

CHAPTER 14

USING MACROS

14.1. Types of Macros	1
14.2. Working with Macros.....	3
14.2.1. Creating Macros.....	3
14.2.2. Opening and Closing Macros	3
14.2.3. Naming a macro.....	4
14.2.4. Deleting a macro.....	4
14.2.5. Saving and Exporting Macros	4
14.2.6. Macro Settings in the Dialog	5
14.3. Writing Macros.....	7
14.3.1. Macro Editor Window	7
14.3.2. Macro Command Properties Tool Window	9
14.4. Macro Commands and Examples	10
14.4.1. Macro Notations and Terminology	10
14.4.2. Data Transfer	12
14.4.3. Arithmetic Operation	13
14.4.4. Logical Operation	14
14.4.5. Calculation.....	16
14.4.6. Data Conversion	18
14.4.7. Conditional Operation	21
14.4.8. Program Control	25
14.4.9. Timer Operation.....	28
14.4.10. Keypad Operation.....	29
14.4.11. Recipe Operation	30
14.4.12. Communication Operation.....	31
14.4.13. System Service.....	32
14.4.14. Screen Operation	33
14.4.15. File Operation	34
14.4.16. Comparison.....	40
14.4.17. String Operation.....	41
14.4.18. Run Operation.....	49
14.4.19. Print Operation.....	50
14.4.20. Sound Operation	52

This chapter explains how you can write macros to perform operations. A macro contains a sequence of macro commands and acts as a simple computer program when it is run. With macros, some tasks that are hard to be performed by the objects can be easily achieved, such as scheduling, data exchanges, conditional operations, and sequential operations.

- Note:** Do not use macros to control systems that can cause life-threatening and serious injury.
- Note:** The real-time OS in the HMI needs to manage multiple tasks at the same time when the application is running. In order to not affect the whole performance, please keep the macro as short as possible.
- Note:** Macros execute individually and are unaware of other macros. When sharing common variables between macros, your application may have possible conflicts. Consider an application where the cycle macro updates the value of an address which is used by the event macro. If the event macro alters the address value before the cycle macro uses that address, the result of the cycle macro will be incorrect.

14.1. Types of Macros

■ Global Macro

A global macro is a macro that can be used by all panel applications in the same project. With global macros, the panel applications in the same project can share common functions without having to keep and maintain the same set of macros locally.

You can set up a password in the Project Information & Protection dialog box to protect global macros. If global macros are under protection, you need to enter a password to remove the protection before using them in your application.

Note that only internal variables can be used in global macros.

■ Local Macro

A local macro is a macro that can only be used by the panel application which the macro is located in.

■ Sub-macro

A sub-macro is a macro that can be run by other macros using the CALL command. When a CALL command is encountered while running a macro, that macro stops running, and the sub-macro starts to run. The last command of a sub-macro must be a RET command which terminates the sub-macro and returns control to the calling macro. You can place RET commands at any location you want. The HMI will resume the execution of the calling macro starting with the command following the CALL command once the called sub-macro terminates.

By implementing common functions in sub-macros for other macros to use, your macros can be modularized, are sharable, easy to read, and easy to maintain.

- **Startup Macro, Main Macro, Event Macro, Time Macro** for the application
- **Open Macro, Cycle Macro, Close Macro** for the screen
- **On Macro, Off Macro, Object Macro** for the object

Select the macro that works best for the occasion you want the macro to run, and for the purpose you want the macro to do.

Run the Macro:	Use:
When the application starts	Startup Macro This macro is run only once when the application starts. The HMI will not display the start-up screen until the macro terminates. You can use Startup Macro to initialize global data and settings for your application. Specify Startup Macro in Panel General Setup dialog box.
While the application is running	Main Macro This macro is run all the time while the application is running. The HMI runs Main Macro cyclically, i.e. it will delay preset time to run Main Macro starting from the first command again each time after it completes the processing of the last command of the macro or when it encounters an END command in the middle of the macro. Specify Main Macro in Panel General Setup dialog box.

Continued

Run the Macro:	Use:
When a specific trigger bit changes from 0 to 1	Event Macro An Event Macro is run whenever the associated trigger bit changes from 0 (off) to 1 (on). An application can have up to four Event Macros which are numbered from 1 to 4. Specify Event Macros in the Panel General Setup dialog box.
Periodically with a preset time interval	Time Macro A Time Macro is run periodically with a preset time interval. An application can have up to four Time Macros which are numbered from 1 to 4. Each Time Macro has a different set of time interval options you can choose to specify how often you want the macro to run. Specify Time Macros in the Panel General Setup dialog box.
When a specific screen is being opened	Open Macro An Open Macro is run once when the associated screen is being opened. The screen will not be displayed until the Open Macro terminates. Specify the Open Macro of a screen in the Screen Properties dialog box.
While a specific screen is open	Cycle Macro A Cycle Macro is run all the time while the associated screen is open. The Cycle Macros runs cyclically, i.e. the Cycle Macro will run starting from the first command again every time after it completes the processing of the last command of the macro, or when an END command is encountered in the middle of the macro. The Cycle Macro terminates immediately when the screen is closed. Specify the Cycle Macro of a screen in the Screen Properties dialog box.
When a specific screen is being closed	Close Macro A Close Macro is run once when the associated screen is being closed. The screen will not be erased until the Close Macro terminates. Specify the Close Macro of a screen in the Screen Properties dialog box.
When a specific button is pressed or released to set a bit to on	On Macro An On Macro is run once when the associated button is pressed or released to set a bit to 1 (on). The setting of the bit will not be performed until the On Macro terminates. Therefore, it is important to keep the On Macro as short as possible in order to not delay the setting of the bit. Both the Bit Buttons and the Toggle Switches can have an On Macro. Specify the On Macro of a button in that button's configuration dialog box.
When a specific button is pressed or released to set a bit to off	Off Macro An Off Macro is run once when the associated button is pressed or released to set a bit to 0 (off). The setting of the bit will not be performed until the Off Macro terminates. So it is important to keep the Off Macro as short as possible in order to not delay the setting of the bit. Both Bit Buttons and Toggle Switches can have an Off Macro. Specify the Off Macro of a button in that button's configuration dialog box.
When a specific object is activated to perform a specific operation	Object Macro An Object Macro is run once when the associated object is activated to perform a specific operation. Whether the macro is run before or after the operation is performed depends on the type of the operation. The objects that can have an Object Macro include Screen Buttons, Function Buttons, and Keypad Buttons. Specify the Object Macro of an object in that object's configuration dialog box.

14.2. Working with Macros

14.2.1. Creating Macros

■ Creating a new and blank macro

- 1) To create a global macro, use the Add... command on the Project > Global Macro menu, or right-click the Global > Global Macros item in the Project Manager tool window to bring out the pop-up menu and then use the Add Macro... command on the pop-up menu.
To create a local macro, use the Add... command on the Panel > Macro menu, or right-click the panel application > Macros item in the Project Manager tool window to bring out the pop-up menu and then use the Add Macro... command on the pop-up menu, or
- 2) In the New Macro dialog box, type the name you want, and hit the ENTER key or click the OK button to validate your choice.

■ Importing an existing macro as a copy macro

- 1) To import a macro as a global macro, right-click the Global > Global Macros item in the Project Manager tool window to bring out the pop-up menu and then use the Import Macro... command on the pop-up menu.
To import a macro as a local macro, right-click the panel application Macros item in the Project Manager tool window to bring out the pop-up menu and then use the Import Macro... command on the pop-up menu
- 2) Click the *.mcr or *.txt file you want to create a new macro from. If you want to open a macro that was saved in a different folder, locate and open the folder first.
- 3) Click Open.

14.2.2. Opening and Closing Macros

■ Opening an existing macro

To open a global macro, select the macro you want to open in Project > Global Macro > Edit menu, or double click the macro in Global > Global Macros item in the Project Manager tool window, or in the Macro settings of the object configuration dialog, select the macro that is located after "-----Global-----" item in the drop-down list.

To open a local macro, select the macro you want to open on Panel > Macro > Edit menu, or double click the macro in the panel application > Macros item in the Project Manager tool window, or in the Macro settings of the object configuration dialog. If global macros exist, select the macro that is located from the beginning to "-----Global-----" item in the drop-down list or select the macro in the drop-down list.

■ Opening a *.txt or *.mcr file within the macro editor window:

You may do the drag-and-drop operation:

- 1) Open the macro editor window by clicking any of the existing macros.
- 2) Drag a selection of *.mcr file or *.txt file into the macro editor window and drop it.

Note: Any macros in the macro editor window will be replaced by macros from the source file.

■ Closing Macro Editor Windows:

To close a single window, select the window and click the close button.

To close all windows, choose Windows... on the Window menu, select all the macro editor windows you want to close in the window dialog and then click Close Window(s) button.

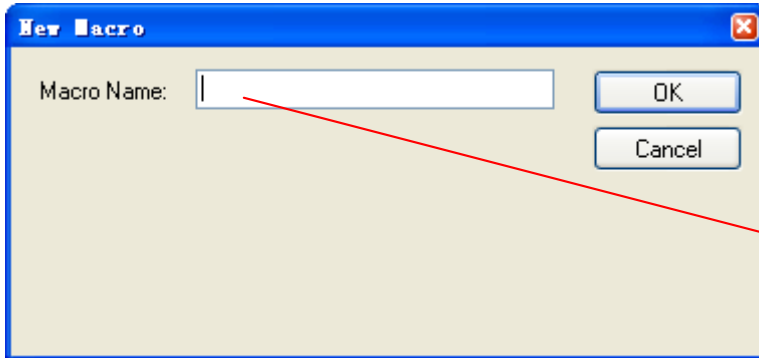
Note: The Macro Command Properties Window will be closed automatically when the macro editor window is closed. Even if the macro editor window is closed, all the changes will be saved, unless the software exits without saving any changes to the file.

■ Closing Macro Command Properties Window:

To close the macro command properties window, click the close button on the Macro Command Properties window or check/uncheck the Macro Command Properties command on the View menu

14.2.3. Naming a macro

When adding a new macro for global use or for the panel application, you need to specify the macro name with the following dialog.



Specify the macro name here. The maximum length for a macro name is 256 characters. Macro names are case insensitive. For example, the names TURN ON and turn on are considered to be the same.

When importing a file as the macro, the file name will be the macro name as the default.

In each panel application, the local macro name has to be unique, but a local macro name can be the same as a global macro name.

■ Renaming a macro from Project Manager:

- 1) Locate the macro you would like to rename
- 2) Right-click on the macro to display the macro item's pop-up menu; and then click Rename, the second menu item.
- 3) Once the macro name is selected, simply type the new name over the selected text, and then press the ENTER key.

14.2.4. Deleting a macro

■ Deleting a macro from Project Manager tool widow:

- 1) Locate the macro you would like to delete
- 2) Right-click on the macro to display the macro item's "pop-up menu"; and then click Delete, the third menu item.

■ Deleting a macro by menu

To delete a global macro, choose Project menu, click Global Macro sub-menu, and select the macro you want to delete on the Delete sub-menu

To delete a local macro, choose Panel menu, click Macro sub-menu, and select the macro you want to delete on the Delete sub-menu

Note: You can only select one macro to delete at a time. If the macro you want to delete is used by an application or object, you will be asked to confirm the delete operation.

