Introduction

This manual describes the details of the various library functions that support the user's software development when using the M60 Series custom release system. Please read this manual thoroughly before starting development, and refer to it as necessary for the applications and purposes of the developed functions.

All functions of the M60 Series custom release system are described in this manual. However, there may be limits to the functions that can be used due to the CNC model or option configuration. Check the CNC specifications before starting.

The following manuals are available as reference. Refer to them as required.

- Custom Release (APLC) Programming Manual ..................... BNP-B2217
- PLC Programming Manual (Ladder) ............................... BNP-B2212
- PLC Programming Manual (Ladder with MELSEC tool) ........ BNP-B2269
- PLC Interface Manual .................................................. BNP-B2211
- DDB Interface Manual .................................................. BNP-B2214

Precautions for using this manual

This manual is written for persons having an understanding of C language. An effort has been made to describe special handling, however, if the item is not described in this manual, please interpret it as “not possible”.
[List of Custom Release Library Functions]

The functions that can be used in the custom release program are as shown below.

1. Screen Display Function I/F

1.1 Custom Screen Control Functions

<table>
<thead>
<tr>
<th>No.</th>
<th>Function name</th>
<th>Outline of function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p_ope()</td>
<td>Function for controlling the transition of the custom screens</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>pcopini()</td>
<td>Function for executing initialization of custom screen when power is turned ON.</td>
<td>1</td>
</tr>
</tbody>
</table>

1.2 Display Request Functions

<table>
<thead>
<tr>
<th>No.</th>
<th>Function name</th>
<th>Outline of function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>enquet()</td>
<td>Function for requesting display</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>TXTYPE</td>
<td>Requests text/title data display</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>CTTYPE</td>
<td>Requests continuous text/title data display</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>NDTYPE</td>
<td>Requests numeric data</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>CNTYPE</td>
<td>Requests continuous numeric data</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>VRATYPE</td>
<td>VRAM direct change (type A)</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>VRBTYPE</td>
<td>VRAM direct change (type B)</td>
<td>18</td>
</tr>
<tr>
<td>8</td>
<td>CLTYPE</td>
<td>Requests deletion (line)</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>WLCTYPE</td>
<td>Requests deletion (matrix)</td>
<td>21</td>
</tr>
<tr>
<td>10</td>
<td>INDTYPE</td>
<td>Requests indirect display</td>
<td>22</td>
</tr>
</tbody>
</table>

1.3 Graphic Display Request Functions

<table>
<thead>
<tr>
<th>No.</th>
<th>Function name</th>
<th>Outline of function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>grastart()</td>
<td>Function for graphic drawing pre-process</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>gramask()</td>
<td>Function for graphic mask control (Compatible with display mask)</td>
<td>26</td>
</tr>
<tr>
<td>3</td>
<td>graclr()</td>
<td>Function for graphic draw deletion (Compatible with all screen clear)</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>enquet()</td>
<td>Function for requesting display</td>
<td>28</td>
</tr>
<tr>
<td>5</td>
<td>GLBCP</td>
<td>Sets drawing start point</td>
<td>29</td>
</tr>
<tr>
<td>6</td>
<td>GLBSLS</td>
<td>Selects line type and plane</td>
<td>30</td>
</tr>
<tr>
<td>7</td>
<td>GLBALIN</td>
<td>Draws absolute value line</td>
<td>31</td>
</tr>
<tr>
<td>8</td>
<td>GLBRLIN</td>
<td>Draws relative value line</td>
<td>32</td>
</tr>
<tr>
<td>9</td>
<td>GLBRPLN</td>
<td>Draws continuous multi-line</td>
<td>33</td>
</tr>
<tr>
<td>10</td>
<td>GLBCRCL</td>
<td>Draws circle</td>
<td>34</td>
</tr>
<tr>
<td>11</td>
<td>GLBAARC</td>
<td>Draws absolute value arc</td>
<td>35</td>
</tr>
<tr>
<td>12</td>
<td>GLBAMLN</td>
<td>Draws non-continuous multi-line</td>
<td>36</td>
</tr>
<tr>
<td>13</td>
<td>GLBAMAR</td>
<td>Draws non-continuous arc</td>
<td>38</td>
</tr>
<tr>
<td>14</td>
<td>graend()</td>
<td>Function for graphic drawing post-process</td>
<td>40</td>
</tr>
</tbody>
</table>
### 1.4 Screen Display Auxiliary Functions

<table>
<thead>
<tr>
<th>No.</th>
<th>Function name</th>
<th>Outline of function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>dspend()</td>
<td>Checks display end</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>smenhi()</td>
<td>Highlights menu</td>
<td>42</td>
</tr>
<tr>
<td>3</td>
<td>smenud()</td>
<td>Displays menu (TXDATA)</td>
<td>44</td>
</tr>
<tr>
<td>4</td>
<td>squrst()</td>
<td>Resets display request</td>
<td>46</td>
</tr>
<tr>
<td>5</td>
<td>texers()</td>
<td>Erases text screen</td>
<td>47</td>
</tr>
<tr>
<td>6</td>
<td>ikeyset()</td>
<td>Reads setting area control data</td>
<td>47</td>
</tr>
<tr>
<td>7</td>
<td>ocurini()</td>
<td>Sets cursor position data</td>
<td>48</td>
</tr>
<tr>
<td>8</td>
<td>omakkcb()</td>
<td>Sets setting area control data</td>
<td>48</td>
</tr>
<tr>
<td>9</td>
<td>ostclr()</td>
<td>Clears setting area buffer</td>
<td>49</td>
</tr>
<tr>
<td>10</td>
<td>setdisp()</td>
<td>Displays setting area title data</td>
<td>49</td>
</tr>
<tr>
<td>11</td>
<td>skey()</td>
<td>Controls the setting area data</td>
<td>50</td>
</tr>
<tr>
<td>12</td>
<td>cursor()</td>
<td>Cursor control</td>
<td>55</td>
</tr>
<tr>
<td>13</td>
<td>scrst40()</td>
<td>Sets 40-character mode screen</td>
<td>56</td>
</tr>
<tr>
<td>14</td>
<td>scrst80()</td>
<td>Sets 80-character mode screen</td>
<td>57</td>
</tr>
</tbody>
</table>

### 2. DDB I/F

#### 2.1 CNC Data Read/Write Functions

<table>
<thead>
<tr>
<th>No.</th>
<th>Function name</th>
<th>Outline of function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ddbrd()</td>
<td>Reads CNC data</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>ddbwt()</td>
<td>Writes CNC data</td>
<td>63</td>
</tr>
<tr>
<td>2</td>
<td>smkonb()</td>
<td>Reads O, N, B data (Compatible with system 1)</td>
<td>64</td>
</tr>
<tr>
<td>3</td>
<td>scaldr()</td>
<td>Calendar function</td>
<td>68</td>
</tr>
<tr>
<td>4</td>
<td>sgetmes()</td>
<td>Message data read function (operation message, setting error)</td>
<td>71</td>
</tr>
<tr>
<td>5</td>
<td>oexsech()</td>
<td>Operation search (Compatible with system 1)</td>
<td>74</td>
</tr>
<tr>
<td>6</td>
<td>ievarrd()</td>
<td>Reads CNC variables (IEEE double)</td>
<td>78</td>
</tr>
<tr>
<td>7</td>
<td>ievarwt()</td>
<td>Writes CNC variables (IEEE double)</td>
<td>80</td>
</tr>
<tr>
<td>8</td>
<td>ievarclear()</td>
<td>Clears CNC variables</td>
<td>81</td>
</tr>
</tbody>
</table>

### 3. Machine Control I/F

#### 3.1 PLC Device Access

<table>
<thead>
<tr>
<th>No.</th>
<th>Function name</th>
<th>Outline of function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>set</td>
<td>Sets the bit device</td>
<td>83</td>
</tr>
<tr>
<td>2</td>
<td>rst</td>
<td>Resets the bit device</td>
<td>83</td>
</tr>
<tr>
<td>3</td>
<td>tst</td>
<td>Tests the bit device</td>
<td>83</td>
</tr>
<tr>
<td>4</td>
<td>set</td>
<td>Sets the word device</td>
<td>84</td>
</tr>
<tr>
<td>5</td>
<td>lst</td>
<td>Tests the word device</td>
<td>84</td>
</tr>
<tr>
<td>6</td>
<td>lst</td>
<td>Sets the long device</td>
<td>85</td>
</tr>
<tr>
<td>7</td>
<td>lst</td>
<td>Tests the long device</td>
<td>86</td>
</tr>
</tbody>
</table>

#### 3.2 PLC Device High-speed Access

<table>
<thead>
<tr>
<th>No.</th>
<th>Function name</th>
<th>Outline of function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>melplcBset</td>
<td>Sets the bit device</td>
<td>89</td>
</tr>
<tr>
<td>2</td>
<td>melplcBrst</td>
<td>Resets the bit device</td>
<td>89</td>
</tr>
<tr>
<td>3</td>
<td>melplcBtst</td>
<td>Tests the bit device</td>
<td>89</td>
</tr>
<tr>
<td>4</td>
<td>melplcWset</td>
<td>Sets the word device</td>
<td>90</td>
</tr>
<tr>
<td>5</td>
<td>melplcWtst</td>
<td>Tests the word device</td>
<td>90</td>
</tr>
<tr>
<td>6</td>
<td>melplcLset</td>
<td>Sets the long device</td>
<td>91</td>
</tr>
<tr>
<td>7</td>
<td>melplcLlst</td>
<td>Tests the long device</td>
<td>91</td>
</tr>
</tbody>
</table>
4. File Release I/F

4.1 File Data Input/Output Functions

<table>
<thead>
<tr>
<th>No.</th>
<th>Function name</th>
<th>Outline of function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>prmake()</td>
<td>Registers machining program No.</td>
<td>94</td>
</tr>
<tr>
<td>2</td>
<td>prrenm()</td>
<td>Changes machining program No.</td>
<td>95</td>
</tr>
<tr>
<td>3</td>
<td>prdele()</td>
<td>Deletes machining program</td>
<td>96</td>
</tr>
<tr>
<td>4</td>
<td>prdir()</td>
<td>Machining program list</td>
<td>97</td>
</tr>
<tr>
<td>5</td>
<td>prcmwt()</td>
<td>Writes a comment</td>
<td>98</td>
</tr>
<tr>
<td>6</td>
<td>prcmrd()</td>
<td>Reads a comment</td>
<td>99</td>
</tr>
<tr>
<td>7</td>
<td>plwrit()</td>
<td>Writes one block of machining program</td>
<td>100</td>
</tr>
<tr>
<td>8</td>
<td>plread()</td>
<td>Reads one block of machining program</td>
<td>101</td>
</tr>
<tr>
<td>9</td>
<td>plinst()</td>
<td>Inserts one block in machining program</td>
<td>102</td>
</tr>
<tr>
<td>10</td>
<td>pldele()</td>
<td>Deletes one block of machining program</td>
<td>103</td>
</tr>
<tr>
<td>11</td>
<td>plcunt()</td>
<td>Counts No. of machining program blocks</td>
<td>103</td>
</tr>
<tr>
<td>12</td>
<td>plrunblk()</td>
<td>Reads machining program being executed</td>
<td>104</td>
</tr>
<tr>
<td>13</td>
<td>prinfo()</td>
<td>Reads machining program information</td>
<td>106</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(No. of registered programs, No. of stored characters)</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>prunchk()</td>
<td>Checks program being run</td>
<td>107</td>
</tr>
</tbody>
</table>

5. General Functions

5.1 Data Conversion Functions

<table>
<thead>
<tr>
<th>No.</th>
<th>Function name</th>
<th>Outline of function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>abtol()</td>
<td>Converts binary character string into 32-bit numeric value</td>
<td>110</td>
</tr>
<tr>
<td>2</td>
<td>ahtol()</td>
<td>Converts hexadecimal character string into 32-bit numeric value</td>
<td>110</td>
</tr>
<tr>
<td>3</td>
<td>atobcd()</td>
<td>Converts decimal character string into 32-bit BCD</td>
<td>111</td>
</tr>
<tr>
<td>4</td>
<td>atol()</td>
<td>Converts decimal character string into 32-bit numeric value</td>
<td>112</td>
</tr>
<tr>
<td>5</td>
<td>atos()</td>
<td>Converts decimal character string into 16-bit numeric value</td>
<td>113</td>
</tr>
<tr>
<td>6</td>
<td>dchtoa()</td>
<td>Converts hexadecimal into a character string</td>
<td>114</td>
</tr>
<tr>
<td>7</td>
<td>ltoa()</td>
<td>Converts decimal into a character string</td>
<td>115</td>
</tr>
<tr>
<td>8</td>
<td>ostrcmp()</td>
<td>Compares two character strings</td>
<td>116</td>
</tr>
<tr>
<td>9</td>
<td>satol()</td>
<td>Converts decimal character string with decimal point into 32-bit numeric data</td>
<td>117</td>
</tr>
</tbody>
</table>
Contents

1. Screen Display Release I/F .................................................................................................. 1
  1.1 Custom Screen Control Functions .................................................................................... 1
  1.1.1 p_ope () Screen Transition and Screen Function Control Function .................. 1
  1.1.2 pcoini () Initialization Function ............................................................................. 1
  1.2 Display Request Functions ............................................................................................. 2
  1.2.1 enquet () Character Display .................................................................................. 2
  1.3 Graphics Drawing Functions .......................................................................................... 23
  1.3.1 grastart () Initialization of Environment for Graphics ........................................... 25
  1.3.2 gramask () Graphic Mask Control ........................................................................ 26
  1.3.3 graclr () Clearing of Graphics ............................................................................... 27
  1.3.4 enquet () Request of Drawing Graphics .................................................................. 28
  1.3.5 graend () End of Graphics ..................................................................................... 40
  1.4 Screen Display Auxiliary Functions ............................................................................. 41
  1.4.1 dspend () Check of Display Completion ............................................................... 42
  1.4.2 smenhi () Menu Highlight ....................................................................................... 42
  1.4.3 smenud () Menu Display .................................................................................... 44
  1.4.4 squrst () Reset of Display Request ........................................................................ 46
  1.4.5 texers () Clear Text Data from Screen .................................................................. 47
  1.4.6 ikeyset () Read Setting Area Control Data Information ..................................... 47
  1.4.7 ocurini () Set Cursor Position Data ....................................................................... 48
  1.4.8 omakkcb () Set Setting Area Control Data ......................................................... 48
  1.4.9 ostclr () Clear Setting Area Buffer ...................................................................... 49
  1.4.10 setdisp () Display Setting Area Title Data .......................................................... 49
  1.4.11 skey () Setting Area Processing Main ................................................................. 50
  1.4.12 cursor () Cursor Display Request ...................................................................... 55
  1.4.13 scrst40 () Set 40-character Mode Screen ......................................................... 56
  1.4.14 scrst80 () Set 80-character Mode Screen ............................................................ 57

2. DDB I/F .......................................................................................................................... 62
  2.1 CNC Data Read/Write Functions ................................................................................. 62
  2.1.1 ddbrd (), ddbwt () CNC Data Read/Write Functions (Type 2) .......................... 63
  2.1.2 smkonb () O, N, B Data Read ............................................................................... 64
  2.1.3 scadr () Calendar Function ................................................................................... 68
  2.1.4 sgetmes () Message Function ............................................................................... 71
  2.1.5 oexsrch () Operation Search ................................................................................ 74
  2.1.6 ievarrd () CNC Variable Read Function (IEEE double type) ......................... 78
  2.1.7 ievarwt () CNC Variable Write Function (IEEE double type) ....................... 80
  2.1.8 ievarclear () CNC Variable Clear Function ....................................................... 81
3. Machine Control I/F

3.1 PLC Device Accessing Functions
   3.1.1 set_  Setting the Bit Device................. 83
   3.1.2 rst_  Resetting the Bit Device............. 83
   3.1.3 tst_  Testing the Bit Device.............. 83
   3.1.4 set_  Setting the Word Device.......... 84
   3.1.5 tst_  Testing the Word Device.......... 84
   3.1.6 lset_ Setting the Long Data in the Word Device... 85
   3.1.7 lst_  Testing the Long Data........... 86

3.2 PLC Device High-speed Access Functions
   3.2.1 melplcBset( ) Setting the Bit Device...... 89
   3.2.2 melplcBrst( )Resetting the Bit Device.... 89
   3.2.3 melplcBtst( ) Testing the Bit Device...... 89
   3.2.4 melplcWset( ) Setting the Word Device.... 90
   3.2.5 melplcWtst( ) Testing the Word Device.... 90
   3.2.6 melplcLset( ) Setting the Long Data..... 91
   3.2.7 melplcLst( ) Testing the Long Data...... 91

4. File Release I/F

4.1 File Data Input/Output Functions
   4.1.1 prmake () Register New Machining Program No........... 94
   4.1.2 prrena () Change Machining Program No............ 95
   4.1.3 prdele () Delete Machining Program.............. 96
   4.1.4 prdir () Read Machining Program Registration State... 97
   4.1.5 prcmwt () Write Comment to Machining Program..... 98
   4.1.6 prcmrd () Read Comment from Machining Program.... 99
   4.1.7 plwrit () Write One Block of Machining Program...... 100
   4.1.8 plread () Read One Block of Machining Program...... 101
   4.1.9 plinst () Insert One Block in Machining Program.... 102
   4.1.10 pldele () Delete One Block of Machining Program.... 103
   4.1.11 plcnt () Read Number of Machining Program Blocks... 103
   4.1.12 plrunblk () Read Machining Program Being Executed... 104
   4.1.13 prinfo () Read Machining Program Information...... 106
   4.1.14 prunchk() Check Program Being Run............. 107

5. General Functions

5.1 Data Conversion Functions
   5.1.1 abtol () Convert Binary Character String into 32-bit Numeric Value.... 110
   5.1.2 ahtol () Convert Hexadecimal Character String into 32-bit Numeric Value........... 110
   5.1.3 atobcd () Convert Decimal Character String into 32-bit BCD.. 111
   5.1.4 atol () Convert Decimal Character String into 32-bit Numeric Value........... 112
   5.1.5 atos () Convert Decimal Character String with Decimal Point into 16-bit Numeric Value........... 113
   5.1.6 dchtoa () Convert Hexadecimal into a Character String......................... 114
   5.1.7 ltoa () Convert Decimal into a Character String.................... 115
   5.1.8 ostrcmp () Compare Two Character Strings................................. 116
   5.1.9 satol () Convert Decimal Character String with Decimal Point into 32-bit Numeric Value........... 117
1. Screen Display Release I/F

1.1 Custom Screen Control Functions

1.1.1 p_ope()  Screen Transition and Screen Function Control Function
This is software created by Mitsubishi and has a function to control "M_OPE". Mainly screen transition and screen functions are controlled.
(Note) "M_OPE" is software used by the user to create original screens, etc.

