Upon receipt of the product and prior to initial operation, read these instructions thoroughly, and retain for future reference.

MOTOMAN INSTRUCTIONS

MOTOMAN-□□□ INSTRUCTIONS
YRC1000 INSTRUCTIONS
YRC1000 OPERATOR’S MANUAL (GENERAL) (SUBJECT SPECIFIC)
YRC1000 MAINTENANCE MANUAL
YRC1000 ALARM CODES (MAJOR ALARMS) (MINOR ALARMS)

Please have the following information available when contacting Yaskawa Customer Support:
• System
• Primary Application
• Software Version (Located on Programming Pendant by selecting: {Main Menu} - {System Info} - {Version})
Robot Serial Number (Located on robot data plate)
Robot Sales Order Number (Located on controller data plate)

Part Number: 178939-1CD
Revision: 1
DANGER

• This manual explains MotoPlus of the YRC1000 system. Read this manual carefully and be sure to understand its contents before handling the YRC1000. Any matter, including operation, usage, measures, and an item to use, not described in this manual must be regarded as "prohibited" or "improper".

• General information related to safety are described in "Chapter 1. Safety" of the YRC1000 INSTRUCTIONS. To ensure correct and safe operation, carefully read "Chapter 1. Safety" of the YRC1000 INSTRUCTIONS.

CAUTION

• In some drawings in this manual, protective covers or shields are removed to show details. Make sure that all the covers or shields are installed in place before operating this product.

• YASKAWA is not responsible for incidents arising from unauthorized modification of its products. Unauthorized modification voids the product warranty.

NOTICE

• The drawings and photos in this manual are representative examples and differences may exist between them and the delivered product.

• YASKAWA may modify this model without notice when necessary due to product improvements, modifications, or changes in specifications. If such modification is made, the manual number will also be revised.

• If your copy of the manual is damaged or lost, contact a YASKAWA representative to order a new copy. The representatives are listed on the back cover. Be sure to tell the representative the manual number listed on the front cover.
NOTES FOR SAFE OPERATION

Read this manual carefully before installation, operation, maintenance, or inspection of the YRC1000.

In this manual, the Notes for Safe Operation are classified as “DANGER”, “WARNING”, “CAUTION”, or “NOTICE”.

**DANGER**
Indicates an imminently hazardous situation which, if not avoided, will result in death or serious injury. Safety Signs identified by the signal word DANGER should be used sparingly and only for those situations presenting the most serious hazards.

**WARNING**
Indicates a potentially hazardous situation which, if not avoided, will result in death or serious injury. Hazards identified by the signal word WARNING present a lesser degree of risk of injury or death than those identified by the signal word DANGER.

**CAUTION**
Indicates a hazardous situation, which if not avoided, could result in minor or moderate injury. It may also be used without the safety alert symbol as an alternative to “NOTICE”.

**NOTICE**
NOTICE is the preferred signal word to address practices not related to personal injury. The safety alert symbol should not be used with this signal word. As an alternative to “NOTICE”, the word “CAUTION” without the safety alert symbol may be used to indicate a message not related to personal injury.

Even items described as “CAUTION” may result in a serious accident in some situations. At any rate, be sure to follow these important items.

**NOTE**
To ensure safe and efficient operation at all times, be sure to follow all instructions, even if not designated as “DANGER”, “WARNING” and “CAUTION”.
Before operating the manipulator, make sure the servo power is turned OFF by performing the following operations. When the servo power is turned OFF, the SERVO ON LED on the programming pendant is turned OFF.

- Press the emergency stop buttons on the front door of the YRC1000, on the programming pendant, on the external control device, etc.
- Disconnect the safety plug of the safety fence. (when in the play mode or in the remote mode)

If operation of the manipulator cannot be stopped in an emergency, personal injury and/or equipment damage may result.

Fig. : Emergency Stop Button

Before releasing the emergency stop, make sure to remove the obstacle or error caused the emergency stop, if any, and then turn the servo power ON.

Failure to observe this instruction may cause unintended movement of the manipulator, which may result in personal injury.

Fig. : Release of Emergency Stop

Observe the following precautions when performing a teaching operation within the manipulator's operating range:

- Be sure to perform lockout by putting a lockout device on the safety fence when going into the area enclosed by the safety fence. In addition, the operator of the teaching operation must display the sign that the operation is being performed so that no other person closes the safety fence.
- View the manipulator from the front whenever possible.
- Always follow the predetermined operating procedure.
- Always keep in mind emergency response measures against the manipulator's unexpected movement toward a person.
- Ensure a safe place to retreat in case of emergency.

Failure to observe this instruction may cause improper or unintended movement of the manipulator, which may result in personal injury.

- Confirm that no person is present in the manipulator's operating range and that the operator is in a safe location before:
  - Turning ON the YRC1000 power
  - Moving the manipulator by using the programming pendant
  - Running the system in the check mode
  - Performing automatic operations

Personal injury may result if a person enters the manipulator's operating range during operation. Immediately press an emergency stop button whenever there is a problem. The emergency stop buttons are located on the front panel of the YRC1000 and on the right of the programming pendant.

- Read and understand the Explanation of the Warning Labels before operating the manipulator.
Definition of Terms Used Often in This Manual

The MOTOMAN is the YASKAWA industrial robot product.

The MOTOMAN usually consists of the manipulator, the controller, the programming pendant, and supply cables.

In this manual, the equipment is designated as follows.

<table>
<thead>
<tr>
<th>Equipment</th>
<th>Manual Designation</th>
</tr>
</thead>
<tbody>
<tr>
<td>YRC1000 controller</td>
<td>YRC1000</td>
</tr>
<tr>
<td>YRC1000 programming pendant</td>
<td>Programming pendant</td>
</tr>
<tr>
<td>Cable between the manipulator and</td>
<td>Manipulator cable</td>
</tr>
<tr>
<td>the controller</td>
<td></td>
</tr>
</tbody>
</table>
Descriptions of the programming pendant keys, buttons, and displays are shown as follows:

<table>
<thead>
<tr>
<th>Equipment</th>
<th>Manual Designation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming Pendant</td>
<td></td>
</tr>
<tr>
<td>Character Keys</td>
<td>The keys which have characters or its symbol printed on them are denoted with [ ].</td>
</tr>
<tr>
<td>Symbol Keys</td>
<td>ex. [ENTER]</td>
</tr>
<tr>
<td>Axis Keys</td>
<td>[Axis Key] and [Numeric Key] are generic names for the keys for axis operation and</td>
</tr>
<tr>
<td>Numeric Keys</td>
<td>number input.</td>
</tr>
<tr>
<td>Keys pressed simultaneously</td>
<td>When two keys are to be pressed simultaneously, the keys are shown with a</td>
</tr>
<tr>
<td></td>
<td>“+” sign between them, ex. [SHIFT]+[COORD]</td>
</tr>
<tr>
<td>Displays</td>
<td>The menu displayed in the programming pendant is denoted with { }.</td>
</tr>
<tr>
<td></td>
<td>ex. {JOB}</td>
</tr>
</tbody>
</table>

**Description of the Operation Procedure**

In the explanation of the operation procedure, the expression "Select • • •" means that the cursor is moved to the object item and [SELECT] is pressed, or that the item is directly selected by touching the screen.

**Registered Trademark**

In this manual, names of companies, corporations, or products are trademarks, registered trademarks, or brand names for each company or corporation. The indications of (R) and ™ are omitted.
# Contents

1 Outline .................................................................................................................... ........................ 1-1
   1.1 Preface .................................................................................................................. 1-1
   1.2 Function Outline ................................................................................................. 1-1
   1.3 MotoPlus Specification ....................................................................................... 1-3
   1.4 Specific Data Type for MotoPlus .......................................................................... 1-4
   1.5 Notes ....................................................................................................................... 1-5

2 Task ................................................................................................................................................ 2-1
   2.1 Multitasking ........................................................................................................... 2-1
   2.2 State Transitions of Task .................................................................................... 2-1
   2.3 Task Scheduling Method ..................................................................................... 2-2
   2.4 Management of Ready-Queue ............................................................................. 2-2
   2.5 Application Task Control ................................................................................... 2-3
   2.6 Processing Time Measurement .......................................................................... 2-3
   2.7 Limit on Protection Regarding Task Control ................................................ 2-4
      2.7.1 Limit on Number of Files Which User Can Load .................................. 2-4
      2.7.2 Limit on Memory Size ................................................................................. 2-4
      2.7.3 Limit on Task Start-up ................................................................................. 2-5
      2.7.4 Function to Check Execution Status of Man-Machine Task ................ 2-5
   2.8 List of Task Control API ..................................................................................... 2-6

3 User Watchdog ....................................................................................................................... 3-1
   3.1 Function Outline ................................................................................................. 3-1
   3.2 Usage Outline ..................................................................................................... 3-1
   3.3 User Watchdog API List .................................................................................... 3-2

4 Ethernet Communications ............................................................................................. 4-1
   4.1 Socket .................................................................................................................. 4-1
   4.2 Client Server Model ........................................................................................... 4-1
   4.3 UDP Communication .......................................................................................... 4-2
   4.4 TCP Communication .......................................................................................... 4-2
   4.5 List of Socket Library Functions ........................................................................ 4-3
   4.6 Operation of Structure and IP Address .......................................................... 4-3
      4.6.1 Sockaddr Structure .................................................................................... 4-3
      4.6.2 Sockaddr_in Structure .............................................................................. 4-4
Contents

4.6.3   Function for IP Address Operation ................................................................. 4-4
4.7   About Byte Order ............................................................................................ 4-5
4.8   Flow of Socket Library System Call ................................................................. 4-5
  4.8.1 Flow of Socket Library System Call with UDP .......................................... 4-5
  4.8.2 Flow of Socket Library System Call with TCP .......................................... 4-6
4.9   Note for Ethernet Programming ................................................................. 4-6

5   Data Transmission with Programming Pendant ........................................... 5-1

6   RS232C Communication Control ................................................................. 6-1

7   Event Control ................................................................................................... 7-1
  7.1   Task Creation .......................................................................................... 7-1
  7.2   Event Notification ................................................................................. 7-1

8   Robot System Control .................................................................................. 8-1
  8.1   System Information .............................................................................. 8-1
  8.2   Variables ................................................................................................ 8-1
  8.3   I/O ........................................................................................................... 8-1
  8.4   Robot Operation .................................................................................... 8-1
  8.5   System Monitor API List ...................................................................... 8-2
  8.6   System Control API List ....................................................................... 8-3

9   Motion Control Service ................................................................................... 9-1
  9.1   Motion Control Service ........................................................................... 9-1
    9.1.1   Outline ......................................................................................... 9-1
    9.1.2   API List ....................................................................................... 9-3
  9.2   Step Motion Service .............................................................................. 9-4
  9.3   JOB Link Operation Control Service ...................................................... 9-5
    9.3.1   Increment Value Move Function ...................................................... 9-5
    9.3.2   Robot Operation Control API List .................................................... 9-5

10  Sensor Control .................................................................................................. 10-1
  10.1   Outline of Provided Functions ................................................................. 10-1
    10.1.1   Path Correction Function ............................................................... 10-1
    10.1.2   Interpolation-Motion Forced Change Function .................................. 10-2
    10.1.3   Speed Override Function ................................................................. 10-2
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1.4 Increment Value Move Function</td>
<td>10-3</td>
</tr>
<tr>
<td>10.2 Outline of Operation</td>
<td>10-4</td>
</tr>
<tr>
<td>10.3 Timing of Path Correction Process</td>
<td>10-5</td>
</tr>
<tr>
<td>10.4 Details of Giving and Receiving Command and Changing Operating Conditions by SKILLSND (Including Path Correction)</td>
<td>10-5</td>
</tr>
<tr>
<td>10.5 Sensor Control API List</td>
<td>10-10</td>
</tr>
<tr>
<td>11 Robot Operation Control Service</td>
<td>11-1</td>
</tr>
<tr>
<td>11.1 Increment Value Move Function</td>
<td>11-1</td>
</tr>
<tr>
<td>11.2 Robot Operation Control API List</td>
<td>11-1</td>
</tr>
<tr>
<td>12 Servo Control</td>
<td>12-1</td>
</tr>
<tr>
<td>12.1 Function Outline</td>
<td>12-1</td>
</tr>
<tr>
<td>12.1.1 Feedback Speed/Torque Retrieve Function</td>
<td>12-2</td>
</tr>
<tr>
<td>12.1.2 Torque Limit Function/Torque Control Function</td>
<td>12-2</td>
</tr>
<tr>
<td>12.2 Example of Usage</td>
<td>12-3</td>
</tr>
<tr>
<td>12.2.1 Detecting Workpiece Position</td>
<td>12-3</td>
</tr>
<tr>
<td>12.2.2 Holding Workpiece with Unknown Size</td>
<td>12-4</td>
</tr>
<tr>
<td>12.3 Servo Control API List</td>
<td>12-4</td>
</tr>
<tr>
<td>13 Memory Pool Control</td>
<td>13-1</td>
</tr>
<tr>
<td>13.1 Memory Pool Control API List</td>
<td>13-1</td>
</tr>
<tr>
<td>14 File Control</td>
<td>14-1</td>
</tr>
<tr>
<td>14.1 Rules for File Name and Size</td>
<td>14-1</td>
</tr>
<tr>
<td>14.2 Rules for USB Connector and USB Memory Stick</td>
<td>14-3</td>
</tr>
<tr>
<td>14.3 Access Right When Opening a File</td>
<td>14-4</td>
</tr>
<tr>
<td>14.4 Available Task Priority</td>
<td>14-4</td>
</tr>
<tr>
<td>14.5 File Control API List</td>
<td>14-5</td>
</tr>
<tr>
<td>15 Existing File Control</td>
<td>15-1</td>
</tr>
<tr>
<td>15.1 Existing File Access API List</td>
<td>15-1</td>
</tr>
<tr>
<td>16 Programming</td>
<td>16-1</td>
</tr>
<tr>
<td>16.1 Rules</td>
<td>16-1</td>
</tr>
<tr>
<td>16.2 C Coding Rules</td>
<td>16-1</td>
</tr>
<tr>
<td>16.3 Program Execution</td>
<td>16-1</td>
</tr>
</tbody>
</table>
16.4 MotoPlus Coding Rules ................................................................. 16-1
16.5 Restrictions on Programming ................................................................. 16-2
  16.5.1 Usable ANSI-C Standard Functions .......................................... 16-2
  16.5.2 Task Priority and Usable API .................................................. 16-3
16.6 Note for Programming ..................................................................... 16-3
16.7 Operation Control Sample Program ........................................... 16-4
  16.7.1 Operation Control Sample Program ........................................... 16-4
    16.7.1.1 Outline ........................................................................ 16-4
    16.7.1.2 Sample Program .......................................................... 16-4
  16.7.2 Independent Control Sample Program ...................................... 16-12
    16.7.2.1 Outline ....................................................................... 16-12
    16.7.2.2 Sample Program .......................................................... 16-13
  16.7.3 Coordinated Control Sample Program ....................................... 16-20
    16.7.3.1 Outline ....................................................................... 16-20
    16.7.3.2 Sample Program .......................................................... 16-21
16.8 Phenomena during Debugging, Potential Coding Problems, and Examination Methods ........................................... 16-27
  16.8.1 Programming Pendant Display Freezes with “Starting up system Start Online Process”, and YRC1000 Does Not Start ......................................................... 16-27
  16.8.2 MotoPlus Application Is Seemingly Not Running ...................... 16-27
  16.8.3 Alarm 4479: “MOTOPLUS MM TASK WATCHDOG ERROR” Occurs ................................................................. 16-28
  16.8.4 Programming Pendant Display or Key Response Is Slow .......... 16-28
  16.8.5 Programming Pendant Communication Error Occurs and Servo Power of YRC1000 Turns Off (Hang-up Status) ......................................................... 16-28
17 Sample Programs ................................................................................. 17-1
  17.1 Outline of Sample Programs .......................................................... 17-1
  17.2 Display of Hello World .................................................................. 17-2
  17.3 Task Control Sample Program ....................................................... 17-3
  17.4 User Watchdog Sample Program .................................................... 17-6
  17.5 Ethernet Sample Programs .............................................................. 17-11
    17.5.1 Preparations ...................................................................... 17-11
    17.5.2 UDP Server Sample Programs (YRC1000 Side) ..................... 17-13
    17.5.3 UDP Client Sample Program (Windows PC Side) ................. 17-17
    17.5.4 TCP Server Sample Programs (YRC1000 Side) ..................... 17-20
    17.5.5 TCP Client Sample Program (Windows PC Side) ................. 17-25
  17.6 Robot System Monitor Samples and Control Programs ................ 17-28
    17.6.1 Variables, I/O Access Sample Program .................................. 17-28
Contents

17.6.2 User Variable Access Sample Program ......................................................... 17-43
17.7 Serial Communication (RS232C)................................................................. 17-48
  17.7.1 Preparations .......................................................................................... 17-48
  17.7.2 Operation Check ................................................................................... 17-48
  17.7.3 Sample Code ......................................................................................... 17-49
17.8 Sensor Control Sample Program................................................................. 17-52
17.9 Servo Control Sample Program................................................................. 17-75
17.10 File Control Sample Program ................................................................. 17-83
  17.10.1 Operation Check ................................................................................ 17-83
  17.10.2 Sample Program ............................................................................... 17-84
17.11 Existing File Access Sample Program....................................................... 17-89
  17.11.1 Preparation ....................................................................................... 17-89
  17.11.2 Operation Check ............................................................................... 17-89
  17.11.3 Sample Program ............................................................................... 17-89
17.12 Coordinate Conversion Sample Program ................................................. 17-93

18 Alarm List ........................................................................................................ 18-1
1 Outline

1.1 Preface

This manual explains the descriptions using the functions provided by MotoPlus (Motoman Professional Programming Language for Superior Use).

1.2 Function Outline

Since MotoPlus provides the following services through APIs, it realizes the desired functions using these APIs in the user's C language program.

1. Application task control
   Application task start/stop, data transmission between application programs by the mailbox and semaphore, exclusive control and execution synchronization

2. Robot control
   Robot control from the application program

3. JOB control
   Start-stop control of the JOB by the application, data transmission between Jobs by variables (byte, integer, double-precision, floating-point, and position variable), and execution synchronization

4. CIO control
   I/O between the application program and the CIO, and read/write of the register

5. Ethernet communication control
   The Ethernet (TCP/IP) communication from the application, especially using the Winsock function library included as an API

6. Programming pendant communication
   Data transmission between the application and the programming pendant application

7. EVENT
   Event notification to the application at every I/O control cycle and interpolation cycle of the system

8. RS232C serial communication control
   RS232C serial communication from the application

9. Sensor control
   The APIs which transfer data to and from a job and change operating conditions such as path correction and speed change, and the instructions which transfer data between a job and MotoPlus application

10. Memory management
    The protected memory management
    Specific instructions, malloc and mfree, support the data area which the system manages for MotoPlus.

11. General-purpose file control
    The function to access multiple general-purpose files by using the fixed area on CMOS as a drive

12. Existing file control
    The function to access existing files (jobs and condition files, etc. which can be loaded and saved by external memory)
1. Outline

1.2 Function Outline

13. Servo control
   Servo control from the application program

14. User watchdog
   Watchdog to monitor whether the application operates normally

15. Coordinate conversion
   Calculation of the manipulator order or inverse kinematics, coordinate conversion, pulse conversion from the feedback pulse to the arithmetic pulse, or linear algebra calculation.
### 1.3 MotoPlus Specification

<table>
<thead>
<tr>
<th>No.</th>
<th>Item</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Number of application files which can be started</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Maximum available memory size (code area + data area (static variable))</td>
<td>2 Mbyte</td>
</tr>
<tr>
<td>3</td>
<td>Task priority</td>
<td>4 level</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For the notification task of the I/O control cycle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For the notification task of the interpolation cycle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• High priority</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Normal priority</td>
</tr>
<tr>
<td>4</td>
<td>Number of tasks which can be started (specified for each priority)</td>
<td>• For the notification task of the I/O control cycle: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For the notification task of the interpolation cycle: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• High priority: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Normal priority: 3</td>
</tr>
<tr>
<td>5</td>
<td>Stack size of each task</td>
<td>40 Kbyte</td>
</tr>
</tbody>
</table>
### 1.4 Specific Data Type for MotoPlus

Regarding the variable definitions in MotoPlus, the following types are used:

- INT8, INT16, INT32, INT64, CHAR, SHORT, LONG, FLOAT, DOUBLE, UCHAR, USHORT, UINT, ULONG, UDLONG, BOOL, STATUS, VOID

These are defined in “MotoPlus.h” as follows:

```c
/* motoPlus scalar types */
typedef char INT8;
typedef short INT16;
typedef int INT32;
typedef long long INT64;
typedef unsigned char UCHAR;
typedef unsigned short USHORT;
typedef unsigned int UINT;
typedef unsigned long ULONG;
typedef unsigned long long UDLONG;
typedef int BOOL;
typedef int STATUS;
typedef void VOID;
```

```
For MotoPlus application programming, note the following points:

- **C language memory operation**
  When programming in C language, the pointer variable which stores the memory address can be used. If the pointer variable is set incorrectly, the system memory area may be rewritten.
  If the system memory area is rewritten, critical problems occur, e.g., the software hangs up (the hang-up status is detected by a CPU exception or the watchdog check function, then the servo power turns OFF and the system shuts down), the robot stops its operation due to an alarm, or the programming pendant becomes inoperable.

- **Task priority and processing time**
  1. Use precautions when you select a task with high priority (the task started with the priority MP_PRI_IO_CLK_TAKE, MP_PRI_IP_CLK_TAKE, or MP_PRI_TIME_CRITICAL). If unusually long processing time is used to describe a program with these priorities, critical problems may occur. (e.g. Due to insufficient processing time for the robot control task, the path of manipulator movement changes, the robot stops at a teaching step, the job tact time increases, or the system alarm occurs; The programming pendant or the HOLD button freezes because the man-machine control task cannot be operated.)
  2. The task “mpUsrRoot” has a high priority to start other application tasks and initialize the entire application quickly. Thus, as the sample program chapter 17.3 “Task Control Sample Program”, make sure to complete the task “mpUsrRoot” after starting up another application or creating a semaphore. If the process which is run with the normal priority is described without completing mpUsrRoot, the same problem as explained in 1. above may occur.

- **Note for programming by using Ethernet**
  For mpSocket() and mpAccept(), memory in the heap area is temporarily used. Thus, if mpSocket() and mpAccept() is repeated without doing mpClose(), the above working memory is consumed and the system memory becomes insufficient. Then the YRC1000 hangs up, the communication error of the programming pendant occurs, and the robot stops its operation.
  Thus, if you need to repeat mpSocket() and mpAccept(), make sure to do mpClose() each time.

---

**NOTE**

For MotoPlus application programming, note the following points:

- **C language memory operation**
  When programming in C language, the pointer variable which stores the memory address can be used. If the pointer variable is set incorrectly, the system memory area may be rewritten.
  If the system memory area is rewritten, critical problems occur, e.g., the software hangs up (the hang-up status is detected by a CPU exception or the watchdog check function, then the servo power turns OFF and the system shuts down), the robot stops its operation due to an alarm, or the programming pendant becomes inoperable.

- **Task priority and processing time**
  1. Use precautions when you select a task with high priority (the task started with the priority MP_PRI_IO_CLK_TAKE, MP_PRI_IP_CLK_TAKE, or MP_PRI_TIME_CRITICAL). If unusually long processing time is used to describe a program with these priorities, critical problems may occur. (e.g. Due to insufficient processing time for the robot control task, the path of manipulator movement changes, the robot stops at a teaching step, the job tact time increases, or the system alarm occurs; The programming pendant or the HOLD button freezes because the man-machine control task cannot be operated.)
  MotoPlus detects such status (when the man-machine task does not run for 3 seconds or more), stops the robot motion by turning the servo power OFF, and gives an alarm. However, make sure to set 100 microseconds as the maximum processing time in the high-priority task (100 microseconds equals to approx. 1000 lines in the C language). The OS waiting time, the message/semaphore waiting time, and the task delay time are not included in this processing time.
  2. The task “mpUsrRoot” has a high priority to start other application tasks and initialize the entire application quickly. Thus, as the sample program chapter 17.3 “Task Control Sample Program”, make sure to complete the task “mpUsrRoot” after starting up another application or creating a semaphore. If the process which is run with the normal priority is described without completing mpUsrRoot, the same problem as explained in 1. above may occur.

- **Note for programming by using Ethernet**
  For mpSocket() and mpAccept(), memory in the heap area is temporarily used. Thus, if mpSocket() and mpAccept() is repeated without doing mpClose(), the above working memory is consumed and the system memory becomes insufficient. Then the YRC1000 hangs up, the communication error of the programming pendant occurs, and the robot stops its operation.
  Thus, if you need to repeat mpSocket() and mpAccept(), make sure to do mpClose() each time.
For MotoPlus application programming, note the following restrictions:

- Only one application file ("*.out" file) can be started. If two or more files are installed, the alarm 1020: "MOTOPLUS APPLICATION LOAD ERROR [1]" occurs.

- The maximum available memory size is 2 Mbyte as the total of the application code and the static variable. If it is exceeded when the YRC1000 is started, the alarm 1020: "MOTOPLUS APPLICATION LOAD ERROR [4]" occurs, and the application will not be loaded.

- Depending on the task priority, only limited APIs can be used. For details, refer to chapter 16.5.2 “Task Priority and Usable API".
2 Task

A task is an execution unit of a program and the MotoPlus application program is run as a task when the power is turned on. To execute multiple tasks simultaneously, use a multitasking structure.

2.1 Multitasking

Executing multiple tasks simultaneously is called multitasking. MotoPlus can run multiple other tasks from the task which has run first. This allows you, for example, to run the task that performs Ethernet communication and the task that directs an action to a robot, and each task can concentrate on its given operation. In this case, although it is necessary to synchronize the timings between the tasks and/or exchange their information, such a function can be realized through the service provided by MotoPlus.

2.2 State Transitions of Task

Tasks experience the state transitions as shown in the figure below. When a task is activated for the first time, it is set to the Running state. However, when there are multiple tasks, one of them is set to the Ready state. This depends on the priority of the tasks. When a task is waiting for an event, it is set to the Asleep state. If the condition that is waited for is met, the state is shifted to the Ready state. A task can be suspended and shifted to the Suspend state according to the request from itself or another task. The descriptions of the states are as follows:

1. Ready state
   The task in this state can be executed at any time and, if the task in execution at the moment loses its execution right for some reason, starts its execution immediately.

2. Running state
   The task in this state is in execution at the moment. Only one task exists in this state at one point in time.

3. Asleep state
   The task in this state is waiting for some external event. When the wait event occurs, it is set to the executable state (Ready state) immediately. The causes of the Asleep state are as follows:
   (1) The task has slept itself for a certain period of time.
   (2) The task is waiting for the arrival of a mail box.
   (3) The task is waiting for a semaphore (use right of shared resource).
2.3 Task Scheduling Method

The task scheduling method is preemptive scheduling based on priority. The tasks in the Ready state with the same priority are executed in a time-shared manner by round robin scheduling.

2.4 Management of Ready-Queue

As shown below, the Ready-Queue is managed based on the FIFO order.

1. A task with the same priority is placed at the end of the tasks with the same priority according to the FIFO order.
2. When the task in execution is preempted during time slice, it is placed at the head of the tasks with the same priority.
3. When the time slice is finished, it is placed at the end of the tasks with the same priority.
2.5 Application Task Control

Load and start-up of an application program:

1. Filename of an application program:
   The filename of a MotoPlus application program must have “out” (*.out) as the extension. Only one file can be loaded or started.

2. Installation to the YRC1000:
   To install an application program file to the YRC1000, load it from the SD/USB in the programming pendant in the MAINTENENCE mode. The loaded application program file will be automatically loaded into the RAM of the main CPU of ACP01 when the YRC1000 is turned ON online (normal startup), then mpUsrRoot() in the application program will be started as a task.

2.6 Processing Time Measurement

If the processing of a high-priority task takes a long time, the entire system will be affected. Thus, measure the processing time of each task when debugging. Make sure that the processing of a high-priority task completes within 100 microseconds excluding the waiting time for the API execution such as task control. APIs are prepared for the processing time measurement. These APIs provide the stopwatch function. Use the APIs to measure multiple lap timings from the start to the stop. With the APIs, you can create or delete the stopwatch, specify the number of the lap-time measurement, start or stop the stopwatch, specify the lap timing, get the elapsed time until the stop, get the lap time, get the number of the valid lap times, and clear the stored lap time.

The simplest procedure is:

Create a stopwatch -> Start -> Stop -> Get the elapsed time
(mpStopWatchCreate) (mpStopWatchStart) (mpStopWatchStop) (mpStopWatchGetTime)

To measure the lap time:

Create a stopwatch (specify the number of the lap-time measurement) -> Clear the lap time
(mpStopWatchCreate) (mpStopWatchReset)

-> Start -> Enter the lap timing -> Stop
(mpStopWatchStart) (mpStopWatchLap) (mpStopWatchStop)

-> Get the lap number -> Get the lap time
(mpStopWatchGetAliveLapNo) (mpStopWatchGetLapTime)

If the compatibility specification (S2C1116) in the processing time measurement is set, the operation in the processing time measurement is changed. For the specification, use the bit specification.

- d0 bit: Acquisition time of mpStopWatchGetTime()
  (0: equivalent to 2 bite data,
   1: equivalent to 4 bite data (default))

- d1 bit: Measurement of the lap time in mpStopWatchStop()
  Execute/Not execute
  (0: Execute the lap time measurement,
   1: Not execute the lap time measurement (default))
2.7 Limit on Protection Regarding Task Control

2.7.1 Limit on Number of Files Which User Can Load

When starting the YRC1000, the application file (*.out) is loaded to the memory of the main CPU. Only one file can be loaded at this time. The number of installed '*.out' files is checked. If multiple files are found, the alarm 1020: “MOTOPLUS APPLICATION LOAD ERROR [1]” occurs, and $B051 will be set to 1 (loading error: the number of files exceeds the limit).

2.7.2 Limit on Memory Size

When loading an application, the code size of the file to be loaded and the size of the static variable defined by the application are checked if they are within the limit values (up to 2 Mbyte). If the limit is exceeded, no file will be loaded, the alarm 1020: “MOTOPLUS APPLICATION LOAD ERROR [2] or [4]” occurs, and $B051 will be set to 4 (loading error: insufficient memory).

* Subcodes of the alarm 1020: “MOTOPLUS APPLICATION LOAD ERROR”:

[1] The number of files exceeds the limit.: Two or more '*.out' files exist in the SD of ACP01.
[2] The memory is insufficient.: Available memory area is less than 3 Mbyte.
[3] The directory cannot be opened.: APPLICATION folder cannot be found.
[4] Load failure (The file cannot be loaded because the file size (code and static RAM) exceeds 2 Mbyte.)
[5] Load failure (Undefined symbol)
[6] Load failure (Others: application overloaded)
[8] No user root task
[9] Failed to create the user root task.

When the file is loaded successfully without the above errors, $B051 will be set to “0”.
2.7 Limit on Protection Regarding Task Control

2.7.3 Limit on Task Start-up

For each task priority, the number of tasks which can be started is limited.

<table>
<thead>
<tr>
<th>Priority</th>
<th>Description</th>
<th>Max. task number</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP_PRI_IO_CLK_TAKE</td>
<td>Notification of I/O control period</td>
<td>1</td>
</tr>
<tr>
<td>MP_PRI_IP_CLK_TAKE</td>
<td>Notification of interpolation period</td>
<td>1</td>
</tr>
<tr>
<td>MP_PRI_TIME_CRITICAL</td>
<td>High priority for general use</td>
<td>1</td>
</tr>
<tr>
<td>MP_PRI_TIME_NORMAL</td>
<td>Normal priority for general use</td>
<td>3</td>
</tr>
</tbody>
</table>

If the number of tasks to be created exceeds the number listed above, the task creation API “mpCreateTask()” returns an error (-1).

2.7.4 Function to Check Execution Status of Man-Machine Task

If unusually long processing time is used to describe a program in a task with high priority (the task priority: MP_PRI_IO_CLK_TAKE, MP_PRI_IP_CLK_TAKE, or MP_PRI_TIME_CRITICAL), the man-machine control task may become inoperable, and critical problems may occur. (e.g. The programming pendant or the HOLD button freezes.) With this function, the status described above is detected, and the servo power is turned OFF to stop the robot motion so that surrounding hazards can be prevented.

The execution status of the man-machine task is monitored, and it is considered as an error if the man-machine task does not run for 3 seconds or more. Then the alarm 4479: “MOTOPLUS MM TASK WATCHDOG ERROR” occurs, and the servo power of the robot will be turned OFF. Also, the specific output: #50901 is turned ON, and the logic such as the operation stop to avoid peripheral hazards can be described in the CIO ladder.

If the man-machine task does not work at all, the alarm 4479: “MOTOPLUS MM TASK WATCHDOG ERROR” cannot be displayed even if it occurs.

In this case, output #50901 by the external output using CIO, and monitor it by a peripheral equipment PLC, etc.

<Specific output I/O specification>

- I/O output number: specific output #50901
  0: normal, 1: man-machine task watchdog time over
- Function
  If the man-machine task does not run for 3 seconds or more, the specific output (#50901) will be turned ON.
  If the man-machine task resumes the operation after that, the specific output (#50901) will be turned OFF.
### 2.8 List of Task Control API

<table>
<thead>
<tr>
<th>MotoPlus</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 mpCreateTask</td>
<td>Creates a task.</td>
</tr>
<tr>
<td>2 mpDeleteTask</td>
<td>Deletes the task.</td>
</tr>
<tr>
<td>3 mpTaskSuspend</td>
<td>Suspends the task.</td>
</tr>
<tr>
<td>4 mpTaskResume</td>
<td>Resumes the task.</td>
</tr>
<tr>
<td>5 mpMsgQCreate</td>
<td>Creates and initializes a message queue.</td>
</tr>
<tr>
<td>6 mpMsgQDelete</td>
<td>Deletes the message queue.</td>
</tr>
<tr>
<td>7 mpMsgQReceive</td>
<td>Receives a message.</td>
</tr>
<tr>
<td>8 mpMsgQSend</td>
<td>Sends a message.</td>
</tr>
<tr>
<td>9 mpErrMsgQRcv</td>
<td>Gets the cause of the error when receiving a message.</td>
</tr>
<tr>
<td>10 mpErrMsgQSnd</td>
<td>Gets the cause of the error when sending a message.</td>
</tr>
<tr>
<td>11 mpSemBCreate</td>
<td>Creates a semaphore.</td>
</tr>
<tr>
<td>12 mpSemDelete</td>
<td>Deletes the semaphore.</td>
</tr>
<tr>
<td>13 mpSemTake</td>
<td>Takes a semaphore.</td>
</tr>
<tr>
<td>14 mpSemGive</td>
<td>Gives a semaphore.</td>
</tr>
<tr>
<td>15 mpErrSemTake</td>
<td>Gets the cause of the error when getting a semaphore.</td>
</tr>
<tr>
<td>16 mpTaskDelay</td>
<td>Delays a task.</td>
</tr>
<tr>
<td>17 mpGetRtc</td>
<td>Gets the clock cycle of the OS.</td>
</tr>
<tr>
<td>18 mpGetSysClkRate</td>
<td>Gets the clock frequency of the OS.</td>
</tr>
<tr>
<td>19 mpStopWatchCreate</td>
<td>Creates a stopwatch.</td>
</tr>
<tr>
<td>20 mpStopWatchDelete</td>
<td>Deletes the stopwatch.</td>
</tr>
<tr>
<td>21 mpStopWatchStart</td>
<td>Starts the stopwatch.</td>
</tr>
<tr>
<td>22 mpStopWatchStop</td>
<td>Stops the stopwatch.</td>
</tr>
<tr>
<td>23 mpStopWatchLap</td>
<td>Records the lap time.</td>
</tr>
<tr>
<td>24 mpStopWatchReset</td>
<td>Resets the stopwatch.</td>
</tr>
<tr>
<td>25 mpStopWatchGetTime</td>
<td>Gets the elapsed time.</td>
</tr>
<tr>
<td>26 mpStopWatchGetLapNum</td>
<td>Gets the number of the lap time counter.</td>
</tr>
<tr>
<td>27 mpStopWatchGetLapTime</td>
<td>Gets the lap time.</td>
</tr>
<tr>
<td>28 mpStopWatchGetAliveLapNo</td>
<td>Gets the range of the valid lap time numbers.</td>
</tr>
</tbody>
</table>
3 User Watchdog

Use the user watchdog to monitor whether the created MotoPlus application is operating normally. Also, if the application is not operating normally, a user-defined process can be performed.

3.1 Function Outline

Up to 2 user watchdog timers can be created. For example, one having a “short timer value” to monitor the execution of critical processes, and the other having a relatively “long timer value” to monitor less critical processes can be used.

If the user watchdog timer times out, the system turns OFF the servo power of all control groups. In addition to turning OFF the servo power, if necessary, the user can define another process to be executed when the timer times out. For example, “output the user alarm”, “output a signal to the user I/O”, and “notify the status to the JOB by writing the user-defined value into the variable” at the time-out.

3.2 Usage Outline

As shown in the following flow chart, to use the user watchdog, create the user watchdog timer first. At this time, set the timer conditions (the delay value of the timer and the process at the time-out) as well.

The user watchdog timer does not start only by creating it. Thus, start the user watchdog timer at the desired timing.

After starting, periodically clear the user watchdog timer at intervals shorter than the set timer.

When the user watchdog timer is no longer necessary, delete it by mpUsrWdogDelete(). Also, to change the conditions of the user watchdog timer such as the delay value and the time-out routine, delete it first, and then create the user watchdog timer again to set new conditions.

In addition, if the user watchdog timer times out, the timer automatically stops. To start the user watchdog again after recovering from the error, execute mpUsrWdogStart().
### User Watchdog API List

<table>
<thead>
<tr>
<th>MotoPlus</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mpUsrWdogCreate</td>
</tr>
<tr>
<td></td>
<td>Creates the user watchdog.</td>
</tr>
<tr>
<td>2</td>
<td>mpUsrWdogDelete</td>
</tr>
<tr>
<td></td>
<td>Deletes the user watchdog.</td>
</tr>
<tr>
<td>3</td>
<td>mpUsrWdogStart</td>
</tr>
<tr>
<td></td>
<td>Starts the user watchdog.</td>
</tr>
<tr>
<td>4</td>
<td>mpUsrWdogClear</td>
</tr>
<tr>
<td></td>
<td>Clears the user watchdog.</td>
</tr>
</tbody>
</table>
4 Ethernet Communications

4.1 Socket

The TCP/IP socket library provides a socket as an end-point that connects the world of program and the world of network. By using a socket, a program can send data to the socket of the destination program by just transmitting the data to the socket. When a program is created by using a socket and if the socket can be connected to the destination socket, it is not necessary for the programmer to think of how the data is sent and received over the TCP/IP network.

4.2 Client Server Model

The program using the IP communication generally consists of a model called the client server model.

The word “client” indicates a program such as a Web browser, which requests processes. The word “server” indicates a program such as a Web server, which performs the processes on the request of the client.

There are 2 major models of the operating environment for the client and server. In many cases, the server and client operate on different machines. The MotoPlus application normally operates on this model. At the same time, it is also possible to operate the server and client on the same machine. In this case, the IP communication is used to perform communication between processes.

**Fig. 4-1: Model 1: Machine communication**

**Fig. 4-2: Model 2: Process communication**
4.3 UDP Communication

The typical protocols for using the IP communication are UDP and TCP. UDP (User Datagram Protocol) is a protocol to use the IP functions directly from a program. It provides a datagram type socket which is connectionless and does not assure the reliability of communication. Since the datagram type socket does not perform retransmission when packets are lost and management of the order of transmission data, data loss or duplication may occur.

However, since UDP has less overhead in communication process than TCP, it is used for the delivery of stream voices and images for which data missing causes few problems and the applications in which real-time property is important.

4.4 TCP Communication

TCP (Transmission Control Protocol) is a protocol to provide a reliable communication between machines by extending the IP functions. Unlike UDP, it is connection-oriented and provides a reliable stream type socket. Since the stream type socket establishes a connection before actual data sending/receiving, it assures reliable data sending/receiving with data loss detection, data arrival order management, etc.

However, its speed is sacrificed due to its high reliability, it is used for the applications in which reliable data sending/receiving is more important than speed.
4.5 List of Socket Library Functions

The list of the socket library functions supported by MotoPlus is shown below.

<table>
<thead>
<tr>
<th>Linux/WinSock</th>
<th>MotoPlus</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 socket</td>
<td>mpSocket</td>
<td>Generating a socket</td>
</tr>
<tr>
<td>2 Bind</td>
<td>mpBind</td>
<td>Setting own IP address and port number to a socket</td>
</tr>
<tr>
<td>3 connect</td>
<td>mpConnect</td>
<td>Sending a connection request from a client to a server</td>
</tr>
<tr>
<td>4 listen</td>
<td>mpListen</td>
<td>Setting a TCP server socket to the Connection ready state</td>
</tr>
<tr>
<td>5 accept</td>
<td>mpAccept</td>
<td>Generating a socket by establishing a client connection with a TCP server</td>
</tr>
<tr>
<td>6 send</td>
<td>mpSend</td>
<td>Data transmission to a connected socket</td>
</tr>
<tr>
<td>7 sendto</td>
<td>mpSendTo</td>
<td>Data transmission to a socket</td>
</tr>
<tr>
<td>8 recv</td>
<td>mpRecv</td>
<td>Data reception form a connected socket</td>
</tr>
<tr>
<td>9 recvfrom</td>
<td>mpRecvFrom</td>
<td>Data reception form a socket</td>
</tr>
<tr>
<td>10 close(Linux)closesocket (Win)</td>
<td>mpClose</td>
<td>Closing a socket</td>
</tr>
<tr>
<td>11 inet_ntoa</td>
<td>mpInetNtoa</td>
<td>Converting an IP address value to a dot notation string address</td>
</tr>
<tr>
<td>12 htonl</td>
<td>mpHtonl</td>
<td>Converting a 32-bit byte order (Host -&gt; Network)</td>
</tr>
<tr>
<td>13 htons</td>
<td>mpHtons</td>
<td>Converting a 16-bit byte order (Host -&gt; Network)</td>
</tr>
<tr>
<td>14 ntohl</td>
<td>mpNtohl</td>
<td>Converting a 32-bit byte order (Network -&gt; Host)</td>
</tr>
<tr>
<td>15 ntohs</td>
<td>mpNtohs</td>
<td>Converting a 16-bit byte order (Network -&gt; Host)</td>
</tr>
<tr>
<td>16 select</td>
<td>mpSelect</td>
<td>Waiting for the socket descriptor status to change</td>
</tr>
<tr>
<td>17 WSAStartup(Win)</td>
<td>---</td>
<td>Starting the use of Ws2_32.dll (WinSock only)</td>
</tr>
<tr>
<td>18 WSACleanup(Win)</td>
<td>---</td>
<td>Terminating the use of Ws2_32.dll (WinSock only)</td>
</tr>
</tbody>
</table>

4.6 Operation of Structure and IP Address

4.6.1 Sockaddr Structure

The sockaddr structure is a basic structure for socket programming. The members of the structure are as follows.

```c
struct sockaddr {
    unsigned char sa_len;
    unsigned char sa_family;
    char sa_data[14];
};
```
4.6.2 Sockaddr_in Structure

The sockaddr_in structure is used to specify the IP address and/or port number of the destination host or the source host. Note that the in_addr structure that sets an IP address stores the IP address as 32-bit numeric data. Thus, for the member; s_addr, of the in_addr structure, its IP address must be set in a network byte order. Similarly, for the member; sin_port, of the sockaddr_in structure, its port number must be set in a network byte order.

```
struct in_addr {
    unsigned long s_addr;
};
```

```
struct sockaddr_in {
    unsigned char sin_len; /* Structure size*/
    unsigned char sin_family; /* Address family */
    unsigned short sin_port; /* Port number */
    struct in_addr sin_addr; /* IP address */
    char sin_zero[8]; /* Unused */
};
```

4.6.3 Function for IP Address Operation

The in_addr structure that sets an IP address stores the IP address as 32-bit numeric data. The following function converts this to a dotted-decimal notation string IP address such as "192.168.0.10.".

Although inet_ntoa() returns a pointer for a dotted-decimal notation string, the memory area indicated by the returned value is statically reserved. Thus, this function call is not reentrant and the value is changed each time the function is called. Therefore, the value must be copied with memcpy() etc. when you want to maintain the value indicated by the returned value.

<table>
<thead>
<tr>
<th>Linux/WinSock</th>
<th>MotoPlus</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>inet_ntoa</td>
<td>mplnetNtoa</td>
<td>Converting an IP address value to a dot notation string address</td>
</tr>
</tbody>
</table>
4.7 About Byte Order

Each computer connected to a network may have mixed 2 types of byte order; little endian and big endian. Therefore, the data on a network is unified to the big endian called network byte order. Thus, when data is sent from a little endian machine, it must be converted into big endian. However, the function which always converts the byte order regardless of the byte order of the target machine is used on a program in consideration of portability. In this case, on a big endian machine, no process will be performed even if the function for byte order conversion is called.

<table>
<thead>
<tr>
<th>Table 4-3: Byte Order Conversion Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
</tr>
<tr>
<td>Converting a 32-bit byte order (Host -&gt; Network)</td>
</tr>
<tr>
<td>Converting a 16-bit byte order (Host -&gt; Network)</td>
</tr>
<tr>
<td>Converting a 32-bit byte order (Network -&gt; Host)</td>
</tr>
<tr>
<td>Converting a 16-bit byte order (Network -&gt; Host)</td>
</tr>
</tbody>
</table>

4.8 Flow of Socket Library System Call

Assuming one-to-one communication, the flow of socket library system call is shown below, on the basis that the client already knows the IP address/port number of the server and the server already knows its own IP address/port number.

4.8.1 Flow of Socket Library System Call with UDP

<table>
<thead>
<tr>
<th>1. socket()</th>
<th>1. socket()</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. bind()</td>
<td>2. bind()</td>
</tr>
<tr>
<td>3. sendto()</td>
<td>3. recvfrom()</td>
</tr>
<tr>
<td>4. recvfrom()</td>
<td>4. sendto()</td>
</tr>
<tr>
<td>5. close()</td>
<td>5. close()</td>
</tr>
</tbody>
</table>
4.8.2 Flow of Socket Library System Call with TCP

Client
1. socket() 2. connect() 3. send() 4. recv() 5. close()

Server
1. socket() 2. bind() 3. listen() 4. accept() 5. recv() 6. send() 7. close()

4.9 Note for Ethernet Programming

For mpSocket() and mpAccept(), memory in the heap area is temporarily used. Thus, if mpSocket() and mpAccept() is repeated without doing mpClose(), the above working memory is consumed and the system memory becomes insufficient. Then the YRC1000 hangs up, the communication error of the programming pendant occurs, and the robot stops its operation. Thus, if you need to repeat mpSocket() and mpAccept(), make sure to do mpClose() each time.

Note
Maximum 20 sockets can be opened at one time.

Note
The port numbers (port 9900 to 10499) which are used by the system software cannot be used for the MotoPlus application. If the MotoPlus application uses the port numbers which are used by the system software, YRC1000 returns an error.
For the MotoPlus application, it is recommended to use the port numbers from 20000 to 23000.
When a programming pendant application is executed on the programming pendant, communications with a MotoPlus application may be needed. If so, data transmission between the programming pendant application and the MotoPlus application will be performed by using the socket communication via Ethernet.

- **Use TCP/IP socket communication.**
  Use the IP address defined as the macro MP_PP_HOST_ADDR for the programming pendant.
6 RS232C Communication Control

Communication by using RS232C of the ACP01 circuit board is available (optional).

For serial communication, 5 APIs (initialization to send and receive, open, close, send, and receive) are prepared for the ACP01 COM1 (CN103 of the ACP01 circuit board) serial port. The FIFO buffer of 1 Kbyte for sending and receiving is prepared, and the send and receive functions perform reading from and writing to the buffer.

<table>
<thead>
<tr>
<th>MotoPlus</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpRsInit</td>
<td>Initializes the specified serial communication port.</td>
</tr>
<tr>
<td>mpRsOpen</td>
<td>Opens the specified serial communication port.</td>
</tr>
<tr>
<td>mpRsClose</td>
<td>Closes the specified serial communication port.</td>
</tr>
<tr>
<td>mpRsSend</td>
<td>Sends data to the serial communication port.</td>
</tr>
<tr>
<td>mpRsRecv</td>
<td>Receives data from the serial communication port.</td>
</tr>
</tbody>
</table>
7 Event Control

7.1 Task Creation

Use mpCreateTask() to create the task to receive the notification as described below.

In this case, specify the priority depending on the desired notification as the first argument of mpCreateTask().

- **I/O Control Cycle**
  Notification Receiving Task ---- Specify MP_PRI_IO_CLK_TAKE. This task will mainly perform accessing to I/O, and monitoring the robot control information.

- **Interpolation Cycle**
  Notification Receiving Task ---- Specify MP_PRI_IP_CLK_TAKE. This task will mainly control robot operations, such as correcting the path of the robot.

```c
io_tid = mpCreateTask(MP_PRI_IO_CLK_TAKE, MP_STACK_SIZE, (FUNCPTR)mp_io_mon_task, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
ip_tid = mpCreateTask(MP_PRI_IP_CLK_TAKE, MP_STACK_SIZE, (FUNCPTR)mp_seg_mon_task, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
```

**NOTE**

For the I/O control cycle notification receiving task and the interpolation cycle notification receiving task, the usable APIs are limited. For details, refer to chapter 16.5.2 “Task Priority and Usable API”.

7.2 Event Notification

The following event notification can be received.

- **I/O Control Cycle**
  The cycle controls the signal between the concurrent I/O and the application. And by using this cycle, the feedback information such as speed or torque from the servo is updated.

- **Interpolation Cycle**
  The cycle calculates the command position of the robot.

Specify the desired notification type and call mpClkAnnounce for each started task as described below.

- **I/O control cycle**
  -------- MP_IO_CLK

- **Interpolation cycle**
  -------- MP_INTERPOLATION_CLK
This makes the system wait with \texttt{mpClkAnnounce()} until the notification of the control cycle. When the notification is received, the control will be resumed.

```c
void mp_io_mon_task(void)
{
    FOREVER
    {
        mpClkAnnounce(MP_IO_CLK);
        // The desired process in the I/O control cycle
    }
}