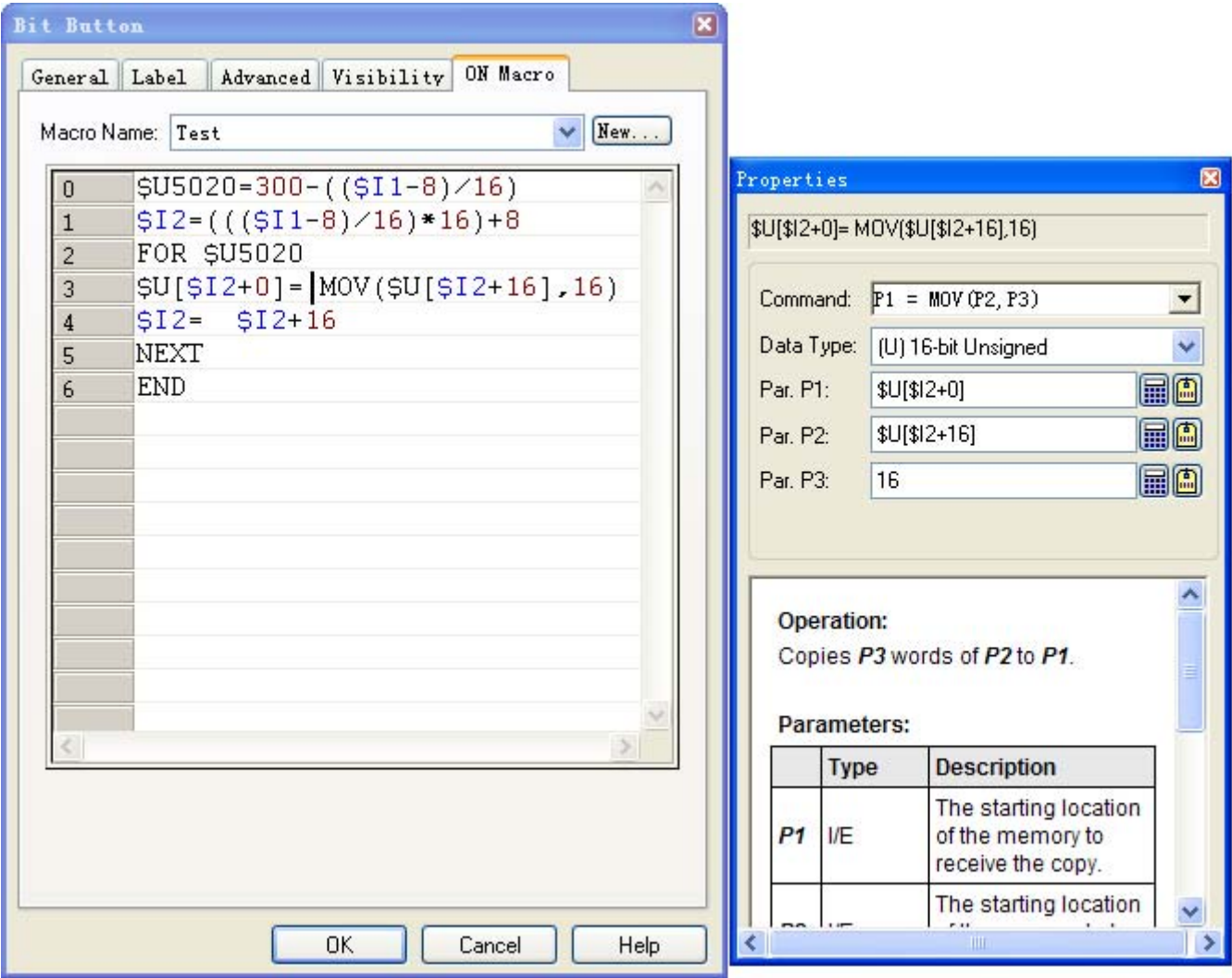
14.2.5. Saving and Exporting Macros

If you have a macro you want to reuse in another application panel, you can export the macro as a .txt file or a .mcr file. You may do the following:

- 1) Locate the macro you would like to export
- 2) Right-click on the macro to display the macro item's "pop-up menu"; and then click Export Macro..., the fourth menu item.
- 3) If you want to save a macro in a different folder, locate and open the folder first, then click Save.

14.2.6. Macro Settings in the Dialog

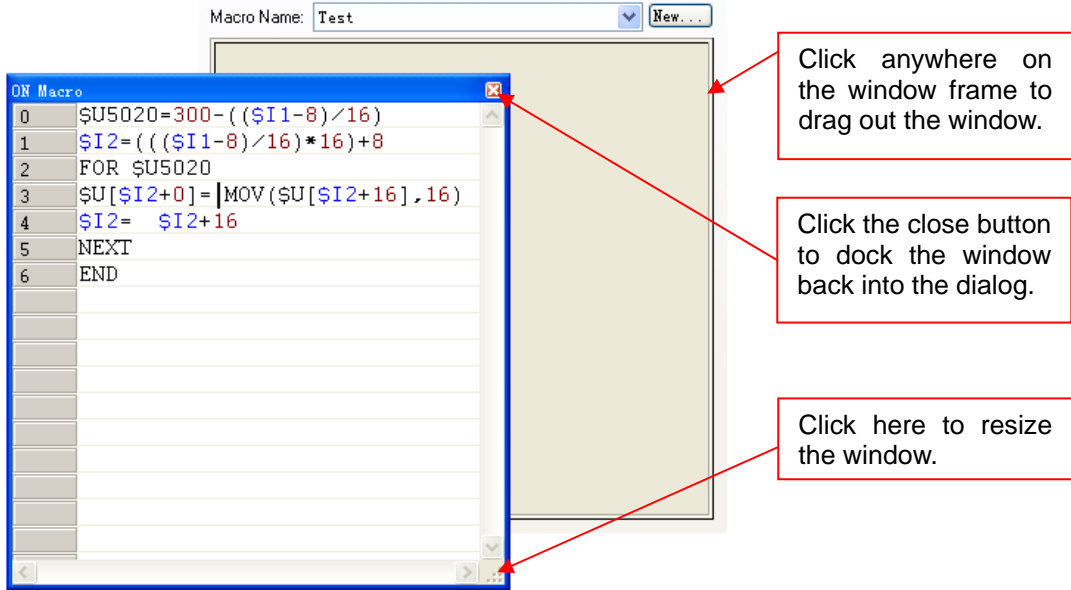
You can open and edit a specified macro or create a new macro in the configuration dialog that contains the macro page. The following is an example of the Macro page in the Bit Button configuration dialog.



The following table describes each property in the General page.

Property	Description
Macro Name	<div>Select an existing local macro or global macro from the drop-down list. The following is a sample in the dropdown list</div> <div><div><div>Test</div><div>SW</div><div>SW SCREEN0</div><div>SW SCREEN1</div><div>SW SCREEN2</div><div>SW SCREEN3</div><div>Test</div><div>-----Global-----</div><div>On</div><div>Monitor</div><div>Debug</div></div><div>Local Macros</div><div>Global Macros</div><div>A separator that is used to separate the local macros and global macros. It shows only when global macros exist.</div></div>
New...	Click the button to bring out the New Macro dialog box to create a new and blank local macro.

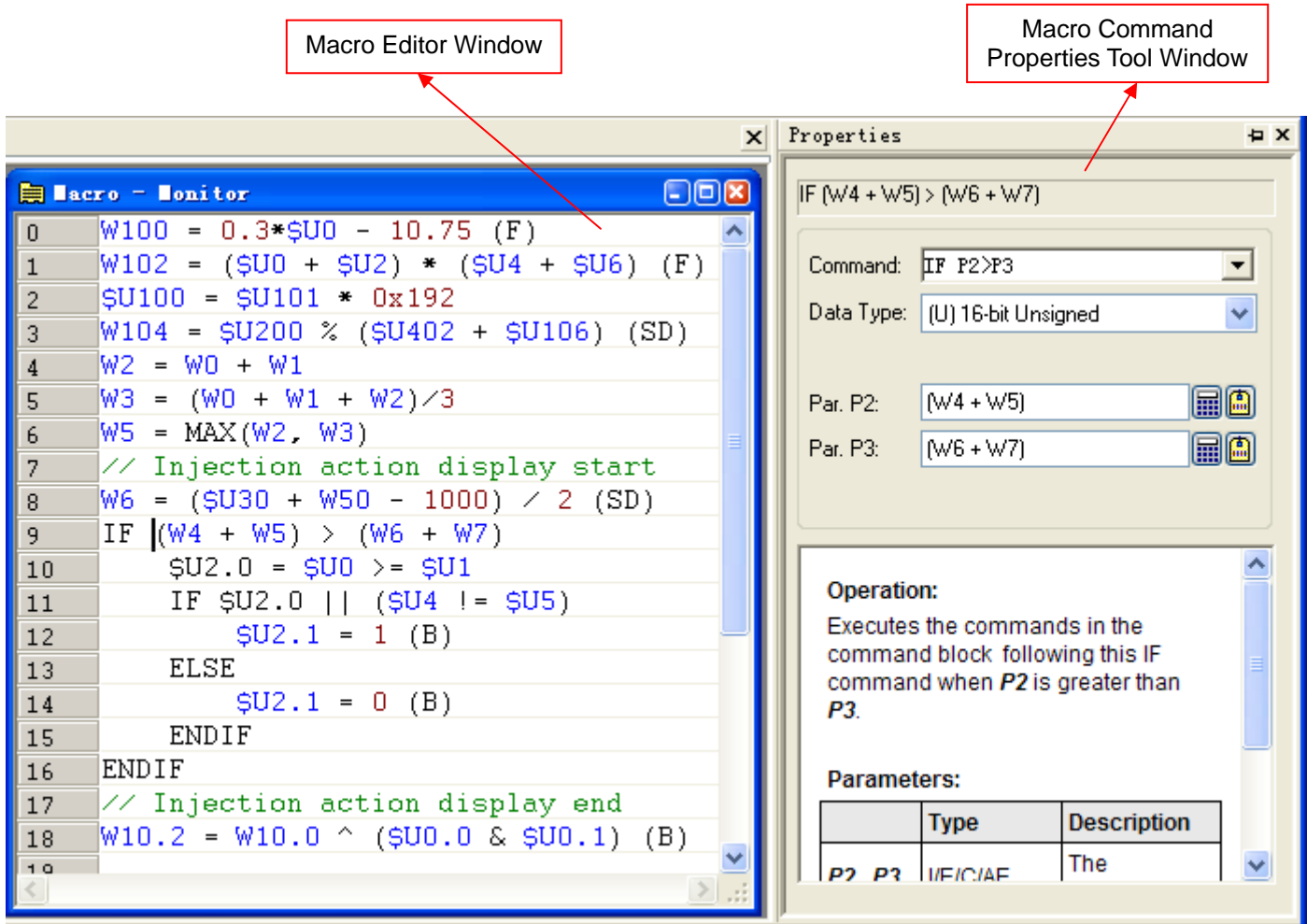
Continued

Property	Description
Macro Editor Window	<p>Write and edit the macros here. For details, see Section 14.3.1. If the editor window is too small, you may drag out the window and resize it. To drag and move the window, left-click anywhere on the window frame and hold down the button, then drag the mouse to move the window outside to another area. It will “float” over the rest of the dialog, allowing you to position it wherever you want it to be. Release the mouse button to let go of the window. Click on the resized tabs located at the bottom right corner of the window to resize the window. Press the close button to dock the window back into the dialog. The following is a sample of the floating macro editor window.</p>  <p>The screenshot shows a floating window titled 'ON Macro' with a list of commands: <pre> 0 \$U5020=300-(((\$I1-8)/16) 1 \$I2=(((\$I1-8)/16)*16)+8 2 FOR \$U5020 3 \$U[\$I2+0]=MOV(\$U[\$I2+16],16) 4 \$I2=\$I2+16 5 NEXT 6 END </pre> Red arrows indicate: <ul style="list-style-type: none"> Click anywhere on the window frame to drag out the window. Click the close button to dock the window back into the dialog. Click here to resize the window. </p>
Properties	<p>A floating dialog allows you specify the macro command. For details, see Section 14.3.2. The macro command properties dialog can be moved anywhere and resized to any size you want. However, it can't be closed until the dialog is closed.</p>

14.3. Writing Macros

In the software, all the macros can be written in the macro development environment that is composed of two elements: the Macro Editor Window and the Macro Command Properties Tool Window.

You will see the following sample of the Macro Development Environment when opening a macro from Project Manager.



14.3.1. Macro Editor Window

The macro editor is a text-based editor with syntax coloring and line numbering. Line numbering in the left margin of the page helps you refer to the specific position of the macro. Syntax coloring gives you visual cues about the structure by using different colors for various elements, such as keywords in black, comments in green, addresses in blue and constants in red.

■ Editing Macro

With the macro editor, you can cut, copy, and paste selected text using menu commands, key combinations or drag-and-drop operations. You can also undo and redo selected editing actions.

You can right-click to display a pop-up menu of editing commands. The editing commands available depend on what the pointer is pointing to.

The macro editor allows the following editing actions:

- Cutting, copying, pasting, and deleting selection of lines, multiple lines or text
- Undoing and redoing editing actions
- Using drag-and-drop editing to move or copy a selection of text within one macro editor window, or between macro editor windows.

The following table shows the supported editing commands.

Menu Command	Key Combination	Description
Cut	CTRL+X	Removes selected text from the active macro editor window.
Copy	CTRL+C	Duplicates selected text in the active macro editor window.
Paste	CTRL+V	Pastes cut or copied text into an active macro editor window.
	DELETE	Deletes text without copying it to the Clipboard.
Undo	CTRL+Z	Reverses the last editing action.
Redo	CTRL+Y	Reapplies the prior editing that has been undone.
	CTRL+A	Selects all texts in the active macro editor

Note that all editing commands require a selection in order to work. Some commands can make a selection based on the current cursor location.

■ Using Comments in Macros

Comments are notes to be ignored when running the macro commands. Macro supports both single-line comments and block comments. Single-line comments begin with two forward slashes (//) and run to the end of the line.