1.1.2 pc locals () Initialization Function
This function carries out initialization when the power is turned ON.
The user's initialization function "mopeini ()" is called.
(Note) "mopeini ()" is an initialization function carried out by the user, and is called once.
- Use this to initialize the custom RAM area, etc.
1.2 Display Request Functions

1.2.1 enquet () Character Display

Note that as shown below, the start position and end position of the data designated with the display request function count column at the upper left of the screen as 1.

**40-character mode**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>to 40</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td></td>
<td>to 80</td>
</tr>
</tbody>
</table>

18 lines

<table>
<thead>
<tr>
<th></th>
<th>681</th>
<th>to 720</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

40 columns

**80-character mode**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>to 80</th>
</tr>
</thead>
<tbody>
<tr>
<td>81</td>
<td></td>
<td>to 160</td>
</tr>
</tbody>
</table>

18 lines

<table>
<thead>
<tr>
<th></th>
<th>1361</th>
<th>to 1440</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

80 columns

When the cursor is displayed, the lower left will be 0 instead of 1.
enquet  Function for requesting display

Function:  Display of characters on the CRT is requested.
Use the command according to the type of characters to be displayed.
The character display commands are shown below.

Command list

<table>
<thead>
<tr>
<th>Command name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TXTYPE</td>
<td>Requests text/title data display</td>
</tr>
<tr>
<td>CTTYPE</td>
<td>Requests continuous text/title data display</td>
</tr>
<tr>
<td>NDTYPE</td>
<td>Requests numeric data</td>
</tr>
<tr>
<td>CNTYPE</td>
<td>Requests continuous numeric data</td>
</tr>
<tr>
<td>VRATYPE</td>
<td>Requests VRAM direct change (type A)</td>
</tr>
<tr>
<td>VRBTYPE</td>
<td>Requests VRAM direct change (type B)</td>
</tr>
<tr>
<td>CLTYPE</td>
<td>Requests deletion (line)</td>
</tr>
<tr>
<td>WCLTYPE</td>
<td>Requests deletion (matrix)</td>
</tr>
<tr>
<td>INDTYPE</td>
<td>Requests indirect display</td>
</tr>
</tbody>
</table>

Format:  
ret = enquet (id, type, flag, datptr, windid);

long id;  Possessory right ID : Specify "pcoptb.ocb.scrnumb".
char type; Command name : Refer to the command list.
char flag; Attribute change data : Specify 0.
char *datpt; Display data pointer : Pointer for data to be displayed
long windid; Window ID  : Specify 1.

Program example

TXDATA 1ot0[] = {6,6,C_RK|NORMATR,4,0x00,0x00,(char **)d1ot00,
                 26,26,C_YK|NORMATR,16,0x00,0x00,(char **)d1ot01,
                 15,15,C_WK|NORMATR,144,0x00,0x00,(char **)d1ot02,
                 2,2,C_WK|NORMATR,245,0x00,0x00,(char **)d1ot03,
                 0};
enquet (pcoptb.ocb.scrnumb,TXTYPE,0,1ot0,1L);
(Note)
For the actual screen display, display process in the CNC is carried out. The display request function carries out a process to set data in the display data table on display process in the CNC. Thus, in the following cases, characters will not be displayed.

- When display data is created on a stack (internal variable). The characters may be displayed due to the processing timing, but the display data must be created on the ROM or on an RAM (global variable) for which the contents are guaranteed.

- When display request reset is carried out before the display is completed. When the display request is reset, the display data that has not been displayed will be invalid. When carrying out display request and display request reset in succession, reset the display request after confirming that the display has been completed.
**TXTYPE**

**Function:** Display of single tile/text data

### Data structure

- **Number of data items**
- **Window width**
- **Display attribute**
- **Display start position**
- **Address type**
- **Setting status**
- **Address information**
- **Pointer**

**Character string ROM/(RAM)**

**NUL character**

This structure name is defined as "TXDATA".

(1) Number of data items, Window width and Display start position

- Number of data items: 16 (8 × 2)
- Window width: 8
- Display start position: 87

In the above case:

- Number of data items: 16 (8 × 2)
- Window width: 8
- Display start position: 87
(2) Display attribute

<table>
<thead>
<tr>
<th>Color CRT</th>
<th>Bit No.</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>B</td>
<td>G</td>
<td>R</td>
<td>B</td>
<td>G</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Black</td>
<td>0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td>0 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Green</td>
<td>0 1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yellow</td>
<td>0 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blue</td>
<td>1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Magenta</td>
<td>1 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cyan</td>
<td>1 1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>White</td>
<td>1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example) 0×60 (Character color) Yellow, (Background color) Black
☆ When highlight designation is set to 1, the color designated for the characters will be displayed as the background color, and the color designated for the background will be displayed as the character colors.

Example) 0×64 (Character color) Black, (Background color) Yellow

| Monochrome CRT | Bit No. | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|               |         |   |   |   |   |   |   |   |   |
|               |         | B | G |    |    |    |    |   |
| Black         | 0 0 0   |   |   |    |    |    |    |   |
| Green         | 0 1 1   |   |   |    |    |    |    |   |
| Cyan          | 1 1 0   |   |   |    |    |    |    |   |
| White         | 1 1 1   |   |   |    |    |    |    |   |

Example) 0×05 Reverse and enlarge

☆ Enlargement designation is valid for the following characters.
If data containing characters for which enlargement designation is not valid, nothing will be displayed.
1) Uppercase alphabet characters, numbers (A to Z, 0 to 9)
2) +, -, (, )
1. Screen Display Release I/F

1.2 Display Request Functions

(3) Address type

<table>
<thead>
<tr>
<th>Bit No.</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Indirect</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Continuous display level**
  - 0: No continuous display (Data is displayed only once)
  - 1: Display each time display task runs (At 60msec interval)
    - Display at 120msec intervals.
  - 2, 3, 4, 5, 6, 7: Display at 240msec intervals.

For the title data (fixed), the data is fixed at 0X00 (direct, and) is not continuously displayed.

* The following figure shows the meanings of the continuous display levels (1 to 7).

![Image of display timing for continuous display level](image)

When designating the continuous display level, the user should make the display task processing time be equal each time.
(4) Setting status
   Set the setting status is fixed at 0x00.

(5) Address information pointer
   The address is set in the address information pointer, so take care when creating the text display data.

Program example

```c
char dkpa10[] = {"[Machining parameter]"};

TXDATA kpal[] = {12,12, C_YK|NORMATR,1,0x00,0x00,(char **)dkpa10,
                  1,1,C_YK|NORMATR,264,0x00,0x00,(char **)&paramet.common.kyoko,
                  0};

Structure's end code
enquet(pcoptb.ocb.scrnumb,TXTYPE,0,kpal,1L);
```
**CTTYPE**

**Function:** Display of continuous title/text data

Data structure:

- Number of data items
- Window width
- Display attribute
- Display start position
- Address type
- Setting status
- Address information
- Pointer
- Number of skips (horizontal)
- Number of skips (vertical)
- Number of displays (direction)
- Number of continuous data items
- Number of bytes to next data

This structure name is defined as "CTXDAT".

(6) Refer to the single title/text data.

(7) Number of displays (direction)

<table>
<thead>
<tr>
<th>Bit No.</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of displays</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Display direction</td>
<td>Vertical</td>
<td>Horizontal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Screen Display Release I/F

1.2 Display Request Functions
The continuous title/text data is used when the following conditions are satisfied as shown below:
There are character string display areas on the CRT that has a set pattern (vertical skip, horizontal skip).
The attributes such as the character display are the same.
The address storing the display character string has a set interval from the top.

In the above case:
- Number of horizontal skips: 6
- Number of vertical skips: 2
- Display direction: Vertical
- Number of displays: 3
- Number of data items: 5
- Number of bytes to next data: 3

(Number of displays (direction) = 3)
The following figure shows a comparison of single title data and continuous title data.

Program example

```c
char dsamp0[] = {"AAAAA"};
char dsamp1[] = {"AAAAA"};
TXDATA samp[] = {5,1,C_WK|NORMATR,83,0x00,0x00,(char **)dsamp0,
                 5,1,C_WK|NORMATR,89,0x00,0x00,(char **)dsamp1,
                 0};
```

Example of single title data

```c
char dsamp0[] = {"A"};
CTXDATA asamp[] = {1,1,C_WK|NORMATR,83.0x00,0x00,(char **)dsamp0,6,1,TATEDIR|5,10,0,
                   0};
```

Example of continuous title data

Comparison of single title data
NDTYPE

**Function:** Display of single numeric data

**Data structure**
- Numeric information
- Pointer
- Display attribute
- Address type
- Display start position
- Variable
- Pointer

**ROM/RAM**
- Display type
- Conversion type
- Number of digits
- Max. value number
- Min. value number
- Setting status

**Variable**
- Direct
- RAM
- Variable
- Pointer

(3) This structure name is defined as "NDATA".
(4) This structure name is defined as "NTYP".

(9) **Display attribute**
Refer to the single title/text data.

(10) **Address type**
Bit No. 7 6 5 1 0

- 0: Direct
- 1: Indirect

Continuous display level (0 to 7)
Refer to item (3) in the single title/text data.

(11) **Display start position**

Figure: Diagram of the data structure and address type with explanations.
1. Screen Display Release I/F

1.2 Display Request Functions

(12) Display type

```
<table>
<thead>
<tr>
<th>Bit No.</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

Variable type
0: char 1: short 2: long 3: double
```

1/2 conversion of numeric value
0: None 1: Added

Fixed to 0

0 0 No digit change according to inch or metric.
0 1 View parameter initialization state, and change digit.
1 1 View machine parameter constant inch state, and change digit.

Sign
0: None 1: Added

☆ The digit change according to inch or metric indicates the following:
When metric is designated for the numeric value with 5 digits in the integer section and 3 digits in the decimal section, the value is converted for inches with 4 digits in the integer section and 4 digits in the decimal section according to the parameter state.

☆ Normally CNC axis data is numerically displayed as 0xAA.

(13) Conversion type
0: Binary display
1: Decimal display
2: Hexadecimal display
3: bit display
4: BCD display

(14) Number of digits

```
<table>
<thead>
<tr>
<th>Bit No.</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

Number of decimal digits (0 to 15)
The decimal point is excluded.
Number of integer digits (0 to 15)
The sign is excluded.
```

☆ If the conversion type is binary, hexadecimal, bit or BCD, the number of digits will be stored without being divided into the decimal section and integer section.

(15) Max. value number, min. value number and setting status
The max. value number and min. value number are fixed to 0.
The setting status is fixed to 0x00.
An example of the single numeric value data is shown below.

```c
NTYP tkpa12[] = {0x82,1,0x53,0,0,0x00};
NTYP tkpa13[] = {0x22,1,0x53,0,0,0x00};
NTYP tkpa14[] = {0x00,1,0x30,0,0,0x00};
NTYP tkpa15[] = {0x22,1,0x53,0,0,0x00};
NTYP tkpa16[] = {0x22,1,0x53,0,0,0x00};
NDATA nkpa1[] = {tkpa12,C_YKINORMATR,0x00,495,(char *)&paramet.common.kkakudo,
                 tkpa13,C_YKINORMATR,0x00,736,(char *)&paramet.common.scale,
                 tkpa14,C_YKINORMATR,0x00,982,(char *)&paramet.common.ovride,
                 tkpa15,C_YKINORMATR,0x00,1056,(char *)&paramet.common.angle,
                 tkpa16,C_YKINORMATR,0x00,1136,(char *)&paramet.common.inn,
                 0};
```

Example of single numeric value display data
### CNTYPE

**Function:** Display of continuous numeric data

- **Display of continuous numeric data**
  - **Data structure**
    - Numeric value information Pointer
    - Display attribute
    - Address type
    - Display start position
  - **Variable Pointer**
    - Number of skips (horizontal)
    - Number of skips (vertical)
    - Number of displays (direction)
    - Number of continuous data items
    - Number of bytes to next data

- **ROM/RAM**
  - **Direct**
    - Display type
    - Conversion type
    - Number of digits
    - Max. value number
    - Min. value number
    - Setting status
  - **Indirect**
    - Head pointer
    - Struct indirect
    - Do not use this struct indirect.

- **Continuous data**

---

This structure name is defined as "CNDATA".
This structure name is defined as "NTYP".

(16) Refer to the single numeric data.
(17) Refer to the continuous title/text data.

---

The continuous numeric data is used when all the following conditions are satisfied as shown above:
The display position of the numeric data on the CRT has a set pattern (vertical skip, horizontal skip).
The attributes such as the display color are the same.
The address storing the numeric data has a set interval from the top.
VRATYPE

Function: VRAM direct change A data

Data structure

<table>
<thead>
<tr>
<th>Display start position</th>
<th>~</th>
<th>~</th>
<th>(18)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-character change data</td>
<td>~</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>1-character change data</td>
<td>~</td>
<td>~</td>
<td></td>
</tr>
</tbody>
</table>

The end code for the VRAM direct change A data is -1.

The VRAM direct change A data is used to rewrite the characters displayed on the CRT as shown below. The data is configured of the 1-character change data between the display start position and end code (-1).
1. Screen Display Release I/F
1.2 Display Request Functions

(18) 1 character change attribute

```
<table>
<thead>
<tr>
<th>Character color</th>
<th>B</th>
<th>G</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Red</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Green</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Yellow</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Blue</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Magenta</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Cyan</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>White</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Background color</th>
<th>B</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Green</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Blue</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Cyan</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

- When highlight designation is set to 1, the color designated for the characters will be displayed as the background color, and the color designated for the background will be displayed as the character colors.

(Note)

For the 1-character change data, if all bits corresponding to the character color and background color are 0, the display color will not be changed. If all bits corresponding to the character codes are 0, the character will not be changed.

An example of the VRAM direct change A data is shown below.

```
short vrama[] = {81, 0xa041, 0xa042, 0xa043, -1};
```
The VRAM direct change B data is used to change the colors of the displayed characters and background on the CRT. The data is configured of the display start position and n-character change data up to the end code (–1).

(19) n-character change data

<table>
<thead>
<tr>
<th>Character color</th>
<th>B</th>
<th>G</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Red</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Green</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Yellow</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Blue</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Magenta</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Cyan</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>White</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
An example of the VRAM direct change B data is shown below.

```c
short vramb[] = {81,0xa005,161,0xa005,241,0xa005,-1};
```

End code
**CLTYPE**

**Function:** Clearing of data

- **Data structure**
  - Clear start position
  - Clear end position

This structure name is defined as "CDATA".

An example of the clear data is shown below.

```c
CDATA erase[] = {5, 105, 0} ;
```

The above clear data can be written as follows.