void mp_seg_mon_task(void)
{
    FOREVER
    {
        mpClkAnnounce(MP_INTERPOLATION_CLK);
        // The desired process in the interpolation cycle
    }
}
```

\begin{note}
The task which uses \texttt{mpClkAnnounce} has a high task priority, so if a time-consuming process (10 microseconds or longer) is done, the whole system may be affected. Thus, perform only the description process whose processing time is short (within several microseconds) such as the fixed-cycle condition monitoring.
\end{note}

\begin{note}
For the task which uses \texttt{mpClkAnnounce}, use APIs following the limitations below.

1. \texttt{mpClkAnnounce (MP\_IO\_CLK)}
   For the I/O control cycle, use only the I/O read/write API and the task control API. If another API is used, the fixed-cycle condition monitoring cannot be performed due to long processing time.

2. \texttt{mpClkAnnounce (MP\_INTERPOLATION\_CLK)}
   For the interpolation cycle, use only the I/O read/write API, the variable read/write API, and the task control API. If another API is used, the fixed-cycle condition monitoring cannot be performed due to long processing time.
\end{note}
8 Robot System Control

8.1 System Information

Regarding the system information, the following information is accessible:

<Readable information>
Robot's current position (pulse command position, Cartesian coordinate command position, feedback pulse position), current speed, current torque, mode (teach, play, or remote) information, motion cycle information, information on whether a job is in execution, information on servo power state, information on playback special mode state (during speed limitation, check operation, etc.), alarm information, information on current master job, currently selected job name, and speed of jog motion

<Writable or controllable information>
Error cancellation, alarm reset, mode (play, teach, or remote), motion cycle information, servo power state, master job name, currently selected job name, start of job execution, job hold, wait state for the completion of job execution, and job deletion

8.2 Variables

The following variables used in a job can be read or written.
Byte type variable (B variable), Integer type variable (I variable), Double-precision type variable (D variable), Floating-point variable (R variable), and Positional variable (Robot axis: P variable, Base axis: BP variable, External axis: EX variable)

8.3 I/O

Regarding I/O, all types of I/Os can be referenced for reading. For writing, user output #10010 - #15127, interface panel #60010 - #60647, network I/O #27010 - #26287, register #1000000 - #1000559 can be referenced.

8.4 Robot Operation

IMOV (incremental operation), MOVJ (joint operation) and MOVL (Cartesian operation) can be executed. The operations are executed step by step.

There is no interlock between the MotoPlus API which writes variables, I/O, or registers and the following modification methods other than MotoPlus, thus the MotoPlus API can write them. Pay attention to the interference with the other methods so that the modification by MotoPlus will not affect the system’s operation.

1. Edit of variables, I/O, or registers in the teach mode by using the programming pendant
2. Change of variables, I/O, or registers by job execution
3. Change of I/O or registers by the concurrent I/O output
<table>
<thead>
<tr>
<th><strong>MotoPlus</strong></th>
<th><strong>Function</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 mpGetVarData</td>
<td>Retrieves variables (B, I, D, R).</td>
</tr>
<tr>
<td>2 mpGetSVarInfo</td>
<td>Retrieves S variable.</td>
</tr>
<tr>
<td>3 mpReadIO</td>
<td>Retrieves the I/O status.</td>
</tr>
<tr>
<td>4 mpMonitor</td>
<td>Retrieves a variable and an I/O value at a time.</td>
</tr>
<tr>
<td>5 mpGetPosVarData</td>
<td>Retrieves the position-type variable.</td>
</tr>
<tr>
<td>6 mpGetUserVars</td>
<td>Retrieves the user variable.</td>
</tr>
<tr>
<td>7 mpGetAlarmStatus</td>
<td>Retrieves the alarm status.</td>
</tr>
<tr>
<td>8 mpGetAlarmCode</td>
<td>Retrieves the alarm code of the system.</td>
</tr>
<tr>
<td>9 mpGetMode</td>
<td>Retrieves the mode information.</td>
</tr>
<tr>
<td>10 mpGetCycle</td>
<td>Retrieves the cycle information.</td>
</tr>
<tr>
<td>11 mpGetServoPower</td>
<td>Retrieves the servo power status.</td>
</tr>
<tr>
<td>12 mpGetPlayStatus</td>
<td>Retrieves the play mode status.</td>
</tr>
<tr>
<td>13 mpGetMasterJob</td>
<td>Retrieves the master job name.</td>
</tr>
<tr>
<td>14 mpGetCurJob</td>
<td>Retrieves the name of the currently selected job.</td>
</tr>
<tr>
<td>15 mpGetSpecialOpStatus</td>
<td>Retrieves the special operation status.</td>
</tr>
<tr>
<td>16 mpGetJobData</td>
<td>Retrieves the selected job date.</td>
</tr>
<tr>
<td>17 mpGetCartPos</td>
<td>Retrieves the current position in Cartesian coordinates.</td>
</tr>
<tr>
<td>18 mpGetCartPosEx</td>
<td>Retrieves the current position in Cartesian coordinates with a specified coordinate frame (Robot, Base, User).</td>
</tr>
<tr>
<td>19 mpGetPulsePos</td>
<td>Retrieves the current pulse position.</td>
</tr>
<tr>
<td>20 mpGetFBPulsePos</td>
<td>Retrieves the current feedback pulse position.</td>
</tr>
<tr>
<td>21 mpGetFBPulsePosEx</td>
<td>Retrieves the current feedback pulse position (the motor-axis, the load-axis).</td>
</tr>
<tr>
<td>22 mpGetServoSpeed</td>
<td>Retrieves the current servo speed.</td>
</tr>
<tr>
<td>23 mpGetFBSpeed</td>
<td>Retrieves the current feedback speed.</td>
</tr>
<tr>
<td>24 mpGetTorque</td>
<td>Retrieves the current torque.</td>
</tr>
<tr>
<td>25 mpGetTorqueEx</td>
<td>Retrieves the current torque (absolute value).</td>
</tr>
<tr>
<td>26 mpGetSysTimes</td>
<td>Retrieves the current system time.</td>
</tr>
<tr>
<td>27 mpGetSysVersionNo</td>
<td>Retrieves the system version.</td>
</tr>
<tr>
<td>28 mpGetRadPos</td>
<td>Retrieves the current position (unit: radian).</td>
</tr>
<tr>
<td>29 mpGetRadPosEx</td>
<td>Retrieves the current position and the unit (unit: radian).</td>
</tr>
<tr>
<td>30 mpGetDegPos</td>
<td>Retrieves the current position (unit: degree).</td>
</tr>
<tr>
<td>31 mpGetDegPosEx</td>
<td>Retrieves the current position and the unit (unit: degree).</td>
</tr>
<tr>
<td>32 mpGetJogSpeed</td>
<td>Retrieves the jog speed.</td>
</tr>
<tr>
<td>33 mpGetJogCoord</td>
<td>Retrieves the jog coordinates.</td>
</tr>
<tr>
<td>34 mpCtrlGrpId2GrpNo</td>
<td>Retrieves the control group number.</td>
</tr>
<tr>
<td>35 mpGetToolData</td>
<td>Retrieves the tool data.</td>
</tr>
</tbody>
</table>
# Robot System Control

## 8.6 System Control API List

<table>
<thead>
<tr>
<th>MotoPlus</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>mpGetCalendar</td>
</tr>
<tr>
<td>37</td>
<td>mpGetJobStepData</td>
</tr>
<tr>
<td>38</td>
<td>mpGetToolNo</td>
</tr>
<tr>
<td>39</td>
<td>mpGetEncoderTemp</td>
</tr>
</tbody>
</table>

### MotoPlus Function

<table>
<thead>
<tr>
<th>MotoPlus</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mpPutVarData</td>
</tr>
<tr>
<td>2</td>
<td>mpPutSVarInfo</td>
</tr>
<tr>
<td>3</td>
<td>mpWriteIO</td>
</tr>
<tr>
<td>4</td>
<td>mpPutPosVarData</td>
</tr>
<tr>
<td>5</td>
<td>mpPutUserVars</td>
</tr>
<tr>
<td>6</td>
<td>mpSetAlarm</td>
</tr>
<tr>
<td>7</td>
<td>mpCancelError</td>
</tr>
<tr>
<td>8</td>
<td>mpResetAlarm</td>
</tr>
<tr>
<td>9</td>
<td>mpSetCycle</td>
</tr>
<tr>
<td>10</td>
<td>mpSetServoPower</td>
</tr>
<tr>
<td>11</td>
<td>mpSetMasterJob</td>
</tr>
<tr>
<td>12</td>
<td>mpSetCurJob</td>
</tr>
<tr>
<td>13</td>
<td>mpStartJob</td>
</tr>
<tr>
<td>14</td>
<td>mpHold</td>
</tr>
<tr>
<td>15</td>
<td>mpWaitForJobEnd</td>
</tr>
<tr>
<td>16</td>
<td>mpDeleteJob</td>
</tr>
<tr>
<td>17</td>
<td>mpiMOV</td>
</tr>
<tr>
<td>18</td>
<td>mpMOVJ</td>
</tr>
<tr>
<td>19</td>
<td>mpMOVL</td>
</tr>
<tr>
<td>20</td>
<td>mpPulseMOVJ</td>
</tr>
<tr>
<td>21</td>
<td>mpPulseMOVL</td>
</tr>
<tr>
<td>22</td>
<td>mpSetJogCoord</td>
</tr>
<tr>
<td>23</td>
<td>mpSetToolNo</td>
</tr>
<tr>
<td>24</td>
<td>mpManualMOV</td>
</tr>
<tr>
<td>25</td>
<td>mpApplicationInfoNotify</td>
</tr>
<tr>
<td>26</td>
<td>mpSetJobPosData</td>
</tr>
</tbody>
</table>
9 Motion Control Service

Provides the service of operating the manipulator and the external axes.

9.1 Motion Control Service

9.1.1 Outline

This service moves manipulators by executing multiple steps in order. The destination position is specified by the data including the target position and the interpolation type, etc. The data is called “target”. The controller has a buffer to store multiple targets. Targets are sent by the user application, and stored in the buffer. When requested to start operation, the targets are executed in the order of being stored in the buffer.

The sent target is stored in the buffer with additional information. The additional information is the data, which is in the controller and supports the target, being called “target profile”.

The target profile includes the coordinate system, speed, etc. The information is like which coordinate system is being based on by the destination position of the sent target, or how fast the operation is executed, etc.
The target profile can be set from the user application with APIs. Setting the voluntary target profile in advance before sending can realize various movements.

For example, if operate with the linear speed of 100 [mm/sec], set the linear speed to 100 [mm/sec] with API, and then send the target.
### API List

<table>
<thead>
<tr>
<th>MotoPlus</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 mpMotStart</td>
<td>Starts the operation (target execution).</td>
</tr>
<tr>
<td>2 mpMotStop</td>
<td>Stops the operation (target execution).</td>
</tr>
<tr>
<td>3 mpMotTargetClear</td>
<td>Clears the target already sent.</td>
</tr>
<tr>
<td>4 mpMotTargetSend</td>
<td>Sends the target.</td>
</tr>
<tr>
<td>5 mpMotTargetreceive</td>
<td>Receives the report of the manipulator having arrived to the destination position.</td>
</tr>
<tr>
<td>6 mpMotSetCoord</td>
<td>Sets the coordinate system of the destination position.</td>
</tr>
<tr>
<td>7 mpMotSetSpeed</td>
<td>Sets the speed of the target.</td>
</tr>
<tr>
<td>8 mpMotSetOrigin</td>
<td>Sets the basis of the destination position.</td>
</tr>
<tr>
<td>9 mpMotSetTask</td>
<td>Sets the task No.</td>
</tr>
<tr>
<td>10 mpMotSetSync</td>
<td>Sets the coordination.</td>
</tr>
<tr>
<td>11 mpMotResetSync</td>
<td>Resets the coordination.</td>
</tr>
<tr>
<td>12 mpMotSetConfig</td>
<td>Sets the robot type.</td>
</tr>
<tr>
<td>13 mpMotSetAccuracy</td>
<td>Sets the positioning level.</td>
</tr>
<tr>
<td>14 mpMotSetAccel</td>
<td>Sets the acceleration speed.</td>
</tr>
<tr>
<td>15 mpMotSetDecel</td>
<td>Sets the deceleration speed.</td>
</tr>
</tbody>
</table>
9.2 Step Motion Service

This service moves the manipulator only one step to the destination position. Use when moving the manipulator to the standby position.

<table>
<thead>
<tr>
<th>MotoPlus</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 mplMOV</td>
<td>Moves the manipulator one step to the position where the increment value is added to the current position by linear interpolation.</td>
</tr>
<tr>
<td>2 mpMOVJ</td>
<td>Moves the manipulator one step to the target cartesian value by joint interpolation.</td>
</tr>
<tr>
<td>3 mpMOVL</td>
<td>Moves the manipulator one step to the target cartesian value by linear interpolation.</td>
</tr>
<tr>
<td>4 mpPulseMOVJ</td>
<td>Moves the manipulator one step to the target pulse value by joint interpolation.</td>
</tr>
<tr>
<td>5 mpPulseMOVL</td>
<td>Moves the manipulator one step to the target pulse value by linear interpolation.</td>
</tr>
</tbody>
</table>
9.3 JOB Link Operation Control Service

This service is used along with the JOB operation.

9.3.1 Increment Value Move Function

This function is provided by mpExRcsIncrementMove(), and the manipulator moves for the increment value specified by the API while executing WAIT instruction.

When specifying the increment value by 0.1mm each to y-direction while the interpolation cycle is set to 4msec, the manipulator moves 500 times in 2 seconds and 50mm to y-direction.

9.3.2 Robot Operation Control API List

<table>
<thead>
<tr>
<th>MotoPlus</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mpExRcsIncrementMove</td>
</tr>
</tbody>
</table>
10 Sensor Control

With the sensor control, the APIs, which transfer data to and from the JOB to change operating conditions such as the path and speed of the manipulator in operation (path correction and speed override), are provided.

10.1 Outline of Provided Functions

10.1.1 Path Correction Function

The path correction function is provided by mpMeiPutCorrPath(), which corrects the taught path by the amount specified by the API. The amounts of correction are added up.

In the example shown below, let the taught path (P1-P2) to move at 25 mm/sec for 2 seconds in X-direction (50 mm in total) be corrected 0.1 mm in Y-direction per interpolation cycle. If the interpolation cycle is 4 msec, it is corrected 500 times for 2 seconds, it thus moves 50 mm both in X-direction and Y-direction.
10.1.2 Interpolation-Motion Forced Change Function

The interpolation-motion forced change function is provided by `mpMeiForcePathEnd()`, which force-quiets the current interpolation motion (without acceleration or deceleration) and changes to the next step. In the example shown below, in the operation of the JOB (P1-P2-P3), when this function is executed during the interpolation motion from P1 to P2, the motion is force-quit and changed to the motion to the next step (P3).

10.1.3 Speed Override Function

The speed override function is provided by `mpMeiPutSpdOverride()`, which changes the current speed of interpolation motion to the speed by the specified ratio. When changing the speed, it is accelerated or decelerated as necessary. In the example shown below, the speed override is set to 50% during the operation with the speed specified in the JOB. When the speed override is set, the speed starts to decelerate, and continues to decelerate until it reaches 50% of the speed specified in the JOB, and then it is fixed.
10.1.4 Increment Value Move Function

This function is provided by mpMeiIncrementMove(), and the manipulator moves for the increment value specified by the API while executing WAIT instruction.

When specifying the increment value by 0.1mm each to Y-direction while the interpolation cycle is set to 4msec, the manipulator moves 500 times in 2 seconds and 50mm to Y-direction.
10.2 Outline of Operation

In this section, the outline of sensor control operation is described by using the path correction as an example. Before using the API which modifies operating conditions such as path correction or speed change, the command (text data defined by the user to start or stop the path correction) must be notified from the JOB to the MotoPlus application by using SKILLSND instruction.

The operating conditions must be modified at each interpolation cycle or at an integral multiple of the interpolation cycle. Even if an API is executed at a cycle shorter than the interpolation cycle, it is not reflected in the manipulator’s motion. Thus, use mpClkAnnounce() in the task in which the API is executed to make the process done at an integral multiple of the interpolation cycle.

The following diagram illustrates the path correction process of the JOB and the MotoPlus application. (simplified description)

1. The JOB sends the command (“CORRON”) to the MotoPlus application by SKILLSND instruction (SKILLSND “CORRON”).
2. The MotoPlus application receives the command by the command receiving task, and if the command content is “CORRON” (start of path correction), requests the path correction task to start the correction.
3. The path correction task continues in an infinite loop. Use mpMeiPutCorrPath() to give the amount of path correction which synchronize with the system’s interpolation cycle by using mpClkAnnounce().
4. The JOB reflects the amount of path correction given by the MotoPlus application to the original taught path while executing MOVL between SKILLSND “CORRON” and SKILLSND “CORROFF”.
5. The JOB sends the command “CORROFF” by executing SKILLSND “CORROFF”.
6. The MotoPlus application receives “CORROFF”, and requests the path correction task to stop the correction.
10.3 Timing of Path Correction Process

By using mpClkAnnounce(), the interpolation task process of the system software can be synchronized with the path correction task process of the MotoPlus application as shown below. The amount of path correction specified by the path correction process (execution of mpMeiPutCorrPath()) is processed at the next interpolation task process of the system software and is reflected to the motion command.

10.4 Details of Giving and Receiving Command and Changing Operating Conditions by SKILLSND (Including Path Correction)

SKILLSND instruction sends a command to the communication area for the control group of the JOB, and the MotoPlus application receives the command by using mpReceiveSkillCommand(). Use the parameter sl_id to specify the communication area from which the command is received.

When the MotoPlus application receives the command, after the preparation (data initialization, etc.) for path correction or speed change (speed override) is completed, the MotoPlus application executes mpEndSkillCommandProcess(). After that, the completion of command receiving process is notified to the system, and then the JOB execution proceeds to the next step of SKILLSND. This is the mechanism to make the manipulator wait before starting its operation until the preparation of the MotoPlus application is completed in order to ensure that the path correction can be performed immediately after the manipulator starts its operation.
10 Sensor Control
10.4 Details of Giving and Receiving Command and Changing Operating Conditions by SKILLSND (Including Path Correction)

Normally, skillCommandReceive() is used to receive SKILL command. On the assumption that it is used in "while" loop, the process is described as follows:

```c
int skillCommandReceive(SYS2MP_SENS_MSG *msg)
{
    // Returns as command-completed response. Execution proceeds to the next step of SKILLSND.
    mpEndSkillCommandProcess (MP_SL_ID1, msg);

    // receiving command issued by SKILLSND instruction
    return (mpSkillCommandReceive (MP_SL_ID1,msg));
}
```

The actual operation of the JOB and the MotoPlus application is described below by using a source code sample.

- **Receiving sensor command**

  The command sent by SKILLSND is received by mpSkillCommandReceive() function. SKILLSND instruction suspends its execution and does not proceed to the next step until mpEndSkillCommandProcess() is executed. Thus, to adjust the timing of completion of SKILLSND, mpEndSkillCommandProcess() and mpReceiveSkillCommand() (which is used to receive the command) can be used independently.

- **Process according to command content**

  After receiving the command, the main command and the sub command are analyzed. For the main command=0 and the sub command=MP_SKILL_SND (=1), the text specified by SKILLSND is sent to comm_link as the command for the process. The process branches according to this content.

  When "CORRON" is sent, as correct_req=1, it requests the path correction task to start the path correction process.

  When "CORROFF" is sent, as correct_req=0, it requests the path correction task to stop the path correction process.

  In addition to SKILLSND, SKILLEND command can be sent at the start of the JOB if the JOB process cannot be continued. In this case, it is sent as the main command=0 and the sub command=MP_SKILL_END (=2).
Selection of the communication area with MotoPlus application used by SKILLSND

The communication area with the MotoPlus application to which SKILLSND sends a command is automatically selected to MP_SL_ID1 to the R1 JOB and MP_SL_ID2 to the R2 JOB.

For the coordinated JOB, the communication area is determined by the slave side robot.
- R1+R2:R1(R1:master): MP_SL_ID2 is selected because R2 is the slave.
- R1+R2:R2(R2:master): MP_SL_ID1 is selected because R1 is the slave.

For an exclusive JOB to the external axis, the communication area is not automatically selected. SLn tag should be put to SKILLSEND, and this tag selects the communication area.
- SKILLSEND SL5 “command”: MP_SL_ID1 is selected.
- SKILLSEND SL6 “command”: MP_SL_ID2 is selected.

NOTE
SKILLSND instruction can be executed only by the JOB to which a robot or an external axis is defined. Incase executing SKILLSND by a JOB to which a robot is not defined, an alarm “4475: CANNOT EXECUTE JOB(NO ROBOT)” occurs.

Also, when it is executed by a concurrent JOB, “4524:CANNOT EXECUTE INST(CONCUR JOB)” alarm occurs.
10 Sensor Control
10.4 Details of Giving and Receiving Command and Changing Operating Conditions by SKILLSND (Including Path Correction)

**SKILLEND command**

SKILLEND command is sent as the main command=0 and the sub command=MP_SKILL_END (=2). After completing MP_SKILL_SND command process, if one of the operations listed below which makes the process unable to continue is performed and then the JOB execution is started, this command is sent.

- JOB selection
- Execution of JOB or file edit (including changing the JOB’s cursor location)
- Execution of jog operation
- Change of step, or FWD (forward) or BWD (backward)
  (For example, in the case after executing MP_SKILL_SND command, the mode is changed from the play mode to the teach mode during playback, the JOB’s cursor is moved, and then the start operation is performed.)

After receiving this command, stop the process performed in real-time such as the path correction. In the example, the stop of the path correction process is requested by correct_req=0.

**Path correction task process**

After initializing the local data for data correction, the path correction task enters an infinite-loop process. At this point, the process can be synchronized with the interpolation cycle by using mpClkAnnounce().

At each interpolation cycle, correct_req is checked, and if the request is issued, the amount of path correction is given by using mpMeiPutCorrPath().

In the example, as the path correction, 1 mm correction at each interpolation cycle is given to Y-axis of the base frame.

In the actual application using sensor information, the amount of path correction is calculated using sequentially changing sensor information by the external input via Ethernet or serial communication.

**Clearing accumulated correction amount**

For the path correction, the amount of path correction per interpolation cycle is issued by using mpMeiPutCorrPath() as a request to the controller, and the controller keeps the accumulated value of difference from the original taught path. Thus, the correction amount is kept accumulated after multiple move instructions. Therefore, to initialize the accumulated value and return to the original taught path, insert the instruction ADVINIT in the JOB.

<<JOB Example>>

```plaintext
NOP
MOVJ   V=10
SKILLSND "CORRON"
MOVL   V=100
MOVL   V=100
SKILLSND "CORROFF"
ADVINIT
MOVL   V=500
END
```

Start of path correction

Accumulates the amount of path correction.

Clears the accumulated amount of path correction, and returns the path to the original taught path.
10 Sensor Control

10.4 Details of Giving and Receiving Command and Changing Operating Conditions by SKILLSND (Including Path Correction)

For details, refer to the example shown in chapter 17.8 “Sensor Control Sample Program”.
### 10.5 Sensor Control API List

<table>
<thead>
<tr>
<th>MotoPlus</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 mpReceiveSkillCommand</td>
<td>Receives a sensor command sent by SKILLSND from the system.</td>
</tr>
<tr>
<td>2 mpEndSkillCommandProcess</td>
<td>Notifies the system that the process to receive the command is completed. After this, the JOB execution proceeds to the next instruction of SKILLSND.</td>
</tr>
<tr>
<td>3 mpMeiGetJobExecTask</td>
<td>Retrieves the task number of the JOB in execution.</td>
</tr>
<tr>
<td>4 mpMeiGetExecControlGroup</td>
<td>Retrieves the control group of the path of the move instruction (instruction in interpolation process) in execution.</td>
</tr>
<tr>
<td>5 mpMeiGetInterpolation</td>
<td>Retrieves interpolation information of the move instruction of the JOB in execution.</td>
</tr>
<tr>
<td>6 mpMeiPutCorrPath</td>
<td>Sets a path correction amount for the path in interpolation process.</td>
</tr>
<tr>
<td>7 mpMeiPutForcePathEnd</td>
<td>Force-quits the move instruction in execution.</td>
</tr>
<tr>
<td>8 mpMeiPutSpdOverride</td>
<td>Sets a speed override amount for the move instruction in execution.</td>
</tr>
<tr>
<td>9 mpMeiIncrementMove</td>
<td>Sets an amount of increment move by the interpolation cycle to the manipulator which is executing WAIT instruction.</td>
</tr>
</tbody>
</table>
11 Robot Operation Control Service

The robot operation control service provides the APIs to operate manipulators.

11.1 Increment Value Move Function

This function is provided by `mpExRcsIncrementMove()`, and the manipulator moves for the increment value specified by the API while executing WAIT instruction.

When specifying the increment value by 0.1mm each to y-direction while the interpolation cycle is set to 4msec, the manipulator moves 500 times in 2 seconds and 50mm to y-direction.

![Diagram](image)

11.2 Robot Operation Control API List

<table>
<thead>
<tr>
<th>MotoPlus</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>mpExRcsIncrementMove</code></td>
</tr>
</tbody>
</table>
12 Servo Control

12.1 Function Outline

By using the servo control, the motor speed and the torque data can be retrieved from MotoPlus in real-time, and the motor torque can be controlled directly.

To use the servo control, the MotoFit function (optional) is necessary.

**CAUTION**

- By using the servo control, the motor may not be able to maintain its position and may move in an unexpected direction. Be sure to confirm safety before using the servo control.
- If the MotoPlus application which uses the servo control hangs up, unintended torque values may be issued to the motor, which may cause a hazardous condition. When using the servo control, be sure to use it together with the user watchdog to take preventive measures such as finishing the servo control if the application hangs up.

**12.1 Function Outline**

As the servo control, the functions in the following table are mainly provided.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feedback speed/torque retrieve function</td>
<td>Retrieves the current feedback speed and torque in real-time.</td>
</tr>
<tr>
<td>Torque limit function</td>
<td>Limits the motor torque. (Only for the external axis)</td>
</tr>
<tr>
<td>Torque control function</td>
<td>Controls the motor torque directly. (Only for the external axis)</td>
</tr>
</tbody>
</table>

**NOTE**

The direction (sign) of the torque for the servo control is the same as the direction of the axis rotation. Also, it coincides with the direction of the speed.

However, it may be different from the direction of the torque retrieved by the system monitor API, mpGetTorque().

For mpGetTorque(), the torque’s normal (forward) direction is not the direction of the axis rotation but the direction of the motor’s normal (forward) rotation. The direction of the axis rotation and the direction of the motor’s normal rotation may be reversed. If so, the direction of the torque retrieved by mpGetTorque() is the reverse of the direction of the speed and the torque for the servo control API.
12 Servo Control
12.1 Function Outline

12.1.1 Feedback Speed/Torque Retrieve Function

The feedback speed and torque of all control groups/all axes can be retrieved by `mpSvsGetVelTrqFb()`.

The system monitor APIs `mpGetFBSpeed()` and `mpGetTorque()` can also return the feedback speed and torque, but the servo control API `mpSvsGetVelTrqFb()` is more real-time because the data are updated at the I/O control cycle.

12.1.2 Torque Limit Function/Torque Control Function

The same procedure is used for the torque limit function and the torque control function.

First, as shown in the following flow chart, execute the torque limit/torque control starting API `mpSvsStartTrqLimit()/mpSvsStartTrqCtrl()`. At this time, specify the group and axis for which the torque limit/torque control is executed. Only the external axis can be specified.

Next, set the value of the torque limit/torque control by the setting API `mpSvsSetTrqLimit()/mpSvsSetTrqCtrl()`. The actual torque limit/torque control is started when the setting API is executed for the first time after the starting API is executed.

The setting API can be executed at the desired intervals. Once the value of the torque limit/torque control is set, it is maintained until the setting API is executed next time.

To end the torque limit/torque control, execute the ending API `mpSvsEndTrqLimit()/mpSvsEndTrqCtrl()`. The setting API can be executed even while the move command is being output to the specified group (while in operation), but the starting API and the ending API can be executed only while the move command is not being output to the specified group (while not in operation).

For the details of each starting, setting, or ending API, refer to “YRC1000 OPTIONS INSTRUCTIONS Reference Manual For New Language Environment MotoPlus Extended Version (HW1483602)”.

---

Start the torque limit/torque control. (Execute `mpSvsStartTrqLimit()/mpSvsStartTrqCtrl()`)

Continue the torque limit?

- **Yes**

  Set the value of the torque limit/torque control. (Execute `mpSvsSetTrqLimit()/mpSvsSetTrqCtrl()`)

  End the torque limit/torque control. (Execute `mpSvsEndTrqLimit()/mpSvsEndTrqCtrl()`)

- **No**
12.2 Example of Usage

12.2.1 Detecting Workpiece Position

While monitoring the torque value by mpSvsGetVelTrqFb(), move the external axis towards the workpiece. When the torque value exceeds a certain value, retrieve the feedback position by mpGetFBPulsePos(), and define the position as the workpiece position.

Or,

Set the torque limit on the external axis by using the servo control. After that, while monitoring the speed by mpSvsGetVelTrqFb(), move the external axis towards the workpiece. When the speed falls below a certain value, retrieve the feedback position by mpGetFBPulsePos(), and define the position as the workpiece position.

CAUTION

- While executing the torque limit/torque control, the motor’s position is not controlled and the motor may move in an unexpected direction. Thus, while executing the torque limit/torque control, monitor the feedback position periodically, and end the torque limit/torque control if the feedback position exceeds the specified range.

- If “0” is set as the value of the torque limit/torque control for the axis subject to gravity and the like, the axis moves in the direction of gravity and the like. Also, if the torque command value is too large, the axis may move. Check the current torque by mpSvsGetVelTrqFb() to confirm that the value to be set as the value of the torque limit/torque control is appropriate, and set the value of the torque limit/torque control before starting the torque limit/torque control.

NOTE

The torque limit and the torque control are not automatically ended by the system due to turning OFF of the servo power or an alarm occurrence. (If the servo power is turned OFF, the execution of the torque limit/torque control will be continued when the servo power is turned ON again.)

To end the torque limit/torque control at turning OFF of the servo power or an alarm occurrence, monitor the status of the servo power and the alarm, and execute mpSvsEndTrqLimit()/mpSvsEndTrqCtrl() as necessary.
12.2.2 Holding Workpiece with Unknown Size

While monitoring the torque value by mpSvsGetVelTrqFb(), move the external axis towards the workpiece. When the torque value exceeds a certain value, execute the torque control of the servo control, and then hold the workpiece with a specified force.

12.3 Servo Control API List

<table>
<thead>
<tr>
<th>MotoPlus</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mpSvsGetVelTrqFb</td>
</tr>
<tr>
<td>2</td>
<td>mpSvsStartTrqLimit</td>
</tr>
<tr>
<td>3</td>
<td>mpSvsSetTrqLimit</td>
</tr>
<tr>
<td>4</td>
<td>mpSvsEndTrqLimit</td>
</tr>
<tr>
<td>5</td>
<td>mpSvsStartTrqCtrl</td>
</tr>
<tr>
<td>6</td>
<td>mpSvsSetTrqCtrl</td>
</tr>
<tr>
<td>7</td>
<td>mpSvsEndTrqCtrl</td>
</tr>
<tr>
<td>8</td>
<td>mpSvsForceInit</td>
</tr>
</tbody>
</table>

Retrieves the current feedback speed and torque.

Starts the torque limit.

Sets the torque limit value.

Ends the torque limit.

Starts the torque control.

Sets the torque command value.

Ends the torque control.

Force-quits the servo control.
13 Memory Pool Control

With these APIs, malloc and mfree are executed in the area controlled for MotoPlus so that the system will not be affected. With MotoPlus, the standard functions malloc and mfree cannot be used. Use these APIs. Make sure to use mpFree to free the area allocated by mpMalloc.

13.1 Memory Pool Control API List

<table>
<thead>
<tr>
<th>MotoPlus</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mpMalloc</td>
</tr>
<tr>
<td>2</td>
<td>mpFree</td>
</tr>
</tbody>
</table>

Allocates memory.
Frees memory.
14 File Control

The user can freely create, read, write, and delete files on the SRAM drive and DRAM drive placed on the CMOS of the controller, and also the files on the USB drive placed in front of the CPU circuit board (JANCD-ACP01). The files created on SRAM and USB will be maintained even if the power is turned OFF, but the files created on DRAM will be deleted if the power is turned OFF.

To use the SRAM drive, the memory expanding option (JANCD-AIF01-2E) of the robot I/F circuit board is necessary.

14.1 Rules for File Name and Size

Rules for file names in MotoPlus are shown as follows:

(1) When specifying a file name, make sure to specify a full pathname including a drive letter.

(2) The drive letter is defined as shown below.
   SRAM drive: MP_SRAM_DEV_DOS  ("MPSRAM1:0")
   DRAM drive: MP_DRAM_DEV_DOS  ("MPRAM1:0")
   USB drive 0: MP_USB0_DEV_DOS   ("MPUSB0")
   USB drive 1: MP_USB1_DEV_DOS   ("MPUSB1")

   <Specification method for USB drive>

   Left side USB connector
   Right side USB connector

   MP_USB0_DEV_DOS("MPUSB0")
   MP_USB1_DEV_DOS("MPUSB1")

(3) After the drive letter, use "\" or "/" to show the root directory in the pathname.

(4) The maximum name length of a single file or a single directory excluding the drive letter and pathname is 32 characters.

(5) Full-width characters cannot be used for the name of a file and a directory.

(6) When creating a file or a directory, it is case-sensitive and the input one is reflected as it is.
   For renaming, this rule is also applied to the name after renaming.

(7) A file and a directory cannot be created nor renamed, using the name that differs only in the case to the existing name of a file or a directory.

(8) When specifying a name of a file or a directory except creating and renaming them, it is case-insensitive.
14 File Control

14.1 Rules for File Name and Size

Examples for (6), (7), (8):

mpCreate("MPUSB0/AbCd.txt"); \Rightarrow "AbCd.txt" is created.
(The case is reflected as they are input.)

mpCreate("MPUSB0/ABCD.TXT"); \Rightarrow "ABCD.TXT" is not created.
("AbCd.txt" exists already.)

mpRemove("MPUSB0/ABCD.TXT"); \Rightarrow "AbCd.txt" is deleted.
(The file is deleted with case ignored.)

(9) The maximum length of a pathname including the drive letter is 255 characters.

(10) The drive capacity of each of SRAM and DRAM is 512 KB.

**NOTE**

- Do not create an application in which file reading (mpRead) or file writing (mpWrite) is performed by multiple tasks. File reading or writing must be performed by one task.

  If the controller’s power supply is turned OFF during file reading or writing by multiple tasks, the file on CMOS memory may encounter an unexpected condition.

- Be sure to save the necessary user-defined files on CMOS in an external memory device. By doing this, even if the file system on CMOS memory is damaged, it can be recovered by initializing the user-defined file in the maintenance mode and loading the file saved in the external memory device.

- For file reading (mpRead) and file writing (mpWrite), the stack of the MotoPlus tasks which executed APIs is used. Therefore, the data size for file reading and file writing should be specified not to exceed the maximum stack size (40 Kbyte), including the size of execution codes and variables, etc. of the MotoPlus task.

  If the specified data size is over the maximum stack size of the MotoPlus task, YRC1000 or the MotoPlus application doesn’t work properly.

  (About the phenomena, refer to chapter 16.8 “Phenomena during Debugging, Potential Coding Problems, and Examination Methods”.)

  (e.g.) mpRead(fd, buffer, 60*1024); : data size: 60Kbyte (over the stack size)

  In this case, set as follows.

  mpRead(fd, buffer, 30*1024);
  mpRead(fd, buffer, 30*1024);
14.2 Rules for USB Connector and USB Memory Stick

Followings are the rules of the USB connector on the CPU circuit board (JANCD-ACP01) and the USB memory stick to be installed.

1. Prohibition of insertion/removal of the USB memory stick during control power ON
   The device recognition process is executed when the USB memory stick is inserted. Do not insert or remove the USB memory stick while the control power supply is ON. Failure to observe this rule may affect the operation of the manipulator (cycle time).

2. Prohibition of disconnection of the control power and insertion/removal of USB memory stick during file access
   Do not disconnect the control power or insert/remove the USB memory stick during file access.
   Failure to observe this rule may breakdown the FAT.

3. User defined file unsupported
   Selecting [FILE] → [INITIALIZE] → [USER DEFINED FILE] of the external memory function doesn’t support file operation (load, save, and initialize) to the USB memory stick.

4. Access speed to the USB memory stick
   The access speed to the USB memory stick is slow in comparison with the access speed to the SRAM drive. When changing the access destination of an API from the SRAM drive to the USB memory stick, check if the API is synchronized with other process properly.

5. Operating temperature range of USB memory stick
   Use a USB memory stick that is guaranteed to work in the range of temperature of the YRC1000 controller.

6. USB memory stick’s falling off by controller vibration
   Prevent the USB memory stick from falling off by the vibration of the controller.
   (Countermeasure example)
   • Fix the USB memory stick with jigs not to fall off, etc.

7. USB connector in front of the CPU circuit board (JANCD-ACP01)
   The USB connector in front of the CPU circuit board (JANCD-ACP01) accepts only the USB memory stick.
   Do not connect a USB hub or other USB devices.

8. Capacity of USB memory stick
   The capacity of the memory stick must be 4Gbyte or less.
14.3 Access Right When Opening a File

With file control APIs, the following access rights can be set when creating or opening a file.

<table>
<thead>
<tr>
<th>Access right</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O_RDONLY</td>
<td>0</td>
<td>Read only</td>
</tr>
<tr>
<td>O_WRONLY</td>
<td>1</td>
<td>Write only</td>
</tr>
<tr>
<td>O_RDWR</td>
<td>2</td>
<td>Read and write</td>
</tr>
<tr>
<td>O_CREAT</td>
<td>0x0200</td>
<td>Creates and opens a file (When executing mpOpen, use this in combination with one of the above three access rights.)</td>
</tr>
</tbody>
</table>

These access rights are set for each file descriptor (fd) allocated when creating a file (mpCreate) or opening a file (mpOpen).

When closing a file (mpClose is executed), fd becomes invalid, and the access rights are revoked.

14.4 Available Task Priority

Only the normal task priority (MP_PRI_TIME_NORMAL) is available for the file control APIs.

If a file is read from the task with the priority higher than that, the following operation will be performed according to the setting of the parameter S2C1101.

This setting can be changed by “FileControl TASK PRY. LIMIT” in the maintenance mode.

<table>
<thead>
<tr>
<th>S2C1101</th>
<th>FileControl TASK PRY. LIMIT</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>LIMITED</td>
<td>The API returns an error.</td>
</tr>
<tr>
<td>1</td>
<td>AUTO</td>
<td>Temporarily changes the task priority to MP_PRI_TIME_NORMAL and executes the API.</td>
</tr>
</tbody>
</table>

For details on how to change “FileControl TASK PRY. LIMIT”, refer to "Chap. 5.3 Setting of File Control Task Priority Limit” in “YRC1000 OPTIONS INSTRUCTIONS User’s Manual For New Language Environment MotoPlus (HW1483600)".
## 14.5 File Control API List

<table>
<thead>
<tr>
<th>MotoPlus</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mpCreate Creates a new file.</td>
</tr>
<tr>
<td>2</td>
<td>mpOpen Opens a file.</td>
</tr>
<tr>
<td>3</td>
<td>mpRemove Deletes a file.</td>
</tr>
<tr>
<td>4</td>
<td>mpClose Closes a file.</td>
</tr>
<tr>
<td>5</td>
<td>mpRename Renames a file.</td>
</tr>
<tr>
<td>6</td>
<td>mpRead Reads data from a file.</td>
</tr>
<tr>
<td>7</td>
<td>mpWrite Writes data to a file.</td>
</tr>
<tr>
<td>8</td>
<td>mpIoctl Controls a device.</td>
</tr>
<tr>
<td>9</td>
<td>mpLseek Moves a file pointer.</td>
</tr>
<tr>
<td>10</td>
<td>mpFstat Retrieves file status information. (Specified by fd)</td>
</tr>
<tr>
<td>11</td>
<td>mpStat Retrieves file status information. (Specified by a file name)</td>
</tr>
<tr>
<td>12</td>
<td>mpOpendir Opens a directory.</td>
</tr>
<tr>
<td>13</td>
<td>mpReaddir Reads one entry from the opened directory.</td>
</tr>
<tr>
<td>14</td>
<td>mpRewinddir Resets a directory descriptor.</td>
</tr>
<tr>
<td>15</td>
<td>mpClosedir Closes a directory.</td>
</tr>
<tr>
<td>16</td>
<td>mpCreatedir Creates a new directory. (Only when specifying the USB drive, this API is usable.)</td>
</tr>
<tr>
<td>17</td>
<td>mpRemovedir Deletes a directory. (Only when specifying the USB drive, this API is usable.)</td>
</tr>
<tr>
<td>18</td>
<td>mpRenamedir Renames a directory. (Only when specifying the USB drive, this API is usable.)</td>
</tr>
</tbody>
</table>
15 Existing File Control

15.1 Existing File Access API List

With these APIs, the JOBs, setting files, etc. are accessed, read out to the RAM drive or the USB drive, and output in text format. In addition, a file on the RAM drive and the USB drive can be written in the system.

To use the SRAM drive, the memory expanding option (JANCD-AIF01-2E) of the robot I/F circuit board is necessary.

For details on rules for the file name, etc. refer to chapter 14.1 “Rules for File Name and Size”.

For details about access to the file on the RAM drive, refer to chapter 14 “File Control”.

15.1 Existing File Access API List

<table>
<thead>
<tr>
<th>MotoPlus</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mpLoadFile</td>
</tr>
<tr>
<td>2</td>
<td>mpSaveFile</td>
</tr>
<tr>
<td>3</td>
<td>mpRefreshFileList</td>
</tr>
<tr>
<td>4</td>
<td>mpGetFileCount</td>
</tr>
<tr>
<td>5</td>
<td>mpGetFileName</td>
</tr>
<tr>
<td>6</td>
<td>mpFdWriteFile</td>
</tr>
<tr>
<td>7</td>
<td>mpFdReadFile</td>
</tr>
<tr>
<td>8</td>
<td>mpFdGetJobList</td>
</tr>
</tbody>
</table>

Calling mpLoadFile, mpSaveFile, mpFdWriteFile, or mpFdReadFile to load or save an existing file prohibits editing operation to prevent conflicts in the system’s internal information. Thus, the editing mode of the JOB, variable, or I/O is initialized, and the programming pendant cannot be temporarily operated. Therefore, so as not to interfere with editing operation by using the programming pendant, use these APIs in the teach mode only when necessary, and do not load or save successively.
16 Programming

16.1 Rules

The rules for source program creation are as follows. The file name must be 32 half-size characters or less, full-size characters are acceptable, and case-sensitive. An extension of ".c" must be used.

16.2 C Coding Rules

Conforms to the standard ANSI-C coding rules.

16.3 Program Execution

For an application program, the file "xxxxxx.out" is automatically loaded when the power is turned ON. Then, mpUsrRoot() function in it will be considered as a task and automatically started. Therefore, an application program must have only one mpUsrRoot(). At the end of mpUsrRoot(), mpExitUsrRoot; must be described. This is a macro which terminates the mpUsrRoot task.

16.4 MotoPlus Coding Rules

1. MotoPlus.h must be included at the head of a program. This include file contains all the header information of the services to be provided.

2. Set void mpUsrRoot (int arg1, int arg2, int arg3, int arg4, int arg5, int arg6, int arg7, int arg8, int arg9, int arg10) as the entry function of an application. Do not specify returned values. Describe mpExitUsrRoot; at the end.