The following is an example of a macro command followed by a single-line comment.

```
IF $U0.0 (B) // Key Down
```

Block comments begin with an opening delimiter (/*) and run to a closing delimiter (*). Comments do not nest.

The following is an example of a block comment.

```
/* $N1001=WH2021
   $N1010=$N1001 */
```

■ Specifying Constants in Macros

To specify a hexadecimal number, use either the h or H suffix. For example, 12abH and 3ABh are valid hexadecimal numbers. You can also use either the “0x” or “0X” prefix. For example, 0x1278abc and 0XFFFF0000 are valid hexadecimal numbers.

To specify a binary number, use either the b or B suffix. For example, 001100111b and 11110000B are valid binary numbers.

For decimal numbers, in most cases, you just type the numbers as they are to specify the constants. However, ambiguity exists when a constant is the same as a valid external variable. For example, if a panel application has a link to a Modicon ModBus slave device, it is impossible to tell whether the number 40001 is a constant or a word address of the controller. To avoid this kind of ambiguity, use the following methods to explicitly declare that a number is a constant:

- 1) Use K, k, D, or d suffix for an integer number. For example, -123K and -123d are valid specifications of constant -123.
- 2) Use either the f or F suffix for a decimal number with decimal point. For example, -12.3F and -12.3f are valid specifications of constant -12.3.



14.3.2. Macro Command Properties Tool Window

The Macro Command Properties Tool Window help you add and modify a macro command quickly and easily.

If you open a macro from Project Manager or Menu Item, the Macro Properties Tool Window will be opened as a docking window. You can easily configure the dockable tool window to automatically be displayed or hide, or tab link with other tool windows, or dock against the edges, or float over. When the Macro Editor is opened, you can also choose to open or close the Macro Command Properties Tool Window by clicking the [Macro Command Properties] menu item under [View] menu.

If you open the macro from an object's configuration dialog box, the Macro Properties Tool Window will float beside the Macro Editor and can be moved anywhere, but it can't be closed.

The following table describes each property in the macro command properties tool window.

Property		Description
Command		Click the dropdown list box to bring up the macro command selection dialog. In the dialog, navigate the keyword of macro commands through tabs and sections by moving the mouse and then clicking the selection. The format of the selected macro command will be shown in the dropdown list after the dialog is closed. To cancel the operation, click anywhere outside the macro command selection dialog.
Data Type		Selects the data type of the macro command from the dropdown list. Different macro commands support different data types. The supported data types for each macro command are some of the following: (S) 16-bit Signed, (U) 16-bit Unsigned, (SD) 32-bit Signed, (UD) 32-bit Unsigned, (F) 32-bit Floating Point, (B) Bit.
Parameter	<Edit Box>	Specifies the bit variable when the Data Type is (B) . Specifies the word variable when the Data Type is (U)/(S) . Specifies the double-word variable when the Data Type is (UD)/(SD)/(F) .
		Click this icon to bring up the Address Input Keypad and specify the desired address for the Variable field.
		Click this icon to bring up the Select Tag dialog box and select the desired tag for the Variable field.
Macro Command Help		Shows the operation and parameter type of the selected macro command.

Note that any modification in the dialog will change the current macro command in the Macro Editor.

14.4. Macro Commands and Examples

14.4.1. Macro Notations and Terminology

The following notations and terminology will be used in the Macro Commands and Examples sections.

■ Notations

- 1) *P1, P2, P3, P4, P5*: Parameters of macro commands.
- 2) *I, E, C, A, CS, M, AE, CE*: Used to indicate the type of parameter a macro command can accept for a specific command parameter.

Abbreviation	Parameter Type
I	Internal Variable
E	External Variable
C	Constant
A	ASCII character string
CS	Character string of the program label
M	Sub-macro name
AE	Arithmetic expression
CE	Comparison expression

- 3) *U, S, UD, SD, F, B*: Used to indicate the types of data a macro command can support.

Abbreviation	Data Type
U	16-bit Unsigned Integer
S	16-bit Signed Integer
UD	32-bit Unsigned Integer
SD	32-bit Signed Integer
F	32-bit Floating Point
B	Bit

■ Terminology

Terminology	Definition
Internal memory	The memory space in the HMI that can be accessed by the panel application. For example, the user memory \$U, the non-volatile memory \$N, the system memory \$S, and the recipe memory \$R are all parts of the internal memory.
Internal variable	An address or a tag referring to an address of a space in the internal memory.
Internal bit variable	An internal variable that refers to a bit in the internal memory. For ease of reading, “internal variable” is used instead of “internal bit variable” when referring to a bit if there is no ambiguity.
Internal word variable	An internal variable that refers to a word in the internal memory. The variables can also be used to refer to a double-word, a block of bytes (byte array), a block of words (word array), and a block of double-words (double-word array). For ease of reading, “internal variable” is used instead of “internal word variable” when referring to a word or a block of memory space if there is no ambiguity.
External memory	The memory space or collection of addressable devices in the controllers that can be accessed by the panel application through communication links.

Continued

Terminology	Definition																																																																										
External variable	An address or a tag referring to an address of a space in the external memory.																																																																										
External bit variable	An external variable that refers to a bit in the external memory. For ease of reading, “external variable” is used instead of “external bit variable” when referring to a bit if there is no ambiguity.																																																																										
External word variable	An external variable that refers to a word in the external memory. The variables can also be used to refer to a double-word, a block of bytes (byte array), a block of words (word array), and a block of double-words (double-word array) if the access unit of the associated addresses is word. If the access unit is double-word, you can only use the variable to refer to a double-word or a block of memory space with a length of a multiple of 4 (bytes). For ease of reading, “external variable” is used instead of “external word variable” when referring to a word or a block of memory space if there is no ambiguity,																																																																										
Expression	<table><tr><th>Type</th><th>Abbreviation</th><th>Description</th></tr><tr><td>Arithmetic Expression</td><td>AE</td><td>Sequences of operators and parameters that are used for computing a value from the parameters.</td></tr><tr><td>Comparison Expression</td><td>CE</td><td>Sequences of operators and parameters that are used for comparing value from the parameters.</td></tr></table>			Type	Abbreviation	Description	Arithmetic Expression	AE	Sequences of operators and parameters that are used for computing a value from the parameters.	Comparison Expression	CE	Sequences of operators and parameters that are used for comparing value from the parameters.																																																															
	Type	Abbreviation	Description																																																																								
	Arithmetic Expression	AE	Sequences of operators and parameters that are used for computing a value from the parameters.																																																																								
	Comparison Expression	CE	Sequences of operators and parameters that are used for comparing value from the parameters.																																																																								
	The software provides the following types of operators for macro expressions:																																																																										
	<table><tr><th>Operators</th><th>Name or Meaning</th><th>Grouping</th><th>Used for</th></tr><tr><td>()</td><td>Parentheses</td><td>Left to right</td><td>AE/CE</td></tr><tr><td>*</td><td>Multiplication</td><td>Left to right</td><td rowspan="7">AE</td></tr><tr><td>/</td><td>Division</td><td>Left to right</td></tr><tr><td>%</td><td>Modulus</td><td>Left to right</td></tr><tr><td>+</td><td>Addition</td><td>Left to right</td></tr><tr><td>-</td><td>Subtraction</td><td>Left to right</td></tr><tr><td><<</td><td>Left shift</td><td>Left to right</td></tr><tr><td>>></td><td>Right shift</td><td>Left to right</td></tr><tr><td><</td><td>Less than</td><td>Left to right</td><td rowspan="6">CE</td></tr><tr><td>></td><td>Greater than</td><td>Left to right</td></tr><tr><td><=</td><td>Less than or equal to</td><td>Left to right</td></tr><tr><td>>=</td><td>Greater than or equal to</td><td>Left to right</td></tr><tr><td>==</td><td>Equality</td><td>Left to right</td></tr><tr><td>!=</td><td>Inequality</td><td>Left to right</td></tr><tr><td>&</td><td>Bitwise AND</td><td>Left to right</td><td rowspan="3">AE</td></tr><tr><td>^</td><td>Bitwise exclusive OR</td><td>Left to right</td></tr><tr><td> </td><td>Bitwise inclusive OR</td><td>Left to right</td></tr><tr><td>&&</td><td>Logical AND</td><td>Left to right</td><td>CE</td></tr><tr><td> </td><td>Logical OR</td><td>Left to right</td><td>CE</td></tr><tr><td>=</td><td>Assignment</td><td>Right to left</td><td>AE/CE</td></tr></table>				Operators	Name or Meaning	Grouping	Used for	()	Parentheses	Left to right	AE/CE	*	Multiplication	Left to right	AE	/	Division	Left to right	%	Modulus	Left to right	+	Addition	Left to right	-	Subtraction	Left to right	<<	Left shift	Left to right	>>	Right shift	Left to right	<	Less than	Left to right	CE	>	Greater than	Left to right	<=	Less than or equal to	Left to right	>=	Greater than or equal to	Left to right	==	Equality	Left to right	!=	Inequality	Left to right	&	Bitwise AND	Left to right	AE	^	Bitwise exclusive OR	Left to right		Bitwise inclusive OR	Left to right	&&	Logical AND	Left to right	CE		Logical OR	Left to right	CE	=	Assignment	Right to left	AE/CE
	Operators	Name or Meaning	Grouping	Used for																																																																							
	()	Parentheses	Left to right	AE/CE																																																																							
	*	Multiplication	Left to right	AE																																																																							
	/	Division	Left to right																																																																								
	%	Modulus	Left to right																																																																								
	+	Addition	Left to right																																																																								
	-	Subtraction	Left to right																																																																								
	<<	Left shift	Left to right																																																																								
	>>	Right shift	Left to right																																																																								
	<	Less than	Left to right	CE																																																																							
	>	Greater than	Left to right																																																																								
	<=	Less than or equal to	Left to right																																																																								
	>=	Greater than or equal to	Left to right																																																																								
	==	Equality	Left to right																																																																								
	!=	Inequality	Left to right																																																																								
	&	Bitwise AND	Left to right	AE																																																																							
	^	Bitwise exclusive OR	Left to right																																																																								
	Bitwise inclusive OR	Left to right																																																																									
&&	Logical AND	Left to right	CE																																																																								
	Logical OR	Left to right	CE																																																																								
=	Assignment	Right to left	AE/CE																																																																								
Note: The above table lists the operators in order of precedence (from highest to lowest precedence). Operators in the same segment of the table have equal precedence and are evaluated in the given order in an expression unless explicitly forced by parentheses.																																																																											

14.4.2. Data Transfer

Assignment (=)

Format	$P1 = P2$	Data Type	U/S/UD/SD/F/B
Function	Assigns the value of $P2$ to $P1$.		
$P1$ (I/E)	The destination.		
$P2$ (I/E/C/AE)	The source.		
Example 1	$\$U2 = 123.45$ (F) /* Assign 123.45 to \$U2 (and \$U3) */		
Example 2	$\$U100.f = 1$ (B) /* Turn on the specified bit */		
Example 3	$W60 = (\$U30 + \$W50 - 1000) / 2$ (SD) /* Write the result of the arithmetic expression to W60. */		
Example 4	$V0.0 = 2 \setminus M0$ (B) /* Assign the bit value of M0 of link 2 to the bit V0.0 of link 1*/		

Logical NOT (= !)