```c
short erase[] = {5, 105, 0} ;
```
WCLTYPE

Function: Clearing of data in window

Data structure

- Clear start position
- Clear end position

This structure name is defined as "WCDATA".

An example of the clear window data is shown below.

\[
\text{WCDATA\ werase[]} = \{5, 105, 0\}
\]

The above clear window data can be written as follows.

\[
\text{short\ werase[]} = \{5, 105, 0\}
\]
INDTYPE

Function: Indirect display request
When the display request function is called once, multiple display requests are carried out.

Data structure

<table>
<thead>
<tr>
<th>Display data 1</th>
<th>QUE type No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display data 2</td>
<td>Attribute change data</td>
</tr>
<tr>
<td>Display data 3</td>
<td>Dummy</td>
</tr>
<tr>
<td>~</td>
<td>Dummy</td>
</tr>
<tr>
<td>~</td>
<td>Display data pointer</td>
</tr>
</tbody>
</table>

Refer to the display request function (enquet) explanation for details. Set 0 for all when they are the end code.

Example

```c
static QUE inddat[] = { 
    NDTYPE, 0, 0, 0, ntork, 
    CTTYPE, 0, 0, 0, atork, 
    CNTYPE, 0, 0, 0, ctork, 
    TXTYPE, 0, 0, 0, tork, 
    0, 0, 0, 0, 0
};

enquet (pcoptb.ocb.scrnumb, INDTYPE, 0, inddat, 1L);
```

★ This command can also be used with the graphics drawing function (Chapter 1.3).
1.3 Graphics Drawing Functions

The specifications for drawing graphics with custom release functions are described in this chapter.

[Graphics environment]

(1) Plane and page configuration

(2) Graphics environment for custom release

The environment when drawing graphics with custom release is as follows.

(Nota) The coordinate system shown in (1) is the hardware coordinate system (Hx, Hy), and the coordinate system actually drawn is the display coordinate system (Cx, Cy).
[Graphics Drawing Procedure]

Graphics are drawn with the following procedure. (The flow is that from when the key is pressed to when the graphic is drawn.)

The functions used in each item are described on the following pages.
1.3.1 grastart ( ) Initialization of Environment for Graphics

**Function:** The graphic environment is initialized for custom release use. Always call this function before drawing graphics with custom release.

The details set with this function are as follow.

<table>
<thead>
<tr>
<th>Setting item</th>
<th>Setting details (common for color/monochrome)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page setting</td>
<td>Page 1</td>
</tr>
<tr>
<td>Origin setting</td>
<td>Lower left of screen (Coordinate: Hx=0, Hy=1635)</td>
</tr>
<tr>
<td>Graphic valid area</td>
<td>Page 1 (one page)</td>
</tr>
<tr>
<td>Graphic mask</td>
<td>All page display/write masks off.</td>
</tr>
</tbody>
</table>

(Nota) The planes targeted for masking are as follow.

- For monochrome: Plane 1
- For color: Planes 1 to 3

**Calling sequence:** grastart ( ) ;

**Note:**

1. Do not call this function from the "M_OPE" initializing function "mopeini ()".
2. Graphic drawing uses page 1 with the custom release.

If graphics are drawn without calling this function, the display may not be correct, and the CNC screen may be affected.
1.3.2 grammask ( ) Graphic Mask Control

Function: The mask is set by designating the mask type (display mask) and the plane.

* Display mask: Displaying designated plane's graphics is prohibited.

Main applications: The display mask can be controlled to quit the display of the screen while holding the contents drawn in the graphic memory. On screens that do not use graphics, the display of graphics can be prohibited by turning the display mask ON.

Calling sequence: ret = grammask (mode, plane) ;

long ret; Return status: 0 Normal completion
-1 Time out error
(The QUE is full so the request was invalidated.)

long mode; Mask type: 31 3 2 1 0

long plane; Plane designation: 31 3 2 1 0

The mask is turned ON/OFF with the plane designation. The mask ON process is carried out to the planes for which the bit is ON. (Prohibition of display)

Example

Example 1) To turn display mask for planes 1, 2 and 3 ON
ret = grammask (1L, 7L);

Example 2) To turn display mask OFF for only plane 1
ret = grammask (1L, 0L);
1.3.3  graclr()  Clearing of Graphics

**Function**: The graphics in the designated plane are cleared.

**Calling sequence**: ret = graclr (plane, startx, starty, lengx, lengy);

- **long plane**: Plane designation: 31 3 2 1 0
- **Plane 1**
- **Plane 2**
- **Plane 3**
- **Plane 4 (Not used)**

- **long startx**: Clear start X point coordinate : Clear start X point coordinate from origin
  - Set as “startx” = 0

- **long starty**: Clear start Y point coordinate : Clear start Y point coordinate from origin
  - Set as “starty” = 0

- **long lengx**: X direction clear width : X coordinate width from clear start point of clear area
  - Set as “lengx” = 640

- **long lengy**: Y direction clear width : Y coordinate width from clear start point of clear area
  - Set as “lengy” = 409
1.3.4 enquet ( ) Request of Drawing Graphics

Function: The drawing of various graphics on the CRT is requested. Differing types of graphics can be drawn according to the type of command designated. The graphic commands are shown below.

**Command list**

<table>
<thead>
<tr>
<th>Command name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLBCP</td>
<td>Sets current pointer.</td>
</tr>
<tr>
<td>GLBSLS</td>
<td>Sets the line type and drawing plane.</td>
</tr>
<tr>
<td>GLBALIN</td>
<td>Absolute value line drawing</td>
</tr>
<tr>
<td>GLBRPLN</td>
<td>Multiple linked line drawing</td>
</tr>
<tr>
<td>GLBCRCL</td>
<td>Circle drawing</td>
</tr>
<tr>
<td>GLBAARC</td>
<td>Absolute value arc drawing</td>
</tr>
<tr>
<td>GLBAMLN</td>
<td>Multiple unlinked line drawing</td>
</tr>
<tr>
<td>GLBAMAC</td>
<td>Multiple unlinked arc drawing</td>
</tr>
</tbody>
</table>

Calling sequence: 

\[
\text{ret} = \text{enquet} (\text{id, type, flag, datptr, windid}) ;
\]

- long id; Possessory right ID : Specify "pcoptb.ocb.scrnumb".
- char type; Command name : Refer to the command list.
- char flag; Attribute change data : Specify 0.
- char *datptr; Display data pointer : Pointer for graphic data to be displayed
- long windid; Window ID : Specify 1.

Program example

To draw a red solid line from coordinates (450, 300) to coordinates (550, 450).

\[
\begin{align*}
\text{long cp} & \text{tbl}[] = \{450,300\}; \\
\text{long slstbl} & \text{[]} = \{0.1\}; \\
\text{long alintbl} & \text{[]} = \{550,450\}; \\
\text{enquet} & (\text{pcoptb.ocb.scrnumb,GLBCP,0,cp} \text{tbl,1L}); \quad \text{Set drawing start point} \\
\text{enquet} & (\text{pcoptb.ocb.scrnumb,GLBSLS,0,slstbl,1L}); \quad \text{Designate drawing line type and color (plane)} \\
\text{enquet} & (\text{pcoptb.ocb.scrnumb,GLBALIN,0,alintbl,1L});
\end{align*}
\]
GLBCP  Setting of drawing start point

Function: Sets the drawing start point (Current pointer) at the point (cx, cy) on the display coordinate system.

Data format: long table [] = {cx, cy} ;

- cx: X axis drawing start point (display coordinate system)
- cy: Y axis drawing start point (display coordinate system)

Program example
long cntlin [] = {319, –204} ;
enquet (pcoptb.ocb.scrnumb, GLBCP, 0, cntlin, 1L) ;
**GLBSLS**  
**Designation of line type and color**

**Function:** Designates the type of line and plane (color) to be drawn with the draw command. The designated line type and color information are retained until commanded again with this command.

**Data format:** long table [] = (line, plane)

<table>
<thead>
<tr>
<th>line</th>
<th>0 Solid line</th>
<th>· · · · · · ·</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Broken line</td>
<td>· · · · · · ·</td>
</tr>
</tbody>
</table>
| 2     | Dot-dashed   | ✓✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗✗ël

<table>
<thead>
<tr>
<th>plane</th>
<th>0 Black</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Green</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Yellow</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Blue</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Magenta</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Cyan</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>White</td>
<td></td>
</tr>
</tbody>
</table>

☆ When using a monochrome CRT, the figure will be drawn with a line type no matter which plane (color) is designated. (The color designation is invalid.)

**Program example**

```c
long pdada [] = {0, 3}
enquet (pcotpib.ocb.scrnumb, GLBSLS, 0, lpdata, 1L) ;
```

After the above program is executed, the line drawn will be a yellow solid line.
GLBALIN  Draw absolute value line

**Function:** Draws a line from the current pointer to the end point (cx, cy) on the display coordinate system.

Data format: long table [] = (cx, cy) ;

- cx: X axis end point (display coordinate system)
- cy: Y axis end point (display coordinate system)

**Program example**

```c
long cntlin [] = {639, -408} ;
enquet (pcoptb.ocb.scrnumb, GLBALIN, 0, cntlin, 1L) ;
```

After drawing, the current pointer (CP) will move to the end point. The line type and color are assigned with "GLBSLS". 
**GLBRLIN**  Draw relative value line

**Function:** Draws a line from the current pointer to a relative value on the display coordinate system.

After drawing, the current pointer (CP) will move to the end point. The line type and color are assigned with "GLBSLS".

**Data format:** long table [] = {lx, ly} ;
- lx: X axis relative value from current pointer
- ly: Y axis relative value from current pointer

**Program example**
```
long lbrlin [] = {320, –204} ;
enquet (pcoptb.ocb.scrnumb, GLBRLIN, 0, lbrlin, 1L) ;
```

![Diagram showing the line drawn from the previous current pointer (319, -204) to the end point (639, -408)]
**GLBRPLN**  Draw polygonal line

**Function:** Draws a polygonal line on the display coordinate system using the current pointer as a start point.

The number of polygonal line fragments and the shift amount designated with a relative value are assigned.

After drawing, the current pointer (CP) will move to the end point. The line type and color are assigned with "GLBSLS".

**Data format:**
```c
typedef struct {
    long datan;
    long *dataptr;
} NBUFF;
long buff[] = {lx1, ly1, lx2, ly2, ..., lxn, lyn};
NBUFF table[] = {n, buff};
n : Number of polygonal line fragments
buff : Shift amount table pointer
```

**Program example**
```c
typedef struct {
    long datan;
    long *dataptr;
} NBUFF;
long zeroX[] = {10, -10, 0, -10, 10, 0};
NBUFF crossd[] = {3, zeroX};
enquet(pcoptb.ocl.scrnumb, GLBRPLN, 0, crossd, 1L);
```

Current pointer before "enquet()"
```
```
(10, -10)  (10, -20)  (20, -20)
```

Current pointer after "enquet()"
```
```
(10, -10)  (20, -20)
```
GLBCRCL  Draw circle

**Function:** Draws a circle (radius \( r \)) on the display coordinate system using the current pointer as the center.

![Diagram of circle with current pointer](image)

After drawing, the current pointer does not move. The line type and color are assigned with "GLBSLS". The circle is drawn in the clockwise direction.

**Data format:** long table \[ \] = \( r \);

\( r \): Radius

**Program example**

```c
long zpntr \[ \] = 10 ;
enquet (pcoptb.ocb.scrnumb, GLBCRCL, 0, &zpntr, 1L) ;
```
GLBAARC  Draw arc

**Function:** Draws an arc on the display coordinate system using the current point as the start point. The arc is drawn toward the end point (cxe, cye) in the direction designated with the ccw parameter. The center is (cxc, cyc)

<table>
<thead>
<tr>
<th>Co</th>
<th>xcx</th>
<th>cxe</th>
<th>+cx</th>
</tr>
</thead>
<tbody>
<tr>
<td>cyc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cye</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CCW: Arc drawing direction
0 – Clockwise
1 – Counterclockwise

After drawing, the current pointer will move to the end point. The line type and color are assigned with "GLBSLS". If the current pointer and the end point are the same, a circle will be drawn.

**Data format:** long table [] = {cxc, cyc, cxe, cye, ccw} ;

- **cxc**: X axis arc center value (CRT coordinate)
- **cyc**: Y axis arc center value (CRT coordinate)
- **cxe**: X axis arc end point (CRT coordinate)
- **cye**: Y axis arc end point (CRT coordinate)
- **ccw**: Arc drawing direction

**Program example**

long clipof [] = {20, -10, 30, -10, 1} ;

enquet (pcoptb.ocb.scrnumb, GLBAARC, 0, clipof, 1L) ;
GLBAMLN   Draw unlinked line

**Function:** A line (segment) is drawn according to the start point and end point assigned with buff on the display coordinate system. The number (n) of lines (segments) to be drawn is also assigned as drawing data.

The drawing above shows the case for n = 4

The coordinates are assigned with absolute values.
After drawing, the current pointer moves to the end point of the line (segment) drawn last.
The line type and color are assigned with "GLBSLS".

**Data format:**
```c
typedef struct {
    long datan;
    short * dataptr;
} NBUFF2;

short buff[] = {p1x, p1y, p2x, p2y, ..., p7x, p7y} ;
NBUFF2 table[] = {n, buff} ;
```

n: Number of lines (segments)
buff: Coordinate value table pointer
p1x, p1y, ... p7x, p7y: Coordinate values (display coordinate system)
Program example

typedef struct {
    long datan;
    short *dataptr;
} NBUFF2;

short gidlin [] = {10, -10, 20, -20, /* L1*/
                 20, -20, 10, -30, /* L2*/
                 10, -40, 40, -10}; /* L3*/

NBUFF2 gpttol [] = {3, gidlin};

enquet (pcoptb.ocb.scrnumb, GLBAMLN, 0, gpttol, 1L);
GLBAMAR  Draw unlinked arcs

**Function:** An arc is drawn according to the arc rotation direction, start point, end point and center point values assigned with buff on the display coordinate system. The number (n) of arcs to be drawn is also assigned as drawing data. The coordinates are assigned with absolute values.

After drawing, the current pointer moves to the end point of the arc drawn last. The line type and color are assigned with "GLBSLS". If the arc's start point and end point are the same, a circle will be drawn.

**Data format:**

```
typedef struct {
    long datan;
    short *dataptr;
} NBUFF2;
```

```
short buff [] = {ccw, p1x, p1y, p2x, p2y, c1x, c1y,
                ccw, p3x, p3y, p4x, p4y, c3x, c3y};
```

```
NBUFF2 table [] = {n, buff};
```

Parameter meanings:
- **n:** Number of arcs to be drawn
- **buff:** Coordinate value table pointer
- **ccw:** Arc drawing direction
  - 0 – Clockwise
  - 1 – Counterclockwise
- **pKx, pKy (K = 1, 3, 5, ...):** Coordinates of arc draw start point
- **pLx, pLy (L = 2, 4, 6, ...):** Coordinates of arc draw end point
- **cMx, cMy (M = 1, 3, 5, ...):** Coordinates of arc center
Program example

typedef struct {
    long datan;
    short *dataptr;
} NBUFF2;

short mardata[] = {1, 20, -10, 20, -30, 20, -20, 0, 30, -15, 30, -45, 30, -30};

NBUFF2 lbamar[] = {2, mardata};
enquet (pcoptb.ocrat.scrnumb, GLBAMAR, 0, lbamar, 1L);
1.3.5  graend( )  End of Graphics

**Function:**  Ends the graphics drawing request.
Always call this function when all graphics drawing requests have been completed.

**Calling sequence:**  graend() ;

**Program example:**

```
gramask (1,0x0f);  \(<-\) Display mask ON
grastart(); \(<-\) Initialize graphics
graclr (0x0f,sx,sy,lx,ly); \(<-\) Clear graphics
enquet (pcoptb.ocb.scrnumb,GLBCP,0,cptbl,1L);
enquet (pcoptb.ocb.scrnumb,GLBSLS,0,slstbl,1L);
enquet (pcoptb.ocb.scrnumb,GLBALIN,0,alintbl,1L);  \}  Drawing request
gramask (1,0x00); \(<-\) Display mask OFF
graend(); \(<-\) End of graphics
```
1.4 Screen Display Auxiliary Functions

List of functions

1. Check of display completion         dspend ()
2. Menu highlight                      smenhi ()
3. Menu display (no language changeover) smenud ()
4. Reset of display request           squrst ()
5. Clear text data from screen         texers ()
6. Read setting area control data information ikeyset ()
7. Set cursor position data           ocurini ()
8. Set setting area control data      omakkcb ()
9. Clear setting area buffer           ostclr ()
10. Display setting area title data    setdisp ()
11. Setting area data control         skey ()
12. Cursor control                    cursor ()
13. Set 40-character mode screen       scrst40 ()
14. Set 80-character mode screen       scrst80 ()
1.4.1 `dspend()` Check of Display Completion

**Format:**
```
dspend();
```

**Example:**
```
enquet(pcoptb. ocb. scrnumb, TXTYPE, 0, ce02, 1L);
enquet(pcoptb. ocb. scrnumb, CTTYPE, 0, ace02, 1L);
enquet(pcoptb. ocb. scrnumb, CNTYPE, 0, cce02, 1L);
enquet(pcoptb. ocb. scrnumb, CTTYPE, 0, alin0, 1L);
dspend();
```

When this function is called, the completion return will not occur until the display of all requested displays is completed.

1.4.2 `smenhi()` Menu Highlight

**Format:**
```
short menatr1[] = {start, atrcnt, End code -1};
```

Always create the same number of menu highlight data items as the number of menus.

```
short menatr2[] = {start, atrcnt, -1};  
```

(This is the display data structure of VRAM direct change B)

```
short *table[] = {menatr1, menatr2, ...};
```

```
smenhi(table, menuno);
```

- `menuno`: Menu number to be highlighted. (0 for menu 1, 4 for menu 5)
- `start`: Highlight display start position
- `atrcnt`: n-character change data

---

**Color CRT**

```
  B G R G B G G G G G G G G G G G
```

**Monochrome CRT**

```
  15 11 10 9 8 7 6 5 4 3 2 1 0
```

- Highlight
- Number of characters
Example

Always create the same number of menu highlight data items as the number of menus.

