation or linking. There is a possibility that the file is destroyed.</td>
</tr>
<tr>
<td>65</td>
<td>41</td>
<td>Bad label file</td>
<td>(Unused)</td>
</tr>
<tr>
<td>66</td>
<td>42</td>
<td>Bad local variables file</td>
<td>Invalid contents of a local variable file (.var) are found on compilation or linking. There is a possibility that the file is destroyed.</td>
</tr>
<tr>
<td>67</td>
<td>43</td>
<td>Bad global variables file</td>
<td>Invalid contents of a global variable file (.gbl) are found on compilation or linking. There is a possibility that the file is destroyed.</td>
</tr>
<tr>
<td>68</td>
<td>44</td>
<td>More than 32 jobs defined</td>
<td>The number of linked jobs exceeds the maximum.</td>
</tr>
<tr>
<td>69</td>
<td>45</td>
<td>Too many global variables</td>
<td>The number of all global variables exceeds the maximum.</td>
</tr>
<tr>
<td>70</td>
<td>46</td>
<td>Bad make file</td>
<td>Invalid contents are found in a make file. There is a possibility that the file is destroyed.</td>
</tr>
<tr>
<td>71</td>
<td>47</td>
<td>ENDI F without IF</td>
<td>If...Endif combination is incorrect in a job program.</td>
</tr>
<tr>
<td>72</td>
<td>48</td>
<td>SELECT CASE nesting over</td>
<td>The number of Select Case nests exceeds the maximum. Extract the program in a Case block as a subroutine to be more structured.</td>
</tr>
<tr>
<td>73</td>
<td>49</td>
<td>SELECT CASE statement missing</td>
<td>Case, Case Else, End Select statement is described, but Select Case statement is not found.</td>
</tr>
<tr>
<td>74</td>
<td>4A</td>
<td>IS operator is not tail</td>
<td>Is operator in Case statement has to be described the tail of the sentence.</td>
</tr>
<tr>
<td>75</td>
<td>4B</td>
<td>CASE ELSE statement missing</td>
<td>Case Else statement has to be described even if there is no procedure for it.</td>
</tr>
<tr>
<td>76</td>
<td>4C</td>
<td>END SELECT statement missing</td>
<td>End Select statement is needed after a Case Else block to terminate a Select Case block.</td>
</tr>
<tr>
<td>77</td>
<td>4D</td>
<td>Sequence error in SELECT CASE block</td>
<td>Procedure in a Select Case block is too complex. It has to be simpler structure by means of structured programming using subroutines.</td>
</tr>
<tr>
<td>78</td>
<td>4E</td>
<td>CASE statements more than 127</td>
<td>Reduce the number of Case sentences to integrate several conditions to one Case sentence.</td>
</tr>
<tr>
<td>79</td>
<td>4F</td>
<td>Statement between SELECT CASE and CASE</td>
<td>There has not to be an executable sentence between Select Case and Case statement.</td>
</tr>
<tr>
<td>80</td>
<td>50</td>
<td>WAIT expression not logical</td>
<td>The condition expression in Wait sentence is not logical expression.</td>
</tr>
<tr>
<td>81</td>
<td>51</td>
<td>Illegal assignment</td>
<td>A value has not to be substituted for HERE, STATUS.</td>
</tr>
<tr>
<td>82</td>
<td>52</td>
<td>Illegal robot type</td>
<td>Invalid value is specified to RobType command.</td>
</tr>
<tr>
<td>83</td>
<td>53</td>
<td>Bad argument type of function or statement</td>
<td>The type of the specified constant or variable is not allowed for the function or statement.</td>
</tr>
<tr>
<td>84</td>
<td>54</td>
<td>Illegal value of argument</td>
<td>Value of the specified constant for the function or statement is out of valid range.</td>
</tr>
<tr>
<td>Code (Dec)</td>
<td>Code (Hex)</td>
<td>Error message</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>85</td>
<td>55</td>
<td>Conditions of CASE statement too long</td>
<td>Condition expression is too long in a Case sentence. Calculate the condition in advance, or divide the condition to multiple Case sentences.</td>
</tr>
<tr>
<td>86</td>
<td>56</td>
<td>Path name not character string type</td>
<td>The argument specifying a file path name is not string type.</td>
</tr>
<tr>
<td>87</td>
<td>57</td>
<td>Parameter not character string type</td>
<td>The argument specifying the parameter is not string type.</td>
</tr>
<tr>
<td>88</td>
<td>58</td>
<td>Illegal type of file number</td>
<td>Number type of the specified file number is not integer.</td>
</tr>
<tr>
<td>89</td>
<td>59</td>