Format	$P1 = ! P2$	Data Type	B
Function	Reverses $P2$ and saves the result in $P1$.		
$P1$ (I/E)	The location to save the result.		
$P2$ (I/E)	The operand.		
Example 1	$\$U2.3 = !\$U3.4$ (B) /* If \$U3.4 is 1 (On), \$U2.3 is 0 (Off) */		

""

Format	$P1 = "P2"$
Function	Copies the quoted ASCII character string $P2$ to $P1$. Note that the string is a null terminated string. If the length of the string is N then N+1 bytes will be copied to $P1$ and the last byte is 0.
$P1$ (I)	The location to save the result.
$P2$ (A)	The quoted ASCII character string.
Example 1	$\$U60 = "TEST"$ /* The null character (00h) will be moved to the low byte of \$U62 */
Example 2	$\$U20 = "ABCDE"$ /* The null character (00h) will be moved to the high byte of \$U22 */

MOV

Format	$P1 = \text{MOV}(P2, P3)$	Data Type	U
Function	Copies $P3$ words of $P2$ to $P1$.		
$P1$ (I/E)	The starting location of the memory to receive the copy.		
$P2$ (I/E)	The starting location of the memory to be copied.		
$P3$ (I/C)	The number of words to be copied.		
Example 1	$\$U100 = \text{MOV}(\$U200, 16)$ /* Copy the 16 words starting from \$U200 to \$U100 */		
Example 2	$W60 = \text{MOV}(\$U200, \$U2)$ /* Copy the word array starting from \$U200 with the size specified in \$U2 to W60. */		
Example 3	$\$U10 = \text{MOV}(2 \setminus D100, 10)$ /* Copy D100 ~ D109 of link 2 to \$U10 ~ \$U19. */		

SETM

Format	$P1 = \text{SETM}(P2, P3)$	Data Type	U
Function	Sets $P3$ words of $P1$ to word value $P2$.		
$P1$ (I/E)	The starting location of the memory to be set.		
$P2$ (I/C)	The set value or the location that holds the set value.		
$P3$ (I/C)	The number of words to be set. The max. no. of words are 512.		
Example 1	$\$U100 = \text{SETM}(0, 16)$ /* Set the 16 words starting from \$U100 to 0. */		
Example 2	$W60 = \text{SETM}(\$U200, \$U2)$ /* Set the words of the word array starting from W60 with the size specified in \$U2 to the value of \$U200.*/		

14.4.3. Arithmetic Operation

Addition (+)

Format	$P1 = P2 + P3$	Data Type	U/S/UD/SD/F
Function	Adds $P2$ and $P3$ and saves the result in $P1$.		
$P1$ (I/E)	The location to save the result.		
$P2, P3$ (I/E/C/AE)	The operands.		
Example 1	$\$U100 = \$U101 + \$U102$ (U)		
Example 2	$W100 = 0.3*\$U0 + 0.1*\$U2 + 0.6*\$U4$ (F)		

Subtraction (-)

Format	$P1 = P2 - P3$	Data Type	U/S/UD/SD/F
Function	Subtracts $P3$ from $P2$ and saves the result in $P1$.		
$P1$ (I/E)	The location to save the result.		
$P2, P3$ (I/E/C/AE)	The operands.		
Example 1	$\$U100 = \$U101 - \$U102$ (U)		
Example 2	$W100 = 0.3*\$U0 - 10.75$ (F)		

Multiplication (*)

Format	$P1 = P2 * P3$	Data Type	U/S/UD/SD/F
Function	Multiplies $P2$ by $P3$ and saves the product in $P1$.		
$P1$ (I/E)	The location to save the product. If the product overflows, the higher bits exceeding the limit will be truncated and the remaining bits will be stored in $P1$.		
$P2, P3$ (I/E/C/AE)	The operands.		
Example 1	$\$U100 = \$U102 * 0x192$		
Example 2	$W100 = (\$U0 + \$U2) * (\$U4 + \$U6)$ (F)		

Division (/)

Format	$P1 = P2 / P3$	Data Type	U/S/UD/SD/F
Function	Divides P2 by P3 and saves the quotient in P1 .		
P1 (I/E)	The location to save the result.		
P2,P3 (I/E/C/AE)	The operands.		
Example 1	\$U100 = \$U101 / \$U102 (U)		
Example 2	W100 = (\$U0 + \$U2) / (\$U4 + \$U6) (F)		

Modulus (%)

Format	$P1 = P2 \% P3$	Data Type	U/S/UD/SD
Function	Divides P2 by P3 and saves the remainder in P1 .		
P1 (I/E)	The location to save the result.		
P2,P3 (I/E/C/AE)	The operands.		
Example 1	\$U100 = \$U30 % 16(U)		
Example 2	W100 = \$U200 % (\$U402 + \$U106) (SD)		

14.4.4. Logical Operation

Bitwise Inclusive OR (|)

Format	$P1 = P2 P3$	Data Type	U/UD/B
Function	Performs bitwise Inclusive OR operation of P2 and P3 and saves the results in P1 .		
P1 (I/E)	The location to save the result.		
P2,P3 (I/E/C)	The operands		
Example 1	W60 = 1111000000001111b \$U100 = 0000111100001111b W60 (U) /* The value of \$U100 is 1111111100001111b */		
Example 2	B15 = \$U1.2 B14 (B) /* If either \$U1.2 or B14 has a value of 1(On), B15 has the value 1(On). Otherwise, B15 has the value 0(Off) */		

Bitwise AND (&)

Format	$P1 = P2 \& P3$	Data Type	U/UD/B
Function	Performs bitwise AND operation of P2 and P3 and saves the results in P1 .		
P1 (I/E)	The location to save the result.		
P2,P3 (I/E/C)	The operands		
Example 1	W60 = 1111000000001111b \$U100 = 0000111100001111b & W60 (U) /* The value of \$U100 is 0000000000001111b */		
Example 2	B15 = \$U1.2 & B14 (B) /* If both \$U1.2 and B14 are 1(On), B15 is set to 1(On). Otherwise B15 is set to 0(Off) */		

Bitwise Exclusive OR (^)

Format	$P1 = P2 \wedge P3$	Data Type	U/UD/B
Function	Performs bitwise Exclusive OR operation of $P2$ and $P3$ and saves the results in $P1$.		
$P1$ (I/E)	The location to save the result.		
$P2, P3$ (I/E/C)	The operands		
Example 1	$W60 = 1111000000001111b$ $\$U100 = 0000111100001111b \wedge W60$ (U) /* The value of $\$U100$ is 1111111100000000b.*/		
Example 2	$B15 = \$U1.2 \wedge B14$ (B) /*If both $\$U1.2$ and $B14$ are 1(On) or 0(Off), the $B15$ is set to 0(Off). Otherwise $B15$ is set to 1(On)*/		

Left Shift (<<)

Format	$P1 = P2 \ll P3$	Data Type	U/UD
Function	Shifts $P2$ to the left by $P3$ bits and saves the results in $P1$. The operation supports the logic shift only.		
$P1$ (I/E)	The location to save the result.		
$P2$ (I/E/C)	The value or the location that holds the value to be shifted.		
$P3$ (I/E/C)	The number of bits to be shifted.		
Example 1	$\$U100 = \$U101 \ll 8$ (U)		
Example 2	$W200 = W100 \ll \$U10$ (UD)		

Right Shift (>>)

Format	$P1 = P2 \gg P3$	Data Type	U/UD
Function	Shifts $P2$ to the right by $P3$ bits and saves the results in $P1$. The operation supports the logic shift only.		
$P1$ (I/E)	The location to save the result.		
$P2$ (I/E/C)	The value or the location that holds the value to be shifted.		
$P3$ (I/E/C)	The number of bits to be shifted.		
Example 1	$\$U100 = \$U101 \gg 8$ (U)		
Example 2	$W200 = W100 \gg \$U10$ (UD)		

Logical AND (&&)

Format	$P1 = P2 \&\& P3$	Data Type	B
Function	Saves 1 in $P1$ if both $P2$ and $P3$ are 1, otherwise saves 0 in $P1$.		
$P1$ (I/E)	The bit to save the result.		
$P2, P3$ (I/E/C)	The operands.		
Example 1	$\$U100.0 = \$U101.0 \&\& \$U101.1$ (B)		

Logical OR (||)

Format	$P1 = P2 \parallel P3$	Data Type	B
Function	Saves 1 in P1 if either or both P2 and P3 are 1, otherwise saves 0 in P1 .		
P1 (I/E)	The bit to save the result.		
P2,P3 (I/E/C)	The operands.		
Example 1	$\$U100.0 = \$U101.0 \parallel \$U101.1$ (B)		

14.4.5. Calculation

MAX

Format	$P1 = \text{MAX}(P2,P3)$	Data Type	U/S/UD/SD/F
Function	Sets P1 to the larger value of P2 and P3 .		
P1 (I/E)	The location to save the result.		
P2,P3 (I/E/C)	The operands.		
Example 1	$\$U100 = \text{MAX}(100, 200)$ /* Set \$U100 to 200 */		

MIN

Format	$P1 = \text{MIN}(P2,P3)$	Data Type	U/S/UD/SD/F
Function	Sets P1 to the smaller value of P2 and P3 .		
P1 (I/E)	The location to save the result.		
P2,P3 (I/E/C)	The operands.		
Example 1	$\$U100 = \text{MIN}(100, 200)$ /* Set \$U100 to 100 */		

BMAX

Format	$P1 = \text{BMAX}(P2,P3)$	Data Type	U/S/UD/SD/F
Function	Finds the maximum in an array starting from P2 with P3 elements and saves the result in P1 .		
P1 (I)	The location to save the result.		
P2 (I)	The starting location of the array.		
P3 (I/C)	The size of the array.		
Example 1	$\$U100 = \text{BMAX}(\$U200, 16)$ (F) /* Find the maximum among 16 floating point numbers starting from \$U200 and save the result in \$U100 */		

BMIN

Format	$P1 = \text{BMIN}(P2,P3)$	Data Type	U/S/UD/SD/F
Function	Finds the minimum in an array starting from P2 with P3 elements and saves the result in P1 .		
P1 (I)	The location to save the result.		
P2 (I)	The starting location of the array.		
P3 (I/C)	The size of the array.		
Example 1	$\$U100 = \text{BMIN}(\$U200, 60)$ (F) /* Find the minimum among 60 floating point numbers starting from \$U200 and save the result in \$U100 */		

SUM

Format	<i>P1</i> = SUM(<i>P2</i> , <i>P3</i>)	Data Type	U/S/UD/SD/F
Function	Calculates the sum of the value in an array starting from <i>P2</i> with <i>P3</i> elements and saves the result in <i>P1</i> .		
<i>P1</i> (I)	The location to save the result.		
<i>P2</i> (I)	The starting location of the array.		
<i>P3</i> (I/C)	The size of the array.		
Example 1	\$U100 = SUM(\$U200, 16) (F) /* Calculate the sum of 16 floating point numbers starting from \$U200 and save the result in \$U100 */		

XSUM

Format	<i>P1</i> = XSUM(<i>P2</i> , <i>P3</i>)	Data Type	U/UD
Function	Calculates one element XOR (Bitwise Exclusive OR) sum of all the <i>P3</i> elements in an array starting from <i>P2</i> and saves the result in <i>P1</i> .		
<i>P1</i> (I)	The location to save the result.		
<i>P2</i> (I)	The starting location of the array.		
<i>P3</i> (I/C)	The size of the array.		
Example 1	\$U100 = XSUM(\$U200, 5) (UD) /* Perform XOR sum of 5 32-bit unsigned numbers starting from \$U200 and save the result in \$U100. Another expression of XOR sum is \$U100 = \$U200 ^ \$U202 ^ \$U204 ^ \$U206 ^ \$U208 (UD) */ \$U100 =1001B \$U101 =1100B \$U102 =0110B \$U120 = XSUM(\$U100,3) /* \$U120=0011B */		

SWAP

Format	SWAP(<i>P1</i> , <i>P2</i>)	Data Type	U
Function	Swaps the low byte and high byte of every word in a word array starting from <i>P1</i> with <i>P2</i> words.		
<i>P1</i> (I)	The starting location of the array.		
<i>P2</i> (I/C)	The size of the array.		
Example 1	\$U120=1111111100000000B \$U121=1000000100000000B SWAP(\$U120, 2) /* The value of \$U120 will be 0000000011111111B, The value of \$U121 will be 000000010000001B */		

14.4.6. Data Conversion

BCD

Format	$P1 = \text{BCD}(P2)$	Data Type	U/UD
Function	Converts binary number $P2$ to a BCD number and saves the result in $P1$.		
$P1$ (I/E)	The location to save the result.		
$P2$ (I/E/C)	The binary number to be converted.		
Example 1	$\$U100 = \text{BCD}(0x1234) (U) /* \text{The value of } \$U100 \text{ will be } 1234. */$		

BIN

Format	$P1 = \text{BIN}(P2)$	Data Type	U/UD
Function	Converts BCD number $P2$ to a binary number and saves the result in $P1$.		
$P1$ (I/E)	The location to save the result.		
$P2$ (I/E/C)	The BCD number to be converted.		
Example 1	$\$U100 = \text{BIN}(1234) (U) /* \text{The value of } \$U100 \text{ will be } 0x1234. */$		

DW

Format	$P1 = \text{DW}(P2)$	Data Type	U/S
Function	Converts 16-bit number $P2$ to a 32-bit number and saves the result in $P1$.		
$P1$ (I/E)	The location to save the result.		
$P2$ (I/E/C)	The 16-bit number to be converted.		
Example 1	$\$U100 = \text{DW}(12345) (S) /* \text{The value of } \$U100 \text{ will be } 12345 \text{ and the value of } \$U101 \text{ will be } 0. */$		
Example 2	$\$U200 = \text{DW}(-12345) (S) /* \text{The value of } \$U200 \text{ will be } -12345 \text{ and the value of } \$U201 \text{ will be } 0xFFFF. */$		