```
short menatr1[] = {1841, 0x0400 | 8, /* menu 1 atr change data */
                  1921, 0x0400 | 8,   /* menu 2 atr change data */
                  1930, 0x0400 | 8,  /* menu 3 atr change data */
                  1939, 0x0400 | 8,   /* menu 3 atr change data */
                  1948, 0x0400 | 8,   /* menu 4 atr change data */
                  1957, 0x0400 | 8,   /* menu 5 atr change data */
short menatr2[] = {1850, 0x0400 | 8, /* menu 2 atr change data */
                  1854, 0x0400 | 8,   /* menu 3 atr change data */
                  1859, 0x0400 | 8,   /* menu 4 atr change data */
                  1939, 0x0400 | 8,   /* menu 5 atr change data */
short menatr3[] = {1868, 0x0400 | 8, /* menu 4 atr change data */
                  1939, 0x0400 | 8,   /* menu 5 atr change data */
short menatr4[] = {1877, 0x0400 | 8, /* menu 5 atr change data */
                  1877, 0x0400 | 8,   /* menu 5 atr change data */
short menatr5[] = {1886, 0x0400 | 8, /* menu 5 atr change data */
                  1939, 0x0400 | 8,   /* menu 5 atr change data */
```

```
short *menatrp[] = { menatr1,
                    menatr2,
                    menatr3,
                    menatr4,
                    menatr5};
```

```c
menuhi ()
{
    smenhi(menatrp, pcoptb.ocb.menuno); /* menu reverse */
    pcoptb.ocb.mncflag & = 0xFFFF;    /* bit4 OFF */
}
```

Note: To return to the highlight display made last time, execute "smenud ()" (menu display) before "smenhi ()".
1.4.3 smenud () Menu Display

**Format:** TXDATA  *menutx [] = {menutxa,
menutxb,  
  ;  
  ;  
  } ;

short menuers [] = {start, end, 0} ;

smenud (menutx, mnuers, index) ;

  menutx: Menu display data address table
  menutxa: Menu display data
  menutxb: Menu area clear data
  start : Menu area clear start position
  end : Menu area clear end position

  index: Index value of menu display data address table
1. Screen Display Release I/F

1.4 Screen Display Auxiliary Functions

Example

Menu display data

Always create as single text data.

TXDATA men1 [] = {16,16,C_WK|NORMATR,1282,0x00,0x00,(char **)dmen10, 0};
TXDATA men2 [] = {50,25,C_WK|NORMATR,1282,0x00,0x00,(char **)dmen20, 8,8,C_WK|NORMATR,1353,0x00,0x00,(char **)dmen21, 0};
TXDATA men3 [] = {16,16,C_WK|NORMATR,1344,0x00,0x00,(char **)dmen30, 17,17,C_WK|NORMATR,1281,0x00,0x00,(char **)dmen31, 0};
TXDATA men4 [] = {16,16,C_WK|NORMATR,1344,0x00,0x00,(char **)dmen40, 8,8,C_WK|NORMATR,1281,0x00,0x00,(char **)dmen41, 0};
CTXDATA men0 [] = {3,1,C_WK|NORMATR,1209,0x00,0x00,(char **)dmen00,5,1,YOKODIR14,4,0, 8,8,C_WK|NORMATR,1201,0x00,0x00,(char **)dmen01,5,1,YOKODIR15,5,0, 0};

/* menu TXDATA address table */
TXDATA *menutx[]={men1, /* menu block 0 */
    men2, /* menu block 1 */
    men3, /* menu block 2 */
    men4 }; /* menu block 3 */

Menu area clear data

Menu display data address table

short meners [] = {80 * 15 + 1, 80 * 18, 0}; /* menu erase */

mendsp ()

Menu display

{ smenud(menutx, meners, pcoptb.ocb.mnbino); /* menu display */

Menu frame display

enqute(pcoptb.ocb.scrnumb, CTTYPE, O, meno, 1L); /* waku display */
pcoptb.ocb.mncflag &= 0xFFF7; /* bit3 OFF */
}

Display example)

Menu area clear start position Menu area clear end position

16
17
18

80 lines

Tool regist Tool life Next page Parameter
1.4.4 `sqrst()` Reset of Display Request

- Resets the display request in the Continuous-display-Que table and once-display-Que table.

**Format:** `sqrst()`;

**Example:** Program as follows to clear the display area from a continuous display area.

```c
mscrcl ()
{
    static short texclr[] = {1, 40, *18, 0}; /* screen clear data */
    sqrst (); /* display Que reset */

    texers (texclr); /* screen clear */

    graclr (7L,0L,0L,640L,409L); /* graphic clear */

    cursor (pcoptb.ocb.scrnumb, 0xFE, 0, 0x8000, 1L); /* cursor OFF */

    pcoptb.ocb.mncflag &= 0xFFFD; /* bit1 OFF */
}
```
1.4.5 texers () Clear Text Data from Screen

Format:
short table [] = {start, end, 0} ;
texers (table) ;
start : Clear start position
end : Clear end position

Example: short scrclr [] = {1, 720, 0} ;
texers (scrclr) ;
Clear data from the entire screen (40-character mode).

1.4.6 ikeyset () Obtain number of setting area, start and end positions, and setting area buffer pointer where cursor 1 is currently displayed.

Syntax:
numb = ikeyset (start, end, textpt);
long numb: Setting area number
0: No setting area has been set.
1 to 32: Setting area number
long *start: Setting area start position data pointer
long *end: Setting area end position data pointer
(Start, end positions: 0 to 719 (1439))
char **textpt: Setting area buffer pointer

Example:
char *textpt;
long start;
long end;
register   long numb;
/* cursor is not in setting */
if (!(numb = ikeyset (&start, &end, &textpt)))
    return;

Note: Always create the setting area control information with "omakkcb ( )" before calling "ikeyset ( )".
1.4.7 ocurini () Set end position of designated setting area number at cursor 1 position (pcoptb.kcbtblr.cursor).

Format: ocurini (number);
        long number ; Setting area number (1 to 32)

Example: ocurini (2L) ;

Set end position of setting area number 2 at "pcoptb.kcbtblr.cursor".

Note: • If an illegal value (for example, 3L for screen with two text items, or 0L) is designated for the setting area number, the cursor will turn off.
     (In this case, the "pcoptb.kcbtblr.cursor" value will be 0 x FFFF.)
• Always create the setting area control information with "omakkcb ()" before calling "ocurini ()".

1.4.8 omakkcb () Create setting area control information kcbtblr from setting area information table kcbtl.

Format: omakkcb (index);
        long index  : Index value of setting area information table having setting area information to be displayed.

Example:

```
    dkcb.c table (Setting area title/text information)
    KCBTBL kcbtbl [] = {7,-1,blif1,flif1,0L
    ,2,-1,btork,btork,0L}; This setting area is displayed.
```

omakkcb (1L) ;
setdisp (   )  ;

Note: With a screen/function of a screen having a setting area, always use the support functions in the following order in the screen display phase.
omakkcb (  ) ;
setdisp (   ) ;
ocurini (  ) ;
1.4.9 ostclr () Clear setting area buffer designated with setting area number.

**Format:**
```
ostclr (numb);
```

Where `numb` is a long number (1 to 32).

**Example:**
```
ostclr (1L);
```

- Clear setting area buffer #0.

**Note:** Always create the setting area control information with "omakkcb ()" before calling "ostclr ()".

1.4.10 setdisp () Display the setting area title with the setting area control information contents.

- The setting area text is displayed as a blank area.
- The setting area buffer contents are cleared.

**Format:**
```
setdisp ( );
```

**Example:**
```
dkcb.c table (Setting area title/text information)

KCBTBL kcbtbl [ ] = {7,−1,blif1,flif1,0L,
2,−1,btork,btork,0L} /* This setting area is displayed. */
};
```

- omakkcb (1L);
- setdisp ( );

**Note:** Always create the setting area control information with "omakkcb ()" before calling "setdisp ()".
1.4.11 skey () Setting Area Processing Main

The following processes are carried out according to the pressed key.

(1) Setting area processing

<table>
<thead>
<tr>
<th>Key type</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>The following processes will be carried out on the setting area containing the cursor.</td>
</tr>
<tr>
<td></td>
<td>• All data in the setting area is right-justified, and a null character is added at the end.</td>
</tr>
<tr>
<td></td>
<td>• If there is set data, the set character bit effective flag is set.</td>
</tr>
<tr>
<td></td>
<td>• The cursor moves to the right end of the setting area where the cursor is located.</td>
</tr>
<tr>
<td></td>
<td>• The add mode is entered.</td>
</tr>
<tr>
<td>Data</td>
<td>The following processes will be carried out on the setting area containing the cursor.</td>
</tr>
<tr>
<td></td>
<td>• Add mode ... The data in the setting area buffer is shifted to the left, and then the key data is added.</td>
</tr>
<tr>
<td></td>
<td>When 2 is pressed: ( 1 2 3)</td>
</tr>
<tr>
<td></td>
<td>• Insert mode ... The key data is added, and the data in the setting area buffer is shifted to the left.</td>
</tr>
<tr>
<td></td>
<td>When 3 is pressed: ( 1 4 2 3)</td>
</tr>
<tr>
<td></td>
<td>• Replace mode ... The data in the setting area buffer is replaced with the key data.</td>
</tr>
<tr>
<td></td>
<td>• If the cursor is in the right end of the setting area, the add mode will be entered.</td>
</tr>
<tr>
<td></td>
<td>• If the cursor is not at the right end of the setting area, the cursor will shift to the right.</td>
</tr>
<tr>
<td>Cursor</td>
<td>When the cursor is in the setting area: The up/down keys will be invalid.</td>
</tr>
<tr>
<td></td>
<td>&lt;Left key&gt;</td>
</tr>
<tr>
<td></td>
<td>• If the cursor is not at the left end, it will shift one to the left, and then the replace mode will be entered.</td>
</tr>
<tr>
<td></td>
<td>( )</td>
</tr>
<tr>
<td></td>
<td>• If the cursor is at the left end, it will shift to the right end of the previous setting area, and then the add mode will be entered.</td>
</tr>
<tr>
<td></td>
<td>( )</td>
</tr>
<tr>
<td></td>
<td>&lt;Right key&gt;</td>
</tr>
<tr>
<td></td>
<td>• If the cursor is not at the right end, it will shift one to the right, and then the replace mode will be entered.</td>
</tr>
<tr>
<td></td>
<td>( )</td>
</tr>
<tr>
<td></td>
<td>• If the cursor is at the right end, it will shift to the right end of the next setting area, and then the add mode will be entered.</td>
</tr>
<tr>
<td></td>
<td>( )</td>
</tr>
</tbody>
</table>
1. Screen Display Release I/F

1.4 Screen Display Auxiliary Functions

<table>
<thead>
<tr>
<th>Key type</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancel</td>
<td>Data is cleared from all setting area buffers, and then the replace mode is entered.</td>
</tr>
</tbody>
</table>
| Tab      | When the cursor is in a setting area:  
Left tab key .........The cursor shift to the right end of the previous column, and then the add mode is entered.  
Right tab key .......The cursor shift to the right end of the next column, and then the add mode is entered.  
The position succeeding the end setting area is the head of the setting area.  
The position preceding the head setting area is the end setting area. |
| Delete   | When the cursor is in a setting area, the character at the cursor position is deleted while right-justifying the remaining data.  
If the cursor is at the right end of the setting area, or the add mode is not entered, the replace mode will be entered. |
| Insert   | When the cursor is in a setting area:  
The cursor is shifted one to the left. ( 1 2 3 4 5 )  
The data before the cursor is shifted one to the left.  
A blank is set at the cursor position. ( 1 2 3 4 5 )  
The insert mode is entered.  
The above processes do not take place if the cursor is at the left end or at the second position. ( 1 2 3 4 5 ) |
| Clear block | When the cursor is in a setting area:  
All data is cleared from the designated setting area buffer.  
The cursor moves to the right end of the setting area.  
The add mode is entered. |

(2) Display of data (text) in setting area buffer using "enquet ( )".
Display of current cursor position with "cursor ( )".

Format: skey ( ) ;

Example: skey ( ) ;

Note: Always create the setting area control information with "omakkcb ( )" before calling "skey ( )".
(3) KCB table data

The KCB table data is used to display the setting area.
The structure of this table is described in this section.
The following figure shows the setting area display data. This data contains the setting area title used to display fixed character string data (parentheses, etc.), and the setting area text used to echo-back the pressed key.

☆ When creating the setting area text data, the number of text data items must be 10 or less per screen. The size of the setting area text must be 1 column (vertical) × 31 columns (horizontal) or less.

This structure name is defined as “KCBTBL”.

```
1. Screen Display Release I/F
1.4 Screen Display Auxiliary Functions
```

```
(3) KCB table data

The KCB table data is used to display the setting area.
The structure of this table is described in this section.
The following figure shows the setting area display data. This data contains the setting area title used to display fixed character string data (parentheses, etc.), and the setting area text used to echo-back the pressed key.

☆ When creating the setting area text data, the number of text data items must be 10 or less per screen. The size of the setting area text must be 1 column (vertical) × 31 columns (horizontal) or less.

This structure name is defined as “KCBTBL”.
```
1. Screen Display Release I/F

1.4 Screen Display Auxiliary Functions

20. Number of setting area
This is the number of setting area texts displayed on the screen.

21. 0xff (fixed).

22. 0L (fixed).

The structure of the setting area title data is the same as the single title data structure.
Refer to page 5 "Display of single title/text data (TXDATA)".

This is a table for referring to the setting area texts displayed on one screen. In each screen, the number of areas in this table equals the number of setting areas.

This is configured of the pointers to the setting area texts of all release screens.

The structure of the setting area text data is the same as the single text data.
Refer to page 5 "Display of single title/text data (TXDATA)".

☆ The structure of the KCB table can be understood by actually referring to an example shown in the figure.
char hclpa25[] = "(# (__) )";
char hkpa110[] = "(# (__) DATA (__________)(__________) )";
char hwoff1[] = "(# (__) DATA (__________) )";
TXDATA bclpa[] = {5,5,C_SK|ORMATR,1281,0x00,0x00,(char **)hclpa25, 0};
TXDATA bkpa1[] = {34,34,C_SK|ORMATR,1281,0x00,0x00,(char **)hkpa110, 0};
TXDATA bwoff[] = {21,21,C_SK|ORMATR,1281,0x00,0x00,(char **)hwoff1, 0};
TXDATA xclpa26[] = {2,2,C_YK|NORMATR,1283,0x00,0x00,(char **)pcoptb.settei.dvar0[0], 0};
TXDATA xkpa112[] = {10,10,C_YK|NORMATR,1292,0x00,0x00,(char **)pcoptb.settei.dvar1[0], 0};
TXDATA xkpa113[] = {10,10,C_YK|NORMATR,1304,0x00,0x00,(char **)pcoptb.settei.dvar1[0], 0};
TXDATA xwoff3[] = {10,10,C_YK|NORMATR,1291,0x00,0x00,(char **)pcoptb.settei.dvar1[0], 0};
char fclpa[] = {0};
char fkpa[] = {0,1,2};
char fwoff[] = {0,3};
TXDATA *tset[] = {xclpa26,xkpa112,xkpa113,
xwoff3};
KCBTBL kcbtbl[] = {1,0xff,bclpa,fclpa,0L,
3,0xff,bkpa1,fkpa1,0L 2,0xff,bwoff,fwoff,0L};

Example of KCB table
1.4.12 cursor () Cursor Display Request

Format: cursor (id, type, (char)0, cursor1, 0);

- long id: Possessory right ID (Designate as "pcoptb.ocb.scrnmb").
- char type: que type number 0xFE: Cursor 1 is valid
- short cursor1: Cursor 1 position

* The cursor position counts the top left end column as 0 as shown below.

### 40-character mode

<table>
<thead>
<tr>
<th>0</th>
<th>40</th>
<th>to 39</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>18 lines</td>
<td>to 79</td>
</tr>
<tr>
<td>680</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 80-character mode

<table>
<thead>
<tr>
<th>0</th>
<th>80</th>
<th>to 79</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>18 lines</td>
<td>to 159</td>
</tr>
<tr>
<td>1360</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example: cursor (pcoptb.ocb.scrnmb, 0xFE, (char)0, pcoptb.kctblr.cursor, 0);

Cursor 1 is displayed at the "pcoptb.kctblr.cursor" value.
1.4.13  scrst40 ()  Set 40-character Mode Screen

**Format:**  
sts = scrst40 () ;  
long sts;  0: Normal completion  -1: Error completion  

**Function:**  Changes the screen display (character display) to the 40-character mode.  

**Example:**  scrst40 () ;
1.4.14  scrst80 ()  Set 80-character Mode Screen

**Format:**  
sts = scrst80 () ;  

    long sts;  0: Normal completion  -1: Error completion

**Function:**  Changes the screen display (character display) to the 80-character mode.

**Example:**  scrst80 () ;

```
+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
```
Example of program using scrst40() and scrst80()