```c
int mpUsrRoot (int arg1, int arg2, int arg3, int arg4, int arg5, int arg6, int arg7, int arg8, int arg9, int arg10)
{
    /* The variable “argc” indicates the number of arguments.
       .
       .
       mpExitUsrRoot;  // Be sure to describe to complete
}
```

The task “mpUsrRoot” has a high priority to start other application tasks and initialize the entire application quickly. Due to its high task priority, if a time-consuming process (100 microseconds or longer) is done in this task, the processing time for the robot control becomes insufficient. Then the system alarm may occur or the programming pendant or the HOLD button may freeze. Thus, as the sample program (refer to chapter 17.3 “Task Control Sample Program”), make sure to complete the task “mpUsrRoot” after starting up another application or creating a semaphore.

---

**NOTE**

The task “mpUsrRoot” has a high priority to start other application tasks and initialize the entire application quickly. Due to its high task priority, if a time-consuming process (100 microseconds or longer) is done in this task, the processing time for the robot control becomes insufficient. Then the system alarm may occur or the programming pendant or the HOLD button may freeze. Thus, as the sample program (refer to chapter 17.3 “Task Control Sample Program”), make sure to complete the task “mpUsrRoot” after starting up another application or creating a semaphore.
16.5 Restrictions on Programming

16.5.1 Usable ANSI-C Standard Functions

The usable functions from the ANSI-C standard functions are listed below. The other functions are not supported. For example, open and close of the memory allocation function and the file control function are not supported. Memory allocation may cause a memory leak and the system may go down due to insufficient memory. Also, if large memory is taken by using a standard function, the system may go down. Regarding the file control function, standard OPEN and CLOSE cannot be done because the SD file of the main CPU has a special access method to prevent the file corruption when the power is turned OFF. Also, the I/O function waiting for input (getc, getchar, gets) cannot be used because it stops the task.

- **ctype.h** Character operation
  - isalnum  isalpha  iscntrl  isdigit  isgraph  islower  isprint  ispunct  isspace  isupper  isxdigit  tolower  toupper
- **locale.h** Locale-specific operation
  - localeconv  setlocale
- **math.h** Mathematics
  - acos  asin  atan  atan2  ceil  cos  cosh  exp  floor  fabs  fmod  frexp  ldexp  log  log10  modf  pow  sin  sinh  sqrt  tan  tanh
- **stdarg.h** Variable number of real arguments
  - va_arg  va_end  va_start
- **stdio.h** Input/Output
  - printf  putc  putchar  puts  sprintf  vfprintf  vprintf  vsprintf
- **stdlib.h** General utility
  - abs  atof  atoi  atol  bsearch  div  exit  getenv  labs  ldiv  qsort  rand  srand  strtod  strtol  strtoul  system
- **string.h** String operation
  - memchr  memcmp  memmove  memset  strcat  strchr  strcmp  strcoll  strcspn  strerror  strlen  strncat  strncmp  strncpy  strpbrk  strrev  strspn  strstr  strtok  strxfrm
## 16.6 Note for Programming

When programming in C language, the pointer variable which stores the memory address can be used. If the pointer variable is set incorrectly, the system memory area may be rewritten.

If the system memory area is rewritten, critical problems occur, e.g., the software hangs up (the hang-up status is detected by the watchdog check function, then the servo power turns OFF and the system shuts down), the robot stops its operation due to an alarm, or the programming pendant becomes inoperable.

### 16.5.2 Task Priority and Usable API

Depending on the task priority, only limited APIs can be used. The task priority and usable APIs are listed in the following table. For high-priority tasks, use only the APIs listed below because the other APIs require long processing time.

<table>
<thead>
<tr>
<th>Priority</th>
<th>Usable API</th>
</tr>
</thead>
<tbody>
<tr>
<td>For the notification task of the I/O control cycle</td>
<td>mpReadIO, mpWriteIO, mpGetUserVars, mpPutUserVars, mpMsgQSend, mpErrMsgQSnd, mpSemGive, mpTaskDelay, mpClkAnnounce, mpStopWatchCreate, mpStopWatchDelete, mpStopWatchStart, mpStopWatchStop, mpStopWatchLap, mpStopWatchReset, mpStopWatchGetTime, mpStopWatchGetLapNum, mpStopWatchGetLapTime, mpStopWatchGetAliveLapNo, mpUsrWdogStart, mpUsrWdogClear, mpSvsGetVelTrqFb, mpSvsSetTrqLimit, mpSvsSetTrqCtrl, mpCtrlGrpld2GrpNo</td>
</tr>
<tr>
<td>For the notification task of the interpolation cycle</td>
<td>Above APIs (APIs usable for the notification task of the I/O control cycle), mpMeiGetJobExecTask, mpMeiGetInterpolation, mpMeiGetExecControlGroup, mpMeiPutCorrPath, mpMeiPutForcePathEnd, mpMeiPutSpdOverride, mpMeiIncrementMove, mpExRcsIncrementMove</td>
</tr>
<tr>
<td>High priority</td>
<td>Above APIs (APIs usable for the notification task of the interpolation cycle), mpGetVarData, mpGetSVarData, mpGetPosVarData, mpPutVarData, mpPutSVarData, mpPutPosVarData, mpSocket, mpListen, mpAccept, mpBind, mpConnect, mpRecv, mpRecvfrom, mpSend, mpSendto, mpClose, mpHtonl, mpHtons, mpNtohl, mpNtohs, mpInetAddr, mpInetNtoa, mpInetNtob, mpGetsockname, mpGetpeername, mpSetsockopt, mpIoctl, mpSelect, mpRsOpen, mpRsClose, mpRsSend, mpRsRecv, mpConvAxesToCartPos, mpConvCartPosToAxes, mpConvPulseToAngle, mpConvAngleToPulse, mpConvFBPulseToPulse, mpMakeFrame, mpInvFrame, mpRotFrame, mpMulFrame, mpZYXeulerToFrame, mpFrameToZYXeuler, mpCrossProduct, mpInnerProduct</td>
</tr>
</tbody>
</table>
16.7 Operation Control Sample Program

16.7.1 Operation Control Sample Program

16.7.1.1 Outline

The following explains the sample program of the case making a manipulator pick up a workpiece and store it into a tray.

Store the position data of 0 to 5 in the figure above to the position variables in advance. This program reads these position variables and sends the target.

16.7.1.2 Sample Program

/* mp_main.c - MotoPlus Test Application for Real Time Process */
#include "motoPlus.h"

void mpTask1();
void mpTask2();

//GLOBAL DATA DEFINITIONS
int nTaskID1;
int nTaskID2;

static int handOn(void);
static int handOff(void);

void mpUsrRoot(int arg1, int arg2, int arg3, int arg4, int arg5,
int arg6, int arg7, int arg8, int arg9, int arg10)
{
    //TODO: Add additional initialization routines.

    //Creates and starts a new task in a separate thread of execution.
//All arguments will be passed to the new task if the function prototype will accept them.
nTaskID1 = mpCreateTask(MP_PRI_TIME_NORMAL, MP_STACK_SIZE, (FUNCPTR)mpTask1, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10);

nTaskID2 = mpCreateTask(MP_PRI_TIME_NORMAL, MP_STACK_SIZE, (FUNCPTR)mpTask2, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10);

//Ends the initialization task.
mpExitUsrRoot;

void mpTask1(void)
{
    int rc;
    int i;
    int id;
    int timeout, delay;
    int grpNo;
    MP_SPEED spd;
    MP_TARGET target;
    MP_USR_VAR_INFO varInfo;

    // timeout = 30[sec]
    timeout = 30000 / mpGetRtc();
    delay = 1000 / mpGetRtc();

    //TODO: Add the code for this task
    FOREVER
    {
        memset(&varInfo, 0, sizeof(varInfo));
        varInfo.var_type = MP_VAR_B;
        varInfo.var_no = 0;

        // waiting for start event.
        printf("\n");
        printf("... waiting for start event.\n");
        while (varInfo.val.b == 0)
        {
            // waiting for start event.
            printf("\n");
            printf("... waiting for start event.\n");
        }
    }
mpTaskDelay(delay);
if ((rc = mpGetUserVars(&varInfo)) < 0)
{
    break;
}
}

if (rc < 0)
{
    printf("%d = mpGetUserVars(MP_VAR_B)n", rc);
    continue;
}

varInfo.val.b = 0; // clear event.
mpPutUserVars(&varInfo);

// get group No.
if (((grpNo = mpCtrlGrpId2GrpNo(MP_R1_GID)) < 0)
{
    printf("%d = mpCtrlGrpId2GrpNo(MP_R1_GID)n", grpNo);
    continue;
}

// initialize motion control.
mpMotStop(0);
mpMotTargetClear(0x0f, 0);

// send targets.
if (((rc = mpMotSetCoord(grpNo, MP_BASE_TYPE, 0)) < 0)
{
    printf("%d = mpMotSetCoord()n", rc);
    continue;
}

memset(&spd, 0, sizeof(spd));
spd.v = 500; // 50.0[mm/sec]
if (((rc = mpMotSetSpeed(grpNo, &spd)) < 0)
{
    printf("%d = mpMotSetSpeed()n", rc);
    continue;
for (i = 1; i < 6; i++)
{
    printf("send No.%d target.\n", i);
    memset(&target, 0, sizeof(target));
    target.id = i;
    target.intp = MP_MOVL_TYPE;
    varInfo.var_type = MP_VAR_P;
    varInfo.var_no = i;
    if ((rc = mpGetUserVars(&varInfo)) < 0)
    {
        printf("%d = mpGetUserVars(MP_VAR_P)\n", rc);
        break;
    }
    else
    {
        memcpy(target.dst.joint, varInfo.val.p.data, sizeof(target.dst.joint));
    }
    if ((rc = mpMotTargetSend((1 << grpNo), &target, timeout)) < 0)
    {
        printf("%d = mpMotTargetSend()\n", rc);
        break;
    }
}
if (rc < 0)
{
    printf("sending target failed.\n");
    continue;
}

handOff(); // open the hand.
printf("\n");
printf("count down ...\n");
for (i = 5; i > 0; i--)
{
    printf("%d\n", i);
mpTaskDelay(delay);

// motion start!
printf("motion start!\n");
if ((rc = mpMotStart(0)) < 0)
{
    printf("%d = mpMotStart()\n", rc);
    continue;
}

// ... waiting for target id = 1.
id = 1;
printf("... waiting for target id = %d.\n", id);
if ((rc = mpMotTargetReceive(grpNo, id, NULL, timeout, 0)) < 0)
{
    printf("%d = mpMotTargetReceive()\n", rc);
    continue;
}
printf("arriving to target id = %d.\n", id);
printf("close the hand!\n");
handOn(); // close the hand.

// ... waiting for target id = 4.
id = 4;
printf("... waiting for target id = %d.\n", id);
if ((rc = mpMotTargetReceive(grpNo, id, NULL, timeout, 0)) < 0)
{
    printf("%d = mpMotTargetReceive()\n", rc);
    continue;
}
printf("arriving to target id = %d.\n", id);
printf("open the hand!\n");
handOff(); // open the hand.

// ... waiting for target id = 5.
id = 5;
printf("... waiting for target id = %d.\n", id);
if ((rc = mpMotTargetReceive(grpNo, id, NULL, timeout, 0)) < 0)
\{ 
    printf("%d = mpMotTargetReceive()\n", rc);
    continue;
\}
printf("arriving to target id = %d.\n", id);

printf("\n");
printf("return to home position count down ...\n");
for (i = 5; i > 0; i--)
{ 
    printf("%d\n", i);
    mpTaskDelay(delay);
}

// send home position.
if ((rc = mpMotSetCoord(grpNo, MP_ANGLE_TYPE, 0)) < 0)
{ 
    printf("%d = mpMotSetCoord()\n", rc);
    continue;
}
memset(&spd, 0, sizeof(spd));
spd.vj = 1000; // 10.00[%]
if ((rc = mpMotSetSpeed(grpNo, &spd)) < 0)
{ 
    printf("%d = mpMotSetSpeed()\n", rc);
    continue;
}

printf("send home position.\n");
memset(&target, 0, sizeof(target));
target.id = id = 0;
target.intp = MP_MOVJ_TYPE;
varInfo.var_type = MP_VAR_P;
varInfo.var_no = 0;
if ((rc = mpGetUserVars(&varInfo)) < 0)
{ 
    printf("%d = mpGetUserVars(MP_VAR_P)\n", rc);
    continue;
}
16 Programming
16.7 Operation Control Sample Program

```c
} else {
    memcpy(target.dst.joint, varInfo.val.p.data, sizeof(target.dst.joint));
}

if ((rc = mpMotTargetSend((1 << grpNo), &target, timeout)) < 0) {
    printf("%d = mpMotTargetSend()\n", rc);
    break;
}

if ((rc = mpMotTargetReceive(grpNo, id, NULL, timeout, 0)) < 0) {
    printf("%d = mpMotTargetReceive()\n", rc);
    continue;
}
printf("complete!\n");
mpTaskDelay(delay);
}

static int handOn(void) {
    MP_IO_DATA req;
    req.ulAddr = 10010;
    req.ulValue = 1;
    return (mpWriteIO(&req, 1));
}

static int handOff(void) {
    MP_IO_DATA req;
    req.ulAddr = 10010;
    req.ulValue = 0;
    return (mpWriteIO(&req, 1));
}

void mpTask2(int arg1, int arg2)
```
{  
    //TODO: Add the code for this task  
}
16.7.2 Independent Control Sample Program

16.7.2.1 Outline

The following is an operation program sample of two manipulators.

R1 and R2 start operation to (1) (in the figure above) at the same time, and arrive to the target position at the same time. Next, each R1 and R2 operates from (2) to (4) independently. Finally, they start operation to (1) again at the same time, and arrive to the target position at the same time.

The position, which is registered with position variables in advance, is loaded and considered to be the target position of (1) to (4) in the figure above.
/* mp_main.c - MotoPlus Test Application for Real Time Process */
#include "motoPlus.h"

#include "motoPlus.h"

#define GROUP_NUM                  (2) // R1, R2
#define TARGET_NUM                 (5)

void mpTask1();
void mpTask2();

//GLOBAL DATA DEFINITIONS
int nTaskID1;
int nTaskID2;

enum {
    START_TARGET,
    POS1_TARGET,
    POS2_TARGET,
    POS3_TARGET,
    END_TARGET,
};

void mpUsrRoot(int arg1, int arg2, int arg3, int arg4, int arg5,
                int arg6, int arg7, int arg8, int arg9, int arg10)
{
    //TODO: Add additional initialization routines.

    //Creates and starts a new task in a separate thread of execution.
    //All arguments will be passed to the new task if the function
    //prototype will accept them.
    nTaskID1 = mpCreateTask(MP_PRI_TIME_NORMAL,
                            MP_STACK_SIZE, (FUNCPTR)mpTask1,
                            arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10);
    nTaskID2 = mpCreateTask(MP_PRI_TIME_NORMAL,
                            MP_STACK_SIZE, (FUNCPTR)mpTask2,
                            arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10);

    //Ends the initialization task.
    mpExitUsrRoot;
}

16 Programming
16.7 Operation Control Sample Program

16.7.2.2 Sample Program
void mpTask1(void)
{
    //TODO: Add the code for this task
    int rc;
    int i;
    int grp;
    int grpNoR1, grpNoR2;
    int timeout, delay;
    MP_SPEED spd;
    MP_USR_VAR_INFO varInfo;
    MP_TARGET target[TARGET_NUM][GROUP_NUM];

    // timeout = 60[sec]
    timeout = 60000 / mpGetRtc();
    delay = 1000 / mpGetRtc();

    FOREVER
    {
        memset(&varInfo, 0, sizeof(varInfo));
        varInfo.var_type = MP_VAR_B;
        varInfo.var_no = 0;

        // waiting for start event.
        printf("\n");
        printf("... waiting for start event.\n");
        while (varInfo.val.b == 0)
        {
            mpTaskDelay(delay);
            if ((rc = mpGetUserVars(&varInfo)) < 0)
            {
                break;
            }
        }

        if (rc < 0)
        {
            printf("%d = mpGetUserVars(MP_VAR_B)\n", rc);
            continue;
        }
    }
}
varInfo.val.b = 0; // clear event.
mpPutUserVars(&varInfo);

if ((grpNoR1 = mpCtrlGrpId2GrpNo(MP_R1_GID)) < 0)
{
    printf("%d = mpCtrlGrpId2GrpNo(MP_R1_GID)\n", grpNoR1);
    continue;
}

if ((grpNoR2 = mpCtrlGrpId2GrpNo(MP_R2_GID)) < 0)
{
    printf("%d = mpCtrlGrpId2GrpNo(MP_R2_GID)\n", grpNoR2);
    continue;
}

grp = (1 << grpNoR1) | (1 << grpNoR2);

// initialize motion control.
mpMotStop(0);
mpMotTargetClear(0x0f, 0);

// set speed
memset(&spd, 0, sizeof(spd));
spd.v = 230; // 23.0 mm/s
if ((rc = mpMotSetSpeed(grpNoR1, &spd)) < 0)
{
    printf("%d = mpMotSetSpeed(R1)\n", rc);
    continue;
}

spd.v = 100; // 10.0 mm/s
if ((rc = mpMotSetSpeed(grpNoR2, &spd)) < 0)
{
    printf("%d = mpMotSetSpeed(R2)\n", rc);
    continue;
}

// set coordinate system.
16 Programming
16.7 Operation Control Sample Program

```c
if ((rc = mpMotSetCoord(grpNoR1, MP_BASE_TYPE, 0)) < 0)
{
    printf("%d = mpMotSetCoord(R1)\n", rc);
    continue;
}

if ((rc = mpMotSetCoord(grpNoR2, MP_BASE_TYPE, 0)) < 0)
{
    printf("%d = mpMotSetCoord(R2)\n", rc);
    continue;
}

// get destination position from P variable.
memset(target, 0, sizeof(target));
for (i = 0; i < TARGET_NUM; i++)
{
    // R1
    target[i][grpNoR1].id = (i + 1);
target[i][grpNoR1].intp = MP_MOVL_TYPE;

    varInfo.var_type = MP_VAR_P;
    varInfo.var_no = i;
    if ((rc = mpGetUserVars(&varInfo)) < 0)
    {
        printf("%d = mpGetUserVars()\n", rc);
        break;
    }
    else
    {
        memcpy(target[i][grpNoR1].dst.joint, varInfo.val.p.data,
               sizeof(varInfo.val.p.data));
    }

    // R2
    target[i][grpNoR2].id = (i + 10);
target[i][grpNoR2].intp = MP_MOVL_TYPE;

    varInfo.var_type = MP_VAR_P;
    varInfo.var_no = (i + 10);
```
if ((rc = mpGetUserVars(&varInfo)) < 0)
{
    printf("%d = mpGetUserVars()\n", rc);
    break;
}
else
{
    memcpy(target[i][grpNoR2].dst.joint, varInfo.val.p.data, sizeof(varInfo.val.p.data));
}

if (rc < 0)
{
    continue;
}

// set task number.
if (((rc = mpMotSetTask(grpNoR1, 0)) < 0)
{
    printf("%d = mpMotSetTask(R1)\n", rc);
    continue;
}

if (((rc = mpMotSetTask(grpNoR2, 0)) < 0)
{
    printf("%d = mpMotSetTask(R2)\n", rc);
    continue;
}

// send target.
if (((rc = mpMotTargetSend(grp, target[START_TARGET], timeout)) < 0)
{
    printf("%d = mpMotTargetSend(%d)\n", rc, START_TARGET);
    continue;
}

// set task number (R1: task1, R2: task0).
if (((rc = mpMotSetTask(grpNoR1, 1)) < 0)
16 Programming
16.7 Operation Control Sample Program

```c
{
    printf("%d = mpMotSetTask(R1)\n", rc);
    continue;
}

// send independent control targets.
if ((rc = mpMotTargetSend(grp, target[POS1_TARGET], timeout)) < 0)
{
    printf("%d = mpMotTargetSend(%d)\n", rc, POS1_TARGET);
    continue;
}

if ((rc = mpMotTargetSend(grp, target[POS2_TARGET], timeout)) < 0)
{
    printf("%d = mpMotTargetSend(%d)\n", rc, POS2_TARGET);
    continue;
}

// send target for only R2 group.
if ((rc = mpMotTargetSend((1 << grpNoR2), target[POS3_TARGET], timeout)) < 0)
{
    printf("%d = mpMotTargetSend(%d)\n", rc, POS3_TARGET);
    continue;
}

// reset task number.
if ((rc = mpMotSetTask(grpNoR1, 0)) < 0)
{
    printf("%d = mpMotSetTask(R1)\n", rc);
    continue;
}