<td>Invalid COM port number</td>
<td>The specified COM port number is out of valid range.</td>
</tr>
<tr>
<td>90</td>
<td>5A</td>
<td>Arm, Local, F, M, S cannot be specified</td>
<td>Arm direction, coordinate type, M-data, F-code, S-code cannot be described to a component expression of position data.</td>
</tr>
<tr>
<td>91</td>
<td>5B</td>
<td>One of Local, F, M, S cannot be omitted</td>
<td>All of coordinate type, M-data, F-code, S-code have to be described to a component expression of position data.</td>
</tr>
<tr>
<td>92</td>
<td>5C</td>
<td>Axis data, Arm cannot be omitted</td>
<td>Axis value or arm direction cannot be omitted in a component expression of position data.</td>
</tr>
<tr>
<td>93</td>
<td>5D</td>
<td>Variable that is not a position is specified</td>
<td>Only position data can be specified to an argument of the function or statement.</td>
</tr>
<tr>
<td>94</td>
<td>5E</td>
<td>Too many robots in the list</td>
<td>The number of robots specified to RobNoList command exceeds the maximum.</td>
</tr>
<tr>
<td>95</td>
<td>5F</td>
<td>Invalid robot number</td>
<td>The specified robot number is out of valid range.</td>
</tr>
<tr>
<td>96</td>
<td>60</td>
<td>Illegal type of robot number</td>
<td>The type of the specified robot number is not integer.</td>
</tr>
<tr>
<td>97</td>
<td>61</td>
<td>Robot number duplicated</td>
<td>A robot number specified in RobNoList command is duplicated.</td>
</tr>
<tr>
<td>98</td>
<td>62</td>
<td>Real number specified in integer constant</td>
<td>In a constant, the value is real type though type declaration is integer.</td>
</tr>
<tr>
<td>99</td>
<td>63</td>
<td>The right side is out of range for the left side</td>
<td>In substitution, the substituted value is out of valid range for the left side.</td>
</tr>
<tr>
<td>100</td>
<td>64</td>
<td>Only available string variable or constant</td>
<td>Only a string constant or variable can be specified to a argument of the function or statement.</td>
</tr>
<tr>
<td>101</td>
<td>65</td>
<td>Timer value is out of range</td>
<td>The value set to a timer is out of valid range.</td>
</tr>
<tr>
<td>102</td>
<td>66</td>
<td>Too many arguments of function</td>
<td>The number of arguments for the function exceeds the maximum.</td>
</tr>
<tr>
<td>103</td>
<td>67</td>
<td>Only string variable available</td>
<td>Only a string variable can be specified to a argument of the function or statement.</td>
</tr>
<tr>
<td>104</td>
<td>68</td>
<td>Cannot use the function without return value</td>
<td>The function that returns nothing cannot be described here.</td>
</tr>
<tr>
<td>105</td>
<td>69</td>
<td>Only user variable can be specified for argument of function</td>
<td>Only a variable can be specified to a output argument in the function. A constant cannot be specified.</td>
</tr>
<tr>
<td>106</td>
<td>6A</td>
<td>Too many axes for arguments</td>
<td>The number of specified axes for the function or statement exceeds the maximum.</td>
</tr>
<tr>
<td>107</td>
<td>6B</td>
<td>Bad usage of operator</td>
<td>The described operator cannot be used here.</td>
</tr>
<tr>
<td>108</td>
<td>6C</td>
<td>Nests of FOR-NEXT overflow</td>
<td>The number of For-Next nests exceeds the maximum. Extract the program in a For-Next block as a subroutine to be more structured.</td>
</tr>
<tr>
<td>Code (Dec)</td>
<td>Code (Hex)</td>
<td>Error message</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>--------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>109</td>
<td>6D</td>
<td>Nests of IF-ENDIF overflow</td>
<td>The number of If-EndIf nests exceeds the maximum. Extract the program in a If-EndIf block as a subroutine to be more structured.</td>
</tr>
<tr>
<td>110</td>
<td>6E</td>
<td>Nests of WHILE-WEND overflow</td>
<td>The number of While-Wend nests exceeds the maximum. Extract the program in a While-Wend block as a subroutine to be more structured.</td>
</tr>
<tr>
<td>111</td>
<td>6F</td>
<td>Variable in NEXT not matched to FOR</td>
<td>A variable in a Next sentence has be the same as the corresponded For sentence.</td>
</tr>
<tr>
<td>112</td>
<td>70</td>
<td>Invalid character code (including space or tab)</td>