```c
#include <mseltb.h>

/* Define several function pointers */
extern void mquerst();
extern void mode40();
extern void mode80();
extern void mscrcl();

long (*mseltb[8])() = {
    (long (*)(()))0L,  /* BTON0:0x0001    */
    (long (*)(()))mquerst,  /* BTON1:0x0002: Display request reset */
    (long (*)(()))mode40,  /* BTON2:0x0004: 40-column mode selection */
    (long (*)(()))mode80,  /* BTON3:0x0008: 80-column mode selection */
    (long (*)(()))mscrcl,  /* BTON4:0x0010: Clearing of screen display area */
    (long (*)(()))0L,  /* BTON5:0x0020    */
    (long (*)(()))0L,  /* BTON6:0x0040    */
    (long (*)(()))0L,  /* BTON7:0x0080    */
    (long (*)(()))0L,  /* BTON8:0x0100    */
};
```

The function table corresponds to the following bits
- Bit1: Display request reset
- Bit2: 40-column mode selection
- Bit3: 80-column mode selection
- Bit4: Clearing of screen display area

In/Out:
- In : –
- Out : –

Machine: M64
Language: C
CPU: R4000
Comp/Asm: multi
# Include file

/* (M64 system) */
/# "define" for controlling the include file is defined here. */
/* System */
#include "o_type.h"
#include "po_typ.h"

/* External reference */

/*---- Data declaration ------------------------------------------------------------------------*/
/*---- Variable declaration -------------------------------------------------------------------*/
extern void sc00(void); /* Screen display */

/*---- Define declaration ----------------------------------------------------------------------*/
/*---- Data declaration ------------------------------------------------------------------------*/
/*     General-purpose function      Only file defined here is used */

/* Function body */

/**/ 
/**/ 
MENUBLK menublk[] = {
/* mmcflag Control flag */
bit0: "mquerst" Display request reset
bit2: "mode40" 40-column mode selection
1.4 Screen Display Auxiliary Functions

bit3: "mode80" 80-column mode selection

/* Menu block 0 */
/* Control Screen processing address Next menu Custom flag */
/* Flag Block No. */
"/" /mmccflag mfuncp mnxtno afflag Menu No. /
0x06, (long (*)()) sc00, –1, 0, /* 0: [Screen display] */
0, (long (*)()) –1, –1, 0, /* 1[ ] */
0, (long (*)()) –1, –1, 0, /* 2[ ] */
0, (long (*)()) –1, –1, 0, /* 3[ ] */
0, (long (*)()) –1, –1, 0, /* 4[ ] */

/* Menu block 1 */
/*mmccflag mfuncp mnxtno afflag Menu No. */
0, (long (*)()) –1, –1, 0, /* 0[ ] */
0, (long (*)()) –1, –1, 0, /* 1[ ] */
0, (long (*)()) –1, –1, 0, /* 2[ ] */
0, (long (*)()) –1, –1, 0, /* 3[ ] */
0, (long (*)()) –1, –1, 0, /* 4[ ] */

/* Menu block 2 */
/*mmccflag mfuncp mnxtno afflag Menu No. */
0, (long (*)()) –1, –1, 0, /* 0[ ] */
0, (long (*)()) –1, –1, 0, /* 1[ ] */
0, (long (*)()) –1, –1, 0, /* 2[ ] */
0, (long (*)()) –1, –1, 0, /* 3[ ] */
0, (long (*)()) –1, –1, 0, /* 4[ ] */

} ;

short sbmbln = (sizeof(menublk) / sizeof(MENUBLK)) ; /* Number of menus set */

<mode40.c>

/**********************************************************************************************/
/*                                                                                           (M64 system) */
/*                                                                                           */
/* <NAME> mseltb.c                                                                                       */
/*                                                                                           */
/* <FUNCTION> m_ope Selection function (bit2) 40-character mode selection                              */
/*                                                                                           */
/* <PROGRAM> APLC                                                                                     */
/*                                                                                           */
/* COPYRIGHT (C) 1998 MITSUBISHI ELECTRIC CORPORATION                                                */
/* ALL RIGHT RESERVED                                                                             */
/**********************************************************************************************/
/*                                                                                           */
/* <Outline>                                                                                       */
/* Set screen to 40-column mode                                                                       */
/* Selective call flag OFF                                                                           */

<In/Out>                                                                                       
In    : None                                                                                          
Out   : None                                                                                          

<Machine> M64                                                                                       
<OS>   VxWorks                                                                                       
>Type  NC                                                                                              
<CPU>  R4000                                                                                           
<Compiler/Assembler> multi                                                                               

/**********************************************************************************************/
/* Include file                                                                                      */
/**********************************************************************************************/
/* ("define" for controlling the include file is defined here.) */
/**********************************************************************************************/
/* System */

#include "o_type.h"
#include "pcoptb.h"
#include "c_def.h"

/**********************************************************************************************/
/* External reference                                                                              */
/**********************************************************************************************/
/*---- Data declaration                                                                            */
extern PCOPTB pcoptb;

/* Work
*/

extern void scrst40();

/*---- Function declaration -------------------------------------------------------------------*/

/* System library */
extern void scrst40();

/****************************************************************************************/
/*     Data definition                        Only file defined here is used           */
/****************************************************************************************/

/*---- Define declaration ----------------------------------------------------------------------*/

/*---- Data declaration ------------------------------------------------------------------------*/

/****************************************************************************************/
/*     General-purpose function      Only file defined here is used                    */
/****************************************************************************************/

/* Function body */

 /****************************************************************************************/

/**/ void mode40( void )
{  scrst40(); /* 40-column mode ON */
    pcoptb.ocb.mncflag &= ~BTON2; /* Selective call flag OFF (bit2)*/
}
## 2. DDB I/F

### 2.1 CNC Data Read/Write Functions

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1.1 CNC Data Read/Write Functions (Type 2)</td>
<td><code>ddbrd()</code>, <code>ddbwt()</code></td>
</tr>
<tr>
<td>2.1.2 O, N, B Data Read</td>
<td><code>smkonb()</code></td>
</tr>
<tr>
<td>2.1.3 Calendar Function</td>
<td><code>scaldr()</code></td>
</tr>
<tr>
<td>2.1.4 Message Function</td>
<td><code>sgetmes()</code></td>
</tr>
<tr>
<td>2.1.5 Operation Search</td>
<td><code>oexsrch()</code></td>
</tr>
<tr>
<td>2.1.6 CNC Variable Read Function (IEEE double type)</td>
<td><code>ievarrd()</code></td>
</tr>
<tr>
<td>2.1.7 CNC Variable Write Function (IEEE double type)</td>
<td><code>ievarwt()</code></td>
</tr>
<tr>
<td>2.1.8 CNC Variable Clear Function</td>
<td><code>ievarclear()</code></td>
</tr>
</tbody>
</table>
2.1.1  

2.1.1  

2. DDB I/F

2.1  CNC Data Read/Write Functions

2.1.1  ddbrd (), ddbwt ()  CNC Data Read/Write Functions (Type 2)

1.  Read function

sts = ddbrd (id1, id2, ddbsz, sysno, axisno, ground, &ddbbf) ;

2.  Write function

sts = ddbwt (id1, id2, ddbsz, sysno, axisno, ground, &ddbbf) ;

long  sts  
Return value
0  Normal completion
–1  No option
–2  Size over
–3  Axis number illegal
–4  Section number illegal
–5  Write protect
–6  System number error

long  id1  
Section number  
Refer to the "DDB Interface Manual" (BNP-B2214).

long  id2  
Sub-section number

long  ddbsz  
Data size
1 – 1 byte
2 – 2 bytes
4 – 4 bytes

long  sysno  
System number
0 – 1st system
1 – 2nd system

long  axisno  
Axis number
0 – No axis
1 – 1st axis
2 – 2nd axis

long  ground  
Ground
0 – Fixed

long & ddbbf  
Read/write data buffer address

3.  Usage example

Example)  G53 offset

<table>
<thead>
<tr>
<th>Section number</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sub-section number</td>
<td>272</td>
</tr>
<tr>
<td>Size</td>
<td>4 bytes</td>
</tr>
<tr>
<td>System</td>
<td>1st system</td>
</tr>
<tr>
<td>Axis</td>
<td>3rd axis</td>
</tr>
<tr>
<td>Ground</td>
<td>0</td>
</tr>
</tbody>
</table>

Set as follows to read the data.

long sts;  Return value
long ddbbf; Read buffer

sts = ddbrd (4, 272, 4, 0, 3, 0, &ddbbf) ;
2.1.2 smkonb () O, N, B Data Read

1. Outline

This is the custom release function that obtains the O (program number), N (sequence number) and B (block number) of the machining program being executed.

With this function, the O, N and B data of the main program being executed in one system, and the O, N and B data of the sub-program can be obtained.

Note) Even when using 2-system, only the data for 1-system will be read out.
2. Details

2-1 Function

The O, N and B data of the main program being executed in 1-system, and the O, N and B data of
the sub-program is obtained.

2-2 Calling sequence

(1) Function name

sts = smkonb (&buff) ;

(2) Argument

<table>
<thead>
<tr>
<th>Top address of work table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designated system ( \text{\textsuperscript{note}} )</td>
</tr>
<tr>
<td>Return information</td>
</tr>
<tr>
<td>Main O number</td>
</tr>
<tr>
<td>Main N number</td>
</tr>
<tr>
<td>Main B number</td>
</tr>
<tr>
<td>MDI data</td>
</tr>
<tr>
<td>Dummy data</td>
</tr>
<tr>
<td>Sub O number</td>
</tr>
<tr>
<td>Sub N number</td>
</tr>
<tr>
<td>Sub B number</td>
</tr>
</tbody>
</table>

\textbf{Note)} Set to 1.

(3) Calling order

1) Set the designated system in the work table. (Only 1-system)

2) Call the top address of the work table as the argument. \textbf{Example)}  \text{\texttt{smkonb (}} \texttt{&buff)}  ;

3) The data will be set in the work table with the ONB data creation function.
(4) Contents of data

<table>
<thead>
<tr>
<th>Data type</th>
<th>Input/ output</th>
<th>Range</th>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designated system</td>
<td>char</td>
<td>Input data</td>
<td>1 to 5</td>
</tr>
<tr>
<td>Return information</td>
<td>char</td>
<td>Output data</td>
<td>0 to 255</td>
</tr>
<tr>
<td>Main O number N B</td>
<td>long long long</td>
<td>Output data Output data Output data</td>
<td>0 to 99999999 0 to 99999 0 to 99999</td>
</tr>
<tr>
<td>Sub O number N B</td>
<td>long long long</td>
<td>Output data Output data Output data</td>
<td>0 to 99999999 0 to 99999 0 to 99999</td>
</tr>
<tr>
<td>MDI data</td>
<td>char</td>
<td>Output data</td>
<td>0 to 1</td>
</tr>
</tbody>
</table>

Notes:
1) If the 9000 address (9000 to 9999) sub-program is being executed when the custom program lock (edit lock C) is ON, 0 will be set in the ONB data for the sub-program.
2) Always set the designated system as 1. Nothing will be set in the main ONB data or sub ONB data if any other value is set.

(5) Return information

Designated system error
If the designated system does not exist, 1 will be set. If the designation is correct, 0 will be set. (If the designated system does not exist, nothing will be set in the main ONB data or sub ONB data.)
typedef struct {
    char systm;  Designated system (Only 1-system)
    char info;  Returned information
    long maino;  Main O number
    long mainn;  Main N number
    long mainb;  Main B number
    char mdi;  MDI data
    char dmyd;  Dummy data
    long subo;  Sub O number
    long subn;  Sub N number
    long subb;  Sub B number
} BUFF;  Declares the structure type.

BUFF buff[3];  Secures the work table.

test ()
{
    buff[0].systm = 1;  Sets system 1.
    smkonb(&buff[0]);  Obtains system 1 ONB data.
}
2.1.3 scaldr () Calendar Function

1. Outline
This function is used to read/write the calendar data from the custom release screen process. The data flow is shown below.
2. Function

Explanation

By using the calendar data read/write function "scaldr ()", the current date can be read and written.

2-1 Using the functions

Calling sequence:  
\[
\text{sts} = \text{scaldr} \left( \text{&buff} \left[0\right] \right);
\]

Argument:  
&buff [0]  Head address of buffer having a 20-byte size.

<table>
<thead>
<tr>
<th>short buff [0] : Command</th>
<th>When 0, calendar data is written. When 1, calendar data is read.</th>
</tr>
</thead>
<tbody>
<tr>
<td>short buff [1] : Year</td>
<td>(0 to 99)</td>
</tr>
<tr>
<td>short buff [2] : Month</td>
<td>(1 to 12)</td>
</tr>
<tr>
<td>short buff [3] : Date</td>
<td>(1 to 31)</td>
</tr>
<tr>
<td>short buff [4] : Hour</td>
<td>(0 to 23)</td>
</tr>
<tr>
<td>short buff [5] : Minute</td>
<td>(0 to 59)</td>
</tr>
<tr>
<td>short buff [6] : Second</td>
<td>(0 to 59)</td>
</tr>
<tr>
<td>short buff [7] : Not used</td>
<td>(Input 0.)</td>
</tr>
<tr>
<td>short buff [8] : Day</td>
<td>(0 to 6: 0 is Sunday, 6 is Saturday)</td>
</tr>
<tr>
<td>short buff [9] : Spare</td>
<td>(Input 0.)</td>
</tr>
</tbody>
</table>

Return value:  
sts  Error status (short)  
Refer to the section 2-2 "List of error details" for details.

Function

(1) Write of calendar data

Input 1 in "buff [0]", and input the current date in "buff [1]" to "buff [8]" following the format given above.  
After inputting all of the data in "buff [0]" to "buff [9]", call this function.

(2) Read of calendar data

Input 0 in "buff [0]", and call this function.  
The current date will be written into "buff [1]" to "buff [8]" following to the format given above.
## 2-2 List of error details

<table>
<thead>
<tr>
<th>sts</th>
<th>Error details</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Command data illegal</td>
</tr>
<tr>
<td>-2</td>
<td>Year data setting range error</td>
</tr>
<tr>
<td>-3</td>
<td>Month data setting range error</td>
</tr>
<tr>
<td>-4</td>
<td>Date data setting range error</td>
</tr>
<tr>
<td>-5</td>
<td>Hour data setting range error</td>
</tr>
<tr>
<td>-6</td>
<td>Minute data setting range error</td>
</tr>
<tr>
<td>-7</td>
<td>Second data setting range error</td>
</tr>
<tr>
<td>-8</td>
<td>Not used</td>
</tr>
<tr>
<td>-9</td>
<td>Day data setting range error</td>
</tr>
<tr>
<td>-10</td>
<td>Spare</td>
</tr>
</tbody>
</table>
2.1.4 sgetmes () Message Function

Function name:  
\[ \text{sts} = \text{sgetmes (para1, para2, \\ & buff[0])} ; \]

Include files  
"mes_def.h".

Argument:  
long para1 : Message section  
  OPEMES : Operation message  
  SETERR : Setting error

long para2 : Message sub-section  
  Set a message number. (Refer to the next page.)

char & buff [0] : Head address of buffer having 20-byte size.

Returned value:  
long sts : 0.......................... Normal completion  
–1 ......................... Section illegal  
–2 ......................... Sub-section illegal  
–100 ..................... Other error

Function  
This function is used to read the message data that the CNC has with the custom software. First, designate the message type with "para1", and designate the message to be read out with "para2". Finally, transfer the head of a buffer having a size of 20 bytes as an argument to this function. For example, to read "E01 SETTING ERROR", set "SETERR" in "para1" and "CE01" in "para2". Refer to the next page for details on the sub-section.

Program example  
\begin{verbatim}
char buff [20] ;  
sgetmes (SETERR, CE01, &buff [0]) ;
\end{verbatim}
### 2. DDB I/F
#### 2.1 CNC Data Read/Write Functions

<table>
<thead>
<tr>
<th>Error message</th>
<th>Error message</th>
<th>Error message</th>
</tr>
</thead>
<tbody>
<tr>
<td>CE01 SETTING ERROR</td>
<td>CE23 NOT ADD I/O</td>
<td>CE50 FILE ERR</td>
</tr>
<tr>
<td>CE02 DATA OVER</td>
<td>CE25 DATA MEMORY ERR</td>
<td>CE51 FILE OPEN ERR</td>
</tr>
<tr>
<td>CE03 No. NOT FOUND</td>
<td></td>
<td>CE52 FILE CLOSE ERR</td>
</tr>
<tr>
<td>CE04 DEV. NOT READY</td>
<td></td>
<td>CE53 FILE SEEK ERR</td>
</tr>
<tr>
<td>CE05 NOT ACCEPTABLE</td>
<td></td>
<td>CE54 FILE READ ERR</td>
</tr>
<tr>
<td>CE06 NO SPEC</td>
<td></td>
<td>CE55 FILE DELETE ERR</td>
</tr>
<tr>
<td>CE07 RESET END</td>
<td></td>
<td>CE57 FILE INSERT ERR</td>
</tr>
<tr>
<td>CE08 PHYSICAL ERR</td>
<td></td>
<td>CE60 IOP ERR</td>
</tr>
<tr>
<td>CE09 TIME OUT</td>
<td>CE35 COMPARE ERROR</td>
<td></td>
</tr>
<tr>
<td>CE10 MEMORY OVER</td>
<td></td>
<td>CE64 PROGRAM No. ERR</td>
</tr>
<tr>
<td>CE11 PROG. No. DUPLI</td>
<td></td>
<td>CE65 PROG. No. DUPLI</td>
</tr>
<tr>
<td>CE12 FILE ENTRY OVER</td>
<td></td>
<td>CE66 NO PROG. NUMBER</td>
</tr>
<tr>
<td>CE13 NB NOT FOUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CE14 PROG. NOT FOUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CE15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CE16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CE17 PARITY H ERR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CE18 PARITY V ERR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CE19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CE20 OVER RUN ERR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CE21 PROGRAM RUNNING</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CE22 CODE CHANGE ERR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error message</td>
<td>Error message</td>
<td>Error message</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>CE70 TOOL No. DUPLI</td>
<td>CM01 SEARCH EXECUTION</td>
<td>CM21 DATA OUT COMPLETE</td>
</tr>
<tr>
<td>CE71 TOOL ENTRY OVER</td>
<td>CM02 SEARCH COMPLETE</td>
<td>CM22 ERASE EXECUTION</td>
</tr>
<tr>
<td></td>
<td>CM03 RESEARCH EXECUTION</td>
<td>CM23 ERASE COMPLETE</td>
</tr>
<tr>
<td></td>
<td>CM04 RESEARCH COMPLETE</td>
<td>CM24 COPY EXECUTION</td>
</tr>
<tr>
<td></td>
<td>CM05 BUFFER EDIT</td>
<td>CM25 COPY COMPLETE</td>
</tr>
<tr>
<td></td>
<td>CM06 MDI NO SETTING</td>
<td>CM26 CONDENSE EXECUTION</td>
</tr>
<tr>
<td></td>
<td>CM07 MDI SETTING COMPLETE</td>
<td>CM27 CONDENSE COMPLETE</td>
</tr>
<tr>
<td></td>
<td>CM08 MDI ENTRY COMPLETE</td>
<td>CM28 MERGE EXECUTION</td>
</tr>
<tr>
<td></td>
<td>CM09 MDI RUNNING</td>
<td>CM29 MERGE COMPLETE</td>
</tr>
<tr>
<td></td>
<td>CM10 PUSH KEY SERCH/PROG</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CM11 EDITTING</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CM12 PROGRAM RUNNING</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CM13 VOICE OUTPUT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CM14 DATA IN EXECUTION</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CM15 DATA WRITING</td>
<td></td>
</tr>
<tr>
<td>CE80 TOP SEARCH ERR</td>
<td>CM16 DATA IN COMPLETE</td>
<td></td>
</tr>
<tr>
<td>CE81 PROGRAM ERROR</td>
<td>CM17 VARIABL CONVERT ERR</td>
<td></td>
</tr>
<tr>
<td>CE82 ALREADY RESERCH</td>
<td>CM18 COMPARE EXECUTION</td>
<td></td>
</tr>
<tr>
<td>CE83 COUNT OVERFLOW</td>
<td>CM19 COMPARE COMPLETE</td>
<td></td>
</tr>
<tr>
<td>CE84 CAN'T IN/OUT</td>
<td>CM20 DATA OUT EXECUTION</td>
<td></td>
</tr>
<tr>
<td>CE85 LINE BUSY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CE86 INPUT DATA ERR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.1.5 oexsrch ()  Operation Search

Calling sequence

oexsrch (&srchinf) ;

SRCHINF srchinf ;

* Create the following table, and transfer the pointer as the argument.

typedef struct {
  short command ; Command
  short status ; Status
  long progno ; Program number
  long seqno ; Sequence number
  long blkno ; Block number
  long blkrep ; Number of block appearances (Reserved for restart search)
  long sys ; Designate system to be searched (0: Fixed to 1-system)
} SRCHINF ;

Function

When this function is called, 1-system operation search of the machining program or MDI program designated in the table is executed.

The table setting data is explained below.

(1) Command ... The mode is designated and the search is started.
(2) Status ... The search status is expressed. (Set by the NC.)

Error cause (Refer with 1 to 255:unsigned char)
1: Setting error
2: Program not found
3: Sequence/block No. not found
4: Program running
5: File error
6: I/O error
7: Program error
8: No spec
9: Top search error (Reserved for restart search)
10: Already restart searched (Reserved for restart search)
11: Server error
12: Disk error
Search execution
Search complete
Error complete (When ON, the error cause is valid)

(3) Program number
- For memory/tape
  Set the number of the program to be searched with a binary.
  For restart search, the program number cannot be omitted (designated ()).
  0 to 99999999 (max. 8 digits)
- For MDI (only operation search)
  Normally set 0.
  Set 1 to reset the search.

(4) Sequence number
Set the number of the sequence to be searched with a binary.
0 to 99999 (max. 5 digits)

(5) Block number
Set the number of the block to be searched with a binary.
0 to 99999 (max. 5 digits)

(6) Number of block appearances (Reserved for restart search)
Set the number of times the block to be searched appears.
For example, when searching for a block in the sub-program, the block is executed the same times
the program is called, so set which execution block to search.
When 0 is set, the first execution block will be searched.
0 to 9999 (max. 4 digits)
### List of set patterns (operation search)

* o, n, b indicate a number (other than 0) is designated.

<table>
<thead>
<tr>
<th>Program No.</th>
<th>Sequence No.</th>
<th>Block No.</th>
<th>Search status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (other than MDI)</td>
<td>o</td>
<td>o</td>
<td>Setting error</td>
</tr>
<tr>
<td>0 (other than MDI)</td>
<td>n</td>
<td>o</td>
<td>Search designated sequence number in selected program</td>
</tr>
<tr>
<td>0 (other than MDI)</td>
<td>n</td>
<td>b</td>
<td>Search designated sequence block number in selected program</td>
</tr>
<tr>
<td>p (including MDI)</td>
<td>o</td>
<td>b</td>
<td>Setting error</td>
</tr>
<tr>
<td>p (including MDI)</td>
<td>n</td>
<td>o</td>
<td>Search head of designated program</td>
</tr>
<tr>
<td>p (including MDI)</td>
<td>n</td>
<td>b</td>
<td>Search designated sequence block number in designated program</td>
</tr>
<tr>
<td>p (including MDI)</td>
<td>o</td>
<td>b</td>
<td>Search for a block that is the designated number of blocks ahead from the head of the designated program Note that if there is a sequence number if the blocks midway, the search cannot be carried out.</td>
</tr>
</tbody>
</table>

**ex)**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G28 XYZ ;</td>
<td>Search is possible with &quot;O100 N0 B1&quot;</td>
</tr>
<tr>
<td>G00 X100. Y50. ;</td>
<td>Search is possible with &quot;O100 N0 B2&quot;</td>
</tr>
<tr>
<td>G01 Z20. F1000 ;</td>
<td>Search is possible with &quot;O100 N0 B3&quot;</td>
</tr>
<tr>
<td>N100 Y50. ;</td>
<td>Search is not possible with &quot;O100 N0 B4&quot; (This block is &quot;O100 N100 B0&quot;)</td>
</tr>
</tbody>
</table>

### Program example

To search block N2 in the machining program No. O100