W

Format	$P1 = \text{W}(P2)$	Data Type	UD/SD
Function	Converts 32-bit number $P2$ to a 16-bit number and saves the result in $P1$. The truncation error may occur.		
$P1$ (I/E)	The location to save the result.		
$P2$ (I/E/C)	The 32-bit number to be converted.		
Example 1	$\$U100 = \text{W}(0x12345678) (UD) /* \text{The value of } \$U100 \text{ will be } 0x5678 */$		
Example 2	$\$U200 = \text{W}(-12345) (SD) /* \text{The value of } \$U200 \text{ will be } -12345 */$		

B2W

Format	$P1 = B2W(P2, P3)$	Data Type	U
Function	Converts P3 -byte array starting from P2 to a P3 -word array and saves the result in P1 . All the high bytes of the word array are set to 0.		
P1 (I)	The location (or the word array) to save the result.		
P2 (I)	The byte array to be converted.		
P3 (I/C)	The size of the byte array.		
Example 1	$\$U200 = 0x45FA$ $\$U201 = 0xEB29$ $\$U100 = B2W(\$U200, 3)$ /* Convert 3 bytes starting from \$U200 to 3 words starting from \$U100, \$U100 will be 0xFA, \$U101 will be 0x45 and \$U102 will be 0x29. */		

W2B

Format	$P1 = W2B(P2, P3)$	Data Type	U
Function	Converts a word array P2 with P3 elements to a byte array and saves the result in the byte array P1 . The conversion discards the high byte of every element of the word array to form a byte array with the same number of elements. The array size cannot exceed 256.		
P1 (I)	The location (or the word array) to save the result.		
P2 (I)	The word array to be converted.		
P3 (I/C)	The size of the word array.		
Example 1	$\$U200 = 0x45FA$ $\$U201 = 0xEB29$ $\$U202 = 0xC781$ $\$U100 = W2B(\$U200, 3)$ /* Convert 3 words starting from \$U200 to 3 bytes starting from \$U100, \$U100 will be 0x29FA and the low byte of \$U101 will be 0x81 */		

A2X

Format	$P1 = A2X(P2)$	Data Type	U
Function	Converts a 4-digit hex number in ASCII character form to a binary number and saves the result in P1 . The character of the fourth digit is in the first word of the word array P2 and the characters of the other digits are in the following words in sequence.		
P1 (I)	The location to save the result.		
P2 (I)	The word array that contains the characters to be converted.		
Example 1	$\$U20 = 49$ // '1' $\$U21 = 50$ // '2' $\$U22 = 69$ // 'E' $\$U23 = 70$ // 'F' $\$U100 = A2X(\$U20)$ /* The value of \$U100 will be 0x12EF. */		

X2A

Format	$P1 = X2A(P2)$	Data Type	U
Function	Converts 16-bit number P2 to a 4-digit hex number in ASCII character form and saves the result in word array P1 . The character of the fourth digit is saved in the first word of P1 and the characters of the other digits are saved in the following words in sequence.		
P1 (I)	The location (or the word array) to save the result.		
P2 (I/C)	The number to be converted.		
Example 1	\$U10 = X2A(0x34AB) /*The 4 words starting from \$U10 will be: 51('3'), 52('4'), 65('A'), 66('B') */		

W2F

Format	$P1 = W2F(P2)$	Data Type	U/S
Function	Converts 16-bit number P2 to a floating point number and saves the result in P1 .		
P1 (I/E)	The location to save the result.		
P2 (I/E/C)	The 16-bit number to be converted.		
Example 1	\$U200 = W2F(\$U10) (S)		

D2F

Format	$P1 = D2F(P2)$	Data Type	UD/SD
Function	Converts 32-bit number P2 to a floating point number and saves the result in P1 .		
P1 (I/E)	The location to save the result.		
P2 (I/E/C)	The 32-bit number to be converted.		
Example 1	\$U200 = D2F(\$U10) (SD)		

F2W

Format	$P1 = F2W(P2)$	Data Type	F
Function	Converts floating point number P2 to a 16-bit number and saves the result in P1 .		
P1 (I/E)	The location to save the result.		
P2 (I/E/C)	The floating point number to be converted.		
Example 1	\$U200 = F2W(\$U10) (F)		

F2D

Format	$P1 = F2D(P2)$	Data Type	F
Function	Converts floating point number P2 to a 32-bit number and saves the result in P1 .		
P1 (I/E)	The location to save the result.		
P2 (I/E/C)	The floating point number to be converted.		
Example 1	\$U200 = F2D(\$U10) (F)		

EXTRACT_BIT

Format	$P1 = \text{EXTRACT_BIT}(P2, P3)$	Data Type	U/UD
Function	Extracts bit $P3$ from $P2$ and saves the result in $P1$.		
$P1$ (I)	The bit to save the result.		
$P2$ (I)	The location to extract the bit.		
$P3$ (I/C)	The number of the bit to be extracted.		
Example 1	$\$U2.0 = \text{EXTRACT_BIT}(\$U10, 31)$ (UD) /* Extract bit 31 of the double word \$U10 and save the result in \$U2.0 */		

14.4.7. Conditional Operation

IF ==

Format	$\text{IF } P2 == P3$	Data Type	U/S/UD/SD/F
Function	Executes the commands in the command block following this IF command when $P2$ is equal to $P3$.		
$P2, P3$ (I/E/C/AE)	The operands.		

IF !=

Format	$\text{IF } P2 != P3$	Data Type	U/S/UD/SD/F
Function	Executes the commands in the command block following this IF command when $P2$ is not equal to $P3$.		
$P2, P3$ (I/E/C/AE)	The operands.		

IF >

Format	$\text{IF } P2 > P3$	Data Type	U/S/UD/SD/F
Function	Executes the commands in the command block following this IF command when $P2$ is greater than $P3$.		
$P2, P3$ (I/E/C/AE)	The operands.		

IF >=

Format	$\text{IF } P2 >= P3$	Data Type	U/S/UD/SD/F
Function	Executes the commands in the command block following this IF command when $P2$ is greater than or equal to $P3$.		
$P2, P3$ (I/E/C/AE)	The operands.		

IF <

Format	$\text{IF } P2 < P3$	Data Type	U/S/UD/SD/F
Function	Executes the commands in the command block following this IF command when $P2$ is less than $P3$.		
$P2, P3$ (I/E/C/AE)	The operands.		

IF <=

Format	IF <i>P2</i> <= <i>P3</i>	Data Type	U/S/UD/SD/F
Function	Executes the commands in the command block following this IF command when <i>P2</i> is less than or equal to <i>P3</i> .		
<i>P2,P3</i> (I/E/C/AE)	The operands.		

IF &

Format	IF <i>P2</i> & <i>P3</i>	Data Type	U/UD
Function	Executes the commands in the command block following this IF command when the result of Bitwise AND between <i>P2</i> and <i>P3</i> is non-zero.		
<i>P2,P3</i> (I/E/C/AE)	The operands.		

IF !&

Format	IF !(<i>P2</i> & <i>P3</i>)	Data Type	U/UD
Function	Executes the commands in the command block following this IF command when the result of Bitwise AND between <i>P2</i> and <i>P3</i> is zero.		
<i>P2,P3</i> (I/E/C/AE)	The operands.		

IF <bit>

Format	IF <i>P2</i>	Data Type	B
Function	Executes the commands in the command block following this IF command if the condition <i>P2</i> is true (1/On).		
<i>P2</i> (I/E/CE)	The condition.		

IF !<bit>

Format	IF ! <i>P2</i>	Data Type	B
Function	Executes the commands in the command block following this IF command if the condition <i>P2</i> is false (0/Off).		
<i>P2</i> (I/E/CE)	The condition.		

ELIF ==

Format	ELIF <i>P2</i> == <i>P3</i>	Data Type	U/S/UD/SD/F
Function	Executes the commands in the command block following this ELIF command when <i>P2</i> is equal to <i>P3</i> .		
<i>P2,P3</i> (I/E/C/AE)	The operands.		

ELIF !=

Format	ELIF <i>P2</i> != <i>P3</i>	Data Type	U/S/UD/SD/F
Function	Executes the commands in the command block following this ELIF command when <i>P2</i> is not equal to <i>P3</i> .		
<i>P2,P3</i> (I/E/C/AE)	The operands.		

ELIF >

Format	ELIF <i>P2</i> > <i>P3</i>	Data Type	U/S/UD/SD/F
Function	Executes the commands in the command block following this ELIF command when <i>P2</i> is greater than <i>P3</i> .		
<i>P2,P3</i> (I/E/C/AE)	The operands.		

ELIF >=

Format	ELIF <i>P2</i> >= <i>P3</i>	Data Type	U/S/UD/SD/F
Function	Executes the commands in the command block following this ELIF command when <i>P2</i> is greater than or equal to <i>P3</i> .		
<i>P2,P3</i> (I/E/C/AE)	The operands.		

ELIF <

Format	ELIF <i>P2</i> < <i>P3</i>	Data Type	U/S/UD/SD/F
Function	Executes the commands in the command block following this ELIF command when <i>P2</i> is less than <i>P3</i> .		
<i>P2,P3</i> (I/E/C/AE)	The operands.		

ELIF <=

Format	ELIF <i>P2</i> <= <i>P3</i>	Data Type	U/S/UD/SD/F
Function	Executes the commands in the command block following this ELIF command when <i>P2</i> is less than or equal to <i>P3</i> .		
<i>P2,P3</i> (I/E/C/AE)	The operands.		

ELIF &

Format	ELIF <i>P2</i> & <i>P3</i>	Data Type	U/UD
Function	Executes the commands in the command block following this ELIF command when the result of Bitwise AND between <i>P2</i> and <i>P3</i> is non-zero.		
<i>P2,P3</i> (I/E/C/AE)	The operands.		

ELIF !&

Format	ELIF !(<i>P2</i> & <i>P3</i>)	Data Type	U/UD
Function	Executes the commands in the command block following this ELIF command when the result of Bitwise AND between <i>P2</i> and <i>P3</i> is zero.		
<i>P2,P3</i> (I/E/C/AE)	The operands.		

ELIF <bit>

Format	ELIF <i>P2</i>	Data Type	B
Function	Executes the commands in the command block following this ELIF command if the condition <i>P2</i> is true (1/On).		
<i>P2</i> (I/E/CE)	The condition.		

ELIF !<bit>

Format	ELIF ! <i>P2</i>	Data Type	B
Function	Executes the commands in the command block following this ELIF command if the condition <i>P2</i> is false (0/Off).		
<i>P2</i> (I/E/CE)	The condition.		

ELSE

Format	ELSE
Function	This command specifies the beginning of the default command block that will be executed if none of the conditions in the preceding IF and/or ELIF commands is true. This is not an executable command.

ENDIF

Format	ENDIF										
Function	This command specifies the end of a command block, which begins at the command following the matching IF, ELIF, or ELSE command. This is not an executable command.										
Example	<p>IF-Command Structures:</p> <table> <tr> <th>Commands and Structures</th><th>Description</th></tr> <tr> <td>IF <condition> ... ENDIF</td><td>Runs the command block between IF and ENDIF when the condition is true, otherwise ignores the command block.</td></tr> <tr> <td>IF <condition> ... ELSE ... ENDIF</td><td>Runs the command block between IF and ELSE when the condition is true, otherwise runs the command block between ELSE and ENDIF.</td></tr> <tr> <td>IF <condition> ... ELIF <condition_2> ... ELIF <condition_3> . . . ELIF <condition_N> ... ENDIF</td><td>Runs the command block between IF and the first ELIF and ignores all the following commands in the structure when condition 1 is true, otherwise examines condition 2. Runs the command block between the first ELIF and the second ELIF and ignores all the following commands in the structure when condition 2 is true, otherwise checks condition 3. Repeats the same operation until condition N is processed. If none of the conditions are true, no command block in this structure is run.</td></tr> <tr> <td>IF <condition> ... ELIF <condition_2> ... ELIF <condition_3> . . . ELIF <condition_N> ... ELSE ... ENDIF</td><td>Runs the command block between IF and the first ELIF and ignores all the following commands in the structure when condition 1 is true, otherwise examines condition 2. Runs the command block between the first ELIF and the second ELIF and ignores all the following commands in the structure when condition 2 is true, otherwise checks condition 3. Repeats the same operation until condition N is processed. Runs the command block between ELSE and ENDIF if none of the conditions are true.</td></tr> </table> <p>Note that there can be up to 20 nested IF-command structures.</p>	Commands and Structures	Description	IF <condition> ... ENDIF	Runs the command block between IF and ENDIF when the condition is true, otherwise ignores the command block.	IF <condition> ... ELSE ... ENDIF	Runs the command block between IF and ELSE when the condition is true, otherwise runs the command block between ELSE and ENDIF.	IF <condition> ... ELIF <condition_2> ... ELIF <condition_3> . . . ELIF <condition_N> ... ENDIF	Runs the command block between IF and the first ELIF and ignores all the following commands in the structure when condition 1 is true, otherwise examines condition 2. Runs the command block between the first ELIF and the second ELIF and ignores all the following commands in the structure when condition 2 is true, otherwise checks condition 3. Repeats the same operation until condition N is processed. If none of the conditions are true, no command block in this structure is run.	IF <condition> ... ELIF <condition_2> ... ELIF <condition_3> . . . ELIF <condition_N> ... ELSE ... ENDIF	Runs the command block between IF and the first ELIF and ignores all the following commands in the structure when condition 1 is true, otherwise examines condition 2. Runs the command block between the first ELIF and the second ELIF and ignores all the following commands in the structure when condition 2 is true, otherwise checks condition 3. Repeats the same operation until condition N is processed. Runs the command block between ELSE and ENDIF if none of the conditions are true.
Commands and Structures	Description										
IF <condition> ... ENDIF	Runs the command block between IF and ENDIF when the condition is true, otherwise ignores the command block.										
IF <condition> ... ELSE ... ENDIF	Runs the command block between IF and ELSE when the condition is true, otherwise runs the command block between ELSE and ENDIF.										
IF <condition> ... ELIF <condition_2> ... ELIF <condition_3> . . . ELIF <condition_N> ... ENDIF	Runs the command block between IF and the first ELIF and ignores all the following commands in the structure when condition 1 is true, otherwise examines condition 2. Runs the command block between the first ELIF and the second ELIF and ignores all the following commands in the structure when condition 2 is true, otherwise checks condition 3. Repeats the same operation until condition N is processed. If none of the conditions are true, no command block in this structure is run.										
IF <condition> ... ELIF <condition_2> ... ELIF <condition_3> . . . ELIF <condition_N> ... ELSE ... ENDIF	Runs the command block between IF and the first ELIF and ignores all the following commands in the structure when condition 1 is true, otherwise examines condition 2. Runs the command block between the first ELIF and the second ELIF and ignores all the following commands in the structure when condition 2 is true, otherwise checks condition 3. Repeats the same operation until condition N is processed. Runs the command block between ELSE and ENDIF if none of the conditions are true.										