<td>2-bytes code of a character is detected in a executable sentence.</td>
</tr>
<tr>
<td>113</td>
<td>71</td>
<td>Relative position missing</td>
<td>It is necessary to specify the relative position.</td>
</tr>
<tr>
<td>114</td>
<td>72</td>
<td>Only long integer variable available</td>
<td>Only 32-bits long integer (&amp;) can be specified. A constant cannot be specified.</td>
</tr>
<tr>
<td>115</td>
<td>73</td>
<td>Invalid axis number</td>
<td>The specified axis number is out of valid range.</td>
</tr>
</tbody>
</table>
# Appendix-D Running Job Errors

<table>
<thead>
<tr>
<th>Code (Dec)</th>
<th>Code (Hex)</th>
<th>Error message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01</td>
<td>Illegal program pointer</td>
<td>There may be OS (Operating System) trouble.</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>Bus error exception</td>
<td>STP: There may be OS trouble. WinSTP: Unused</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>Address error exception</td>
<td>STP: There may be OS trouble. WinSTP: Unused</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>Illegal instruction exception</td>
<td>STP: There may be OS trouble. WinSTP: Unused</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>Zero divide exception</td>
<td>STP: There may be OS trouble. WinSTP: Unused</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>CHK instruction exception</td>
<td>(Unused)</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>TRAPV instruction exception</td>
<td>(Unused)</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>Privilege violation</td>
<td>(Unused)</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>Format error</td>
<td>(Unused)</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>Line 1010 emulator exception</td>
<td>(Unused)</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>Line 1111 emulator exception</td>
<td>(Unused)</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>(Unused)</td>
<td>(Unused)</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>(Unused)</td>
<td>(Unused)</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>Arithmetic co-processor exception</td>
<td>(Unused)</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>Program not downloaded</td>
<td>Download a program to STP.</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>Calculation overflow</td>
<td>The result of calculation exceeds the maximum value.</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>Divided by Zero</td>
<td>Dividing a value by zero is executed.</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>RESUME without error</td>
<td>Resume statement is available only after a job error has occurred.</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>Written to out of area</td>
<td>Out of variable memory area is accessed when substitution. There is a possibility of OS trouble.</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>Invalid internal data</td>
<td>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.</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>Array accessed out of range</td>
<td>A subscript indicates out of the defined area of the array.</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>Nests of FOR-NEXT overflow</td>
<td>Extract the program in a For-Next block as a subroutine to be more structured.</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>FOR and NEXT not a pair</td>
<td>For or Next statement is not described.</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>Undefined command</td>
<td>The command (function or statement) is not supported at present.</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>RETURN without GOSUB</td>
<td>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.</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>Incorrect usage of command or function</td>
<td>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.</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>OPEN already executed</td>
<td>Open statement is executed for the file or communication port that is already opened.</td>
</tr>
<tr>
<td>Code (Dec)</td>
<td>Code (Hex)</td>