```c
srchinf.command = BITO ;
srchinf.progno = 100 ;
srchinf.seqno = 2 ;
srchinf.blkno = 0 ;
oexsrch(&srchinf) ;
if ( srchinf.status & BIT2 )
    Error process
```

BITO: 0x0001  
BIT2: 0x0004  
BIT9: 0x0200
To search for the head of the MDI program

```c
srchinf.command = BIT0 | BIT9;
srchinf.progno = 0;
srchinf.seqno = 0;
srchinf.blkno = 0;
oexsrch (&srchinf);
if ( srchinf.status & BIT2 )
    Error process
```

To reset the MDI program search

```c
srchinf.command = BITO | BIT9;
srchinf.progno = -1;
oexsrch (&srchinf);
if ( srchinf.status & BIT2 )
    Error process
```
### 2.1.6 ievarrd () CNC Variable Read Function (IEEE double type)

**Function**
Reads the CNC variables with the standard "double" format (IEEE double).

**Format**
```c
sts=ievarrd (mode,varno,syst,ground,varptr);
```
- long `sts`; Status
  - 1: Blank variable retrieved
  - 0: Normal completion
  - -1: Mode error
  - -2: Variable number error
  - -3: System number error
  - -4: Ground designation error
  - -5: File unformatted error
- long `mode`; Variable mode
  - 0: Common variable
  - 3: Local variable
- long `varno`; Variable number

<table>
<thead>
<tr>
<th>Variable</th>
<th>Variable number</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common variable</td>
<td></td>
<td>Machining center</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lathe</td>
</tr>
<tr>
<td>100</td>
<td>100 to 149, 500 to 549</td>
<td>○</td>
</tr>
<tr>
<td>200</td>
<td>100 to 199, 500 to 599</td>
<td>○</td>
</tr>
<tr>
<td>300</td>
<td>100 to 199, 500 to 699</td>
<td>○</td>
</tr>
<tr>
<td>600</td>
<td>100 to 199, 500 to 999</td>
<td>○</td>
</tr>
<tr>
<td>700</td>
<td>100 to 199, 400 to 999</td>
<td>○</td>
</tr>
<tr>
<td>Local variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 to 33 (32), 101 to 133 (132),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>201 to 233 (232), 301 to 333 (332),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>401 to 433 (432)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>* The 100th place is the level designation.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>201 → #1 in level 2.</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** The maximum number in the local variables not shown in parentheses is for the machining center. The number for the lathe is shown in parentheses.

If a variable not provided in the options is designated, –2: Variable number error will occur.

```c
long syst; System number 0: 1st system 1: 2nd system
long ground; Ground designation 0: Fixed
long varptr; Variable data pointer
```

Common variables for each system valid only for (#100 to #199) (Fixed to 0 in other cases.)
Example

Read the common variable #510 with standard "double".

```c
extern long ievarrd ();

mvalrd ()
{
    long    sts;
    double  dbuff;
    ...
    ...
    sts = ievarrd (0L, 510L, 0L, 0L, &dbuff);
    if (sts < 0)
    {
        merror ();
    }
    ...
    ...
}
```
2.1.7  ievarwt ()  CNC Variable Write Function (IEEE double type)

Function
Writes the CNC variables with the standard "double" format (IEEE double).

Format

\[ \text{sts} = \text{ievarwt} (\text{mode}, \text{varno}, \text{syst}, \text{ground}, \text{varptr}); \]

<table>
<thead>
<tr>
<th>long</th>
<th>sts;</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>mode;</td>
<td>Variable mode</td>
</tr>
<tr>
<td>long</td>
<td>varno;</td>
<td>Variable number</td>
</tr>
<tr>
<td>long</td>
<td>syst;</td>
<td>System number</td>
</tr>
<tr>
<td>double   *</td>
<td>varptr;</td>
<td>Variable data pointer</td>
</tr>
</tbody>
</table>

0: Normal completion  
-1: Mode error  
-2: Variable number error  
-3: System number error  
-4: Ground designation error  
-5: File unformatted error  

long mode; Variable mode  
0: Common variable  
3: Local variable  

long varno; Variable number  
(Refer to section 2.1.7.)  

long syst; System number  
0: 1st system  
1: 2nd system  
2: . .  

long ground; Ground designation  
0: Fixed  
Common variables for each system valid only for (# 100 to # 199) (Fixed to 0 in other cases.)  

double *varptr; Variable data pointer

Program example
Write to the 1st system FORE ground common variable # 120 with standard "double".

extern long ievarwt ();

mvalwt ()
{
  long   sts;
  double   dbuff;
  .
  .
  .
  dbuff = 0.001;
  sts = ievarwt (0L,120L,0L,0L,&dbuff);
  if (sts != 0)
  {
    merror ();
  }
  .
  .
  .
}
2.1.8  ievarclear ()  CNC Variable Clear Function

Function
The CNC variable is "cleared".

Format
sts = ievarclear (mode, varno, syst, ground);

long sts;  Status
0: Normal completion
–1: Mode error
–2: Variable number error
–3: System number error
–4: Ground designation error
–5: File unformatted error

long mode;  Variable mode
0: Common variable
3: Local variable

long varno;  Variable number
(Refer to section 2.1.7.)

long syst;  System number
0: 1st system
1: 2nd system
. .

Common variables for each system valid only for (#100 to #199)
(Fixed to 0 in all cases.)

long ground;  Ground designation
0: Fixed

Program example
Clear the 1st system FORE ground common variable #120.

extern long ievarclear ();
mvalclear ()
{
    long sts;
    .
    .
    sts = ievarclear (0L, 120L, 0L, 0L);
    if (sts != 0)
    {
        merror ();
    }
    .
    .
}
3. Machine Control I/F

3.1 PLC Device Accessing Functions

[Function List]

1. Setting the bit device set_
2. Resetting the bit device rst_
3. Testing the bit device tst_
4. Setting the word device set_
5. Testing the word device tst_
6. Setting the long data lset_
7. Testing the long data ltst_


3. Machine Control I/F

3.1 PLC Device Accessing Functions

3.1.1 set_ □ Setting the Bit Device

Format
set_ □ (no);
□ : Devices name
- For PLC 4B — X, Y, M, L, F, E, U, W, S, I, J, G
- For GPPQ/W — X, Y, M, L, F
long no: Device number

Function
Turns ON the designated bit device.

Example
set_X (0x40) ; Turns ON device X40.

3.1.2 rst_ □ Resetting the Bit Device

Format
rst_ □ (no);
□ : Devices name
- For PLC 4B — X, Y, M, L, F, E, U, W, S, I, J, G
- For GPPQ/W — X, Y, M, L, F
long no: Device number

Function
Turns OFF the designated bit device.

Example
rst_X (0x40) ; Turns OFF device X40.

3.1.3 tst_ □ Testing the Bit Device

Format
tst_ □ (no);
□ : Devices name
- For PLC4B — X, Y, M, L, F, E, U, W, S, I, J, G, TO, CO, QO, BO
- For GPPQ/W — X, Y, M, L, F, TO, CO
long fg: Results of bit device test
long no: Device number

Function
Returns the bit status (ON/OFF); device OFF: returns 0.
ON : returns numeric data other than 0.

Example
extern long tst_X () ; "extern" declaration to byte size.
tst_X (0x40) ; Test the status of device X40.

(Note)
The "extern" declaration must be executed.
3.1.4  set_□ Setting the Word Device

Format
set_□ (no, setdt) ;
□ : Devices name  D, R, T, C
long no : Device number
long setdt : Data to be set

Function
Sets data in the designated word device.

Example
set_R (1900, 100) ; Sets 100 into device R1900.

(Note)
The set data must not exceed the word size range.
If the set data exceeds the word size, the data that is set will not be guaranteed.

3.1.5  tst_□ Testing the Word Device

Format
tstdt = tst_□ (no) ;
□ : Devices name  D, R, T, C
long tstdt : Results of word device test
long no : Device number

Function
Returns the data stored in the designated word device with a word size.

Example
extern long tst_R () ; "extern" declaration to word size
tstdt = tst_R (1900) ; Returns the data stored in device R1900.

(Note)
The "extern" declaration must be executed.
3.1.6  lset_ □  Setting the Long Data

Format
lset_ □ (no, setdt) ;  
□ : Devices name  D, R  
long no : Device number  
long setdt: Long data to be set

Function
Sets the data in the designated word device with a long size.

Example
lset_R (1900, 0x12345678) ; Sets 0x12345678 in devices R1900 and R1901.

(Note)
The set data must not exceed the long size range.  
If the set data exceeds the long size, the data that is set will not be guaranteed.  
When using a word device with a long size, use two devices. (no and no +1).  
Note that the high-order two bytes and low-order two bytes of the set data will be interchanged.  
An actual example is given below.

Device configuration:

<table>
<thead>
<tr>
<th>Device</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1900</td>
<td>0x5678</td>
</tr>
<tr>
<td>R1901</td>
<td>0x1234</td>
</tr>
</tbody>
</table>

Execute "lset_R (1900,0x12345678)"; →
3.1.7  ltst_□  Testing the Long Data

Format

\[
\text{tstdt} = \text{ltst}_\square (\text{no}) ;
\]

\(\square\) : Devices name  D, R  

long tstdt : Results of the word device test 

long no : Device number

Function

Returns the data stored in the designated word devices with a long size.

Example

eextern long tst_R () ; "extern" declaration to long size. 

tstdt = tst_R (1900); Returns the data in device R1900 with a long size.

(Note)

The "extern" declaration must be executed.
When using a word device with a long size, use two devices. (no and no +1).
Note that the high-order two bytes and low-order two bytes of the returned data will be interchanged.  
An actual example is given below.

<table>
<thead>
<tr>
<th>Device</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1900</td>
<td>0x01234</td>
</tr>
<tr>
<td>R1901</td>
<td>0x05678</td>
</tr>
</tbody>
</table>

Execute "tstdt=ltst_R (1900)"; \(\rightarrow\) tstdt:0x56781234

[Note]

Designate the device X and Y devices numbers as a hexadecimal, and the other device numbers as a decimal.
### 3.2 PLC Device High-speed Access Functions

The following functions are PLC device access functions used within the PLC function called with the CALL command in the ladder program.

| Function list |
|---------------|----------------|
| 1. Setting the bit device | `melplcBset` |
| 2. Resetting the bit device | `melplcBrst` |
| 3. Testing the bit device | `melplcBtst` |
| 4. Setting the word device | `melplcWset` |
| 5. Testing the word device | `melplcWtst` |
| 6. Setting the long data | `melplcLset` |
| 7. Testing the long data | `melplcLtst` |
### 3. Machine Control I/F

#### 3.2 PLC device high-speed access functions

<table>
<thead>
<tr>
<th>High-speed access to bit device function</th>
</tr>
</thead>
<tbody>
<tr>
<td>- <code>melplcBset( )</code> : Turns the designated bit device ON.</td>
</tr>
<tr>
<td>- <code>melplcBrst( )</code> : Turns the designated bit device OFF.</td>
</tr>
<tr>
<td>- <code>melplcBtst( )</code> : Checks the designated bit device.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>High-speed access to word device function</th>
</tr>
</thead>
<tbody>
<tr>
<td>- <code>melplcWset( )</code> : Sets the designated value as a word size in the designated word device.</td>
</tr>
<tr>
<td>- <code>melplcWtst( )</code> : Checks the designated word device.</td>
</tr>
<tr>
<td>- <code>melplcLset( )</code> : Sets the designated value as a double-word size in the designated word device.</td>
</tr>
<tr>
<td>- <code>melplcLtst( )</code> : Checks the designated word device with a double-word size.</td>
</tr>
</tbody>
</table>
3.2.1 melplcBset ( ) Setting the Bit Device

Format
long melplcBset (Devices name
For PLC 4B – X, Y, M, L, F, E, Ti, TO, Cl, CO, U, W, S, I, J, Qi, QO, BI, BO, G
For GPPQ/W – X, Y, M, L, F, SM, Ti, TO, CI, CO)
()

(explain)
Sets the designated bit device ON.

Return value
0 is returned when completed normally.
If the designated device number is not found, –1 is returned.

3.2.2 melplcBrst ( ) Resetting the Bit Device

Format
long melplcBrst (Devices name
For PLC 4B – X, Y, M, L, F, E, Ti, TO, Cl, CO, U, W, S, I, J, Qi, QO, BI, BO, G
For GPPQ/W – X, Y, M, L, F, SM, Ti, TO, CI, CO)
()

Explanation
Turns the designated bit device OFF.

Return value
0 is returned when completed normally.
If the designated device number is not found, –1 is returned.

3.2.3 melplcBtst ( ) Testing the Bit Device

Format
long melplcBtst (Devices name
For PLC 4B – X, Y, M, L, F, E, Ti, TO, Cl, CO, U, W, S, I, J, Qi, QO, BI, BO, G
For GPPQ/W – X, Y, M, L, F, SM, Ti, TO, CI, CO)
()

Explanation
Tests (reads) the designated bit device. The test results (ON: 1, OFF: 0) are set in bValue.

Return value
0 is returned when completed normally.
If the designated device number is not found, –1 is returned.
3.2.4 melplcWset( ) Setting the Word Device

Format
\[
\text{long melplcWset( } (\text{ Devices name: R, D}) \\
( \\
\quad \text{long devNo} \quad \text{Device number to be set} \\
\quad \text{short wValue} \quad \text{Word size value set in device} \\
( )
\]

Explanation
Sets the designated value as a word size in the designated word device

Return value
0 is returned when completed normally.
If the designated device number is not found, –1 is returned.

3.2.5 melplcWtst( ) Testing the Word Device

Format
\[
\text{long melplcWtst( } (\text{ Devices name: R, D}) \\
( \\
\quad \text{long devNo} \quad \text{Device number to be set} \\
\quad \text{short *wValue Address to store tested values} \\
\)
\]

Explanation
Tests (reads) the designated word device as a word size.
The test results are set in "wValue".

Return value
0 is returned when completed normally.
If the designated device number is not found, –1 is returned.
3.2.6  melplcLset  () Setting the Long Data

Format
long melplcLset ( Devices name: R, D)

( long  devNo    Device number to be set
long   IValue   Double-word size value set in device
)

Explanation
Sets the designated value as a double-word size in the designated word device (register).

Limit
Only even number device numbers can be designated.

Return value
0 is returned when completed normally.
If the designated device number is not found, –1 is returned. If the device number is odd, –2 is returned.

3.2.7  melplcLst () Testing the Long Data

Format
long melplcLst ( Devices name: R, D)

( long  devNo    Device number to be set
short *IValue   Address to store tested values
)

Explanation
Tests (reads) the designated word device (register) as a double-word size.
The test results are set in "IValue".

Limit
Only even number device numbers can be designated.

Return value
0 is returned when completed normally.
If the designated device number is not found, –1 is returned. If the device number is odd, –2 is returned.

Precautions
The types of devices that can be handled differ between the PLC4B and GPPQ/W ladders.
However, the library accessing the bit device automatically converts the assignment. For example, if "melplcBsetU(0)" is executed when using the GPPQ/W ladder, the same results as when "melplcBsetX (0x4C))" is executed will be attained. Refer to the "MELDAS 64 Series GPPQ/GPPW Interface Function Specifications (BNP-B3941-057)" for details on the assignment conversion.
4. File Release I/F

4.1 File Data Input/Output Functions

This is an interface used to read/write the machining programs, etc., in the CNC memory from the custom release system.

Function list

<table>
<thead>
<tr>
<th>No.</th>
<th>Function outline</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Register new machining program No.</td>
<td>prmake ()</td>
</tr>
<tr>
<td>2</td>
<td>Change machining program No.</td>
<td>prrena ()</td>
</tr>
<tr>
<td>3</td>
<td>Delete machining program</td>
<td>prdele ()</td>
</tr>
<tr>
<td>4</td>
<td>Read machining program registration state</td>
<td>prdir ()</td>
</tr>
<tr>
<td>5</td>
<td>Write comment to machining program</td>
<td>prcmwt ()</td>
</tr>
<tr>
<td>6</td>
<td>Read comment from machining program</td>
<td>prcmrd ()</td>
</tr>
<tr>
<td>7</td>
<td>Write one block of machining program</td>
<td>plwrit ()</td>
</tr>
<tr>
<td>8</td>
<td>Read one block of machining program</td>
<td>plread ()</td>
</tr>
<tr>
<td>9</td>
<td>Insert one block in machining program</td>
<td>plinst ()</td>
</tr>
<tr>
<td>10</td>
<td>Delete one block of machining program</td>
<td>pldele ()</td>
</tr>
<tr>
<td>11</td>
<td>Read number of machining program blocks</td>
<td>plrcunt ()</td>
</tr>
<tr>
<td>12</td>
<td>Read machining program being executed</td>
<td>plrunblk ()</td>
</tr>
<tr>
<td>13</td>
<td>Read machining program information</td>
<td>prinfo ()</td>
</tr>
<tr>
<td>14</td>
<td>Check program being run</td>
<td>prunchk ()</td>
</tr>
</tbody>
</table>
[Explanation of functions]

Each function is explained with the following format.

(1) `prmake()` Register new machining program No.  

Calling sequence