// send target.
if ((rc = mpMotTargetSend(grp, target[END_TARGET], timeout)) < 0)
{
    printf("%d = mpMotTargetSend(%d)\n", rc, END_TARGET);
    continue;
}
```
printf("sending target is complete!\n");

// motion start!
printf("motion start!\n");
if ((rc = mpMotStart(0)) < 0)
{
    printf("%d = mpMotStart()\n", rc);
    continue;
}

if ((rc = mpMotTargetReceive(grpNoR1, TARGET_NUM, NULL, timeout, 0)) < 0)
{
    printf("%d = mpMotTargetReceive(R1)\n", rc);
    continue;
}

printf("complete!\n");
mpTaskDelay(delay);
}

void mpTask2(int arg1, int arg2)
{
    //TODO: Add the code for this task
}
16.7.3 Coordinated Control Sample Program

16.7.3.1 Outline

The following explains about a program sample that the manipulator moves from end to end of the work while coordinating with the station.

This program reads the position which is registered in advance as position variables, and the manipulators operate while regarding the variables as motion starting position or arrival position.
/* mp_main.c - MotoPlus Test Application for Real Time Process */
#include "motoPlus.h"

//
#define GROUP_NUM               (2) // R1, S1
#define TARGET_NUM              (2)

//
void mpTask1();
void mpTask2();

//GLOBAL DATA DEFINITIONS
int nTaskID1;
int nTaskID2;

void mpUsrRoot(int arg1, int arg2, int arg3, int arg4, int arg5, int arg6,
                int arg7, int arg8, int arg9, int arg10)
{
    //TODO: Add additional initialization routines.

    //Creates and starts a new task in a separate thread of execution.
    //All arguments will be passed to the new task if the function
    //prototype will accept them.
    nTaskID1 = mpCreateTask(MP_PRI_TIME_NORMAL,
                            MP_STACK_SIZE, (FUNCPTR)mpTask1,
                            arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10);
    nTaskID2 = mpCreateTask(MP_PRI_TIME_NORMAL,
                            MP_STACK_SIZE, (FUNCPTR)mpTask2,
                            arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10);

    //Ends the initialization task.
    mpExitUsrRoot;
}

void mpTask1(void)
{
    //TODO: Add the code for this task
    int rc;
    int i;
    int grp;
int grpNoR1, grpNoS1;
int timeout, delay;
MP_SPEED spd;
MP_TARGET target[TARGET_NUM][GROUP_NUM];
MP_USR_VAR_INFO varInfo;

// timeout = 60[sec]
timeout = 60000 / mpGetRtc();
delay = 1000 / mpGetRtc();

FOREVER
{
    memset(&varInfo, 0, sizeof(varInfo));
    varInfo.var_type = MP_VAR_B;
    varInfo.var_no = 0;

    // waiting for start event.
    printf("\n");
    printf("... waiting for start event.\n");
    while (varInfo.val.b == 0)
    {
        mpTaskDelay(delay);
        if ((rc = mpGetUserVars(&varInfo)) < 0)
        {
            break;
        }
    }

    if (rc < 0)
    {
        printf("%d = mpGetUserVars(MP_VAR_B)\n", rc);
        continue;
    }

    varInfo.val.b = 0; // clear event.
    mpPutUserVars(&varInfo);

    // initialize motion control.
    mpMotStop(0);
    mpMotTargetClear(0x0f, 0);
if (((grpNoR1 = mpCtrlGrpId2GrpNo(MP_R1_GID)) < 0)
{
    printf("%d = mpCtrlGrpId2GrpNo(MP_R1_GID)\n", grpNoR1);
    continue;
}

if (((grpNoS1 = mpCtrlGrpId2GrpNo(MP_S1_GID)) < 0)
{
    printf("%d = mpCtrlGrpId2GrpNo(MP_S1_GID)\n", grpNoS1);
    continue;
}

grp = (1 << grpNoR1) | (1 << grpNoS1);

// reset coordinated motion.
if (((rc = mpMotResetSync(grpNoR1)) < 0)
{
    printf("%d = mpMotResetSync(R1)\n", rc);
    continue;
}

// set speed.
memset(&spd, 0, sizeof(spd));
spd.v = 230; // 23.0 mm/s
if (((rc = mpMotSetSpeed(grpNoR1, &spd)) < 0)
{
    printf("%d = mpMotSetSpeed(R1)\n", rc);
    continue;
}

memset(&spd, 0, sizeof(spd));
spd.vj = 5000; // 50.00 %
if (((rc = mpMotSetSpeed(grpNoS1, &spd)) < 0)
{
    printf("%d = mpMotSetSpeed(S1)\n", rc);
    continue;
}
/ set coordinate system,
if ((rc = mpMotSetCoord(grpNoR1, MP_BASE_TYPE, 0)) < 0)
{
    printf("%d = mpMotSetCoord(R1)\n", rc);
    continue;
}

if ((rc = mpMotSetCoord(grpNoS1, MP_PULSE_TYPE, 0)) < 0)
{
    printf("%d = mpMotSetCoord(S1)\n", rc);
    continue;
}

// get destination position from P variable (or EX variable).
memset(target, 0, sizeof(target));
for (i = 0; i < TARGET_NUM; i++)
{
    // R1
    target[i][grpNoR1].id = (i + 1);
    target[i][grpNoR1].intp = MP_MOVL_TYPE;
    varInfo.var_type = MP_VAR_P;
    varInfo.var_no = i;
    if ((rc = mpGetUserVars(&varInfo)) < 0)
    {
        printf("%d = mpGetUserVars()\n", rc);
        break;
    }
    else
    {
        memcpy(target[i][grpNoR1].dst.joint, varInfo.val.p.data,
               sizeof(varInfo.val.p.data));
    }

    // S1
    target[i][grpNoS1].id = (i + 10);
    target[i][grpNoS1].intp = MP_MOVJ_TYPE;
    varInfo.var_type = MP_VAR_EX;
varInfo.var_no = i;
if ((rc = mpGetUserVars(&varInfo)) < 0)
{
    printf("%d = mpGetUserVars\n", rc);
    break;
}
else
{
    memcpy(target[i][grpNoS1].dst.joint,
        varInfo.val.ex.data, sizeof(varInfo.val.ex.data));
}
}
if (rc < 0)
{
    continue;
}

// send target.
i = 0;
if ((rc = mpMotTargetSend(grp, target[i], timeout)) < 0)
{
    printf("%d = mpMotTargetSend(%d)\n", rc, i);
    continue;
}

// set coordinated motion.
if ((rc = mpMotSetSync(grpNoR1, grpNoS1, 0)) < 0)
{
    printf("%d = mpMotSetSync\n", rc);
    continue;
}

// send coordinated motion target.
i++;
if ((rc = mpMotTargetSend(grp, target[i], timeout)) < 0)
{
    printf("%d = mpMotTargetSend(%d)\n", rc, i);
    continue;
printf("sending target is complete!\n");

// motion start!
printf("motion start!\n");
if ((rc = mpMotStart(0)) < 0)
{
    printf("%d = mpMotStart()\n", rc);
    continue;
}

if ((rc = mpMotTargetReceive(grpNoR1, TARGET_NUM, NULL, timeout, 0)) < 0)
{
    printf("%d = mpMotTargetReceive()\n", rc);
    continue;
}

printf("complete!\n");
mpTaskDelay(delay);
}

void mpTask2(int arg1, int arg2)
{
    //TODO: Add the code for this task
}
16.8 Phenomena during Debugging, Potential Coding Problems, and Examination Methods

16.8.1 Programming Pendant Display Freezes with “Starting up system Start Online Process”, and YRC1000 Does Not Start

- **Cause 1:** A local variable which exceeds the stack size is defined, and it is accessed (written) at start-up.
  
  If the 7segLED of the AIF01 circuit board is C. (with the dot blinking), it is likely that the stack area of the task is exceeded.

  Delete the MotoPlus application in the MAINTENANCE mode, and check whether the YRC1000 starts normally.

  If it starts normally, it is likely that the MotoPlus application program has a problem.

  The stack size is 20 Kbyte per task. Make sure that no local variable defined in each function of the MotoPlus application exceeds the stack size.

- **Cause 2:** In a high-priority task with the task priority other than the normal priority (MP_PRI_TIME_NORMAL), the changes in the flag status is performed in “while loop” with the endless-wait status without delaying the task.

  Delete the MotoPlus application in the MAINTENANCE mode, and check whether the YRC1000 starts normally.

  If it starts normally, it is likely that the MotoPlus application program has a problem.

  If the 7segLED of the AIF01 circuit board is D. (with the dot blinking), check whether the application is in the endless-wait status without using mpTaskDelay() in the high-priority task of the application.

16.8.2 MotoPlus Application Is Seemingly Not Running

- **Cause 1:** A local variable which exceeds the stack size is defined and accessed (written) under certain conditions, and the stack area is destroyed.

  If the MotoPlus application destroys the stack area, it is likely that the controller OS detects it, turns the task into an error, and stops the operation. In the task in which a problem probably occurred, add a process which periodically turns ON/OFF an I/O. If the task is running, you can check in the I/O window on the programming pendant that the specified I/O turns ON/OFF. If you cannot check whether it turns ON/OFF, the task may not be running. Then, check the following:

  - Whether the local variable definition is smaller than the stack size
  - Whether the array number of the local variable defined as an array is within the expected maximum value
  - Whether the pointer operation related to the local variable is correct
16.8 Phenomena during Debugging, Potential Coding Problems, and Examination Methods

16.8.3 Alarm 4479: “MOTOPLUS MM TASK WATCHDOG ERROR” Occurs

- **Cause 1:** In a high-priority task with the task priority other than the normal priority (MP_PRI_TIME_NORMAL), the changes in the flag status is performed in “while loop” with the endless-wait status without delaying the task.

  Under the conditions after start-up, check whether the application is in the endless-wait status without using mpTaskDelay() in the high-priority task of the application.

- **Cause 2:** In a high-priority task with the task priority other than the normal priority (MP_PRI_TIME_NORMAL), a process which takes a long time is performed.

  After start-up, check whether a process which takes a long time is performed under certain conditions.
  If such a process exists, review the design to move the process to a normal-priority task.

16.8.4 Programming Pendant Display or Key Response Is Slow

- **Cause 1:** In a high-priority task with the task priority other than the normal priority (MP_PRI_TIME_NORMAL), a process which takes a long time is performed.

  After start-up, check whether a process which takes a long time is performed under certain conditions.
  If a process which takes more than 1 msec is performed, the programming pendant display or key response becomes slow, even though the alarm 4479: “MOTOPLUS MM TASK WATCHDOG ERROR” does not occur.

- **Cause 2:** In a MotoPlus application task with the normal priority (MP_PRI_TIME_NORMAL), the system is in the endless-wait status without using mpTaskDelay().

  A normal-priority task and the task which runs the programming pendant display have the same priority. If the application is in the endless-wait status without using mpTaskDelay() in a MotoPlus application task, the processing time for the programming pendant display will be affected and the response will become slow.

16.8.5 Programming Pendant Communication Error Occurs and Servo Power of YRC1000 Turns Off (Hang-up Status)

- **Cause 1:** A CPU exception occurred due to pointer misoperation.

  In this case, “000E-0F3D412D” (“0F3D412D” is the program execution address of the exception occurrence, and it differs depending on where the exception occurred.) is displayed chronologically in the 7segLED of the AIF01 circuit board. It is likely that the pointer control of the MotoPlus application task is incorrect. Thus, determine where the error occurred by using printf() or outputing the number of the program execution state to the user variable.
17. Sample Programs

17.1 Outline of Sample Programs

The sample programs can be found under "\samples" in the DVD. Copy the sample program to a folder on the PC. (In the example, it is copied to C:\MotoPlusData\samples.) Each folder has source files, and their operations can be checked following the procedures below.

1. Start MotoPlus IDE, and open "*.mpProj" in each folder under C:\MotoPlusData\samples.
2. Execute "build", then "Build project" under the menu of MotoPlus IDE.
3. "*.out" (executable file) is created in "\out" in the folder where "*.mpProj" locates under C:\MotoPlusData\samples.
4. Copy the created "*.out" file to the SD or the USB. Then, start the YRC1000 in the MAINTENANCE mode, and install the file by "LOAD (USER APPLICATION)" under "MotoPlus APL." menu. (For details, refer to “Chap.4. Installation and Start-up of Application Program” in “YRC1000 OPTIONS INSTRUCTIONS User’s Manual For New Language Environment MotoPlus (HW1483600)”.)
5. Restart the YRC1000 in the normal mode.

With the above procedure, the application program automatically starts after the start-up of the YRC1000.

NOTE

The sample program in the DVD is created in the default installation folder (C:\Program Files\Yaskawa\MotoPlusIDE). Thus, if the installation folder is changed from the default, the build settings of the sample project must be initialized.

For details, refer to “Chap.3.7 Changing Build Settings” in “YRC1000 OPTIONS INSTRUCTIONS User’s Manual For New Language Environment MotoPlus (HW1483600)”. 
17.2 Display of Hello World

The display of "HELLO WORLD," which is the most primitive source code, is described below. This code is "HelloWorld.c" located under the folder C:\MotoPlusData\sample\Hello World\srcHelloWorld. When connected to a PC via Telnet after the power is turned on, the following message will be displayed on the Telnet terminal window of the PC after the start-up of the YRC1000 is completed.

->Hello World
->Hello World
->Hello World

The above is repeated every 2.5 seconds.

//**** HelloWorld.c ****

#include "motoPlus.h"

void mpUsrRoot(int arg1, int arg2, int arg3, int arg4, int arg5, int arg6, int arg7, int arg8, int arg9, int arg10)
{
    tid = mpCreateTask (MP_PRI_TIME_NORMAL, MP_STACK_SIZE, (FUNCPTR) moto_plus0_task, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10);

    mpExitUsrRoot;
}

void moto_plus0_task(void)
{
    while (1)
    {
        puts("Hello World!");
        mpTaskDelay(2500);
    }
}
17.3 Task Control Sample Program

The following is the sample of task control. An example using 2 source codes is explained here. They are “TaskCtl1.c” and “TaskCtl1.2” located in the folder C:\MotoPlusData\Sample\Multi Task control\SrcMultiTaskCtrl.

The first source file TaskCtl1.c activates 2 tasks, moto_plus0_task and moto_plus1_task, using mpCreateTask() to create one semaphore. After the semaphore is created, they are terminated automatically.

//**** TaskCtl1.c ****

#include "motoPlus.h"

// for GLOBAL DATA DEFINITIONS
SEM_ID semid;

// for IMPORT API & FUNCTIONS
extern void moto_plus0_task(void);
extern void moto_plus1_task(void);

// for LOCAL DEFINITIONS
static int tid1, tid2;

void mpUsrRoot(int arg1, int arg2, int arg3, int arg4, int arg5,
        int arg6, int arg7, int arg8, int arg9, int arg10)
{
    tid1 = mpCreateTask(MP_PRI_TIME_NORMAL, MP_STACK_SIZE,
                        (FUNCPTR)moto_plus0_task, arg1, arg2, arg3, arg4, arg5, arg6,
                        arg7, arg8, arg9, arg10);
    tid2 = mpCreateTask(MP_PRI_TIME_NORMAL, MP_STACK_SIZE,
                        (FUNCPTR)moto_plus1_task, arg1, arg2, arg3, arg4, arg5, arg6,
                        arg7, arg8, arg9, arg10);
    semid = mpSemBCreate(SEM_Q_FIFO, SEM_EMPTY); // Binary semaphore
    puts("Exit mpUsrRoot!");
    mpExitUsrRoot;//(or) mpSuspendSelf;
}
The second source file TaskCtl2.c includes the description of moto_plus0_task and moto_plus1_task. moto_plus0_task transmits a semaphore to moto_plus1_task using mpSemGive() every 2.5 seconds. Each time moto_plus1_task receives a semaphore transmitted using mpSemTake(), it displays the number of receipts after “moto_plus1_task Running.”

```c
//**** TaskCtl2.c ****
#include "motoPlus.h"

// for API & FUNCTIONS
extern void moto_plus0_task(void);
extern void moto_plus1_task(void);

// for DATA
extern SEM_ID semid;

void moto_plus0_task(void)
{
    puts("Activate moto_plus0_task!");
    while (1)
    {
        mpSemGive(semid);
        mpTaskDelay(2500);
    }
}

void moto_plus1_task(void)
{
    STATUS status;
    unsigned int run_cnt;

    //float get_time;

    puts("Activate moto_plus1_task!");

    run_cnt = 0;
    while (1)
    {
        run_cnt++;
    }
}
status = mpSemTake(semid, WAIT_FOREVER);
if (status == ERROR)
{
    printf("semTake Error![%d]\n", run_cnt);
}
else
{
    printf("moto_plus1_task Running![%d]\n", run_cnt);
}
}
### 17.4 User Watchdog Sample Program

The sample program of the user watchdog is described below. The sample program is in the folder “UserWatchDog”. This sample program consists of a high-priority task mp_seg_mon_task, which operates the user watchdog, and a task mp_para_get_task, which accesses to D variables. The D variables accessed by mp_para_get_task are used as the triggers and parameters to operate the user watchdog. The D variables used are shown in the following table.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>D050</td>
<td>Execution of mpUsrWdogCreate function</td>
<td>Execution condition: “0” → “1”</td>
</tr>
<tr>
<td>D051</td>
<td>Execution of mpUsrWdogDelete function</td>
<td>Execution condition: “0” → “1”</td>
</tr>
<tr>
<td>D052</td>
<td>Execution of mpUsrWdogStart function</td>
<td>Execution condition: “0” → “1”</td>
</tr>
<tr>
<td>D053</td>
<td>Execution of mpWdogClear function</td>
<td>Execution condition: “0” → “1”</td>
</tr>
<tr>
<td>D054</td>
<td>Index of the operated user watchdog</td>
<td>0 or 1</td>
</tr>
<tr>
<td>D055</td>
<td>Delay time of the user watchdog with the index “0”</td>
<td>Unit: msec</td>
</tr>
<tr>
<td>D056</td>
<td>Delay time of the user watchdog with the index “1”</td>
<td>Unit: msec</td>
</tr>
<tr>
<td>D058</td>
<td>Clear interval of the user watchdog with the index “0”</td>
<td>Unit: interpolation cycle</td>
</tr>
<tr>
<td>D059</td>
<td>Clear interval of the user watchdog with the index “1”</td>
<td>Unit: interpolation cycle</td>
</tr>
<tr>
<td>D061</td>
<td>Start and periodic clear of the user watchdog with the index “0”</td>
<td>Execution condition: “0” → “1”</td>
</tr>
<tr>
<td>D062</td>
<td>Start and periodic clear of the user watchdog with the index “1”</td>
<td>Execution condition: “0” → “1”</td>
</tr>
</tbody>
</table>

Enter “0” → “1” in the variable D050 to D053 to execute the corresponding API of the user watchdog.

For MotoPlus, up to 2 user watchdog timers can be operated simultaneously. D054 works as the index to specify the 2 user watchdog timers.

D055 and D056 are the delay time (unit: ms) of the user watchdog timer.

D061 and D062 are the triggers to “start” and “periodically clear” the user watchdog. Specify the clear interval by D058 and D059. The user watchdog is cleared at the intervals of D058 (or D059) × interpolation cycle. If it is not cleared within the delay time, the time-out routine function fncwdRoutine0() or fncwdRoutine1(), which is specified at the creation of the user watchdog, is executed, and the value is output to the register number 50 or 51.

Since the “creation” of the user watchdog takes a long processing time, only the “start” and “clear” of it are executed with a high-priority task.
**User Watchdog Sample Program**

```c
void mpUsrRoot(int arg1, int arg2, int arg3, int arg4, int arg5,
    int arg6, int arg7, int arg8, int arg9, int arg10)
{
    semid = mpSemBCreate(SEM_Q_FIFO, SEM_EMPTY);
    ip_tid  = mpCreateTask(MP_PRI_IP_CLK_TAKE,
            MP_STACK_SIZE, (FUNCPTR)mp_seg_mon_task,
            arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10);
    tn_tid = mpCreateTask(MP_PRI_TIME_NORMAL,
            MP_STACK_SIZE, (FUNCPTR)mp_para_get_task,
            arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10);
    mpExitUsrRoot;
}

static void mp_para_get_task(void)
{
    FOREVER
    {
        GetParameter( );
        mpTaskDelay(1000);  // 1sec wait;
    }
}

static void mp_seg_mon_task(void)
{
    int   rc, i;
    LONG  Cnt[2];
    LONG  DVvalue_last[2];

    wdRoutine[0] = (MP_USR_WDG_ROUTINE)fncwdRoutine0;
    wdRoutine[1] = (MP_USR_WDG_ROUTINE)fncwdRoutine1;

    GetParameter( );

    FOREVER
    {
        if (mpClkAnnounce(MP_INTERPOLATION_CLK) == ERROR)
```
mpTaskSuspend(ip_tid);

for( i = 0; i < 2; i++ )
{
    //D061,D062
    if( (DVvalue_last[i] == 0) && (DVvalue[11 + i] == 1) )
    {
        Cnt[i] = 0;
        rc = mpUsrWdogStart( handle[i] );
    }
    DVvalue_last[i] = DVvalue[11 + i];

    if( DVvalue[11 + i] == 1 )
    {
        if( ClearCnt[i] > Cnt[i] )
        {
            ++Cnt[i];
            if( ClearCnt[i] == Cnt[i] )
            {
                rc = mpUsrWdogClear( handle[i] );
                Cnt[i] = 0;
            }
        }
    }
}

static void GetParameter(void)
{
    int rc;
    int index;
    LONG DVvalue050, DVvalue051, DVvalue052, DVvalue053;

    //Save DVar oldvalue
    DVvalue050 = DVvalue[0]; //WdogCreate
    DVvalue051 = DVvalue[1]; //WdogDelete
    DVvalue052 = DVvalue[2]; //WdogStart
    DVvalue053 = DVvalue[3]; //WdogClear
//Get DVar value
if( mpGetMultiDVar(50, DVvalue, DVAL_NUM) == 0 )
{
    //Get index value: D054
    if( DVvalue[4] < 2 )
        index = (int)DVvalue[4];
    else
        index = 0;

    //Get delay value: D055, D056
    delay[0] = (int)DVvalue[5];
    delay[1] = (int)DVvalue[6];

    //Get ClearCnt value: D058, D059
    ClearCnt[0] = DVvalue[8];
    ClearCnt[1] = DVvalue[9];

    //WdogCreate
    if( (DVvalue050 == 0) && (DVvalue[0] == 1) ) //D050 "0"->"1"
        handle[index] = mpUsrWdogCreate( delay[index], wdRoutine[index] );

    //WdogDelete
    if( (DVvalue051 == 0) && (DVvalue[1] == 1) ) //D051 "0"->"1"
        rc = mpUsrWdogDelete( handle[index] );

    //WdogStart
    if( (DVvalue052 == 0) && (DVvalue[2] == 1) ) //D052 "0"->"1"
        rc = mpUsrWdogStart( handle[index] );

    //WdogClear
    if( (DVvalue053 == 0) && (DVvalue[3] == 1) ) //D053 "0"->"1"
        rc = mpUsrWdogClear( handle[index] );
}
}

static void fncwdRoutine0( MP_WDG_HANDLE lhandle )
{
    if( WriteIO(1000050, 50) != 0)
        puts("Error!! (WriteIO_0)\n");
17 Sample Programs
17.4 User Watchdog Sample Program

else
    printf("wdRoutine0!! (WriteIO_0) [handle = %d]\n", lhandle);
}

static void fncwdRoutine1(MP_WDG_HANDLE lhandle)
{
    if( WriteIO(1000051, 51) != 0)
        puts("Error!! (WriteIO_1)\n");
    else
        printf("wdRoutine1!! (WriteIO_1) [handle = %d]\n", lhandle);
}
17.5 Ethernet Sample Programs

Server programs which echo back the send data of a client as MotoPlus application of the YRC1000 are described below. They are “TcpSvr1.c” and “TcpSvr2.c” located under the folder C:\MotoPlusData\sample\Ethernet control\TCP\srcTcpSvr, and “UdpSvr1.c” and “UdpSvr2.c” located under the folder C:\MotoPlusData\sample\Ethernet control\UDP\srcUdpSvr.

17.5.1 Preparations

With this sample program, the PC is used as the client. The following software must be installed on the PC to be used in advance.

- Microsoft Windows XP/Vista
- Microsoft Visual C++ 2008

To use the Ethernet function in MotoPlus, the Ethernet function (optional) of the YRC1000 must be set in advance. Refer to “YRC1000 OPTIONS INSTRUCTIONS FOR ETHERNET FUNCTION (HW1483358)” for the details of the ethernet function of the YRC1000.

1. Ethernet Cable Connections
   Connect an Ethernet cable (shielded, category 5 or higher) to the LAN connector CN106 (LAN2) at the front of the ACP01 circuit board in the CPU rack of the YRC1000.

2. Settings of IP Address etc.
   The Ethernet communication settings tailored to the user’s environment such as the IP address of the YRC1000 can be made in the maintenance mode. Configure the settings according to the following procedure.

   (1) Turn OFF the power then back ON while pressing {MAIN MENU} to start the maintenance mode.

   (2) Change the security mode to the “MANAGEMENT MODE”.

   (3) From the main menu, select {SYSTEM}, then {Security} and change the security mode to the “MANAGEMENT MODE”.

   (4) Open "DETAIL" of "HOST SETUP" in the network function setting window.

   (5) From the main menu, select {SYSTEM}, {SETUP}, {OPTION FUNCTION}, then {NETWORK} and select "DETAIL" of “HOST SETUP.”
Set the IP address etc. in the host setting window. As in the case of the YRC1000, set the Windows PC side so that it has the same subnet address. In the case of the following example for the YRC1000 setting, set the PC side as follows; IP address: 192.168.255.xx (xx is a decimal number between 0 - 255, but it must differ from that of the YRC1000 side), subnet mask: 255.255.255.0.

3. Restrictions on IP Address Setting
   The Ethernet function of the YRC1000 does not support the local IP address "10.0.0.xx". (xx is a decimal number from 0 to 255.) DO NOT set it for the IP address"10.0.0.xx".

4. Restrictions on Setting the IP Address
   In the Ethernet function of the YRC1000, the system uses specific ports exclusively for UDP and TCP. Thus, the port numbers smaller than 10040 cannot be used in the MotoPlus application. The ports used by the system may be added or changed depending on the system version. When the optional functions related to the Ethernet function of the YRC1000 are used, be sure not to use the port numbers described in the instruction manuals for the functions.
17.5 Ethernet Sample Programs

17.5.2 UDP Server Sample Programs (YRC1000 Side)

This program converts the string sent by the client of Windows PC to uppercase characters, and echoes it back.

<How to execute the sample programs>

1. Create the application program from the sample programs ("UdpSvr1.c" and "UdpSvr1.c" under the folder C:\MotoPlusData\sample\Ethernet control\UDP\srcUdpSvr) and install them on the YRC1000.

2. Restart the YRC1000.

3. The name of the client application at the Windows PC side is client.exe located in the folder C:\MotoPlusData\sample\Ethernet control\UDP\clientForWindows. Start it with the DOS prompt.

4. When the client application is run, it prompts you to enter the port number and IP address of the server. Enter 11000 as the port number, and enter the IP address as set in advance.

5. Then, the data sent to or received from the server is displayed each time the Enter key is pressed after characters are entered by using the keyboard.

6. Enter “exit<CR>” to terminate the client. Start it before starting the client.

<Execution example of DOS prompt>

C:\MotoPlusData\sample\Ethernet control\UDP\clientForWindows>client
Simple UDP client
Please type PORT No. >11000
Please type IP addr. >192.168.255.1
Please type the ASCII characters
> a
  send(1) a
  recv(2) A
> b
  send(3) b
  recv(4) B
> exit
  send(5) exit
  recv(6) EXIT

C:\MotoPlusData\sample\Ethernet control\UDP\clientForWindows>
<Sample programs>

The first sample program UdpSvr1.c starts moto_plus0_task when the program is started and then terminates itself.

/* UdpSvr1.c */
#include "motoPlus.h"

// for GLOBAL DATA DEFINITIONS
SEM_ID semid;

// for IMPORT API & FUNCTIONS
extern void moto_plus0_task(void);

// for LOCAL DEFINITIONS
static int tid1;

void mpUsrRoot(int arg1, int arg2, int arg3, int arg4, int arg5,
    int arg6, int arg7, int arg8, int arg9, int arg10)
{
    tid1 = mpCreateTask(MP_PRI_TIME_NORMAL, MP_STACK_SIZE,
        (FUNCPTR)moto_plus0_task, arg1, arg2, arg3, arg4, arg5,
        arg6, arg7, arg8, arg9, arg10);
    semid = mpSemBCreate(SEM_Q_FIFO, SEM_EMPTY);  // Binary semaphore
    puts("Exit mpUsrRoot!");
    mpExitUsrRoot;  //(or) mpSuspendSelf;
}

The next sample program "UdpSvr2.c" calls and runs the UDP server program.