14.4.8. Program Control

JMP

Format	JMP <i>P1</i>
Function	Unconditionally jumps to the program point specified by label <i>P1</i> .
<i>P1</i> (CS)	The label of the program point.
Example 1	<pre>IF \$U10 == 0 JMP SKIP /* Skip the command "\$U20 = \$U10 / 2". */ ENDIF \$U20 = \$U10 / 2 SKIP: \$U10 = 1</pre>

<label>

Format	<i>P1</i>:
Function	This is not an executable command. The <i>P1</i> is the label of the program point where it is positioned.
<i>P1</i> (CS)	The character string as the label of the program point. Remember to have the character ':' after the label.
Example 1	<pre>IF \$U10 == 0 JMP SKIP /* Skip the command "\$U20 = \$U10 / 2". */ ENDIF \$U20 = \$U10 / 2 SKIP: \$U10 = 1</pre>

JMP ==

Format	JMP(<i>P1</i>,<i>P2</i> == <i>P3</i>)	Data Type	U/S/UD/SD/F
Function	Jumps to the program point specified by label <i>P1</i> when <i>P2</i> is equal to <i>P3</i> .		
<i>P1</i> (CS)	The label of the program point.		
<i>P2</i>,<i>P3</i> (I/E/C/AE)	The operands.		

JMP !=

Format	JMP(<i>P1</i>,<i>P2</i> != <i>P3</i>)	Data Type	U/S/UD/SD/F
Function	Jumps to the program point specified by label <i>P1</i> when <i>P2</i> is not equal to <i>P3</i> .		
<i>P1</i> (CS)	The label of the program point.		
<i>P2</i>,<i>P3</i> (I/E/C/AE)	The operands.		

JMP >

Format	JMP(<i>P1</i>,<i>P2</i> > <i>P3</i>)	Data Type	U/S/UD/SD/F
Function	Jumps to the program point specified by label <i>P1</i> when <i>P2</i> is greater than <i>P3</i> .		
<i>P1</i> (CS)	The label of the program point.		
<i>P2</i>,<i>P3</i> (I/E/C/AE)	The operands.		

JMP >=

Format	JMP(<i>P1</i>,<i>P2</i> >= <i>P3</i>)	Data Type	U/S/UD/SD/F
Function	Jumps to the program point specified by label <i>P1</i> when <i>P2</i> is greater than or equal to <i>P3</i> .		
<i>P1</i> (CS)	The label of the program point.		
<i>P2,P3</i> (I/E/C/AE)	The operands.		

JMP <

Format	JMP(<i>P1</i>,<i>P2</i> < <i>P3</i>)	Data Type	U/S/UD/SD/F
Function	Jumps to the program point specified by label <i>P1</i> when <i>P2</i> is less than <i>P3</i> .		
<i>P1</i> (CS)	The label of the program point.		
<i>P2,P3</i> (I/E/C/AE)	The operands.		

JMP <=

Format	JMP(<i>P1</i>,<i>P2</i> <= <i>P3</i>)	Data Type	U/S/UD/SD/F
Function	Jumps to the program point specified by label <i>P1</i> when <i>P2</i> is less than or equal to <i>P3</i> .		
<i>P1</i> (CS)	The label of the program point.		
<i>P2,P3</i> (I/E/C/AE)	The operands.		

JMP &

Format	JMP(<i>P1</i>,<i>P2</i> & <i>P3</i>)	Data Type	U/UD
Function	Jumps to the program point specified by label <i>P1</i> when the result of Bitwise AND between <i>P2</i> and <i>P3</i> is non-zero.		
<i>P1</i> (CS)	The label of the program point.		
<i>P2,P3</i> (I/E/C/AE)	The operands.		

JMP !&

Format	JMP(<i>P1</i>,!(<i>P2</i> & <i>P3</i>))	Data Type	U/UD
Function	Jumps to the program point specified by label <i>P1</i> when the result of Bitwise AND between <i>P2</i> and <i>P3</i> is zero.		
<i>P1</i> (CS)	The label of the program point.		
<i>P2,P3</i> (I/E/C/AE)	The operands.		

JMP <bit>

Format	JMP(<i>P1</i>,<i>P2</i>)	Data Type	B
Function	Jumps to the program point specified by label <i>P1</i> if the condition <i>P2</i> is true (1/On).		
<i>P1</i> (CS)	The label of the program point.		
<i>P2,P3</i> (I/E/CE)	The operands.		

JMP !<bit>

Format	JMP(<i>P1</i> ,! <i>P2</i>)	Data Type	B
Function	Jumps to the program point specified by label <i>P1</i> if the condition <i>P2</i> is false (0/Off).		
<i>P1</i> (CS)	The label of the program point.		
<i>P2</i> , <i>P3</i> (I/E/CE)	The operands.		

CALL

Format	CALL <i>P1</i>
Function	Goes to sub-macro <i>P1</i> .
<i>P1</i> (Sub-macro name)	The sub-macro to be called.
Example 1	CALL CommonFunction_01 /* Go to sub-macro named CommonFuncation_01 */

RET

Format	RET
Function	Returns to the calling macro. This command can only be used in sub-macros.

FOR

Format	FOR <i>P2</i>	Data Type	U
Function	Runs the commands within the FOR loop by <i>P1</i> times. A FOR loop is enclosed by a matching pair of FOR and NEXT commands. There can be up to 20 nested FOR loops.		
<i>P1</i> (I/C)	Total times to run the FOR loop		
Example 1	FOR 10 \$U100 = \$U100 + 1 /* This command will be executed 10 times */ FOR 12 \$U200 = \$U200 + 1 /* This command will be executed 120 times */ NEXT NEXT		

NEXT

Format	NEXT
Function	This command indicates the end of a FOR loop. It is not an executable command.
Example 1	Example: \$U1 = 10 \$U2 = 12 FOR \$U1 \$U100 = \$U100 + 1 /* This command will be executed 10 times. */ FOR \$U2 \$U200 = \$U200 + 1 /* This command will be executed 120 times. */ NEXT NEXT

STOP

Format	STOP
Function	<p>Stops the macro immediately. If the macro is a Cycle macro, it will run again starting from the first command when the associated window is opened again. If the macro is a Main macro, it will run again starting from the first command when restarting the application.</p> <p>This command cannot be used in sub-macros.</p>

END

Format	END
Function	<p>Indicates the end of macro and stops the macro in the current cycle. It can be put anywhere in a macro to stop the macro at any point. If the macro is a cyclic macro, such as the Main macro and the Cycle macro, it is stopped just in the current cycle and will be run again starting from the first command in the next cycle.</p> <p>This command cannot be used in sub-macros.</p>

14.4.9. Timer Operation**SET_T**

Format	SET_T(P1,P2)	Data Type	U															
Function	Starts the timer P1 using the timer control block in P2 .																	
P1 (C)	The ID of the timer. There are 8 timers available and the IDs are 0 to 7.																	
P2 (l)	<div>The starting location of the memory block (or word array) that is used as a Timer Control Block for the timer. The structure of the Timer Control Block is shown below:</div> <table><tr><th>Word No.</th><th>Data Item</th><th>Description</th></tr><tr><td>0</td><td>Type of operation</td><td>0: One-shot; 1: Square-wave</td></tr><tr><td>1</td><td>Current timer value</td><td>The timer increases the value of this word by 1 every 100ms.</td></tr><tr><td>2</td><td>Timer limit</td><td>When the current timer value reaches the timer limit, the timer will perform one of the following operations according to the type of operation: 1) If the type of operation is One-shot (0), sets the time-up flag to 1, resets the current timer value to 0, and stops itself. 2) If the type of operation is Square-wave (1), toggles the time-up flag, resets the current timer value to 0, and continues the timing operation.</td></tr><tr><td>3</td><td>Time-up flag</td><td>This word will be set to 0 or 1 when the current timer value is equal to the timer limit.</td></tr></table> <div>The timer will use the associated Timer Control Block as its private memory, so do not use any words in the block for other purposes. A Timer Control Block requires 4 words.</div>			Word No.	Data Item	Description	0	Type of operation	0: One-shot; 1: Square-wave	1	Current timer value	The timer increases the value of this word by 1 every 100ms.	2	Timer limit	When the current timer value reaches the timer limit, the timer will perform one of the following operations according to the type of operation: 1) If the type of operation is One-shot (0), sets the time-up flag to 1, resets the current timer value to 0, and stops itself. 2) If the type of operation is Square-wave (1), toggles the time-up flag, resets the current timer value to 0, and continues the timing operation.	3	Time-up flag	This word will be set to 0 or 1 when the current timer value is equal to the timer limit.
Word No.	Data Item	Description																
0	Type of operation	0: One-shot; 1: Square-wave																
1	Current timer value	The timer increases the value of this word by 1 every 100ms.																
2	Timer limit	When the current timer value reaches the timer limit, the timer will perform one of the following operations according to the type of operation: 1) If the type of operation is One-shot (0), sets the time-up flag to 1, resets the current timer value to 0, and stops itself. 2) If the type of operation is Square-wave (1), toggles the time-up flag, resets the current timer value to 0, and continues the timing operation.																
3	Time-up flag	This word will be set to 0 or 1 when the current timer value is equal to the timer limit.																
Example 1	<pre>\$U100 = 1 /* Type of operation is Square-wave. */ \$U101 = 0 /* Initialize the current timer value to 0. */ \$U102 = 5 /* Timer limit is 0.5 second (5*100ms). */ \$U103 = 0 /* Initialize the time-up flag to 0. */ SET_T(3, \$U100) /* Use timer #3 to generate a 1 Hz square wave on \$U103.0 */</pre>																	

STOP_T

Format	STOP_T(<i>P1</i>)	Data Type	U
Function	Stops the timer <i>P1</i> .		
<i>P1</i> (C)	The ID of the timer.		
Example 1	STOP_T(1) /* Stop timer #1 */		

WAIT_T

Format	WAIT_T(<i>P1</i>)	Data Type	U
Function	Waits for the time-up of timer <i>P1</i> . The macro command following this one will not be executed until the timer reaches its limit.		
<i>P1</i> (C)	The ID of the timer.		
Example 1	<pre> \$U100 = 0 /* Type of operation is One-shot. */ \$U101 = 0 /* Initialize the current timer value to 0. */ \$U102 = 5 /* Timer limit is 0.5 second (5*100ms). */ \$U103 = 0 /* Initialize the time-up flag to 0. */ SET_T(7, \$U100) /* Starts timer #7 as a 0.5 second timer. */ WAIT_T(7) /* Wait 0.5 second */ </pre>		