```c
sts = prmake (mode, name)  
```

- `sts` : Status number (Refer to section [Status])
- `mode` : Program type: Refer to section 4.1 (3).
- `name` : Program number (1 to 99999999)

Function

The machining program designated with the argument "name" is registered into the NC memory.

The details of the registered program are only EOR (%).

With "prmake()", the machining programs for the number of systems are registered.

If a machining program with the same number exists, 10 will be set in the return value `sts`.

Example of C language description

```c
Program example

To register machining program number O100.
sts = prmake (1, 100);
```

(1) The program number setting range is as follows.
- **Machining program** : Range of 1 to 99999999
- **MDI data** : Insignificant (fixed to 0)
- **Fixed cycle** : (G code) * 10

The G code can be used in the range of G70 to G89.

(Example) Set 710 for the G71 program.

(2) Maximum number of registerable characters and number of programs in MDI and fixed cycle

- **MDI data** : Number of registerable characters = 500
- **Fixed cycle** : Number of registerable characters = 12000
  Number of registerable programs = 24
For the mode setting, the high-order bytes are the system number, and the low-order bytes are the program type.

<table>
<thead>
<tr>
<th>Program type</th>
<th>System number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Machining program</td>
</tr>
<tr>
<td>2</td>
<td>MDI data</td>
</tr>
<tr>
<td>3</td>
<td>Fixed cycle program</td>
</tr>
<tr>
<td>4</td>
<td>Not used</td>
</tr>
<tr>
<td>0: $1, 1: $2</td>
<td></td>
</tr>
</tbody>
</table>

4.1.1 prmake () Register New Machining Program No.

**Format**

\[
\text{sts} = \text{prmake} (\text{mode}, \text{name})
\]

- unsigned long \(\text{sts}\) : Status number (Refer to section [Status])
- unsigned long \(\text{name}\) : Program number (1 to 99999999)

**Function**

The machining program designated with the argument "name" is registered into the NC memory. The details of the registered program are only EOR (%).

- With "prmake()", the machining programs for the number of systems are registered.
- If a machining program with the same number exists, 10 will be set in the return value \(\text{sts}\).

**Program example**

To register machining program number O100.

\[
\text{sts} = \text{prmake} (1, 100) ;
\]
4.1.2 prrena () Change Machining Program No.

**Format**

```
sts = prrena (mode, name1, name2)
```

- `unsigned long sts` : Status number (Refer to section [Status])
- `unsigned long mode` : Program type: Refer to section 4.1 (3). Insignificant (Fixed to 0)
- `unsigned long name1` : Old program number
- `unsigned long name2` : New program number

**Function**

The number of the machining program designated with argument "name1" is changed to the number designated with argument "name2". If the machining program designated with argument "name1" is not registered in the NC memory, 11 will be set in the return value "sts". If the machining program designated in argument "name2" already exists, 10 will be set in the return value "sts".

**Program example**

To change machining program O100 to O200.
```
sts = prrena (1, 100, 200) ;
```
4.1.3 prdele () Delete Machining Program

**Format**

```
sts = prdele (mode, name)
```

- `unsigned long sts` : Status number (Refer to section [Status])
- `unsigned long mode` : Program type: Refer to section 4.1 (3). Insignificant (Fixed to 0)
- `unsigned long name` : Program number (1 to 99999999)

**Function**

The machining program designated with argument "name" is deleted. With "prdele ()", the machining programs corresponding to the number of systems will be deleted. If the machining program designated with argument "name" is not registered in the NC memory, 11 will be set in the return value "sts".

**Program example**

To delete the machining program O100.
```
sts = prdele (1, 100) ;
```
4.1.4 prdir () Read Machining Program Registration State

Format

\[ \text{sts} = \text{prdir}\left(\text{mode}, \text{start}, \text{size}, &\text{buff}[0]\right) \]

- \text{unsigned long \text{sts}} : Status number (Refer to section \text{[Status]})
- \text{unsigned long \text{mode}} : Program type: Refer to section 4.1 (3).
  Insignificant (Fixed to 0)
- \text{unsigned long \text{start}} : Index number of program registered in NC memory.
  \((0 \leq \text{start})\)
- \text{unsigned long \text{size}} : Number of programs to be read out
- \text{unsigned long \text{buff}[N]} : Registered program number read buffer \((N \geq \text{size})\)

Function

The numbers of the machining programs registered in the NC memory are read into the designated buffer.
If the reading of all program numbers in the NC memory is completed while setting the program numbers corresponding to the designated number of programs (size), –1 will be set in the corresponding buffer.

Program example

When there are five machining programs (O101 to O105) in the NC memory, and three are to be read out at a time.

<table>
<thead>
<tr>
<th>NC memory</th>
<th>(\text{buff}[0])</th>
<th>(\text{buff}[1])</th>
<th>(\text{buff}[2])</th>
</tr>
</thead>
<tbody>
<tr>
<td>O101</td>
<td>101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O102</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O103</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O104</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O105</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{long \text{buff}[3]};
\text{sts} = \text{prdir}\left(1,0,3,&\text{buff}[0]\right);
\]

\[
\text{buff}[0] \quad 101 \\
\text{buff}[1] \quad 102 \\
\text{buff}[2] \quad 103 \\
\]

\[
\text{sts} = \text{prdir}\left(1,3,3,&\text{buff}[0]\right);
\]

\[
\text{buff}[0] \quad 104 \\
\text{buff}[1] \quad 105 \\
\text{buff}[2] \quad -1 \\
\]
4.1.5 prcmwt () Write Comment to Machining Program

Format
sts = prcmwt (mode, name, &comm [0])

unsigned long sts : Status number (Refer to section [Status])
unsigned long mode : Program type: Refer to section 4.1 (3).
                        Insignificant (Fixed to 0)
unsigned long name : Program number (1 to 99999999)
unsigned char comm [18] : Storage buffer for comment to be written

Function
The comment character strings in the buffer are written as the comment for the machining program designated with the argument "name".
For the comment, a character string containing up to 18 characters to the end code (NULL) is written.
If the machining program designated with argument "name" is not registered in the NC memory, 11 will be set in the return value "sts".

Program example
To set the comment for the machining program "O100" to "MITSUBISHI".
sts = prcmwt (1, 100, "MITSUBISHI") ;
4.1.6  prcmrd ()  Read Comment from Machining Program

Format

sts = prcmrd (mode, name, &comm [0])

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned long</td>
<td>sts</td>
<td>Status number (Refer to section [Status])</td>
</tr>
<tr>
<td>unsigned long</td>
<td>mode</td>
<td>Program type: Refer to section 4.1 (3). Insignificant (Fixed to 0)</td>
</tr>
<tr>
<td>unsigned long</td>
<td>name</td>
<td>Program number (1 to 99999999)</td>
</tr>
<tr>
<td>unsigned char</td>
<td>comm [18]</td>
<td>Storage buffer for comment to be read</td>
</tr>
</tbody>
</table>

Function
The comment character string of the machining program designated with the argument "name" is read into the buffer.
The comment buffer must have enough space for 18 characters.

Program example
To read the comment for the machining program "O100".
sts = prcmrd (1, 100, &comm [0]) ;
4.1.7 plwrit () Write One Block of Machining Program

Format

\[ \text{sts} = \text{plwrit} (\text{mode}, \text{name}, \text{block}, \&\text{buff}[0]) \]

- **unsigned long sts**: Status number (Refer to section [Status])
- **unsigned long mode**: Program type: Refer to section 4.1 (3). Insignificant (Fixed to 0)
- **unsigned long name**: Program number (1 to 99999999)
- **unsigned long block**: Block number (block \( \geq 1 \))
- **unsigned char buff [N]**: Storage buffer for one block of data to be written (0\(<N\leq248\))

Function

One block of data in the buffer (buff [N]) is written into the designated machining program. One block of the NC memory machining program has up to 248 bytes including EOB(\;). The block number is counted from 1. The zero (NULL) and space (\('\)\) codes in the machining program are ignored. If the designated machining program is not registered in the NC memory, 11 will be set in the return value “sts”. If the block number does not exist, 12 will be set in the return value.

Program example

To set head block of machining program “O100" to “G28X0.Y0.;”

\[ \text{sts} = \text{plwrit} (1, 100, 1, "G28X0.Y0.;") ; \]

```
O100
G28X0.;
G90G92G53X0.Y0.;
G52X0.Y0.;
M02;
%
```

```
O100
G28X0.Y0.;
G90G92G53X0.Y0.;
G52X0.Y0.;
M02;
%
```
4.1.8 plread () Read One Block of Machining Program

Format

\[
\text{unsigned long } \text{sts} : \text{Status number (Refer to section [Status])}
\]

\[
\text{unsigned long } \text{mode} : \quad \text{Program type: Refer to section 4.1 (3).}\ 
\quad \text{Insignificant (Fixed to 0)}
\]

\[
\text{unsigned long } \text{name} : \text{Program number (1 to 99999999)}
\]

\[
\text{unsigned long } \text{block} : \text{Block number (block } \geq 1)\]

\[
\text{unsigned char } \text{buff [N]} : \text{Storage buffer of one block of data to be read. (N = 248)}
\]

Function

One block of data in the designated machining program is read out the designated buffer (buff [N]). One block of the NC memory machining program has up to 248 bytes including EOB(;). The block number is counted from 1.

If the designated machining program is not registered in the NC memory, 11 will be set in the return value "sts". If the block number does not exist, 12 will be set in the return value.

Program example

To read the head block of machining program "O100".
\[
\text{sts} = \text{plread (1, 100, 1, &buff [0]) ;}
\]
4.1.9 plinst () Insert One Block in Machining Program

Format

\[
\text{sts} = \text{plinst} \left( \text{mode}, \text{name}, \text{block}, \&\text{buff [0]} \right)
\]

- unsigned long \( \text{sts} \): Status number (Refer to section 4.1 [Status])
- unsigned long \( \text{mode} \): Program type: Refer to section 4.1 (3). Insignificant (Fixed to 0)
- unsigned long \( \text{name} \): Program number (1 to 99999999)
- unsigned long \( \text{block} \): Block number (0 ≤ block)
- unsigned char \( \text{buff [N]} \): Storage buffer of one block of data to be inserted (N ≤ 248)

Function

One block of data in the designated buffer (buff [N]) is inserted into the designated machining program.

One block of the NC memory machining program has up to 248 bytes including EOB(;). The block number is counted from 1.

To insert the block data at the head of the machining program, designate 0 as the block number.

To add the block data at the end of the machining program, designate the number of blocks in the corresponding program as the block number.

Program example

To insert "G28X0.Y0.;" at the head block of the machining program "O100".

\[
\text{sts} = \text{plinst} \left(1, 100, 0, \text{"G28X0.Y0.;"} \right)
\]

\[
\begin{array}{l}
\text{O100} \\
\text{G90G92G53X0.Y0.} \\
\text{G52X0.Y0.} \\
\text{M02;} \\
\text{\%}
\end{array} \quad \begin{array}{l}
\text{O100} \\
\text{G28X0.Y0.} \\
\text{G90G92G53X0.Y0.} \\
\text{G52X0.Y0.} \\
\text{M02;} \\
\text{\%}
\end{array}
\]
4.1.10 pldele () Delete One Block of Machining Program

**Format**

\[
\text{sts} = \text{pldele} (\text{mode}, \text{name}, \text{block})
\]

- **sts** : Status number (Refer to section [Status])
- **unsigned long mode** : Program type: Refer to section 4.1 (3). Insignificant (Fixed to 0)
- **unsigned long name** : Program number (1 to 99999999)
- **unsigned long block** : Block number (block \( \geq 1 \))

**Function**

Deletes a block from the designated machining program.

**Program example**

To delete the head of the machining program "O100".
\[
\text{sts} = \text{pldele} (1, 100, 1);
\]

4.1.11 plcunt () Read Number of Machining Program Blocks

**Syntax**

\[
\text{sts} = \text{plcunt} (\text{mode}, \text{name}, &\text{blockno})
\]

- **unsigned long sts** : Status number (Refer to section [Status])
- **unsigned long mode** : Program type: Refer to section 4.1 (3). Insignificant (Fixed to 0)
- **unsigned long name** : Program number (1 to 99999999)
- **unsigned long blockno** : Number of blocks

**Function**

Returns the designated number of machining program blocks to "blockno".

**Program example**

To read the total number of blocks in the machining program "O100".
\[
\text{sts} = \text{plcunt} (1, 100, &\text{blockno}) ;
\]
4.1.12 plrunblk () Read Machining Program Being Executed

Format