<td>Error message</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>File accessed without OPEN</td>
<td>The file or communication port that is not opened is accessed.</td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>Data nothing to receive</td>
<td>(Unused)</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>Calculation underflow</td>
<td>The result of calculation becomes less than the minimum value.</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>Receiving buffer overflow</td>
<td>(Unused)</td>
</tr>
<tr>
<td>32</td>
<td>20</td>
<td>Stack error at FOR-NEXT execution</td>
<td>(Unused)</td>
</tr>
<tr>
<td>33</td>
<td>21</td>
<td>Character string expression too complex</td>
<td>Divide a string expression to the smaller ones.</td>
</tr>
<tr>
<td>34</td>
<td>22</td>
<td>Character string too long</td>
<td>The string has to be treated as the smaller ones.</td>
</tr>
<tr>
<td>35</td>
<td>23</td>
<td>Type of variable or data mismatched</td>
<td>Invalid type of a variable or data is specified.</td>
</tr>
<tr>
<td>36</td>
<td>24</td>
<td>Incorrect format command</td>
<td>(Unused)</td>
</tr>
<tr>
<td>37</td>
<td>25</td>
<td>Cannot convert data type</td>
<td>Automatic type conversion cannot be executed. Check the combination of two types of data.</td>
</tr>
<tr>
<td>38</td>
<td>26</td>
<td>Arithmetic co-processor error</td>
<td>(Unused)</td>
</tr>
<tr>
<td>39</td>
<td>27</td>
<td>Data receiving (parity, overrun, framing) error</td>
<td>The error occurs on RS232C communication. There are possibilities of communication speed inconsistency, noise, or disconnection.</td>
</tr>
<tr>
<td>40</td>
<td>28</td>
<td>OPENed file already used</td>
<td>Open statement is executed with the file number that is already used.</td>
</tr>
<tr>
<td>41</td>
<td>29</td>
<td>Stack control error</td>
<td>OS cannot manage the inner stack correctly. There is a possibility of OS trouble.</td>
</tr>
<tr>
<td>42</td>
<td>2A</td>
<td>Nests of GOSUB-RETURN overflow</td>
<td>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.</td>
</tr>
<tr>
<td>43</td>
<td>2B</td>
<td>COM line not connected</td>
<td>On RS232C transmission, DSR signal becomes OFF. Check the disconnection.</td>
</tr>
<tr>
<td>44</td>
<td>2C</td>
<td>Sending buffer overflow</td>
<td>(Unused)</td>
</tr>
<tr>
<td>45</td>
<td>2D</td>
<td>JOB START without JOB OFF</td>
<td>The specified job has to be Job Off state before Job Start statement is executed.</td>
</tr>
<tr>
<td>46</td>
<td>2E</td>
<td>Position memory access out of range</td>
<td>(Unused)</td>
</tr>
<tr>
<td>47</td>
<td>2F</td>
<td>Network already opened</td>
<td>The already opened network is opened without close.</td>
</tr>
<tr>
<td>48</td>
<td>30</td>
<td>Network open overflow</td>
<td>The number of opened networks exceeds the maximum.</td>
</tr>
<tr>
<td>49</td>
<td>31</td>
<td>Network not opened</td>
<td>A network is accessed without open.</td>
</tr>
<tr>
<td>50</td>
<td>32</td>
<td>Network writing size error</td>
<td>Writing size specified to NetWrite function is out of valid range.</td>
</tr>
<tr>
<td>51</td>
<td>33</td>
<td>Network CR (Communication Reference) undefined</td>
<td>Network definition is not created correctly.</td>
</tr>
<tr>
<td>52</td>
<td>34</td>
<td>Own station number specified</td>
<td>Own station number is specified for the network communication.</td>
</tr>
<tr>
<td>53</td>
<td>35</td>
<td>Specified COM port not implemented.</td>
<td>The specified COM port is not available, or not implemented.</td>
</tr>
<tr>
<td>Code (Dec)</td>
<td>Code (Hex)</td>
<td>Error message</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>---------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>54</td>
<td>36</td>
<td>Communication buffer overflow</td>
<td>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.</td>
</tr>
<tr>
<td>55</td>
<td>37</td>
<td>Invalid baud rate for serial COM</td>
<td>The specified value of RS232C communication speed is invalid.</td>
</tr>
<tr>
<td>56</td>
<td>38</td>
<td>Invalid parity for serial COM</td>
<td>The specified value of RS232C parity is invalid.</td>
</tr>
<tr>
<td>57</td>
<td>39</td>