14.4.10. Keypad Operation

KB_MCR

Format	KB_MCR(<i>P1</i>)	Data Type	U
Function	Accepts or ignores the character/command currently input by the associated keypad button. This command must be used only in a macro that is run by a keypad button. A keypad button runs the specified macro when it is pressed. You can use this command in a keypad button macro to accept or ignore the current input of that button.		
<i>P1</i> (I/C)	The value or the location that holds the value to determine the acceptance of the keypad button input. If the value is 0, the input will be accepted; Otherwise the input will be ignored.		
Example 1	KB_MCR(1) /* Ignore the current input */		

KPD_TEXT

Format	KPD_TEXT(<i>P1</i>)	Data Type	U
Function	The memory block (or byte array) that contains the null-terminated ASCII character string to be used to initialize the keypad display and buffer.		
<i>P1</i> (I)	The memory block (or byte array) that contains the null-terminated ASCII character string to be used to initialize the keypad display and buffer.		
Example 1	<pre> \$U100 = "initial text" KPD_TEXT(\$U100) /* Initialize the keypad display and buffer using the string "initial text". */ </pre>		

14.4.11. Recipe Operation

RB2ROM

Format	<i>P1</i> = RB2ROM(<i>P2</i>)	Data Type	U
Function	Saves the data of recipe block <i>P2</i> to the flash ROM and saves the completion code in <i>P1</i> .		
<i>P1</i> (I)	The word to receive the completion code. If the completion code is 0, the operation succeeded; otherwise the operation failed.		
<i>P2</i> (I/C)	The ID of the recipe block to be saved. The option "Need space in flash ROM to save backup" must be selected for the recipe block.		
Example 1	\$U10 = RB2ROM(3) /* Save recipe block #3 to the flash ROM. */		

ROM2RB

Format	<i>P1</i> = ROM2RB(<i>P2</i>)	Data Type	U
Function	Restores the data of recipe block <i>P2</i> from the flash ROM and saves the completion code in <i>P1</i> .		
<i>P1</i> (I)	The word to receive the completion code. If the completion code is 0, the operation succeeded; otherwise the operation failed.		
<i>P2</i> (I/C)	The ID of the recipe block to be restored. The option "Need space in flash ROM to save backup" must be selected for the recipe block.		
Example 1	\$U10 = ROM2RB(3) /* Restore recipe block #3 from the flash ROM. */		

REF_RCP_OBJ

Format	REF_RCP_OBJ(<i>P1</i>)	Data Type	U
Function	Refreshes the recipe objects associated with the specified recipe block <i>P1</i> . The recipe objects include recipe selectors and recipe tables. You can use this command to update the display of associated objects after changing the data of a recipe block in a macro program.		
<i>P1</i> (I/C)	The ID of the associated recipe block.		
Example 1	REF_RCP_OBJ(3) /* Refresh the recipe objects associated with recipe block #3 */		

14.4.12. Communication Operation

EN_LINK

Format	EN_LINK(<i>P1</i> , <i>P2</i> , <i>P3</i>)	Data Type	U
Function	Enables communication link <i>P1</i> or sub-link <i>P2</i> of communication link <i>P1</i> when <i>P3</i> is 1. Disables the specified communication link or sub-link when <i>P3</i> is 0.		
<i>P1</i> (I/C)	The number of the communication link to be enabled or disabled.		
<i>P2</i> (I/C)	The node address of the sub-link to be enabled or disabled. If the specified communication link has no sub-link, this parameter is ignored. If the specified communication link has sub-links and you want to enable or disable the link itself, set this parameter to 0.		
<i>P3</i> (I/C)	To enable the specified communication link or sub-link, set this parameter to 1. To disable the specified communication link or sub-link, set this parameter to 0.		
Example 1	ENABLE_LINK(1, 20, 0) /* Disable the sub-link, whose node address is 20, of communication link 1. */		

LINK_STS

Format	<i>P1</i> = LINK_STS(<i>P2</i> , <i>P3</i>)	Data Type	U																																																												
Function	Gets the status of communication link <i>P2</i> or sub-link <i>P3</i> of communication link <i>P2</i> and saves the result in <i>P1</i> .																																																														
<i>P1</i> (I/C)	<p>The word to receive the status of the specified communication link or sub-link. The status is a 16-bit value. The following table lists the meaning of each status value.</p> <table border="1"> <thead> <tr> <th>Status Value</th><th>Meaning</th><th>Status Value</th><th>Meaning</th></tr> </thead> <tbody> <tr><td>0</td><td>OK</td><td>14</td><td>Device busy</td></tr> <tr><td>1</td><td>Overflow error</td><td>15</td><td>Unknown error</td></tr> <tr><td>2</td><td>Break error</td><td>16</td><td>Link disabled</td></tr> <tr><td>3</td><td>Parity error</td><td>17</td><td>Initialization failure</td></tr> <tr><td>4</td><td>Framing error</td><td>18</td><td>Failed to send data</td></tr> <tr><td>5</td><td>No response</td><td>19</td><td>Failed to receive data</td></tr> <tr><td>6</td><td>Unrecognized response</td><td>20</td><td>Failed to open connection</td></tr> <tr><td>7</td><td>Timeout</td><td>21</td><td>Connection not ready</td></tr> <tr><td>8</td><td>Inactive CTS</td><td>22</td><td>Invalid sub-link</td></tr> <tr><td>9</td><td>Checksum error</td><td>23</td><td>Invalid COM port</td></tr> <tr><td>10</td><td>Command rejected</td><td>24</td><td>Error</td></tr> <tr><td>11</td><td>Invalid address</td><td>255</td><td>Condition uncertain</td></tr> <tr><td>12</td><td>Invalid range</td><td>65535</td><td>Failed to get status</td></tr> <tr><td>13</td><td>Invalid request</td><td></td><td></td></tr> </tbody> </table>			Status Value	Meaning	Status Value	Meaning	0	OK	14	Device busy	1	Overflow error	15	Unknown error	2	Break error	16	Link disabled	3	Parity error	17	Initialization failure	4	Framing error	18	Failed to send data	5	No response	19	Failed to receive data	6	Unrecognized response	20	Failed to open connection	7	Timeout	21	Connection not ready	8	Inactive CTS	22	Invalid sub-link	9	Checksum error	23	Invalid COM port	10	Command rejected	24	Error	11	Invalid address	255	Condition uncertain	12	Invalid range	65535	Failed to get status	13	Invalid request		
Status Value	Meaning	Status Value	Meaning																																																												
0	OK	14	Device busy																																																												
1	Overflow error	15	Unknown error																																																												
2	Break error	16	Link disabled																																																												
3	Parity error	17	Initialization failure																																																												
4	Framing error	18	Failed to send data																																																												
5	No response	19	Failed to receive data																																																												
6	Unrecognized response	20	Failed to open connection																																																												
7	Timeout	21	Connection not ready																																																												
8	Inactive CTS	22	Invalid sub-link																																																												
9	Checksum error	23	Invalid COM port																																																												
10	Command rejected	24	Error																																																												
11	Invalid address	255	Condition uncertain																																																												
12	Invalid range	65535	Failed to get status																																																												
13	Invalid request																																																														
<i>P2</i> (I/C)	The number of the communication link.																																																														
<i>P3</i> (I/C)	The node address of the sub-link. If the specified communication link has no sub-link, this parameter is ignored.																																																														
Example 1	\$U100 = LINK_STS(2, 0) /* Get the status of communication link 2 and save it to \$U100. */																																																														
Example 2	\$U12 = LINK_STS(1, 128) /* Get the status of the sub-link, whose node address is 128, of communication link 1 and save it to \$U12. */																																																														

14.4.13. System Service

GET_RTC

Format	GET_RTC(<i>P1</i>)	Data Type	U																											
Function	Gets the data of the real time clock and saves the result in <i>P1</i> .																													
<i>P1</i> (I)	<div>The starting location of the memory block that is used as an RTC data block to receive the operation result. The structure of the RTC data block is shown below:</div> <table><thead><tr><th>Data Item</th><th>Data Type/Size</th><th>Word No.</th></tr></thead><tbody><tr><td>Second</td><td>16-bit Unsigned Integer</td><td>0</td></tr><tr><td>Minute</td><td>16-bit Unsigned Integer</td><td>1</td></tr><tr><td>Hour</td><td>16-bit Unsigned Integer</td><td>2</td></tr><tr><td>RTC adjustment</td><td>16-bit Signed Integer</td><td>3</td></tr><tr><td>Day</td><td>16-bit Unsigned Integer</td><td>4</td></tr><tr><td>Month</td><td>16-bit Unsigned Integer</td><td>5</td></tr><tr><td>Year</td><td>16-bit Unsigned Integer</td><td>6</td></tr><tr><td>Day of week</td><td>16-bit Unsigned Integer</td><td>7</td></tr></tbody></table> <div>Second: 0-59; Minute: 0-59; Hour: 0-23; RTC adjustment: -63-63; Day: 1-31; Month: 1-12; Year: 0(2000)-99(2099); Day of week: 0(Sunday)-6(Saturday) An RTC data block requires 8 words.</div>			Data Item	Data Type/Size	Word No.	Second	16-bit Unsigned Integer	0	Minute	16-bit Unsigned Integer	1	Hour	16-bit Unsigned Integer	2	RTC adjustment	16-bit Signed Integer	3	Day	16-bit Unsigned Integer	4	Month	16-bit Unsigned Integer	5	Year	16-bit Unsigned Integer	6	Day of week	16-bit Unsigned Integer	7
Data Item	Data Type/Size	Word No.																												
Second	16-bit Unsigned Integer	0																												
Minute	16-bit Unsigned Integer	1																												
Hour	16-bit Unsigned Integer	2																												
RTC adjustment	16-bit Signed Integer	3																												
Day	16-bit Unsigned Integer	4																												
Month	16-bit Unsigned Integer	5																												
Year	16-bit Unsigned Integer	6																												
Day of week	16-bit Unsigned Integer	7																												
Example 1	GET_RTC(\$U100) /* Get the data of the real time clock. The second will be in \$U100 and the day-of-week will be in \$U107. */																													

SET_RTC

Format	SET_RTC(<i>P1</i>)	Data Type	U
Function	Sets the real time clock using the data in <i>P1</i> .		
<i>P1</i> (I)	The starting location of the memory block that is used as an RTC data block to contain the new settings for the real time clock. See the description of GET_RTC to know the structure of the RTC data block.		
Example 1	<pre> \$U100 = 0 // Second \$U101 = 30 // Minute \$U102 = 8 // Hour \$U103 = 0 // Adjustment \$U104 = 1 // Day \$U105 = 7 // July \$U106 = 10 // Year 2010 \$U107 = 4 // Thursday SET_RTC(\$U100) /* Set the real time clock to 8:30:00 July 1st 2010 Thursday */ </pre>		

SYS

Format	SYS(<i>P1,P2,P3</i>)	Data Type	U
Function	Requests system service <i>P1</i> with the arguments <i>P2</i> and <i>P3</i> . This command is reserved for system use.		
<i>P1</i> (I)	The code of the system service.		
<i>P2,P3</i> (I/C)	The arguments of the system service.		

14.4.14. Screen Operation

OPEN_WS

Format	OPEN_WS <i>P1</i>	Data Type	U
Function	The number of the window screen to be opened. This command will not open the specified screen if it is a normal screen or menu screen. The macro commands following this command will not be executed until the opened window screen is closed. Also, when a screen's Cycle macro is waiting for the closing of the window screen opened by this command, that screen cannot be closed or switched by any means.		
<i>P1</i> (I/C)	The number of the window screen to be opened. If the screen number indicates the normal screen or menu screen, no screen will be opened.		

CLOSE_WS

Format	CLOSE_WS
Function	Closes the window screen that was opened by the macro command OPEN_WS.