/* UdpSvr2.c */
/* Copyright 2009 YASKAWA ELECTRIC All Rights reserved. */

#include "motoPlus.h"

// for API & FUNCTIONS
void moto_plus0_task(void);
void ap_UDP_Sserver(ULONG portNo);
#define PORT        11000 //Arbitrary port number that is not used by the system
#define BUFF_MAX    1023

void moto_plus0_task(void)
{

puts("Activate moto_plus0_task!");
ap_UDP_Sserver(PORT);     // Start of Ethernet UDP server processing
mpSuspendSelf;            // suspend
}

char *strupr(char *string);

void ap_UDP_Sserver(ULONG portNo)
{
    int     sockHandle;
    struct sockaddr_in serverSockAddr;
    struct sockaddr_in clientSockAddr;
    int     sizeofSockAddr;
    int     rc;

    printf("Simple UDP server\n");

    while(1)
    {
        sockHandle = mpSocket(AF_INET, SOCK_DGRAM, 0); // Creation of
                                                  // connectionless socket
        if (sockHandle < 0)
            return;
        memset(&serverSockAddr, 0, sizeof(serverSockAddr));
        serverSockAddr.sin_family = AF_INET;
        serverSockAddr.sin_addr.s_addr = INADDR_ANY;
        serverSockAddr.sin_port = mpHtons(PORT);     // Set the IP address and port
                                                 // number of own side to the socket.
        rc = mpBind(sockHandle, (struct sockaddr *)&serverSockAddr,
                     sizeof(serverSockAddr));
        if (rc < 0)
            goto closeSockHandle;
        memset(&clientSockAddr, 0, sizeof(clientSockAddr));
        sizeofSockAddr = sizeof(clientSockAddr);
        while (1)
        {
            int     bytesRecv;
            int     bytesSend;
char    buff[BUFF_MAX + 1];

memset(buff, 0, sizeof(buff));

bytesRecv = mpRecvFrom(sockHandle, buff, BUFF_MAX, 0, (struct sockaddr *)&clientSockAddr, &sizeofSockAddr); // Receiving
if (bytesRecv < 0)
    break;

/* Converts the received data to uppercase characters and returns it */
strupr(buff);
bytesSend = mpSendTo(sockHandle, buff, bytesRecv, 0, (struct sockaddr *)&clientSockAddr, sizeof(clientSockAddr)); // Sending

if (bytesSend != bytesRecv)
    break;

if (strncmp(buff, "EXIT", 4) == 0 || strncmp(buff, "exit", 4) == 0)
    break;
}
closeSockHandle:
    mpClose(sockHandle); // Close the socket.
}

char *strupr(char *string);
{
    int   i;
    int   len;

    len = strlen(string);
    for (i = 0; i < len; i++)
    {
        if (isalpha(string[i])
        {
            string[i] = toupper(string[i]);
        }
    }
    return (string);
}
17.5 Ethernet Sample Programs

17.5.3 UDP Client Sample Program (Windows PC Side)

When using Microsoft Visual C++, link Ws2_32.lib.

```c
#include "stdafx.h"
#include <winsock2.h>
#define PORT 11000 // Port number specified by the server side
#define BUFF_MAX 256

int _tmain(int argc, _TCHAR* argv[])
{
    SOCKET sockHandle;
    struct sockaddr_in serverSockAddr;
    struct sockaddr_in clientSockAddr;
    unsigned long serverAddr;
    int sizeofSockAddr;
    int bytesKeyin;
    int bytesReceived;
    int bytesSend;
    int cntRcvSnd;
    int rc;
    char buff[BUFF_MAX + 1];
    WSADATA data;

    printf("Simple UDP client\n");
    printf(" Please type the ASCII characters\n");

    WSAStartup(MAKEWORD(2,0), &data);

    sockHandle = socket(AF_INET, SOCK_DGRAM, 0);
    if(sockHandle < 0)
    {
        printf("Error : socket() rc = %d\n", sockHandle);
        exit(1);
    }

    memset( &clientSockAddr, 0, sizeof(clientSockAddr) );
    clientSockAddr.sin_family = AF_INET;
    clientSockAddr.sin_addr.s_addr = INADDR_ANY;
    clientSockAddr.sin_port = htons(PORT);
```
rc = bind(sockHandle, (struct sockaddr *)&clientSockAddr, sizeof(clientSockAddr));
if (rc < 0)
{
    printf("Error : bind() rc = %d\n", rc);
    goto closeSockHandle;
}
memset( &serverSockAddr, 0, sizeof(serverSockAddr) );
serverSockAddr.sin_family = AF_INET;
serverAddr = inet_addr( "192.168.255.1" ); // IP address of the YRC1000 (server)
serverSockAddr.sin_addr.s_addr = serverAddr;
serverSockAddr.sin_port = htons(PORT);
sizeofSockAddr = sizeof(serverSockAddr);

printf(" > ");
cntRcvSnd = 0;
while (1)
{
    /* Transmits the character(s) entered from the keyboard. */
    fgets(buff, BUFF_MAX, stdin);
    if(buff[0] == '\n')
    {
        break;
    }

    bytesKeyin = strlen(buff);
    bytesSend = sendto(sockHandle, buff, bytesKeyin, 0, (struct sockaddr *)
            &serverSockAddr, sizeof(serverSockAddr));
    if (bytesSend != bytesKeyin)
    {
        printf("Error : sendto() rc = %d\n", bytesSend);
        break;
    }

    /* Waiting for incoming data */
    cntRcvSnd++;
    printf("send(%d) %s", cntRcvSnd, buf);
    bytesReceived = recvfrom(sockHandle, buff, BUFF_MAX, 0, (struct sockaddr *)
            &serverSockAddr, &sizeofSockAddr);
if (bytesReceived < 0)
{
    printf("Error : recvfrom() rc = %d\n", bytesReceived);
    break;
}

cntRcvSnd++;
buff[bytesReceived] = '\0';
printf("recv(\%d) %s", cntRcvSnd, buff);
printf(" > ");
}
closeSockHandle:
    closesocket(sockHandle);
    WSACleanup();
    exit(0);
}
17.5 Ethernet Sample Programs

17.5.4 TCP Server Sample Programs (YRC1000 Side)

This program also converts the string sent by the client of Windows PC to uppercase characters, and echoes it back.

<How to execute the sample programs>

1. Create the application program from the sample programs ("TCPSvr1.c" and "TCPSvr1.c" under the folder C:\MotoPlusData\sample\Ethernet control\TCP\srcTcpSvr) and install them on the YRC1000.

2. Restart the YRC1000.

3. The name of the client application at the Windows PC side is client.exe located in the folder C:\MotoPlusData\sample\Ethernet control\TCP\clientForWindows. Start it with the DOS prompt.

4. When the client application starts, it prompts you to enter the port number and IP address of the server. Enter 11000 as the port number, and enter the IP address as set in advance.

5. Then, the data sent to or received from the server is displayed each time the Enter key is pressed after characters are entered by using the keyboard.

6. Enter “exit<CR>” to terminate the client. Start it before starting the client.

<Execution example of DOS prompt>

C:\MotoPlusData\sample\Ethernet control\TCP\clientForWindows>client
Simple TCP client
Please type PORT No. >11000
Please type IP addr. >192.168.255.1
Please type the ASCII characters
> a
send(1) a
recv(2) A
> b
send(3) b
recv(4) B
> exit
send(5) exit
recv(6) EXIT

C:\MotoPlusData\sample\Ethernet control\TCP\clientForWindows>
<Sample programs>

The first sample program TcpSvr1.c is the same as UdpSvr1. It starts moto_plus0_task when the program is started, then terminates itself.

/* UdpSvr1.c */
#include "motoPlus.h"

// for GLOBAL DATA DEFINITIONS
SEM_ID semid;

// for IMPORT API & FUNCTIONS
extern void moto_plus0_task(void);

// for LOCAL DEFINITIONS
static int tid1;

void mpUsrRoot(int arg1, int arg2, int arg3, int arg4, int arg5,
    int arg6, int arg7, int arg8, int arg9, int arg10)
{
    tid1 = mpCreateTask(MP_PRI_TIME_NORMAL, MP_STACK_SIZE,
        (FUNCPTR)moto_plus0_task, arg1, arg2, arg3, arg4, arg5, arg6, arg7,
        arg8, arg9, arg10);
    semid = mpSemBCreate(SEM_Q_FIFO, SEM_EMPTY); // Binary semaphore
    puts("Exit mpUsrRoot!");
    mpExitUsrRoot; // (or) mpSuspendSelf;
}

The next sample program "TcpSvr2.c" calls and runs the TCP server program.

/* TcpSvr2.c */
/* Copyright 2009 YASKAWA ELECTRIC All Rights reserved. */

#include "motoPlus.h"

// for API & FUNCTIONS
void moto_plus0_task(void);
void ap_TCP_Sserver(ULONG portNo);
define PORT 11000
#define BUFF_MAX 1023
void moto_plus0_task(void)
{
    puts("Activate moto_plus0_task!");
    ap_TCP_Sserver(PORT);
    mpSuspendSelf;
}

char *strupr(char *string);

void ap_TCP_Sserver(ULONG portNo)
{
    int     sockHandle;
    struct  sockaddr_in    serverSockAddr;
    int     rc;

    printf("Simple TCP server\n");

    sockHandle = mpSocket(AF_INET, SOCK_STREAM, 0); // Creation of socket
    if (sockHandle < 0)
        return;

    memset(&serverSockAddr, 0, sizeof(serverSockAddr));
    serverSockAddr.sin_family = AF_INET;
    serverSockAddr.sin_addr.s_addr = INADDR_ANY;
    serverSockAddr.sin_port = mpHtons(PORT); // Set the IP address and port
    number of own side to the socket.

    rc = mpBind(sockHandle, (struct sockaddr *)&serverSockAddr, sizeof(serverSockAddr));
    if (rc < 0)
        goto closeSockHandle;

    rc = mpListen(sockHandle, SOMAXCONN); // Put the socket into the
    connection standby state
    if (rc < 0)
        goto closeSockHandle;

    while (1)
    {
        int     acceptHandle;

struct sockaddr_in clientSockAddr;
int sizeofSockAddr;

memset(&clientSockAddr, 0, sizeof(clientSockAddr));
sizeofSockAddr = sizeof(clientSockAddr);

acceptHandle = mpAccept(sockHandle, (struct sockaddr *)&clientSockAddr, &sizeofSockAddr); // Establishment of connection and creation of socket for data sending/receiving

if (acceptHandle < 0)
    break;

while(1)
{
    int bytesRecv;
    int bytesSend;
    char buff[BUF_MAX + 1];

    memset(buff, 0, sizeof(buff));

    bytesRecv = mpRecv(acceptHandle, buff, BUF_MAX, 0); // Receiving

    if (bytesRecv < 0)
        break;

    /* Converts the received data to uppercase characters and returns it */
    strupr(buff);
    bytesSend = mpSend(acceptHandle, buff, bytesRecv, 0); // Sending

    if (bytesSend != bytesRecv)
        break;

    if (strncmp(buff, "EXIT", 4) == 0 || strncmp(buff, "exit", 4) == 0)
        break;
}

mpClose(acceptHandle); // Closes the socket for data sending/receiving

closeSockHandle:
mpClose(sockHandle); // Closes the socket for connection

    return;
}

cchar *strupr(char *string)
{
    int     i;
    int     len;

    len = strlen(string);
    for (i = 0; i < len; i++)
    {
        if (isalpha(string[i]))
        {
            string[i] = toupper(string[i]);
        }
    }
    return (string);
17.5.5 TCP Client Sample Program (Windows PC Side)

When using Microsoft Visual C++, link Ws2_32.lib.

```c
#include "stdafx.h"
#include <winsock2.h>
#define PORT        11000 // Port number specified by the server side
#define BUFF_MAX    256

int _tmain(int argc, _TCHAR* argv[]) {
    SOCKET sockHandle;
    struct sockaddr_in serverSockAddr;
    int cntRcvSnd;
    WSADATA data;
    int rc;

    printf("Simple TCP client\n");
    printf(" Please type the ASCII characters\n");

    WSAStartup(MAKEWORD(2,0), &data);

    sockHandle = socket(AF_INET, SOCK_STREAM, 0);
    if(sockHandle < 0) {
        printf("Error : socket() rc = %d\n", sockHandle);
        exit(1);
    }

    memset( &serverSockAddr, 0, sizeof(serverSockAddr) );
    serverSockAddr.sin_family = AF_INET;
    serverSockAddr.sin_addr.s_addr = inet_addr( "192.168.255.1" );
    serverSockAddr.sin_port = htons(PORT);

    rc = connect(sockHandle, (struct sockaddr *)&serverSockAddr, sizeof(serverSockAddr));
    if (rc < 0) {
        printf("Error : connect() rc = %d\n", rc);
        goto closeSockHandle;
    }

    memset( &serverSockAddr, 0, sizeof(serverSockAddr) );
    serverSockAddr.sin_family = AF_INET;
    serverSockAddr.sin_addr.s_addr = inet_addr( "192.168.255.1" );
    serverSockAddr.sin_port = htons(PORT);

    rc = connect(sockHandle, (struct sockaddr *)&serverSockAddr, sizeof(serverSockAddr));
    if (rc < 0) {
        printf("Error : connect() rc = %d\n", rc);
        goto closeSockHandle;
    }

    closeSockHandle:
    return 0;
}
```
printf(" > ");
cntRcvSnd = 0;
while (1)
{
    int bytesKeyin;
    int bytesSend;
    int bytesRecv;
    char buff[BUFF_MAX + 1];

    fgets(buff, BUFF_MAX, stdin);

    bytesKeyin = strlen(buff);

    bytesSend = send(sockHandle, buff, bytesKeyin, 0);
    if (bytesSend != bytesKeyin)
    {
        printf("Error : send() rc = %d\n", bytesSend);
        break;
    }

    cntRcvSnd++;
    printf("send(%d) %s", cntRcvSnd, buff);

    bytesRecv = recv(sockHandle, buff, bytesSend, 0);
    if (bytesRecv < 0)
    {
        printf("Error : recv() rc = %d\n", bytesRecv);
        break;
    }

    cntRcvSnd++;
    buff[bytesRecv] = '\0';
    printf("recv(%d) %s", cntRcvSnd, buff);

    if (strcmp(buff, "EXIT") == 0)
        break;

    printf(" > ");
closeSockHandle:
    closesocket(sockHandle);

WSACleanup();

exit(0);
17.6 Robot System Monitor Samples and Control Programs

The following is a sample program which controls the variables and I/Os of a robot system.

17.6.1 Variables, I/O Access Sample Program

The first source file TaskCtl1.c is the same as that of chapter 17.2 “Display of Hello World”, which starts 2 tasks, moto_plus0_task and moto_plus1_task, by using mpCreateTask() to create one semaphore. After the semaphore is created, they are terminated automatically.

//**** TaskCtl1.c ****

#include "motoPlus.h"

// for GLOBAL DATA DEFINITIONS
SEM_ID semid;

// for IMPORT API & FUNCTIONS
extern void moto_plus0_task(void);
extern void moto_plus1_task(void);

// for LOCAL DEFINITIONS
static int tid1, tid2;

void mpUsrRoot(int arg1, int arg2, int arg3, int arg4, int arg5,
    int arg6, int arg7, int arg8, int arg9, int arg10)
{
    tid1 = mpCreateTask(MP_PRI_TIME_NORMAL, MP_STACK_SIZE,
        (FUNCPTR)moto_plus0_task, arg1, arg2, arg3, arg4, arg5,
        arg6, arg7, arg8, arg9, arg10);
    tid2 = mpCreateTask(MP_PRI_TIME_NORMAL, MP_STACK_SIZE,
        (FUNCPTR)moto_plus1_task, arg1, arg2, arg3, arg4, arg5,
        arg6, arg7, arg8, arg9, arg10);
    semid = mpSemBCreate(SEM_Q_FIFO, SEM_EMPTY); // Binary semaphore

    puts("Exit mpUsrRoot!");
    mpExitUsrRoot; // (or) mpSuspendSelf;
}


The second source file is “VariableIOCtrl.c” which is the program having actual access to variables and I/Os. This program changes the position variable P010 periodically. This is realized by 2 tasks, moto_plus0_task and moto_plus1_task. moto_plus0_task sends a semaphore to moto_plus1_task at a constant period. This period can be specified with the byte variable B001. The period is 50 msec when B001 = 0, and, with other values, the period is B001 x 10 (msec).

Each time moto_plus1_task receives a semaphore in an endless loop, moto_plus1_task performs its process. In other words, it performs the processing at the period with which moto_plus0_task sends a semaphore. The content of processing varies depending on the byte variable B000. Nothing is performed while B000 = 0. However, when the variable becomes 1 (B000 = 1), the current Cartesian coordinate position of the robot is read at that moment. Then, the positions 50 mm above and below the read Cartesian coordinate position in the Z-axis direction of the robot coordinate are written in the position variable P010 at each period. At the timing of writing, I/O #10012 and #10013 are inverted and a pulse of 20 msec is output to #10014.

The motions generated by combining this application program and the job “MOTOPLUS-TEST1.JBI” are shown in the table below. The robot moves from the first point to the second point, moves from the second point to the vicinity of the first point with vertical swing, and then repeats a vertical swing motion after it reaches to the X, Y positions of the first point.
<table>
<thead>
<tr>
<th>Motion of JOB “MOTOPLUS-TEST1.JBI”</th>
<th>Motion of MotoPlusApI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SET B000</strong></td>
<td>Initialization instruction</td>
</tr>
<tr>
<td><strong>SET B001 10</strong></td>
<td>Setting of MotoPlusApI period</td>
</tr>
<tr>
<td><strong>MOVJ VJ=20.00</strong></td>
<td>Motion to the first point</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MOVJ VJ=20.00</strong></td>
<td>Motion to the second point</td>
</tr>
<tr>
<td>*LABEL1 MOVL P010 V=138 UNTIL IN#(954) JUMP *LABEL1</td>
<td>Moves from the second point to P10 (the first point position or a position 50 mm above the position: set by MotoPlusApI). Whenever IN#954 is turned ON, a motion toward the newly read P10 is performed again.</td>
</tr>
</tbody>
</table>
17.6 Robot System Monitor Samples and Control Programs

<JOB MOTOPLUS-TEST.JBI>

//NAME MOTOPLUS-TEST1
NOP
**** Initialize ***
SET B000 0
**** set running interval of MOTO
**** of MOTOPLUS APL[100msec]
SET B001 10
MOVJ C00000 VJ=20.00
**** Set command to MOTOPLUS APL
**** MOTOPLUS APL Latch currentpos
**** and give + or - 50mm z-dir
**** pos from got currentpos
**** via P010
**** then pulseout for MOVL UNTIL
SET B000 1
MOVJ C00001 VJ=20.00
*LABEL1
MOVL P010 V=23.0 UNTIL IN#(954)=ON
JUMP *LABEL1
END

<Adding CIO to user ladder>

Connect #10014(OUT#(5)) through which MotoPlusApl outputs its pulse to IN#(954) of MOVL P010 V=23.0 UNTIL IN#(954)=ON.

STR #10014
OUT #01201
When the YRC1000 is started and B000 is set to “1,” it can be confirmed that UT#0003 (#10012) and OUT#0004 (#10013) repeat ON/OFF alternately.

The following figure shows the window on which the job “MOTOPLUS-TEST.JBI,” position variable P010 and current orthogonal position are displayed simultaneously. This is the state when “MOVL P010 V=138 UNTIL IN#(954)” is stopped during its execution and the target position or Z-coordinate value of P010 is 614.000 mm. The current Z-coordinate value is 636.499 mm and it is moving to the position 614.000 mm that is 50 mm lower in the Z direction.
<Sample programs>

/*** VariableIOCtrl.c****

#include "motoPlus.h"

// for External definition
extern void moto_plus0_task(void);
extern void moto_plus1_task(void);

// for DATA
extern SEM_ID semid;

// for LOCAL DEFINITIONS
static unsigned int run_cnt; // for moto_plus0_task.
static unsigned int cycletime;

#define CYCLETIME  50 // [msec] Default cycle time (If B00=0 then used)
#define RTC_TIME   1   // [msec]
#define MOVE_DISTANCE 50000 // [micro meter] Up & Down distance
#define PULSE_TIME  20 // [msec] Pulse out width

#define ON 1
#define OFF 0

/******************
 ** useful function for application **
******************/

extern STATUS GetBVar(UINT16 index, long *value);
extern STATUS GetMultiBVar(UINT16 startIdx, long *value, long nbr);
extern STATUS SetBVar(UINT16 index, long value);
extern STATUS GetIVar(UINT16 index, long *value);
extern STATUS SetIVar(UINT16 index, long value);
extern STATUS GetSVar(UINT16 index, char *value);
extern STATUS SetDVar(UINT16 index, long value);
extern STATUS SetMultiDVar(UINT16 startIdx, long *values, long nbr);
extern STATUS GetMultiDVar(UINT16 startIdx, long *values, long nbr);
extern STATUS GetXVar(UINT16 type, UINT16 index, char *value);
extern STATUS GetIo(UINT32 ioAddr, USHORT *signal);
extern STATUS SetIo(UINT32 ioAddr, USHORT signal);
extern STATUS SetMultiIo(UINT32 ioStartAddr, USHORT *signal, long nbr);
extern STATUS WriteIO(UINT32 addr, UINT16 value);
extern STATUS ReadIO(UINT32 addr, UINT16* value);

typedef struct
{
    USHORT     usType;
    USHORT     usIndex;
    CHAR     VariableType;
    CHAR     config;
    USHORT     ToolNo_UserNo;
    LONG     ulPosdata[9];
} MP_POSVAR_DATA2;

void PulseOut(io_adr, time)
{
    LONG rt = 0;

    rt = SetIo(io_adr, ON);
    mpTaskDelay(time);
    rt = SetIo(io_adr, OFF);
    return;
}

void moto_plus0_task(void)
{
    LONG BVvalue = 0;
    LONG rt = 0;
    puts("Activate moto_plus0_task!");

    run_cnt = 0;
    cycletime = 0;

    while (1)
    {
        run_cnt++;

        mpSemGive(semid);
rt = GetBVar(1, &BVvalue); // B001
if (BVvalue == 0)
    cycletime = CYCLETIME/RTC_TIME;
else
    cycletime = BVvalue * 10;
mpTaskDelay(cycletime);
}
}

void moto_plus1_task(void)
{
    STATUS status;
    unsigned int run_cnt;
    unsigned int turn;

    unsigned int sase1state;

    LONG BVvalue = 0;
    LONG rt = 0;

    //float get_time;
    MP_POSVAR_DATA2 mp_posvar_data;

    MP_CTRL_GRP_SEND_DATA mp_ctrl_grp_send_data;
    MP_CART_POS_RSP_DATA mp_cart_pos_rsp_data;

    ///

    puts("Activate moto_plus1_task!");

    run_cnt = 0;
    turn = 0;
    case1state = 0;

    while (1)
    {
        run_cnt++;
        status = mpSemTake(semid, WAIT_FOREVER);
if (status == ERROR)
{
    printf("semTake Error![%d]n", run_cnt);
}

rt = GetBVar(0, &BVvalue); // Rfering B000

if (BVvalue != 1) case1state = 0; // initialize case1stae if (b000 != 1)
switch (BVvalue)
{
    case 0: // Initialize data
        turn=0;
        break;
    case 1: // only first time get current cart pos,
        // but all the time set P10 got current pos + or - distance
        if (case1state == 0)
        {
            // get current cart position
            mp_ctrl_grp_send_data.sCtrlGrp = 0; // R1
            if (mpGetCartPos(&mp_ctrl_grp_send_data, &mp_cart_pos_rsp_data) != 0)
            {
                puts("get cart pos error");
                break;
            }
            // Copy current position to P variable
            memset(&mp_posvar_data.ulPosdata[0], 0, (sizeof(long) * 9)); // all clear
            memcpy(&mp_posvar_data.ulPosdata[0], &mp_cart_pos_rsp_data.lPos[0], (sizeof(long) * 6));
            mp_posvar_data.config = (unsigned char)(mp_cart_pos_rsp_data.sConfig);
            mp_posvar_data.ToolNo_UserNo = 0;
            sase1state = 1;
        }
        mp_posvar_data.usType = MP_RESTYPE_VAR_ROBOT;
        mp_posvar_data.usIndex = 10; // P var number (P010)
        mp_posvar_data.VariableType = 16; // Cartesian (base coordinates)

        if (turn == 0) // set 50mm or -50mm to z direction data depending on turn value
17 Sample Programs
17.6 Robot System Monitor Samples and Control Programs

```c
{
    turn = 1;
    mpPutPosVarData((MP_POSVAR_DATA*) &mp_posvar_data, 1);

    rt = SetIo(10012, ON);  // #10012 --> ON
    rt = SetIo(10013, OFF);  // #10013 --> OFF
    PulseOut(10014, PULSE_TIME);  // #10014 pulse out 20msec
}
else
{
    turn = 0;
    mpPutPosVarData((MP_POSVAR_DATA*) &mp_posvar_data, 1);

    rt = SetIo(10012, OFF);  // #10012 --> OFF
    rt = SetIo(10013, ON);   // #10013 --> ON
    PulseOut(10014, PULSE_TIME);  // #10014 pulse out 20msec
}
break;

    case 2:
    case 3:
        break;

    default:
        break;
}
```
The third source file is the sample program “funclib.c” with simplified variables and I/Os. It simplifies programs by describing arguments in the data structure of an interface.

```c
//**** funclib.c ****

#include "motoPlus.h"
/
** useful function for application **
****************************************/}

STATUS GetBVar(UINT16 index, long *value)
{
    MP_VAR_INFO info;

    info.usType = MP_RESTYPE_VAR_B;
    info.usIndex = index;

    return mpGetVarData(&info, value, 1);
}

STATUS GetMultiBVar(UINT16 startIdx, long *value, long nbr)
{
    int i;
    MP_VAR_INFO info[20];

    for(i=0; i<nbr; i++)
    {
        info[i].usType = MP_RESTYPE_VAR_B;
        info[i].usIndex = startIdx + i;
    }

    return mpGetVarData(info, value, nbr);
}

STATUS SetBVar(UINT16 index, long value)
{
    MP_VAR_DATA info;

    info.usType = MP_RESTYPE_VAR_B;
```
17 Sample Programs
17.6 Robot System Monitor Samples and Control Programs

```c
info.usIndex = index;
info.ulValue = value;

return mpPutVarData(&info, 1);
}

STATUS GetIVar(UINT16 index, long *value)
{
    MP_VAR_INFO info;

    info.usType = MP_RESTYPE_VAR_I;
    info.usIndex = index;

    return mpGetVarData(&info, value, 1);
}

STATUS SetIVar(UINT16 index, long value)
{
    MP_VAR_DATA info;

    info.usType = MP_RESTYPE_VAR_I;
    info.usIndex = index;
    info.ulValue = value;

    return mpPutVarData(&info, 1);
}

STATUS GetSVar(UINT16 index, char *value)
{
    MP_VAR_INFO info;

    info.usType = 10/*MP_RESTYPE_VAR_S*/;
    info.usIndex = index;

    return mpGetVarData(&info, (long *)value, 1);
}

STATUS SetDVar(UINT16 index, long value)
{
    
```
```c
MP_VAR_DATA info;

info.usType = MP_RESTYPE_VAR_D;
info.usIndex = index;
info.ulValue = value;

return mpPutVarData(&info, 1);
}

STATUS SetMultiDVar(UINT16 startIdx, long *values, long nbr)
{
    int i;
    MP_VAR_DATA info[64];

    nbr = (nbr > 64) ? 64 : nbr;

    for(i=0; i<nbr; i++)
    {
        info[i].usType = MP_RESTYPE_VAR_D;
        info[i].usIndex = startIdx + i;
        info[i].ulValue = values[i];
    }

    return mpPutVarData(info, nbr);
}

STATUS GetMultiDVar(UINT16 startIdx, long *values, long nbr)
{
    int i;
    MP_VAR_INFO info[64];

    nbr = (nbr > 64) ? 64 : nbr;

    for(i=0; i<nbr; i++)
    {
        info[i].usType = MP_RESTYPE_VAR_D;
        info[i].usIndex = startIdx + i;
    }
```
Sample Programs

17.6 Robot System Monitor Samples and Control Programs

```c
return mpGetVarData(info, values, nbr);
}

STATUS GetXVar(UINT16 type, UINT16 index, char *value)
{
    MP_VAR_INFO info;
    info.usType = type;
    info.usIndex = index;
    return mpGetVarData(&info, (long *)value, 1);
}

STATUS GetIo(UINT32 ioAddr, USHORT *signal)
{
    MP_IO_INFO nfo;
    nfo.ulAddr = ioAddr;
    return mpReadIO(&nfo, signal, 1);
}

STATUS SetIo(UINT32 ioAddr, USHORT signal)
{
    MP_IO_DATA dta;
    dta.ulAddr = ioAddr;
    dta.ulValue = signal;
    return mpWriteIO(&dta, 1);
}

STATUS SetMultiIo(UINT32 ioStartAddr, USHORT *signal, long nbr)
{
    MP_IO_DATA dta[20];
    int i;
    if( nbr > 20 )
        return ERROR;
    for(i=0; i<nbr; i++)
        // rest of the method
```
17 Sample Programs
17.6 Robot System Monitor Samples and Control Programs

```c
{  
dta[i].ulAddr = ioStartAddr+i;
    dta[i].ulValue = signal[i];
}
return mpWriteIO(dta, nbr);
}

STATUS WriteIO(UINT32 addr, UINT16 value)
{
    MP_IO_DATA sData;
    sData.ulAddr = addr;
    sData.ulValue = value;

    return mpWriteIO(&sData, 1);
}

STATUS ReadIO(UINT32 addr, UINT16* value)
{
    MP_IO_INFO sData;
    sData.ulAddr = addr;

    return mpReadIO (&sData, value, 1);
}
```
17.6 Robot System Monitor Samples and Control Programs

17.6.2 User Variable Access Sample Program

The following is a sample program of the user variable access by using the API. Unlike the other APIs, API of the user variable access can reach to the variable in high speed. As for the following example, write and retrieve in the beginning of the B, I, D, R, S and P variables. It is verified that the contents of the writing and retrieving are the same.

```c
/** mpMain.c ****/
/* Copyright 2012 YASKAWA ELECTRIC All Rights reserved. */
#include "motoPlus.h"

#define VAR_TYPE_NUM 6 // B, I, D, R, S, P
#define COORD_NUM  6 // X, Y, Z, Rx, Ry, Rz

void mp_usr_var_acc_task();
int cmp_usr_var_info(MP_USR_VAR_INFO*, MP_USR_VAR_INFO*);
int cmp_pos_var_info(MP_P_VAR_BUFF *, MP_P_VAR_BUFF *);

int nTaskID1;
char var_type_name[VAR_TYPE_NUM][8] = {
    "B",
    "I",
    "D",
    "R",
    "S",
    "P"
};

void mpUsrRoot(int arg1, int arg2, int arg3, int arg4, int arg5,
                int arg6, int arg7, int arg8, int arg9, int arg10)
{
    nTaskID1 = mpCreateTask(MP_PRI_TIME_NORMAL, MP_STACK_SIZE,
                            (FUNCPTR)mp_usr_var_acc_task, arg1, arg2,
                            arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10);
    mpExitUsrRoot;
}

void mp_usr_var_acc_task()
{
    int i = CLEAR;
    STATUS status = ERROR;
    MP_USR_VAR_INFO winfo[VAR_TYPE_NUM];
    MP_USR_VAR_INFO rinfo[VAR_TYPE_NUM];
```
memset(&wInfo, 0, sizeof(wInfo));
memset(&rInfo, 0, sizeof(rInfo));

for (i = 0; i < VAR_TYPE_NUM; i++)
{
    wInfo[i].var_type = rInfo[i].var_type = MK_VAR_TYPE(MP_VAR_B + i);
    wInfo[i].var_no = rInfo[i].var_no = 0;
}

// set data
wInfo[0].val.b = 100;
wInfo[1].val.i = 200;
wInfo[2].val.d = 300;
wInfo[3].val.r = 400.004;
sprintf(wInfo[4].val.s, "Put S var");
wInfo[5].val.p.dtype = MP_ROBO_DTYPE;
wInfo[5].val.p.tool_no = 0;
wInfo[5].val.p.fig_ctrl = (MP_FIG_SIDE | MP_FIG_R180);
for (i = 0; i < COORD_NUM; i++)
{
    wInfo[5].val.p.data[i] = 100 + i;
}

FOREVER
{
    for (i = 0; i < VAR_TYPE_NUM; i++)
    {
        status = mpPutUserVars(&wInfo[i]);
        if (status != OK)
        {
            printf("#%s%03d Error: mpPutUserVars(): %d\n",
                   var_type_name[i], wInfo[i].var_no, status);
        }

        status = mpGetUserVars(&rInfo[i]);
        if (status != OK)
        {
            printf("#%s%03d Error: mpGetUserVars(): %d\n",
                   var_type_name[i], rInfo[i].var_no, status);
        }
    }
status = cmp_usr_var_info(&wInfo[i], &rInfo[i]);
if (status != OK)
{
    printf("%s var Error: %d: cmp_usr_var_info()\n",
           var_type_name[i], status);
}
else
{
    printf("Compare %s variable data OK\n",
           var_type_name[i]);
}
}

mpTaskDelay(5000);
}

int cmp_usr_var_info(MP_USR_VAR_INFO *info1, MP_USR_VAR_INFO *info2)
{
    int ret = OK;
    int var_type = MK_VAR_TYPE(info1->var_type);

    if (info1->var_type == info2->var_type)
    {
        switch(var_type)
        {
        case MP_VAR_B:
            if (info1->val.b != info2->val.b)
            {
                ret = ERROR;
            }
            break;
        case MP_VAR_I:
            if (info1->val.i != info2->val.i)
            {
                ret = ERROR;
            }
            break;
        case MP_VAR_D:
            break;
        case MP_VAR_L:
            if (info1->val.i != info2->val.i)
            {
                ret = ERROR;
            }
            break;
        case MP_VAR_D:
            break;
        }
    }
    return ret;
}
if (info1->val.d != info2->val.d)
{
    ret = ERROR;
}
break;
case MP_VAR_R:
    if (info1->val.r != info2->val.r)
    {
        ret = ERROR;
    }
    break;
case MP_VAR_S:
    if (strncmp(info1->val.s, info2->val.s, STR_VAR_SIZE) != 0)
    {
        ret = ERROR;
    }
    break;
case MP_VAR_P:
    ret = cmp_pos_var_info(&(info1->val.p), &(info2->val.p));
    break;
default:
    break;
}
else
{
    ret = ERROR;
}
return (ret);

int cmp_pos_var_info(MP_P_VAR_BUFF *buf1, MP_P_VAR_BUFF *buf2)
{
    int i = CLEAR;
    int ret = OK;

    if (buf1->dtype != buf2->dtype)
    {
        ret = ERROR;
    }
}
if ((buf1->dtype == MP_USER_DTYPE) &&
    (buf1->uf_no != buf2->uf_no))
{
    ret = ERROR;
}

if (buf1->tool_no != buf2->tool_no)
{
    ret = ERROR;
}

if ((buf1->dtype >= MP_BASE_DTYPE) &&
    (buf1->fig_ctrl != buf2->fig_ctrl))
{
    ret = ERROR;
}

for (i = CLEAR; i < MP_GRP_AXES_NUM; i++)
{
    if (buf1->data[i] != buf2->data[i])
    {
        ret = ERROR;
        break;
    }
}

return (ret);
17.7 Serial Communication (RS232C)

The following is the sample code for RS232C serial communication. The string sent by the client of Windows PC is converted to uppercase characters and echoed back.