<td>Invalid character length for serial COM</td>
<td>The specified value of RS232C character length is invalid.</td>
</tr>
<tr>
<td>58</td>
<td>3A</td>
<td>Invalid stop bits for serial COM</td>
<td>The specified value of RS232C stop bits is invalid.</td>
</tr>
<tr>
<td>59</td>
<td>3B</td>
<td>No available file number</td>
<td>All file numbers are used now. Reduce the number of concurrently used files.</td>
</tr>
<tr>
<td>60</td>
<td>3C</td>
<td>Specified COM port not available</td>
<td>The specified COM port number is out of valid range.</td>
</tr>
<tr>
<td>61</td>
<td>3D</td>
<td>File number out of range</td>
<td>The specified file number is out of valid range.</td>
</tr>
<tr>
<td>62</td>
<td>3E</td>
<td>Can not open COM port</td>
<td>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.</td>
</tr>
<tr>
<td>63</td>
<td>3F</td>
<td>Serial COM parameter format error</td>
<td>The format of the specified RS232C communication parameter is invalid.</td>
</tr>
<tr>
<td>64</td>
<td>40</td>
<td>No such file or directory</td>
<td>The specified file or directory is not found.</td>
</tr>
<tr>
<td>65</td>
<td>41</td>
<td>File open error</td>
<td>The specified storage file cannot be opened.</td>
</tr>
<tr>
<td>66</td>
<td>42</td>
<td>File write error</td>
<td>The specified storage file cannot be written.</td>
</tr>
<tr>
<td>67</td>
<td>43</td>
<td>File read error</td>
<td>The specified storage file cannot be read.</td>
</tr>
<tr>
<td>68</td>
<td>44</td>
<td>Received HRCS data format error</td>
<td>The format of a received HRCS protocol frame is invalid.</td>
</tr>
<tr>
<td>69</td>
<td>45</td>
<td>Communication TxRDY off</td>
<td>RS232C TxRDY signal becomes OFF unexpectedly.</td>
</tr>
<tr>
<td>70</td>
<td>46</td>
<td>Communication RxRDY off</td>
<td>RS232C RxRDY signal becomes OFF unexpectedly.</td>
</tr>
<tr>
<td>71</td>
<td>47</td>
<td>System-call or API error</td>
<td>In WinSTP, a Windows API (System-call) error occurs.</td>
</tr>
<tr>
<td>72</td>
<td>48</td>
<td>Mode error in communication hardware</td>
<td>In WinSTP communication, a hardware mode error occurs.</td>
</tr>
<tr>
<td>73</td>
<td>49</td>
<td>General I/O error in communication hardware</td>
<td>In WinSTP communication, a general I/O error occurs.</td>
</tr>
<tr>
<td>74</td>
<td>4A</td>
<td>Break status detected in communication hardware</td>
<td>In WinSTP communication, the break signal is detected.</td>
</tr>
<tr>
<td>75</td>
<td>4B</td>
<td>Transmission timeout in communication hardware</td>
<td>In WinSTP communication, a transmission timeout occurs.</td>
</tr>
<tr>
<td>76</td>
<td>4C</td>
<td>Other error of communication</td>
<td>In communication, a miscellaneous error occurs.</td>
</tr>
<tr>
<td>77</td>
<td>4D</td>
<td>Re-open PROCON or HOST failed</td>
<td>When closing a communication port, re-setup of the hardware fails.</td>
</tr>
<tr>
<td>78</td>
<td>4E</td>
<td>Specified file cannot communicate</td>
<td>The specified file number is not for a communication port.</td>
</tr>
<tr>
<td>79</td>
<td>4F</td>
<td>Undefined error detected</td>
<td>Undefined error about communication. There is a possibility of OS trouble.</td>
</tr>
<tr>
<td>Code (Dec)</td>
<td>Code (Hex)</td>
<td>Error message</td>
<td>Explanation</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>80</td>
<td>50</td>
<td>Communication timeout with the robot</td>
<td>A robot does not respond.</td>
</tr>
<tr>
<td>81</td>
<td>51</td>
<td>Error received from the robot</td>
<td>A robot responds an error.</td>
</tr>
<tr>
<td>82</td>
<td>52</td>
<td>Illegal response format of the robot</td>
<td>The response format of the robot is illegal or not supported by STP.</td>
</tr>
<tr>
<td>83</td>
<td>53</td>
<td>Data Not registered in the robot (Unused)</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>54</td>
<td>Not SEQ mode in the robot (Unused)</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>55</td>
<td>Too many robots for communication</td>
<td>The number of robots that are concurrently communicated with STP exceeds the maximum.</td>
</tr>
<tr>
<td>86</td>
<td>56</td>
<td>Duplicated sending to robots</td>