```
sts = plrunblk (mode, req_num, req_len, read_num, rdbuf)
```

```
unsigned long sts : Status number (Refer to section [Status])

unsigned long mode :
  Ground designation: 0 FIXED
  System number: Refer to section 4.1 (3).

unsigned long req_num :
  1: Only block being executed
  2: Block being executed and
  3: Block being executed and
  .
  .
  .
  16: Block being executed and
       next 15 blocks

unsigned long req_len : Maximum number of characters in one block read
(including EOB)

PLRUNBLK *rdbuf : Pointer to read information buffer
(When reading number of blocks, the read information
buffers are prepared as arrays for the number of blocks to be
read, so set the pointer to the first element of the read
information buffer array.)
```

[Read information buffer (PLRUNBLK) structure]

```
typedef struct {
  char *readbuf;
  long char_num;
} PLRUNBLK;
```

Function

This function reads the current machining program block and blocks following the next command in
the program being run into the read information buffer array designated with argument "rdbuf".
The number of blocks to be read is designated with "req_num". However, if the remaining number of
blocks in the program (from the block being executed) is less than the designated number of blocks,
only the remaining number of blocks will be read. The number of blocks actually read is returned to
"read_num".
The maximum number of characters in the machining program block to be returned to the read
information buffer is designated with "req_len". The area designated with the read information buffer
readbuf must be the size designated by "req_len" for all blocks. The number of characters actually
read will be returned to "char_num". However, if the blocks to be read exceed the number of
characters designated with "req_len", the valid number of characters actually read to each buffer
will be the maximum number of characters –1 because a NULL code is added at the end.
Program example
To read the block being executed and the next block during program execution.
```
char actbf[50];
char nextbf[50];
PLRUNBLK rdbuf[2];
long read_num;

mode = 0;
req_num = 2;
req_len = 50;
rdbuf[0].readbuf = &actbf[0];
rdbuf[1].readbuf = &nextbf[0];
sts = plrunblk(mode, req_num, req_len,
    &read_num, &buff[0]);
```

<Machining program>
```
O100;
G28 XYZ ;
G00 X100. ;  \text{Block being executed}  
G01 X150. Y150. F100. ;
```

<Results>
```
actbf []
G00 X100. ;
```
```
nextbf []
G01 X150. Y150. F100. ;
```
```
rdbuf[0].char_num: 10
read_num: 2
```
```
rdbuf[1].char_num: 22
```

<Note>
(1) Limits to number of read blocks
During the disk operation, there may be cases when the designated number of blocks cannot be read. The actually read blocks will be returned to "read_num". For the block data that could not be read, a NULL code will be added at the head.
(Note that this will occur frequently in programs in which one block exceeds 200 characters.)

(2) Read character strings
Normally, one space is placed between the character strings, but in some cases, two spaces may be placed.

(3) Specifications of read block during search state
The block data can be read when the search is completed (when program is not being run) when using memory operation. For the block data read out at that time, the block being executed will be empty, and the next block will be the head of the program. Note that the number of blocks including the executed block will be returned to "read_num".
4.1.13  prinfo () (old prsize)  Read Machining Program Information

Format
\[
\text{sts} = \text{pinfo} \ (\text{mode}, \ &\text{buff}[0])
\]
\[
\text{unsigned long} \quad \text{sts} : \text{Status number (Refer to section [Status])}
\]
\[
\text{unsigned long} \quad \text{mode} : \begin{array}{l}
\text{Program type: Refer to section 4.1 (3).} \\
\text{Insignificant (Fixed to 0)}
\end{array}
\]
\[
\text{unsigned long} \quad \text{buff}[N] : \text{Machining program information (N=4)}
\]
\[
\text{buff}[0] : \text{Number of registered machining programs}
\]
\[
\text{buff}[1] : \text{Remaining number of registerable machining programs}
\]
\[
\text{buff}[2] : \text{Number of registered characters}
\]
\[
\text{buff}[3] : \text{Remaining number of registerable characters}
\]

Function
The information such as the number of registered machining programs, etc., is returned to the buff.
4.1.14 *prunchk () (old pdrchk)* Check Program Being Run

**Format**

```c
sts = prunchk (name)
```

<table>
<thead>
<tr>
<th>long</th>
<th>sts</th>
<th>Check results</th>
</tr>
</thead>
<tbody>
<tr>
<td>–1</td>
<td>Running</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Not being run</td>
<td></td>
</tr>
</tbody>
</table>

| unsigned long | name      | Program number (1 to 99999999) |

**Function**

Whether the designated machining program is currently being run or not is checked, and the results are returned to "sts".

If the program is being run, do not edit the program.
### 4.1 File Data Input/Output Functions

#### Status

<table>
<thead>
<tr>
<th>Status No.</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal completion</td>
</tr>
<tr>
<td>1</td>
<td>Argument illegal (&quot;mode&quot; value is incorrect)</td>
</tr>
<tr>
<td>2</td>
<td>Argument illegal (&quot;name&quot; value is incorrect)</td>
</tr>
<tr>
<td>3</td>
<td>Argument illegal (&quot;block&quot; value is incorrect)</td>
</tr>
<tr>
<td>5</td>
<td>Argument illegal (&quot;bknum&quot; value is incorrect)</td>
</tr>
<tr>
<td>6</td>
<td>Operation mode illegal (Not in memory operation)</td>
</tr>
<tr>
<td>8</td>
<td>Argument illegal (System number is incorrect)</td>
</tr>
<tr>
<td>10</td>
<td>Same program number found</td>
</tr>
<tr>
<td>11</td>
<td>Designated program number not found</td>
</tr>
<tr>
<td>12</td>
<td>Designated block number not found</td>
</tr>
<tr>
<td>13</td>
<td>Number of registered programs exceeded</td>
</tr>
<tr>
<td>14</td>
<td>Number of write characters exceeded</td>
</tr>
<tr>
<td>15</td>
<td>No EOB (;) code</td>
</tr>
<tr>
<td>16</td>
<td>Number of characters exceeded</td>
</tr>
<tr>
<td>17</td>
<td>EOR (%) found before EOB</td>
</tr>
<tr>
<td>30</td>
<td>Executing program</td>
</tr>
<tr>
<td>80</td>
<td>File is already opened</td>
</tr>
<tr>
<td>81</td>
<td>Write protected</td>
</tr>
<tr>
<td>82</td>
<td>No DDB function option</td>
</tr>
<tr>
<td>99</td>
<td>File system error</td>
</tr>
<tr>
<td>100</td>
<td>Not running machining program (generated with plrunblk)</td>
</tr>
</tbody>
</table>

#### Note
- Do not delete or change the file while the program is running.
- Do not use the file input/output function with the initialization function "mopeini()".
- Include "o_def.h" in the file where these functions are called.
5. General Functions

5.1 Data Conversion Functions

Function List

1. Convert binary character string into 32-bit numeric value  abtol ()
2. Convert hexadecimal character string into 32-bit numeric value ahtol ()
3. Convert decimal character string into 32-bit BCD atobcd ()
4. Convert decimal character string into 32-bit numeric value atol ()
5. Convert decimal character string into 16-bit numeric value atos ()
6. Convert hexadecimal into a character string dchtoa ()
7. Convert decimal into a character string ltoa ()
8. Compare two character strings ostrcmp ()
9. Convert decimal character string with decimal point into 32-bit numeric data satol ()
5.1.1 abtol () Convert Binary Character String into 32-bit Numeric Value

Format

\[
\text{short st} : \text{Error status} \quad \text{Normal completion} \rightarrow 0 \\
\text{Error occurrence} \rightarrow -1 \\
\text{char asc[]} : \text{Binary character string to be converted (0, 1 or blank)} \\
\text{long *longd} : \text{Pointer of variable where converted value is to be stored}
\]

Example

```c
char asc[] = {"10110001"};
```

A null character must be placed at the end of the character string.

```c
main()
{
    short st;
    long longd;
    st = abtol (asc,&longd);
}
```

The converted value (longd) is 177.

5.1.2 ahtol () Convert Hexadecimal Character String into 32-bit Numeric Value

Format

\[
\text{short st} : \text{Error status} \quad \text{Normal completion} \rightarrow 0 \\
\text{Error occurrence} \rightarrow -1 \\
\text{char asc[]} : \text{Hexadecimal character string to be converted (0 to 9, A to F, or _)} \\
\text{long *longd} : \text{Pointer of variable where converted value is to be stored}
\]

Example

```c
char asc[] = {"A0B1CDEF"};
```

A null character must be placed at the end of the character string.

```c
main()
{
    short st;
    long longd;
    st = ahtol (asc,&longd);
}
```

The converted value (longd) is -1598960145.
5.1.3 atobcd () Convert Decimal Character String into 32-bit BCD

**Format**

```
st = atobcd (asc, longd)
```

- `short st` : Error status  
  - Normal completion → 0  
  - Error occurrence → -1
- `char asc[]` : Decimal character string to be converted (0 to 9 or `_`)  
- `long *longd` : Pointer of variable where converted BCD data is to be stored

**Example**

```c
char asc[] = {"12345678"};

A null character must be placed at the end of the character string.

main()
{
    short st;
    long longd;

    st = atobcd (asc,&longd);
}
```

The converted BCD (longd) is 0x12345678.  

(305419896)
5.1.4  atol ()  Convert Decimal Character String into 32-bit Numeric Value

Format
st = atol (asc, longd)

- short st : Error status
  - Normal completion → 0
  - Error occurrence → −1
- char asc [] : Decimal character string to be converted (+, −, 0 to 9 or _)
- long *longd : Pointer of variable where converted value is to be stored

Example
char asc[] = {"1092387456"}
A null character must be placed at the end of the character string.

main()
{
  short st;
  long longd;
  st = atol (asc, &longd);
}
The converted value (longd) is−1092387456.
(0xBEE37D80)
5.1.5  atos () Convert Decimal Character String (Integer with Sign) into 16-bit Numeric Value

Format

\[
\text{short} \quad \text{st} : \text{Error status} \quad \text{Normal completion} \rightarrow 0 \quad \text{Error occurrence} \rightarrow -1
\]

\[
\text{char} \quad \text{asc} [] : \text{Decimal character string to be converted (+, –, 0 to 9 or \_)}
\]

\[
\text{short} \quad \text{*shortd} : \text{Pointer of variable where converted value is to be stored}
\]

Example

\[
\text{char} \quad \text{asc[]} = \{"–10923"\}
\]

\[
\text{main()}
\]

\[
\text{short} \quad \text{st;}
\]

\[
\text{short} \quad \text{shortd;}
\]

\[
\text{st} = \text{atos (asc, &shortd)};
\]

The converted value (shortd) is –10923.

(0xD555)
5.1.6  dchtoa () Convert Hexadecimal into a Character String (0 to 9, A to F)

**Format**

\[
\text{st} = \text{dchtoa} (\text{longd}, \text{keta}, \text{asc})
\]

long  \quad \text{longd} : Hexadecimal value to be converted  
short \quad \text{keta} : Designation of number of converted character string digits  
char   \quad \text{asc} [] : Converted character string

**Example**

```c
main()
{
    char  \quad \text{asc}[8];
    long  \quad \text{longd} = 0xF0E1D2;
    dchtoa (longd, 8, asc);
}
```

The hexadecimal character string is right-justified.

If the number of characters in the converted hexadecimal character string is less than the designated value, "0" will be added at the head.  
A null character is not added to the end.
5.1.7 Itoa () Convert Decimal into a Character String (0 to 9)

**Format**

\[
\text{dt} = \text{ltoa} (\text{longd}, \text{keta}, \text{asc})
\]

- long dt : Data (longd value)
- long longd : Decimal value to be converted
- short keta : Designation of number of converted character string digits
- char asc [] : Converted character string

**Example**

```c
main()
{
    char asc[10];
    long longd = 56473829;
    long dt;

    dt = ltoa (longd, 10, asc);
}
```

The decimal character string is right-justified.

If the number of characters in the converted decimal character string is less than the designated value, "0" will be added at the head. A null character is not added to the end.
5.1.8 ostrcmp () Compare Two Character Strings

**Format**

\[
\text{st} = \text{ostrcmp (longd, keta, asc)}
\]

- short \( \text{st} \) : Status (Code difference of characters that first differ)
  - Negative value : \( \text{str1} < \text{str2} \)
  - 0 : \( \text{str1} = \text{str2} \)
  - Positive value : \( \text{str1} > \text{str2} \)

char *str1 : Character string 1
char *str2 : Character string 2

**Example**

```c
char str1 = \"MITSUBISHI\" ;
char str2 = \"METSUBUSHI\" ;
char str3 = \"MITSUBISHI\" ;

main()
{
    short code;
    code = ostrcmp (str1, str2); // The character string differs.
    The code values are \'I\'-\'E\', or in other words, the code value is 4.

    code = ostrcmp (str1, str3); // The character strings are identical.
    The code value is 0.
}
```
5. General Functions

5.1  Data Conversion Functions

5.1.9  satol () Convert Decimal Character String (with Sign or Decimal Point) into long Data

**Format**

\[ \text{st} = \text{satol (str, val, num)} \]

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>short st</td>
<td>Error status Normal completion → 0</td>
</tr>
<tr>
<td>Error occurrence → −1</td>
<td></td>
</tr>
<tr>
<td>char *str</td>
<td>Pointer of decimal character string to be converted</td>
</tr>
<tr>
<td>long *val</td>
<td>Pointer of variable where converted value is to be stored</td>
</tr>
<tr>
<td>short num</td>
<td>Number of digits in fraction (num ≥ 0)</td>
</tr>
</tbody>
</table>

**Example**

```c
static char string[] = "–12345.678";

main()
{
    short st;
    long val;

    st = satol(string, &val, 3);
}
```

Examples of using this function are shown below.

<table>
<thead>
<tr>
<th>Character string to be converted</th>
<th>Number of digits after the decimal point</th>
<th>Value after conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;–12345.678&quot;</td>
<td>2</td>
<td>Error</td>
</tr>
<tr>
<td>&quot;–12345.678&quot;</td>
<td>3</td>
<td>–12345678</td>
</tr>
<tr>
<td>&quot;–12345.678&quot;</td>
<td>4</td>
<td>–123456780</td>
</tr>
<tr>
<td>&quot;–123 &quot;</td>
<td>3</td>
<td>123000</td>
</tr>
</tbody>
</table>

This function is used when the setting area buffer contents are to be converted into values, etc.
<table>
<thead>
<tr>
<th>Sub-No.</th>
<th>Date of revision</th>
<th>Revision details</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>March 2003</td>
<td>First edition created.</td>
</tr>
</tbody>
</table>
Notice

Every effort has been made to keep up with software and hardware revisions in the contents described in this manual. However, please understand that in some unavoidable cases simultaneous revision is not possible.

Please contact your Mitsubishi Electric dealer with any questions or comments regarding the use of this product.

Duplication Prohibited

This instruction manual may not be reproduced in any form, in part or in whole, without written permission from Mitsubishi Electric Corporation.

© 2003 MITSUBISHI ELECTRIC CORPORATION
ALL RIGHTS RESERVED
<table>
<thead>
<tr>
<th>MODEL</th>
<th>M60 Series</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
<td>008-254</td>
</tr>
<tr>
<td>Manual No.</td>
<td>BNP-B2219E(ENG)</td>
</tr>
</tbody>
</table>