17.7.1 Preparations

To check the sample program by MotoPlus, follow the procedure below.

Connect the PC and the YRC1000 by using a standard D-SUB 9-pin cable (cross cable).

Prepare a cable for RS232C with the specifications as follows:

<table>
<thead>
<tr>
<th>YRC1000 D-Sub 9 pin female</th>
<th>PC D-Sub 9 pin female</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD 2</td>
<td>3 SD</td>
</tr>
<tr>
<td>SD 3</td>
<td>2 RD</td>
</tr>
<tr>
<td>SG 5</td>
<td>7 RS</td>
</tr>
<tr>
<td>RS 7</td>
<td>8 CS</td>
</tr>
<tr>
<td>CS 8</td>
<td>5 SG</td>
</tr>
<tr>
<td>FG HOOD CASE</td>
<td></td>
</tr>
</tbody>
</table>

Prepare the client function on the PC side. HyperTerminal is useful for testing. Make settings as shown below.

17.7.2 Operation Check

Type lowercase characters (e.g. abc) by using the PC’s keyboard, and they will be converted to uppercase characters (e.g. ABC) and displayed in the console of HyperTerminal.
### Sample Code

```c
//mpMain.c
/
//
//This contains mpUsrRoot which is the entry point for your MotoPlus application

//ADDITIONAL INCLUDE FILES
//(Note that MotoPlus.h should be included in every source file)
#include "motoPlus.h"

//GLOBAL DATA DEFINITIONS
int nTaskID1;
int nTaskID2;

//FUNCTION PROTOTYPES
void mpTask1(void);
void mpTask2(int arg1, int arg2);

//FUNCTION DEFINITIONS
void mpUsrRoot(int arg1, int arg2, int arg3, int arg4, int arg5, int arg6,
    int arg7, int arg8, int arg9, int arg10)
{
    //TODO: Add additional initialization routines.

    //Creates and starts a new task in a separate thread of execution.
    //All arguments will be passed to the new task if the function
    //prototype will accept them.
    nTaskID1 = mpCreateTask(MP_PRI_TIME_NORMAL, MP_STACK_SIZE,
        (FUNCPTR)mpTask1,arg1, arg2, arg3, arg4,
        arg5, arg6, arg7, arg8, arg9, arg10);
    nTaskID2 = mpCreateTask(MP_PRI_TIME_NORMAL, MP_STACK_SIZE,
        (FUNCPTR)mpTask2,arg1, arg2, arg3, arg4,
        arg5, arg6, arg7, arg8, arg9, arg10);

    //Ends the initialization task.
    mpExitUsrRoot;
}

#define MP_BUFF_SIZE 1024
static char *strupr(char *string);
```
void mpTask1(void)
{
  //TODO: Add the code for this task
  MP_RS_CONFIG rs_config;
  int rc;
  ULONG snd_len;
  UCHAR snd_buff[MP_BUFF_SIZE];
  UCHAR rcv_buff[MP_BUFF_SIZE];
  ULONG rcv_len;
  LONG portHandle;

  printf("Simple RS232C test\n");

  rs_config.dataBit = mpRsDataBit_8;
  rs_config.stopBit = mpRsStopBit_one;
  rs_config.parity = mpRsParity_even;
  rs_config.baudRate = mpRsBaudrate_19200;

  printf("mpRsInit() : ");
  rc = mpRsInit(MP_RS_COM1, &rs_config);
  printf("rc = %d\n", rc);
  printf("mpRsOpen() : ");
  rc = mpRsOpen(MP_RS_COM1);
  printf("rc = %d\n", rc);

  if( rc >= 0 )
  {
    portHandle = rc;

    while(1)
    {
      memset( rcv_buff, 0, sizeof(rcv_buff));
      memset( snd_buff, 0, sizeof(snd_buff));

      rcv_len = mpRsRecv(portHandle, rcv_buff, MP_BUFF_SIZE);
      if( rcv_len > 0 )
      {
        snd_len = rcv_len;
      }
    }
17.7 Serial Communication (RS232C)

```c
memcpy( snd_buff, rcv_buff, sizeof(snd_buff) );
strupr( snd_buff );
rcv_len = mpRsSend(portHandle, snd_buff, snd_len);
}
}
mpRsClose(portHandle);
}

void mpTask2(int arg1, int arg2)
{
    //TODO: Add the code for this task
}

static char *strupr(char *string)
{
    int i;
    int len;

    len = strlen(string);
    for (i=0; i < len; i++)
    {
        if (isalpha(string[i]))
        {
            string[i] = toupper(string[i]);
        }
    }
    return (string);
}
17.8 Sensor Control Sample Program

As examples of simple path correction by the sensor control APIs, the following 4 applications are described here.

- mpMeiPutCorrPath(): Makes corrections to the Y-axis by 1 mm every 1 second.
- mpMeiPutSpdOverride(): Sets the speed override to 150%.
- mpMeiPutSpdOverride(): Cancels the speed override.
- mpMeiPutForcePathEnd(): Terminates forcibly the job step.

To change operating conditions by using the sensor control APIs, send a skill command from the job, have the MotoPlus application interpret the command contents to execute the target operation.

To execute each skill command, create the following 4 macro jobs.

<table>
<thead>
<tr>
<th>Number</th>
<th>Macro job name</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macro 1</td>
<td>MCRPATH</td>
<td>&quot;corrpath&quot;</td>
</tr>
<tr>
<td>Macro 2</td>
<td>MCRSPD1</td>
<td>&quot;spdoverride_on&quot;</td>
</tr>
<tr>
<td>Macro 3</td>
<td>MCRSPD2</td>
<td>&quot;spdoverride_off&quot;</td>
</tr>
<tr>
<td>Macro 4</td>
<td>MCRFEND</td>
<td>&quot;forcepathend&quot;</td>
</tr>
</tbody>
</table>

Create the job to send a skill command as described below.

1. Select {NEW JOB CREATE}, and create a robot macro job "MCRPATH".
2. Press [INFORM LIST], and select {OTHER}, then {SKILLSND}.

- Enter a command in the argument. ("corrpath" for the following example.)
  MotoPlus application receives and interprets the entered command to select a processing method.
– In addition, with SKILLSND command, the following attribute (ATTR) can be set.

<table>
<thead>
<tr>
<th>ATTR=</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>After executing the command, does not stop the look-ahead processing.</td>
</tr>
<tr>
<td>2</td>
<td>After executing the command, continues the JOB without waiting for the response of the MotoPlus application.</td>
</tr>
<tr>
<td>3</td>
<td>Enables both of the above two.</td>
</tr>
<tr>
<td>Other than above</td>
<td>After executing the command, stops the look-ahead processing and waits for the response of the MotoPlus application.</td>
</tr>
</tbody>
</table>

– Set the attribute as described in the following. Move the cursor to the instruction, then press [SELECT].

– For “ATTRIBUTE”, enter one of the above values in “ATTR=”. 
3. Register the created macro job as a macro instruction.

- For details about macro instructions, refer to “YRC1000 OPTIONS INSTRUCTIONS FOR MACRO COMMAND FUNCTION (HW1483378)”.

- Create the other 3 macro jobs in the same manner as above, then set as shown below.

4. After the registration is completed, “MACRO” is added to the display when [INFORM LIST] is pressed. Then, these macro instructions can be called.
The contents of the jobs created above are shown as follows:

“MCRCORON” Starts path correction.

/JOB
//NAME MCRCORON
//POS
///NPOS 0,0,0,0,0,0
//INST
///DATE 2010/10/22 10:58
///ATTR SC,RW,MC
///GROUP1 RB1
NOP
SKILLSND “corrpah”
END

“MCRCOROF” Calls when path correction is completed. ADVINIT clears the accumulated amount of path correction.

/JOB
//NAME MCRCOROF
//POS
///NPOS 0,0,0,0,0,0
//INST
///DATE 2010/10/22 10:58
///ATTR SC,RW,MC
///GROUP1 RB1
NOP
ADVINIT
END

“MCRSPDON” Starts speed override.

/JOB
//NAME MCRSPDON
//POS
///NPOS 0,0,0,0,0,0
//INST
///DATE 2010/10/22 12:49
17 Sample Programs
17.8 Sensor Control Sample Program

///ATTR SC,RW,MC
///GROUP1 RB1
NOP
SKILLSND “spdoverride_on”
END

“MCRSPDOF”   Ends speed override.

/JOB
//NAME MCRSPDOF
//POS
///NPOS 0,0,0,0,0,0
//INST
///DATE 2010/10/22 12:44
///ATTR SC,RW,MC
///GROUP1 RB1
NOP
SKILLSND “spdoverride_off”
END

“MCRFEND”

/JOB
//NAME MCRFEND
//POS
///NPOS 0,0,0,0,0,0
//INST
///DATE 2010/10/22 11:00
///ATTR SC,RW,MC
///GROUP1 RB1
NOP
SKILLSND “forcepathend”
END
17.8 Sensor Control Sample Program

- **Master job**

  "MJOBPATH.JBI"
  "MJOBSPD1.JBI"
  "MJOBSPD2.JBI"
  "MJOBFEND.JBI"

  The user coordinates are set, a macro is called, and a robot motion job is called.

  By setting the user coordinates to the point of the wrist-axis end when all the manipulator axes are set to 0 pulse, all the manipulators move in the same way.

  The called macro and motion job are different depending on the job.

  Because P variable 0 is used as the manipulator’s home position when setting the user coordinates, set the pulses of all the axes to 0 for P variable #P000 as shown below.
"MJOBPATH"

/JOB
//NAME MJOBPATH
//POS
///NPOS 0,0,3,0,0
//TOOL 0
//POSTYPE PULSE
///PULSE
P00000=0,0,0,0,0,0
///POSTYPE BASE
//RECTAN
///RCONF 1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
P00001=920.000,0.000,614.000,180.0000,0.0000,0.0000
P00002=820.000,-100.000,614.000,180.0000,0.0000,0.0000
///INST
///DATE 2010/10/22 11:02
///ATTR SC,RW
///GROUP1 RB1
NOP
' HOME POSITION
MOVJ P000 VJ=20.00
',
GETS PX001 $PX001
GETE D001 P001 (1)
ADD D001 100000
SETE P001 (1) D001
',
GETS PX002 $PX001
GETE D002 P002 (2)
ADD D002 -100000
SETE P002 (2) D002
',
MFRAME UF#(1) PX000 PX001 PX002
',
MACRO1 MJ#(0)  // MCRCORON
CALL JOB:TESTJOB1
MACRO1 MJ#(1)  // MCRCOROF

END
“MJOBSPDON”

/JOB
//NAME MJOBSPDON
//POS
///NPOS 0,0,0,3,0,0
///TOOL 0
///POSTYPE PULSE
///PULSE
P00000=0,0,0,0,0,0
///POSTYPE BASE
///RECTAN
///RCONF 1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
P00001=920.000,0.000,614.000,180.0000,0.0000,0.0000
P00002=820.000,-100.000,614.000,180.0000,0.0000,0.0000
///INST
///DATE 2010/10/22 12:50
///ATTR SC,RW
///GROUP1 RB1
NOP
' HOME POSITION
MOVJ P000 VJ=20.00
,
GETS PX001 $PX001
GETE D001 P001 (1)
ADD D001 100000
SETE P001 (1) D001
,
GETS PX002 $PX001
GETE D002 P002 (2)
ADD D002 -100000
SETE P002 (2) D002
,
MFRAME UF#(1) PX000 PX001 PX002
,
MACRO1 MJ#(2) // MCRSPDON
CALL JOB:TESTJOB2
END
“MJOBSPD2”

/JOB
//NAME MJOBSPD2
//POS
///NPOS 0,0,0,3,0,0
///TOOL 0
///POSTYPE PULSE
///PULSE
P00000=0,0,0,0,0,0
///POSTYPE BASE
///RECTAN
///RCONF 1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
P00001=920.000,0.000,614.000,180.000,0.000,0.000
P00002=820.000,-100.000,614.000,180.000,0.0000,0.0000
///INST
///DATE 2010/10/22 12:51
///ATTR SC,RW
///GROUP1 RB1
NOP
' HOME POSITION
MOVJ P000 VJ=20.00
',
GETS PX001 $PX001
GETE D001 P001 (1)
ADD D001 100000
SETE P001 (1) D001
',
GETS PX002 $PX001
GETE D002 P002 (2)
ADD D002 -100000
SETE P002 (2) D002
',
MFRAME UF#(1) PX000 PX001 PX002
',
MACRO1 MJ#(3)??// MCRSPDOF
CALL JOB:TESTJOB2
END
“MJOBFEND” Force-quit the interpolation motion.

/JOB
//NAME MJOBFEND
//POS
///NPOS 0,0,0,3,0,0
///TOOL 0
///POSTYPE PULSE
///PULSE
P00000=0,0,0,0,0,0
///POSTYPE BASE
///RECTAN
///RCONF 1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
P00001=920.000,0.000,614.000,180.0000,0.0000,0.0000
P00002=820.000,-100.000,614.000,180.0000,0.0000,0.0000
///INST
///DATE 2010/10/22 12:50
///ATTR SC,RW
///GROUP1 RB1
NOP
' HOME POSITION
MOVJ P000 VJ=20.00
,' GETS PX001 $PX001
GETE D001 P001 (1)
ADD D001 100000
SETE P001 (1) D001
,' GETS PX002 $PX001
GETE D002 P002 (2)
ADD D002 -100000
SETE P002 (2) D002
,' MFRAME UF#(1) PX000 PX001 PX002
,' MACRO1 MJ#(4)
CALL JOB:TESTJOB2
END
Robot motion job “TESTJOB1.JBI”

With this job, only the x-coordinate is changed from the user coordinates set by the master job to make the manipulator move right to left repeatedly.

Because P variable #P003 is used as the target value, set as shown below.

/JOB
//NAME TESTJOB1
//POS
//NPOS 0,0,0,2,0,0
//TOOL 0
//POSTYPE PULSE
//PULSE
P00000=0,0,0,0,0,0
//USER 1
//POSTYPE USER
//RECTAN
//RCONF 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
P00003=100.000,0.000,0.000,0.0000,0.0000,0.0000
//INST
//DATE 2010/10/20 15:54
//ATTR SC,RW
//GROUP1 RB1
NOP
*LABEL
MOVL P000 V=83.3
MOVL P003 V=83.3
JUMP *LABEL
END
Robot motion job “TESTJOB2.JBI”

With this job, the manipulator moves to the same target point as TESTJOB1.JBI only once, and does not repeat the movement.

P variable #P003 must be set just like TESTJOB1.JBI.

/JOB
//NAME TESTJOB2
//POS
///NPOS 0,0,0,2,0,0
///TOOL 0
///POSTYPE PULSE
///PULSE
P00000=0,0,0,0,0,0
///USER 1
///POSTYPE USER
///RECTAN
///RCONF 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
P00003=300.000,0.000,0.000,0.0000,0.0000,0.0000
///INST
///DATE 2010/10/22 11:18
///ATTR SC,RW
///GROUP1 RB1
NOP
MOVL P000 V=33.3
MOVL P003 V=33.3
END
The following is an example of the application which receives a skill command and changes operating conditions.

Description of each task:

- **CmdRcv_Task**
  Waits for the skill command from the job, and decides the process to run according to the command contents (comm_link).

- **CorrPath_Task**
  Runs the process decided by CmdRcv_Task.
  The process of changing operating conditions such as path correction and speed change must be executed every interpolation cycle of the manipulator's motion. Thus, this task's priority is set to MP_PRI_IO_CLK_TIME, and the interpolation cycle is notified by mpClkAnnounce to run the process.

```c
/* mp_main.c - MotoPlus Test Application for Real Time Process */
#include "motoPlus.h"

//FUNCTION PROTOTYPES
void CmdRcv_Task(void);
void CorrPath_Task(void);

//GLOBAL DATA DEFINITIONS
int nTaskID1;
int nTaskID2;
char Process = 0;

//FUNCTION DEFINITIONS
void mpUsrRoot(int arg1, int arg2, int arg3, int arg4, int arg5, int arg6, int arg7, int arg8, int arg9, int arg10)
{
  nTaskID1 = mpCreateTask(MP_PRI_TIME_NORMAL, MP_STACK_SIZE, (FUNCPTR)CmdRcv_Task, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10);

  nTaskID2 = mpCreateTask(MP_PRI_IP_CLK_TAKE, MP_STACK_SIZE, (FUNCPTR)CorrPath_Task, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10);

  mpExitUsrRoot;
}

//Command Receive Task
```
void CmdRcv_Task(void)
{
    SYS2MP_SENS_MSG msg;

    memset(&msg, CLEAR, sizeof(SYS2MP_SENS_MSG));

    FOREVER
    {
        mpEndSkillCommandProcess(MP_SL_ID1, &msg);

        if (mpReceiveSkillCommand(MP_SL_ID1, &msg) == ERROR)
        {
            printf("mpReceiveSkillCommand Error\n\r");
            mpTaskDelay(1000);
        }

        printf("main_comm %d\n\r", msg.main_comm);
        printf("sub_comm %d\n\r", msg.sub_comm);
        printf("exe_tsk %d\n\r", msg.exe_tsk);
        printf("exe_apl %d\n\r", msg.exe_apl);
        printf("comm %s\n\r", msg.cmd);

        switch(msg.main_comm)
        {
            case MP_SKILL_COMM:
                switch(msg.sub_comm)
                {
                    case MP_SKILL_SEND:
                        if(strcmp(msg.cmd, "corrpath") == 0 && Process == 0)
                        {
                            printf("Command Receive CorrPath\n\r");
                            Process = 1;
                        }
                        else if(strcmp(msg.cmd, "spdoverride_on") == 0
                                && Process == 0)
                        {
                            printf("Command Receive SpdOverride_on\n\r");
                            Process = 2;
                        }
                }
        }
    }
}  
else if(strcmp(msg.cmd, "spdoverride_off") == 0  
    && Process == 0)
    {
        printf("Command Receive SpdOverride_off\n\r");
        Process = 3;
    }
    
else if(strcmp(msg.cmd, "forcepathend") == 0  
    && Process == 0)
    {
        printf("Command Receive ForcePathEnd\n\r");
        Process = 4;
    }
    
else
    {
        Process = 0;
        printf("Unknown Command\n\r");
    }
break;

case MP_SKILL_END:
    Process = 0;
    printf("MP_SKILL_END\n\r");
break;

default:
    printf("Unknown Sub Command\n\r");
break;
}
break;

case MP_SL_RST_COMM:

    Process = 0;
    switch(msg.sub_comm)
    {
    case MP_SL_SOFTWARE_RST:
    
    }
17 Sample Programs
17.8 Sensor Control Sample Program

```c
void CorrPath_Task(void)
{
    int ret;
    long dy = 0;
    int cnt = 0;

    MP_POS_DATA corrpath_src_p;
    CTRLG_T ctrl_grp = 1;
    long spd_src_p[MP_GRP_NUM];

    printf("MP_SL_SOFTWARE_RST\n\r");
    break;

    case MP_SL_ALM_RST:
        printf("MP_SL_ALM_RST\n\r");
        break;

    case MP_START_SEG_CLK:
        printf("MP_START_SEG_CLK\n\r");
        break;

    case MP_SE_PRM_TRANS:
        printf("MP_SE_PRM_TRANS\n\r");
        break;

    default:
        printf("Unknown Sub Command\n\r");
        break;
    }
    break;

    default:
        printf("Unknown Main Command\n\r");
        break;
    }
    }
}
```
memset(&corrpath_src_p, CLEAR, sizeof(MP_POS_DATA));
memset(&spd_src_p, CLEAR, sizeof(long) * MP_GRP_NUM);

corrpath_src_p.ctrl_grp = 1;
corrpath_src_p.grp_pos_info[0].pos_tag.data[0] = 0x3f;
corrpath_src_p.grp_pos_info[0].pos_tag.data[2] = 0;
corrpath_src_p.grp_pos_info[0].pos[0] = 0;
corrpath_src_p.grp_pos_info[0].pos[1] = 0;
corrpath_src_p.grp_pos_info[0].pos[2] = 0;
corrpath_src_p.grp_pos_info[0].pos[3] = 0;
corrpath_src_p.grp_pos_info[0].pos[4] = 0;
corrpath_src_p.grp_pos_info[0].pos[5] = 0;

spd_src_p[0] = 15000;

FOREVER
{
    mpClkAnnounce(MP_INTERPOLATION_CLK);

    switch(Process)
    {
        //PutCorrPath
        case 1:
            corrpath_src_p.grp_pos_info[0].pos[1] = dy;
            ret = mpMeiPutCorrPath(MP_SL_ID1, &corrpath_src_p);
            if(ret != 0)
            {
                printf("mpMeiPutCorrPath Error  ret = %d\n\r", ret);
            }
            dy = 0;
            cnt++;

        //wait 1s
        if(cnt > 250)
        {
            cnt = 0;
            dy = 1000;
        }
    }
}
//SpdOverride_on
case 2:
    spd_src_p[0] = 15000;
    ret = mpMeiPutSpdOverride(MP_SL_ID1, ctrl_grp, spd_src_p);
    if(ret != 0)
    {
        printf("mpMeiPutSpdOverride Error ret = %d\n\r", ret);
    }
    Process = 0;
    cnt = 0;
    break;

//SpdOverride_off
case 3:
    //speed override cancel
    ret = mpMeiPutSpdOverride(MP_SL_ID1, OFF, NULL);
    if(ret != 0)
    {
        printf("mpMeiPutSpdOverride Error ret = %d\n\r", ret);
    }
    Process = 0;
    cnt = 0;
    break;

//ForcePathEnd
case 4:
    cnt++;

    //wait 3s
    if(cnt > 750)
    {
        ret = mpMeiPutForcePathEnd(MP_SL_ID1, ctrl_grp);
        if(ret != 0)
        {
            printf("mpMeiPutForcePathEnd Error ret = %d\n\r", ret);
        }
    }
17.8 Sensor Control Sample Program