<td>A next communication command is sent to a robot before the response.</td>
</tr>
<tr>
<td>87</td>
<td>57</td>
<td>Received position data invalid</td>
<td>Invalid position data is received from a robot.</td>
</tr>
<tr>
<td>88</td>
<td>58</td>
<td>Robot response without robot no.</td>
<td>A robot number is not found in the response from a robot that has to manage the robot number.</td>
</tr>
<tr>
<td>89</td>
<td>59</td>
<td>Different robot no. in response</td>
<td>The robot number received from a robot is not same as the sent one.</td>
</tr>
<tr>
<td>90</td>
<td>5A</td>
<td>Robot memory index out of range</td>
<td>The specified index (subscript) of robot memory is out of valid range.</td>
</tr>
<tr>
<td>91</td>
<td>5B</td>
<td>Robot is not ONLINE mode</td>
<td>Operation mode of the robot to communicate is not ONLINE.</td>
</tr>
<tr>
<td>92</td>
<td>5C</td>
<td>Robot motion stopped incompletely</td>
<td>A robot stopped before it completes the motion to the programmed position.</td>
</tr>
<tr>
<td>93</td>
<td>5D</td>
<td>Robot axis no. error</td>
<td>The specified axis number is out of valid range.</td>
</tr>
<tr>
<td>94</td>
<td>5E</td>
<td>Operands overflow</td>
<td>In robot function or statement, the number of specified arguments (operands) exceeds the maximum.</td>
</tr>
<tr>
<td>95</td>
<td>5F</td>
<td>Number of robot axes overflow</td>
<td>In robot function or statement, the number of specified axes exceeds the maximum.</td>
</tr>
<tr>
<td>96</td>
<td>60</td>
<td>OPEN in the through mode</td>
<td>A communication port in Through Mode is opened.</td>
</tr>
<tr>
<td>97</td>
<td>61</td>
<td>CLOSE in the through mode</td>
<td>A communication port in Through Mode is closed.</td>
</tr>
<tr>
<td>98</td>
<td>62</td>
<td>Sending data too long</td>
<td>The size of sending data at one time exceeds the maximum.</td>
</tr>
<tr>
<td>99</td>
<td>63</td>
<td>Minus value specified as a parameter</td>
<td>Minus value is specified to an argument of a robot function or statement.</td>
</tr>
<tr>
<td>100</td>
<td>64</td>
<td>Null string detected</td>
<td>Null string is specified to a function or statement.</td>
</tr>
<tr>
<td>101</td>
<td>65</td>
<td>Accessed out of string</td>
<td>A function or statement for string operation accesses the area out of the string.</td>
</tr>
<tr>
<td>102</td>
<td>66</td>
<td>Invalid code specified</td>
<td>A character code out of 0 to 255 is specified to a function or statement for string operation.</td>
</tr>
<tr>
<td>103</td>
<td>67</td>
<td>Cannot execute for current robot type</td>
<td>For the controller specified by RobType, the function or statement cannot be executed.</td>
</tr>
<tr>
<td>104</td>
<td>68</td>
<td>Not available to write time or date</td>
<td>In WinSTP, system date and time cannot be written.</td>
</tr>
<tr>
<td>Code (Dec)</td>
<td>Code (Hex)</td>
<td>Error message</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>105</td>
<td>69</td>
<td>Format error of time or date</td>
<td>The format to write system date or time is invalid.</td>
</tr>
<tr>
<td>106</td>
<td>6A</td>
<td>Bad robot no. in response from robot</td>
<td>The format of the robot number received from a robot is invalid.</td>
</tr>
<tr>
<td>107</td>
<td>6B</td>
<td>Robot number out of range</td>
<td>The specified robot number is out of valid range.</td>
</tr>
<tr>
<td>108</td>
<td>6C</td>
<td>Robot number duplicated</td>
<td>The same robot number is registered two times and more.</td>
</tr>
<tr>
<td>109</td>
<td>6D</td>
<td>Robot number not found in OPEN list</td>
<td>The specified robot number is not defined by RobNoList of Open statement.</td>
</tr>
<tr>
<td>110</td>
<td>6E</td>
<td>Specified job not downloaded</td>
<td>The specified job has not been downloaded.</td>
</tr>
<tr>
<td>111</td>
<td>6F</td>
<td>LRC error of HRCS protocol</td>
<td>LRC error is detected on HRCS protocol communication.</td>
</tr>
<tr>
<td>112</td>
<td>70</td>
<td>Too many arguments</td>
<td>The number of the specified arguments exceeds the maximum.</td>
</tr>
<tr>
<td>113</td>
<td>71</td>
<td>Invalid parameter descriptor</td>
<td>The format of a parameter descriptor is invalid.</td>
</tr>
<tr>
<td>114</td>
<td>72</td>
<td>Invalid type of argument</td>
<td>Type of the specified argument of a function or statement is invalid.</td>
</tr>
<tr>
<td>115</td>
<td>73</td>
<td>Parameter value out of range</td>
<td>The value of parameter specified to a function or statement is out of valid range.</td>
</tr>
<tr>
<td>116</td>
<td>74</td>
<td>Cannot execute when robot is moving</td>
<td>The function or statement cannot be executed when a robot is moving.</td>
</tr>
<tr>
<td>117</td>
<td>75</td>
<td>Invalid position data</td>
<td>The content of position data is invalid.</td>
</tr>
<tr>
<td>118</td>
<td>76</td>
<td>Position data empty</td>
<td>All elements of position data are zero.</td>
</tr>
<tr>
<td>119</td>
<td>77</td>
<td>M,F,S data all zero</td>
<td>All M-data, S-code and F-code of position data are zero.</td>
</tr>
<tr>
<td>120</td>
<td>78</td>
<td>Robot controlled by other job</td>
<td>(Unused)</td>
</tr>
<tr>
<td>121</td>
<td>79</td>
<td>Number of FOR-NEXT overflow</td>
<td>The number of all For-Next statements exceeds the maximum.</td>
</tr>
<tr>
<td>122</td>
<td>7A</td>
<td>MOVE executed without ENABLE</td>
<td>Enable statement has to be executed before Move statement.</td>
</tr>
<tr>
<td>123</td>
<td>7B</td>
<td>Invalid job priority</td>
<td>The value of the specified job priority is out of valid range.</td>
</tr>
<tr>
<td>124</td>
<td>7C</td>
<td>Specified job not found</td>
<td>The specified job is not found in the system.</td>
</tr>
<tr>
<td>125</td>
<td>7D</td>
<td>Cannot execute on the current platform</td>
<td>In the current platform (type of a controller or STP), the function or statement is not available.</td>
</tr>
<tr>
<td>126</td>
<td>7E</td>
<td>Invalid data of robot collision check</td>
<td>The downloaded data of robot collision check is invalid.</td>
</tr>
<tr>
<td>127</td>
<td>7F</td>
<td>Robot collision check data overflow</td>
<td>Data overflow occurs on the setup of robot collision check when Open or Close statement is executed.</td>
</tr>
<tr>
<td>128</td>
<td>80</td>
<td>Safety length for robot collision not defined</td>
<td>In the downloaded data of robot collision check, safety length is not defined.</td>
</tr>
<tr>
<td>129</td>
<td>81</td>
<td>Internal function error of robot collision</td>
<td>An unexpected error occurs in the internal function of robot collision check.</td>
</tr>
<tr>
<td>130</td>
<td>82</td>
<td>Robot collision detected</td>
<td>Robot collision is detected by the internal function of robot collision check.</td>
</tr>
<tr>
<td>131</td>
<td>83</td>
<td>Local-World coordinates conversion data not defined</td>
<td>In the downloaded data of robot collision check, Local-World coordinates conversion data is not defined.</td>
</tr>
<tr>
<td>Code (Dec)</td>
<td>Code (Hex)</td>
<td>Error message</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>--------------------------------------</td>
<td>--------------------------------------------------------------</td>
</tr>
<tr>
<td>132</td>
<td>84</td>
<td>Invalid specified robot number</td>
<td>The specified robot number is not defined in a robot controller.</td>
</tr>
<tr>
<td>133</td>
<td>85</td>
<td>SG/SP data group error</td>
<td>The specified group name of SG/SP data is incorrect.</td>
</tr>
<tr>
<td>134</td>
<td>86</td>
<td>SG/SP data invalid</td>
<td>The specified value of SG/SP data is invalid.</td>
</tr>
</tbody>
</table>
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
  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
  ' Created: 2004.1.15
  ' (R)(C) All rights reserved by HIRATA Corporation.
  ' Revisions:
  ' '04.08.24 XXXX   YYYYYYY 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 #NO.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
    GoTo *MAIN.LOOP
  End Select

• 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
Appendix

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.
<table>
<thead>
<tr>
<th>Definition type</th>
<th>Header file name</th>
<th>Constant name</th>
</tr>
</thead>
</table>
| I/O number      | Io.hed          | Input: I.XX...XX  
|                 |                 | 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)