```c
} else {
    printf("mpMeiPutForcePathEnd done\n\r");
    Process = 0;
    cnt = 0;
}
break;

default:
    dy = 0;
    cnt = 0;
    break;
}
}
```
17 Sample Programs

17.8 Sensor Control Sample Program

Execution results

As shown below, when “MJOBPATH.JBI” is executed, the y-coordinate of the manipulator’s current position with the MotoPlus application running is different from that with the MotoPlus application not running.

When MotoPlus application is NOT run

![Image of current position when MotoPlus is not run]

When MotoPlus application is run

![Image of current position when MotoPlus is run]
17 Sample Programs
17.8 Sensor Control Sample Program

When “MJOBSPD1.JBI” or “MJOBSPD2.JBI” is executed, display the job's cycle time to compare the values when the override is set and when it is canceled.

When “MJOBSPD1.JBI” is executed

When “MJOBSPD2.JBI” is executed
17 Sample Programs
17.8 Sensor Control Sample Program

When “MJOBFEND.JBI” is executed, stop the manipulator’s motion before it arrives at the target point to check the values when the MotoPlus application is run.

When MotoPlus application is NOT run

![Diagram showing current position when MotoPlus is not run]

When MotoPlus application is run

![Diagram showing current position when MotoPlus is run]
17.9 Servo Control Sample Program

The sample program of the servo control is described below. The sample program is in the folder “ServoControl”. This sample program consists of 2 tasks, moto_plus0_task and moto_plus1_task. The task moto_plus0_task issues a semaphore at periodic intervals, and the task moto_plus1_task executes the specified process each time it receives the semaphore. (By doing this, the process in moto_plus1_task is executed at any intervals.)

In moto_plus1_task, the servo control is executed according to the values set for the B variables and D variables. The B variables used in moto_plus1_task are shown in the following table.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>B000</td>
<td>Execution of servo control API</td>
<td>1: execution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Other than 1: no execution</td>
</tr>
<tr>
<td>B001</td>
<td>Semaphore sending cycle</td>
<td>0: 50 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Other than 0: Input value (unit: ms)</td>
</tr>
<tr>
<td>B002</td>
<td>Function to be executed</td>
<td>0: no function executed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: mpSvsStartTrqLimit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: mpSvsSetTrqLimit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3: mpSvsEndTrqLimit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4: mpSvsStartTrqCtrl</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5: mpSvsSetTrqCtrl</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6: mpSvsEndTrqCtrl</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7: mpSvsGetVelTrqFb</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8: mpCtrlGrpld2GrpNo</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Other than above: mpSvsForceInit</td>
</tr>
</tbody>
</table>

Set 1 to B000 to make the servo control executable.

With B001, the cycle of sending a semaphore can be set.

Set a value to B002 to execute the corresponding API of the servo control. When the execution is finished, 0 is set to B002 by the sample program.
The D variables are used to set the parameters to execute the APIs of the servo control. The D variables used are shown in the following table.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>D001</td>
<td>Torque unit</td>
<td>0: 0.01%, 1: $10^{-8}$ Nm</td>
</tr>
<tr>
<td>D004</td>
<td>Execution axis 1</td>
<td>0: no execution, 1: execution</td>
</tr>
<tr>
<td>D005</td>
<td>Execution axis 2</td>
<td>0: no execution, 1: execution</td>
</tr>
<tr>
<td>D006</td>
<td>Execution axis 3</td>
<td>0: no execution, 1: execution</td>
</tr>
<tr>
<td>D007</td>
<td>Execution axis 4</td>
<td>0: no execution, 1: execution</td>
</tr>
<tr>
<td>D008</td>
<td>Execution axis 5</td>
<td>0: no execution, 1: execution</td>
</tr>
<tr>
<td>D009</td>
<td>Execution axis 6</td>
<td>0: no execution, 1: execution</td>
</tr>
<tr>
<td>D010</td>
<td>Execution axis 1: group</td>
<td>Specify a control group number.</td>
</tr>
<tr>
<td>D011</td>
<td>Execution axis 1: axis</td>
<td>Specify a control axis number.</td>
</tr>
<tr>
<td>D012</td>
<td>Execution axis 1: upper limit value for torque limit</td>
<td></td>
</tr>
<tr>
<td>D013</td>
<td>Execution axis 1: lower limit value for torque limit</td>
<td></td>
</tr>
<tr>
<td>D014</td>
<td>Execution axis 1: torque command value</td>
<td></td>
</tr>
<tr>
<td>D015</td>
<td>Execution axis 2: group</td>
<td>Specify a control group number.</td>
</tr>
<tr>
<td>D016</td>
<td>Execution axis 2: axis</td>
<td>Specify a control axis number.</td>
</tr>
<tr>
<td>D017</td>
<td>Execution axis 2: upper limit value for torque limit</td>
<td></td>
</tr>
<tr>
<td>D018</td>
<td>Execution axis 2: lower limit value for torque limit</td>
<td></td>
</tr>
<tr>
<td>D019</td>
<td>Execution axis 2: torque command value</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D035</td>
<td>Execution axis 6: group</td>
<td>Specify a control group number.</td>
</tr>
<tr>
<td>D036</td>
<td>Execution axis 6: axis</td>
<td>Specify a control axis number.</td>
</tr>
<tr>
<td>D037</td>
<td>Execution axis 6: upper limit value for torque limit</td>
<td></td>
</tr>
<tr>
<td>D038</td>
<td>Execution axis 6: lower limit value for torque limit</td>
<td></td>
</tr>
<tr>
<td>D039</td>
<td>Execution axis 6: torque command value</td>
<td></td>
</tr>
</tbody>
</table>

In this sample program, up to 6 axes for which the servo control is executed can be specified. Specify the axes by using D004 to D009.

Specify the control group number of the control group for which the servo control is executed in D010, D015, ... D035.

Specify the axis number of the axis for which the servo control is executed in D011, D016, ... D036. Use a number which begins with 0 for the axis number. (Enter 0 for the first axis.)
The values in D010, D015, ..., D035 and D011, D016, ..., D036 are used to set the member variables of g_svs_grp_config, which is the parameter of mpSvsStartTrqLimit(), mpSvsSetTrqLimit(), mpSvsStartTrqCtrl(), and mpSvsSetTrqCtrl().

\[
g_{\text{svs grp config}}.\text{ctrl grp} \mid= 0x01 \ll \text{value in D010, D015, ..., D035 (grp_no)}
\]

\[
g_{\text{svs grp config}}.\text{axes config}[	ext{value in D010, D015, ..., D035}]
\mid= 0x01 \ll \text{value in D011, D016, ..., D036 (axis_no)}
\]

Enter the torque limit value in D012, D017, ..., D037 and D013, D018, ..., D038. At this time, specify the unit of the torque by using D001.

The torque limit value and the unit of the torque are set as the member variables of trq_lmt, which is the parameter of mpSvsSetTrqLimit().

\[
\text{trq lmt.data[ grp_no ].max[axis_no] = value in D012, D017, ..., D037}
\]

\[
\text{trq lmt.data[ grp_no ].min[axis_no] = value in D013, D018, ..., D038}
\]

\[
\text{trq lmt.unit = value in D001}
\]

Set the torque command value in D014, D019, ..., D039 in the same way as the torque limit value. The torque command value and the unit of the torque are set as the member variables of trq_ctl, which is the parameter of mpSvsSetTrqCtrl().

\[
\text{trq ctl.data[ grp_no ][axis_no] = value in D014, D019, ..., D039}
\]

\[
\text{trq ctl.unit = value in D001}
\]

**<Sample programs (main part only)>**

```c
/**** motoSvapiTest.c ****/

void mpUsrRoot(int arg1, int arg2, int arg3, int arg4, int arg5,
    int arg6, int arg7, int arg8, int arg9, int arg10)
{
    tid1 = mpCreateTask(MP_PRI_TIME_NORMAL, MP_STACK_SIZE,
        (FUNCPTR)moto_plus0_task, arg1, arg2, arg3, arg4,
        arg5, arg6, arg7, arg8, arg9, arg10);
    tid2 = mpCreateTask(MP_PRI_TIME_NORMAL, MP_STACK_SIZE,
        (FUNCPTR)moto_plus1_task, arg1, arg2, arg3, arg4,
        arg5, arg6, arg7, arg8, arg9, arg10);
    semid = mpSemBCreate(SEM_Q_FIFO, SEM_EMPTY);
    mpExitUsrRoot; // (or) mpSuspendSelf;
}

void moto_plus0_task(void)
{
```
17  Sample Programs
17.9  Servo Control Sample Program

(void moto_plus1_task(void)
{

    /* Initialization */
    run_cnt = 0;
    svhsvs_handle = 0;

    while (1)
    {
        run_cnt++;
        status = mpSemTake(semid, WAIT_FOREVER);
        if (status == ERROR)
            printf("semTake Error! [%d]n", run_cnt);

        /* Gets the variable "B" */
        memset(&BVar, CLEAR, sizeof(MULTIBVAR));
        rt = GetMultiBVar(BVAR_START_NUM, (long*)&BVar, BVAR_SIZE);
        CmdType = BVar.CmdType;   // B000
        NewCmd = BVar.NewCmd;   // B002

        /* Between task variables */
        if (BVar.CycleTime == 0)
            g_CycleTime = CYCLETIME/RTC TIME; // 50[msec]
        else
            g_CycleTime = BVar.CycleTime;

        /* CmdType(B000) of the value switch the contents of command */
        switch (CmdType)
        {
        }}
case NO_OP:
    break;

/* Command from the variable       */
case VAR_OPE: /* */

if( (LstCmd == NewCmd) && (NewCmd != 0) )
{
    rt = SetBVar(2, CLEAR);
    break;
}

switch (NewCmd) /* B002 */
{
    case VAR_OPE_0: /* When the idle processing */

        /* Variable(B002) has completed or cleared */
        if(LstCmd != 0)
            LstCmd = CLEAR;
        break;

    case VAR_OPE_1: /* mpSvsStartTrqLimit */

        (Retrieval of D variable and setting of the parameter information of
         mpSvsStartTrqLimit(). The code is omitted.)

        svs_handle = mpSvsStartTrqLimit(&g_svs_grp_config);
        rt = SetBVar(2, CLEAR);   //B002
        LstCmd = VAR_OPE_1;

        break;

    case VAR_OPE_2: /* mpSvsSetTrqLimit */

        if (svs_handle > 0)
        {
            (Retrieval of D variable and setting of the parameter information of
             mpSvsSetTrqLimit(). The code is omitted.)

            rt = mpSvsSetTrqLimit(svs_handle, &g_svs_grp_config, &trq_lmt);
rt = SetBVar(2, CLEAR);
LstCmd = VAR_OPE_2;
break;

case VAR_OPE_3: /* mpSvsEndTrqLimit */

rt_func = mpSvsEndTrqLimit(svs_handle);
rt = SetBVar(2, CLEAR);
LstCmd = VAR_OPE_3;
break;

case VAR_OPE_4: /* mpSvsStartTrqCtrl */

(svss handle = mpSvsStartTrqCtrl(&g_svsgrp_config);
rt = SetBVar(2, CLEAR);
LstCmd = VAR_OPE_4;
break;

case VAR_OPE_5: /* mpSvsSetTrqCtrl */

if (svs_handle > 0)
{
(svss handle = mpSvsSetTrqCtrl(svss handle, &g_svsgrp_config, &trq_ctl);
}
rt = SetBVar(2, CLEAR);
LstCmd = VAR_OPE_5;
break;

case VAR_OPE_6: /* mpSvsEndTrqCtrl */
rt_func = mpSvsEndTrqCtrl(svs_handle);
rt = SetBVar(2, CLEAR);
LstCmd = VAR_OPE_6;
break;

case VAR_OPE_7: /* mpSvsGetVelTrqFb */

memset(&vel_fb, CLEAR, sizeof(long) * MP_GRP_NUM * MP_GRP_AXES_NUM);
memset(&trq_fb, CLEAR, sizeof(MP_TRQ_CTL_VAL));

(Retrieval of D variable and setting of the parameter information of mpSvsGetVelTrqFb(). The code is omitted.)

rt = mpSvsGetVelTrqFb(vel_fb, &trq_fb);

if(rt >= 0)
{ 
  (The retrieved speed and torque information is shown. The code is omitted.)
}

rt = SetBVar(2, CLEAR);
LstCmd = VAR_OPE_7;
break;

case VAR_OPE_8: /* mpCtrlGrpld2GrpNo */

puts("[GrouplD] -> [GrpNo]");
puts("---------------");
printf("MP_R1_GID -> %2d\n", mpCtrlGrpld2GrpNo(MP_R1_GID));
printf("MP_R2_GID -> %2d\n", mpCtrlGrpld2GrpNo(MP_R2_GID));
printf("MP_B1_GID -> %2d\n", mpCtrlGrpld2GrpNo(MP_B1_GID));
printf("MP_B2_GID -> %2d\n", mpCtrlGrpld2GrpNo(MP_B2_GID));
printf("MP_S1_GID -> %2d\n", mpCtrlGrpld2GrpNo(MP_S1_GID));
printf("MP_S2_GID -> %2d\n", mpCtrlGrpld2GrpNo(MP_S2_GID));
rt = SetBVar(2, CLEAR);
LstCmd = VAR_OPE_8;
break;
default:
  /* Perform a forced reset */
  rt = mpSvsForceInit();
  LstCmd = NewCmd;
  rt = SetBVar(2, CLEAR);
  break;

  } break;

default:
  break;

  }

  }

  }
17.10 File Control Sample Program

The following is the sample program which uses the file control APIs.
It creates the file “TESTFILE1.dat” on the RAM drive, and performs functions such as reading from or writing to the file, seeking, retrieving file information, opening a directory, reading an entry, etc.

17.10.1 Operation Check

Results such as the following can be checked via Telnet.

-> mpRead Success
  File Write Test

mpIoctl Success
  File name = SRAM:\TESTFILE1.dat
mpRead Success
  Write Test

mpFstat Complete
  st_dev = 250466920
  st_ino = 0
  st_mode = 33279
  st_nlink = 1
  st_uid = 0
  st_gid = 0
  st_rdev = 0
  st_size = 17
  st_atime = 1284681600
  st_mtime = 1284724294
  st_ctime = 1284724294
  st_blksize = 512
  st_blocks = 1
  st_attrib = 0

mpReaddir Success
  TESTFILE1.dat
  End
17.10.2 Sample Program

//mpMain.c

//
//This contains mpUsrRoot which is the entry point for your MotoPlus application

//ADDITIONAL INCLUDE FILES
//(Note that MotoPlus.h should be included in every source file)
#include "motoPlus.h"

//GLOBAL DATA DEFINITIONS
int nTaskID1;

//FUNCTION PROTOTYPES
void mpTask1(void);

//FUNCTION DEFINITIONS
void mpUsrRoot(int arg1, int arg2, int arg3, int arg4, int arg5, int arg6, int arg7, int arg8, int arg9, int arg10)
{
    nTaskID1 = mpCreateTask(MP_PRI_TIME_NORMAL, MP_STACK_SIZE, (FUNCPTR)mpTask1,
        arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10);

    //Ends the initialization task.
    mpExitUsrRoot;
}

void mpTask1(void)
{
    char filename[100];

    char str[100];
    char ReadStr[100];

    int fd;
    DIR* pDir;

    char tempc[256];
int ret;
struct dirent* ret_dir;
STATUS ret_sts;
struct stat pStat;

memset(filename, '0', 100);
memset(str, '0', 100);
strcpy(filename, MP_DRAM_DEV_DOS);
strcat(filename, "\TESTFILE1.dat");

mpTaskDelay(20000);

ret_sts = mpRemove(filename);
if(ret_sts != 0)
{
    printf("mpRemove Error\n\r");
}

fd = mpCreate(filename, O_RDWR);
if(fd >= 0)
{
    strcpy(str, "File Write Test\n\r");
    ret = mpWrite(fd, str, (size_t)strlen(str));
    if(ret == -1)
    {
        printf("mpWrite Error \n\r");
    }

    ret = mpLseek(fd, 0, SEEK_SET);
    if(ret == -1)
    {

printf("mpLseek Error \n\r");
}

memset(ReadStr, '0', 100);

ret = mpRead(fd, ReadStr, 100);
if(ret == -1)
{
    printf("mpRead Error \n\r");
}
else
{
    printf("mpRead Success\n\r");
    printf(" %s\n\r", ReadStr);
}

ret = mpIoctl(fd, FIOGETNAME, (int)tempc);
if(ret == -1 )
{
    printf("mpioctl Error \n\r");
}
else
{
    printf("mpioctl Success\n\r");
    printf(" File name = %s\n\r", tempc);
}

ret = mpLseek(fd, 5, SEEK_SET);
if(ret == -1)
{
    printf("mpLseek Error \n\r");
}

memset(ReadStr, '0', 100);

ret = mpRead(fd, ReadStr, 100);
if(ret == -1)
17 Sample Programs
17.10 File Control Sample Program

```c
{
    printf("mpRead Error \n\r");
}
else
{
    printf("mpRead Success\n\r");
    printf(" %s\n\r", ReadStr);
}

ret_sts = mpFstat(fd, &pStat);
if(ret_sts == -1)
{
    printf("mpFstat Error\n\r");
}
else
{
    printf("mpFstat Complete\n\r");
    printf("st_dev = %u\n\r", pStat.st_dev);
    printf("st_ino = %u\n\r", pStat.st_ino);
    printf("st_mode = %u\n\r", pStat.st_mode);
    printf("st_nlink = %d\n\r", pStat.st_nlink);
    printf("st_uid = %d\n\r", pStat.st_uid);
    printf("st_gid = %d\n\r", pStat.st_gid);
    printf("st_rdev = %u\n\r", pStat.st_rdev);
    printf("st_size = %u\n\r", pStat.st_size);
    printf("st_atime = %u\n\r", pStat.st_atime);
    printf("st_mtime = %u\n\r", pStat.st_mtime);
    printf("st_ctime = %u\n\r", pStat.st_ctime);
    printf("st_blksize = %u\n\r", pStat.st_blksize);
    printf("st_blocks = %u\n\r", pStat.st_blocks);
    printf("st_attrib = %u\n\r\n\r", pStat.st_attrib);
}

ret_sts = mpClose(fd);
if(ret_sts == -1)
{
    printf("mpClose Error \n\r");
}
```
17 Sample Programs
17.10 File Control Sample Program

```c
else
{
    printf("mpCreate Error!\n");
}

pDir = mpOpendir(MP_DRAM_DEV_DOS);

if(pDir > 0)
{
    ret_dir = mpReaddir(pDir);
    if(ret_dir->d_name !="")
    {
        printf("mpReaddir Success\n");
        printf(" %s\n", ret_dir->d_name);
    }
    else
    {
        printf("mpReaddir Failed\n");
    }
    ret_sts = mpClosedir(pDir);

    if(ret_sts == -1)
    {
        printf("mpClosedir Failed\n");
    }
}
else
{
    printf("mpOpendir Failed\n");
}

printf("End\n");
```
17.11 Existing File Access Sample Program

The following sample program reads the job “TESTJOB1.JBI” on the RAM drive, edits the content and writes it to the controller as the job with another name “TESTJOB2.JBI”.

17.11.1 Preparation

Create the job “TESTJOB1.JBI” in advance.
The job content can be anything.

17.11.2 Operation Check

-> ExFile Test Start

Read TESTJOB1 End
Change Job Name
mpGetFileCount FileCount = 2
mpGetFileName FileName = TESTJOB1.JBI
mpGetFileName FileName = TESTJOB2.JBI

17.11.3 Sample Program

//mpMain.c

//This contains mpUsrRoot which is the entry point for your MotoPlus application

//ADDITIONAL INCLUDE FILES
//(Note that MotoPlus.h should be included in every source file)
#include "motoPlus.h"

//GLOBAL DATA DEFINITIONS
int nTaskID1;

//FUNCTION PROTOTYPES
void mpTask1(void);
//FUNCTION DEFINITIONS
void mpUsrRoot(int arg1, int arg2, int arg3, int arg4, int arg5, int arg6, int arg7, int arg8, int arg9, int arg10)
{
    nTaskID1 = mpCreateTask(MP_PRI_TIME_NORMAL, MP_STACK_SIZE, (FUNCPTR)mpTask1,
                        arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10);
    mpExitUsrRoot;
}

void mpTask1(void)
{
    int fd;
    int i;
    char R_FileData[3000];
    char W_FileData[3000];
    char filename[100];
    LONG FileCnt;
    LONG ret;
    size_t File_len = 0;
    memset(R_FileData, '\0', sizeof(char) * 3000);
    memset(W_FileData, '\0', sizeof(char) * 3000);
    mpTaskDelay(30000);

    printf("ExFile Test Start\n\r");
    ret = mpSaveFile(MP_DRV_ID_DRAM, "", "TESTJOB1.JBI");
    if(ret == -1)
        printf("mpSaveFile Error\n\r");
    memset(filename, '\0', sizeof(char) * 100);
    strcpy(filename, MP_DRAM_DEV_DOS);
    strcat(filename, "\TESTJOB1.JBI");
    fd = mpOpen(filename, O_RDONLY, 0);
    if(fd == -1)
        printf("mpOpen Error\n\r");
Sample Programs
17.11 Existing File Access Sample Program

```c
ret = mpRead(fd, R_FileData, 3000);
if(ret == -1)
    printf("mpRead Error\n\r");

ret = mpClose(fd);
if(ret == -1)
    printf("mpClose Error\n\r");

printf("\n\r");
File_len = strlen(R_FileData);
strcpy(W_FileData, R_FileData);

printf("\n\r");
printf("Read TESTJOB1 End\n\r\n\r");

//Change Job Name
for(i=0; i<File_len; i++)
{
    if(strncmp(&R_FileData[i], "//NAME", 6) == 0)
    {
        printf("Change Job Name\n\r");
        strncpy(&W_FileData[i], "//NAME TESTJOB2", 15);
        break;
    }
}

printf("%s", W_FileData);
memset(filename, '0', sizeof(char) * 100);
strcpy(filename, MP_DRAM_DEV_DOS);
strcat(filename, "\TESTJOB2.JBI");

fd = mpCreate(filename, O_RDWR);
if(fd < 0)
    printf("mpCreate Error\n\r");

ret = mpWrite(fd, W_FileData, File_len);
if(ret == -1)
```
printf("mpWrite Error\n\r");

ret = mpClose(fd);
if(ret == -1)
    printf("mpClose Error\n\r");

ret = mpLoadFile(MP_DRV_ID_DRAM, "", "TESTJOB2.JBI");
if(ret != 0)
    printf("mpLoadFile Error %d\n\r", ret);

ret = mpRefreshFileList(MP_EXT_ID_JBI);
if(ret == -1)
    printf("mpRefreshFileList Error\n\r");

FileCnt = mpGetFileCount();
if(ret == -1)
    printf("mpGetFileCount Error\n\r");
else
    printf("mpGetFileCount FileCount = %d\n\r", FileCnt);

for(i = 0; i < FileCnt; i++)
{
    ret = mpGetFileName(i, filename);
    if(ret == -1)
        printf("mpGetFileName Error\n\r");
    else
        printf("mpGetFileName FileName = %s\n\r", filename);
}
}
17.12 Coordinate Conversion Sample Program

In this section, the sample program of the coordinate conversion is explained and it is stored in "Kinematics" folder. This program is monitoring the robot coordinate system present value. Also, the positions of each axis are monitored at every 1mm when the robot is moved from the present position to X direction. And the robot axis coordinate value on the base coordinate system at the final reaching point is calculated.

//**** Kinematics ****/

#include "motoPlus.h"

void mpTask1();

//GLOBAL DATA DEFINITIONS
int nTaskID1;

void mpUsrRoot(int arg1, int arg2, int arg3, int arg4, int arg5, int arg6, int arg7, int arg8, int arg9, int arg10)
{
    nTaskID1 = mpCreateTask(MP_PRI_TIME_NORMAL, MP_STACK_SIZE, (FUNCPTR)mpTask1, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10);
    mpExitUsrRoot;
}

void mpTask1(void)
{
    int i;
    int rc;
    int err_counter;
    int grp_no;
    long pulse[MP_GRP_AXES_NUM];
    long angle[MP_GRP_AXES_NUM];
    MP_COORD rob_coord;
    MP_COORD bas_coord;
    MP_COORD coord;
    MP_FRAME rob_frame = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
17.12 Coordinate Conversion Sample Program

```c
MP_FRAME bas_frame = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
MP_FRAME frame = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
MP_CTRL_GRP_SEND_DATA sData;
MP_PULSE_POS_RSP_DATA rPls_data;
MP_FB_PULSE_POS_RSP_DATA rFB_data;

err_counter = 0;
printf("Sample Kinematics test\n");

while (1)
{
    printf("\n--Robot minute movement--\n");
    sData.sCtrlGrp = 0;
    memset(&rFB_data, 0, sizeof(rFB_data));
    mpGetFBPulsePos(&sData, &rFB_data);

    grp_no = mpCtrlGrpId2GrpNo(MP_R1_GID);

    memset(pulse, 0, sizeof(pulse));
    printf("mpConvFBPulseToPulse()\n");
    while ((rc = mpConvFBPulseToPulse(grp_no, rFB_data.lPos, pulse)) < 0)
    {
        err_counter++;
        if (rc != E_KINEMA_FAILURE)
        {
            printf("function err\n");
            return;
        }
    }

    mpTaskDelay(1);
}
printf("pulse = \n");
for (i = 0; i < MP_GRP_AXES_NUM; i++)
{
    printf("%ld", pulse[i]);

    if (i != (MP_GRP_AXES_NUM - 1))
        printf(",");
}
```

17 Sample Programs
17.12 Coordinate Conversion Sample Program

```c
{
    printf(" , ");
}
else
{
    printf("\n");
}
}

printf("mpConvPulseToAngle()\n");
memset(angle, 0, sizeof(angle));
while ((rc = mpConvPulseToAngle(grp_no, pulse, angle)) < 0)
{
    err_counter++;
    if (rc != E_KINEMA_FAILURE)
    {
        printf("function err\n");
        return;
    }

    mpTaskDelay(1);
}
printf("angle = {");
for (i = 0; i < MP_GRP_AXES_NUM; i++)
{
    printf("%ld", angle[i]);
    if (i != (MP_GRP_AXES_NUM - 1))
    {
        printf(" , ");
    }
    else
    {
        printf("\n");
    }
}

printf("mpConvAxesToCartPos()\n");
memset(&rob_coord, 0, sizeof(rob_coord));
```
while ((rc = mpConvAxesToCartPos(grp_no, angle, 0, NULL, &rob_coord)) < 0)
{
    err_counter++;
    if (rc != E_KINEMA_FAILURE)
    {
        printf("function err\n");
        return;
    }

    mpTaskDelay(1);
}

printf("rob_coord = {%ld, %ld, %ld, %ld, %ld, %ld, %ld, %ld} \n", rob_coord.x, rob_coord.y, rob_coord.z, rob_coord.rx, rob_coord.ry, rob_coord.rz, rob_coord.ex1, rob_coord.ex2);

printf("mpConvCartPosToAxes()\n");
do {
    while (mpConvCartPosToAxes(grp_no, &rob_coord, 0, 0, angle, MP_KINEMA_DELTA, angle) < 0)
    {
        err_counter++;
        if (rc != E_KINEMA_FAILURE)
        {
            printf("function err\n");
            return;
        }

        mpTaskDelay(1);
    }

    printf("angle = {%};
    for (i = 0; i < MP_GRP_AXES_NUM; i++)
    {
        printf("%ld", angle[i]);

        if (i != (MP_GRP_AXES_NUM - 1))
        {
            printf(",");
        }
17.12 Coordinate Conversion Sample Program

```c
    } else {
        printf("\n");
    }
}

    } while ((rob_coord.x = (rob_coord.x + (1 * MP_DIST_UNIT))) < (500 * MP_DIST_UNIT));

printf("err_counter = %d\n", err_counter);

printf("n--base coordinates position--\n");
sData.sCtrlGrp = 8;
memset(&rPls_data, 0, sizeof(rPls_data));
mpGetPulsePos(&sData, &rPls_data);

if ((grp_no = mpCtrlGrpId2GrpNo(MP_B1_GID)) >= 0) {
    printf("mpConvPulseToAngle()\n");
    memset(angle, 0, sizeof(angle));
    while ((rc = mpConvPulseToAngle(grp_no, rPls_data.lPos, angle)) < 0) {
        err_counter++;
        if (rc != E_KINEMA_FAILURE)
            {
                printf("function err\n");
                return;
            }

mpTaskDelay(1);
}

printf("angle = {\n");
for (i = 0; i < MP_GRP_AXES_NUM; i++) {
    printf("%ld", angle[i]);
}

if (i != (MP_GRP_AXES_NUM - 1)) {
    printf(",");
}
```
17 Sample Programs
17.12 Coordinate Conversion Sample Program

```c
else
{
    printf("\n");
}

printf("mpConvAxesToCartPos()\n");
memset(&bas_coord, 0, sizeof(bas_coord));
while ((rc = mpConvAxesToCartPos(grp_no, angle, 0, NULL, &bas_coord)) < 0)
{
    err_counter++;
    if (rc != E_KINEMA_FAILURE)
    {
        printf("function err\n");
        return;
    }

    mpTaskDelay(1);
}

printf("bas_coord = {\ld, \ld, \ld, \ld, \ld, \ld, \ld} \n",
        bas_coord.x, bas_coord.y, bas_coord.z, bas_coord.rx,
        bas_coord.ry, bas_coord.rz, bas_coord.ex1,
        bas_coord.ex2);

printf("mpZYXeulerToFrame()\n");
while ((rc = mpZYXeulerToFrame(&rob_coord, &rob_frame)) < 0)
{
    err_counter++;
    printf("function err\n");
    return;
}

printf("rob_frame = {\f, \f, \f, \f, \f, \f, \f} \n",
        rob_frame.nx, rob_frame.ny, rob_frame.nz,
        rob_frame.ox,
        rob_frame.oy, rob_frame.oz, rob_frame.ax,
        rob_frame.ay,
```
rob_frame.az, rob_frame.px, rob_frame.py, rob_frame.pz);

while ((rc = mpZYXeulerToFrame(&bas_coord, &bas_frame)) < 0)
{
    err_counter++;
    printf("function err\n");
    return;
}

printf("bas_frame = {%f, %f, %f, %f, %f, %f, %f, %f, %f, %f, %f, %f} \n",
    bas_frame.nx, bas_frame.ny, bas_frame.nz, bas_frame.ox,
    bas_frame.oy, bas_frame.oz, bas_frame.ax, bas_frame.ay,
    bas_frame.az, bas_frame.px, bas_frame.py, bas_frame.pz);

printf("mpMulFrame()\n");
while ((rc = mpMulFrame(&bas_frame, &rob_frame, &frame)) < 0)
{
    err_counter++;
    printf("function err\n");
    return;
}

printf("Mul_frame = {%f, %f, %f, %f, %f, %f, %f, %f, %f, %f, %f, %f} \n",
    frame.nx, frame.ny, frame.nz, frame.ox, frame.oy,
    frame.oz, frame.ax, frame.ay, frame.az, frame.px, frame.py, frame.pz);

printf("mpFrameToZYXeuler()\n");
memset(&coord, 0, sizeof(coord));
while ((rc = mpFrameToZYXeuler(&frame, &coord)) < 0)
{
    err_counter++;
    printf("function err\n");
    return;
}
17 Sample Programs
17.12 Coordinate Conversion Sample Program

```c
printf("base_pos = {%ld, %ld, %ld, %ld, %ld, %ld, %ld, %ld} \n",
    coord.x, coord.y, coord.z, coord.rx,
    coord.ry, coord.rz, coord.ex1, coord.ex2);
}

printf("err_counter = %d\n", err_counter);
err_counter = 0;

mpTaskDelay(10000);
}
```
<table>
<thead>
<tr>
<th>Alarm Number</th>
<th>Alarm Name</th>
<th>Subcode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1020</td>
<td>MOTOPLUS APPLICATION LOAD ERROR</td>
<td>1</td>
<td>The number of loaded files exceeds the limit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>The memory is insufficient. (Available memory area is less than 3 Mbyte.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Failed to open the directory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Load failure (Failed to open the file.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Load failure (Undefined symbol)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Load failure (Others: application overloaded)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>Failed to initialize the API library.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>No user root task</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>Failed to create the user root task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>Failed to create RAM-Disk.</td>
</tr>
<tr>
<td>4478</td>
<td>MM TASK NO RESPONSE</td>
<td></td>
<td>The process requested from the MotoPlus application to the MM task was not completed within the specified time.</td>
</tr>
<tr>
<td>4479</td>
<td>MOTOPLUS MM TASK WATCHDOG ERROR</td>
<td></td>
<td>The man-machine task does not run for 3 seconds or more.</td>
</tr>
</tbody>
</table>