14.4.15. File Operation

FILE_IO

Format	P1 = FILE_IO(P2,P3)		Data Type	U																																																													
Function	Performs the file operation specified by P2 and P3 using default filename and saves the completion code in P1.																																																																
P1 (I)	The word to receive the completion code of the operation. If the completion code is 0, the operation succeeded; otherwise the operation failed.																																																																
P2,P3 (I/C)	P2 specifies the type of file operation. P3 specifies the ID of the data source. The following table describes how to set P2 and P3.																																																																
	<table><thead><tr><th>File Operation</th><th>P2</th><th>P3</th><th>Default Filename Format</th></tr></thead><tbody><tr><td>Save Logged Data (.txt)</td><td>1</td><td rowspan="2">Data logger ID (0-15)</td><td>DL<ID>_<Date>_<Time>.txt</td></tr><tr><td>Save Logged Data (.csv)</td><td>14</td><td>DL<ID>_<Date>_<Time>.csv</td></tr><tr><td>Save Logged Alarms (.txt)</td><td>2</td><td rowspan="2">0</td><td>AL_<Date>_<Time>.txt</td></tr><tr><td>Save Logged Alarms (.csv)</td><td>15</td><td>AL_<Date>_<Time>.csv</td></tr><tr><td>Save Alarm Counts (.txt)</td><td>3</td><td rowspan="2">0</td><td>AC_<Date>_<Time>.txt</td></tr><tr><td>Save Alarm Counts (.csv)</td><td>16</td><td>AC_<Date>_<Time>.csv</td></tr><tr><td>Save Recipe Data (.txt)</td><td>4</td><td rowspan="3">Recipe block ID (0-15)</td><td>RB<ID>.txt</td></tr><tr><td>Save Recipe Data (.csv)</td><td>17</td><td>RB<ID>.csv</td></tr><tr><td>Save Recipe Data (.prd)</td><td>5</td><td>RB<ID>.prd</td></tr><tr><td>Print Screen to File (256-color .bmp)</td><td>6</td><td rowspan="2">Screen number (1-7999)</td><td>S<ID>_<Date>_<Time>.bmp</td></tr><tr><td>Print Screen to File (64K-color .bmp)</td><td>7</td><td>S<ID>_<Date>_<Time>.bmp</td></tr><tr><td>Save Logged Operations (.txt)</td><td>9</td><td>0</td><td>OL_<Date>_<Time>.txt</td></tr><tr><td>Save Logged Operations (.csv)</td><td>18</td><td>0</td><td>OL_<Date>_<Time>.csv</td></tr><tr><td>Save Logged Data (.ldf)</td><td>10</td><td>Data logger ID (0-15)</td><td>DL<ID>_<Date>_<Time>.ldf</td></tr><tr><td>Take Picture (.bmp)</td><td>12</td><td rowspan="2">USB camera ID (0-3)</td><td>CAM<ID>_<Date>_<Time>.bmp</td></tr><tr><td>Take Picture (.jpg)</td><td>13</td><td>CAM<ID>_<Date>_<Time>.jpg</td></tr></tbody></table>	File Operation	P2	P3	Default Filename Format	Save Logged Data (.txt)	1	Data logger ID (0-15)	DL<ID>_<Date>_<Time>.txt	Save Logged Data (.csv)	14	DL<ID>_<Date>_<Time>.csv	Save Logged Alarms (.txt)	2	0	AL_<Date>_<Time>.txt	Save Logged Alarms (.csv)	15	AL_<Date>_<Time>.csv	Save Alarm Counts (.txt)	3	0	AC_<Date>_<Time>.txt	Save Alarm Counts (.csv)	16	AC_<Date>_<Time>.csv	Save Recipe Data (.txt)	4	Recipe block ID (0-15)	RB<ID>.txt	Save Recipe Data (.csv)	17	RB<ID>.csv	Save Recipe Data (.prd)	5	RB<ID>.prd	Print Screen to File (256-color .bmp)	6	Screen number (1-7999)	S<ID>_<Date>_<Time>.bmp	Print Screen to File (64K-color .bmp)	7	S<ID>_<Date>_<Time>.bmp	Save Logged Operations (.txt)	9	0	OL_<Date>_<Time>.txt	Save Logged Operations (.csv)	18	0	OL_<Date>_<Time>.csv	Save Logged Data (.ldf)	10	Data logger ID (0-15)	DL<ID>_<Date>_<Time>.ldf	Take Picture (.bmp)	12	USB camera ID (0-3)	CAM<ID>_<Date>_<Time>.bmp	Take Picture (.jpg)	13	CAM<ID>_<Date>_<Time>.jpg			
File Operation	P2	P3	Default Filename Format																																																														
Save Logged Data (.txt)	1	Data logger ID (0-15)	DL<ID>_<Date>_<Time>.txt																																																														
Save Logged Data (.csv)	14		DL<ID>_<Date>_<Time>.csv																																																														
Save Logged Alarms (.txt)	2	0	AL_<Date>_<Time>.txt																																																														
Save Logged Alarms (.csv)	15		AL_<Date>_<Time>.csv																																																														
Save Alarm Counts (.txt)	3	0	AC_<Date>_<Time>.txt																																																														
Save Alarm Counts (.csv)	16		AC_<Date>_<Time>.csv																																																														
Save Recipe Data (.txt)	4	Recipe block ID (0-15)	RB<ID>.txt																																																														
Save Recipe Data (.csv)	17		RB<ID>.csv																																																														
Save Recipe Data (.prd)	5		RB<ID>.prd																																																														
Print Screen to File (256-color .bmp)	6	Screen number (1-7999)	S<ID>_<Date>_<Time>.bmp																																																														
Print Screen to File (64K-color .bmp)	7		S<ID>_<Date>_<Time>.bmp																																																														
Save Logged Operations (.txt)	9	0	OL_<Date>_<Time>.txt																																																														
Save Logged Operations (.csv)	18	0	OL_<Date>_<Time>.csv																																																														
Save Logged Data (.ldf)	10	Data logger ID (0-15)	DL<ID>_<Date>_<Time>.ldf																																																														
Take Picture (.bmp)	12	USB camera ID (0-3)	CAM<ID>_<Date>_<Time>.bmp																																																														
Take Picture (.jpg)	13		CAM<ID>_<Date>_<Time>.jpg																																																														
<p>Note:</p> <p><ID>: ID of the data logger, ID of the recipe block, ID of the USB camera, or number of the screen <Date>: The date when saving the data. <Time>: The time when saving the data. You can select the formats of <Date> and <Time> on the Custom page in the General Setup dialog box.</p>																																																																	

FILE_IO_N

Format	<i>P1</i> = FILE_IO_N(<i>P2</i> , <i>P3</i> , <i>P4</i>)		Data Type	U
Function	Performs the file operation specified by <i>P2</i> and <i>P3</i> using filename <i>P4</i> and saves the completion code in <i>P1</i> .			
<i>P1</i> (I)	The word to receive the completion code of the operation. If the completion code is 0, the operation succeeded; otherwise the operation failed.			
<i>P2</i> , <i>P3</i> (I/C)	<i>P2</i> specifies the type of file operation. <i>P3</i> specifies the ID of the data source. The following table describes how to set <i>P2</i> and <i>P3</i> .			
	File Operation	<i>P2</i>	<i>P3</i>	
	Save Logged Data (.csv/.txt)	31	Data logger ID (0-15)	
	Save Logged Alarms (.txt)	32	0	
	Save Alarm Counts (.txt)	33	0	
	Save Recipe Data (.csv/.txt)	34	Recipe block ID (0-15)	
	Save Recipe Data (.prd)	35	Recipe block ID (0-15)	
	Print Screen to File (256-color .bmp)	36	Screen number (1-7999)	
	Print Screen to File (64K-color .bmp)	37	Screen number (1-7999)	
	Save Logged Operations (.txt)	39	0	
	Save Logged Data (.ldf)	40	Data logger ID (0-15)	
	Take Picture (.bmp)	42	USB camera ID (0-3)	
	Take Picture (.jpg)	43	USB camera ID (0-3)	
<i>P4</i> (I)	The byte array that contains the specified filename or full pathname. The name must be a valid Windows pathname with ASCII characters only. The character string must be null terminated and each character occupies one byte. The maximum length of the string is 127. All the folders stated in the full pathname must already exist or the file operation will fail.			

MKDIR

Format	<i>P1</i> = MKDIR(<i>P2</i>)
Function	Creates a new directory with the specified name <i>P2</i> and saves the result to <i>P1</i> .
<i>P1</i> (I)	The word to receive the completion code of the operation. If the completion code is 0, the operation succeeded; otherwise the operation failed.
<i>P2</i> (I)	The byte array that contains the name of the new directory. The name must be a valid directory name with or without pathname and has only ASCII characters in it.

OPEN_FILE

Format	$P1 = \text{OPEN_FILE}(P2,P3)$	Data Type	U												
Function	Creates or opens a file.														
$P1$ (I)	<p>The starting location of the memory block that is used as a File Information Block to receive the operation result. The structure of the File Information Block is shown below:</p> <table><tr><th>Data Item</th><th>Data Type/Size</th><th>Word No.</th></tr><tr><td>File handle</td><td>32-bit Unsigned Integer</td><td>0 and 1</td></tr><tr><td>File size</td><td>32 bit Unsigned Integer</td><td>2 and 3</td></tr><tr><td>Filename</td><td>Byte array with 81 elements</td><td>4 through 44</td></tr></table> <p>The file handle is zero if the operation failed. The file size is zero for a newly created file. The filename is a null-terminated character string. The maximum allowable size is 80. It is set when the file is successfully opened. A File Information Block requires 45 words.</p>			Data Item	Data Type/Size	Word No.	File handle	32-bit Unsigned Integer	0 and 1	File size	32 bit Unsigned Integer	2 and 3	Filename	Byte array with 81 elements	4 through 44
Data Item	Data Type/Size	Word No.													
File handle	32-bit Unsigned Integer	0 and 1													
File size	32 bit Unsigned Integer	2 and 3													
Filename	Byte array with 81 elements	4 through 44													
$P2$ (I)	The byte array that contains the filename or the full pathname of the file to be opened. The name is a null-terminated string and has only ASCII characters in it.														
$P3$ (I/C)	<p>Specifies the purpose of opening the file.</p> <table><tr><th>Purpose</th><th>Value</th></tr><tr><td>Read</td><td>0</td></tr><tr><td>Write</td><td>1</td></tr><tr><td>Append</td><td>3</td></tr><tr><td>Read CSV File</td><td>5</td></tr></table>			Purpose	Value	Read	0	Write	1	Append	3	Read CSV File	5		
Purpose	Value														
Read	0														
Write	1														
Append	3														
Read CSV File	5														
Example 1	$\$U10 = \text{"test.txt"}$ $\$U100 = \text{OPEN_FILE}(\$U10, 0)$ /* Open the file "test.txt" for the read operation. The double word \$U100 will contain the file handle. The double word \$U102 will contain the file size. The byte array \$U104 will contain the filename. */														

READ_FILE

Format	$P1 = \text{READ_FILE}(P2, P3, P4)$	Data Type	U
Function	Reads $P4$ bytes from file $P2$ to buffer $P3$ and saves the result in $P1$.		
$P1$ (I)	The word to receive the number of bytes that were actually read. If the operation failed, the number is 65535 (0xFFFF).		
$P2$ (I)	The file handle of the file to be read.		
$P3$ (I)	The memory block to receive the data read from the file.		
$P4$ (I/C)	Number of bytes to be read from the file. The maximum you can specify is 32767(0x7FFF).		
Example 1	<pre>\$U200 = READ_FILE(\$U100, \$U150, 20) /* Read 20 bytes from the file specified by the file handle in \$U100 and saves the data in the memory block starting from \$U150. */</pre>		

WRITE_FILE

Format	<i>P1</i> = WRITE_FILE(<i>P2,P3,P4</i>)	Data Type	U
Function	Writes <i>P4</i> bytes of data in <i>P3</i> to file <i>P2</i> and saves the completion code in <i>P1</i> .		
<i>P1</i> (I)	The word to receive the completion code of the operation. If the completion code is 0, the operation succeeded; otherwise the operation failed.		
<i>P2</i> (I)	The file handle of the file.		
<i>P3</i> (I)	The memory block (or byte array) that stores the data to be written to the file.		
<i>P4</i> (I/C)	Number of bytes to be written to the file.		
Example 1	\$U200=WRITE_FILE(\$U100,\$U150,30) /* Write 30 bytes of data stored in the memory block starting from \$U150 to the file specified by the file handle in \$U100. */		

CLOSE_FILE

Format	<i>P1</i> = CLOSE_FILE(<i>P2,P3</i>)	Data Type	U
Function	Closes an opened file <i>P2</i> and saves the completion code in <i>P1</i> .		
<i>P1</i> (I)	The word to receive the completion code of the operation. If the completion code is 0, the operation succeeded; otherwise the operation failed.		
<i>P2</i> (I)	The file handle of the file to be closed.		
Example 1	\$U200=CLOSE_FILE(\$U100) /* Close the file specified by the file handle in \$U100. */		

DELETE_FILE

Format	<i>P1</i> = DELETE_FILE(<i>P2</i>)	Data Type	U
Function	Deletes a file named <i>P2</i> and saves the completion code in <i>P1</i> .		
<i>P1</i> (I)	The word to receive the completion code of the operation. If the completion code is 0, the operation succeeded; otherwise the operation failed.		
<i>P2</i> (I)	The byte array that contains the filename or the full pathname of the file to be deleted. The name is a null-terminated string and has only ASCII characters in it.		
Example 1	\$U10 = "test.txt" \$U200 = DELETE_FILE(\$U10) /* Delete the file "test.txt". */		

RENAME_FILE

Format	<i>P1</i> = RENAME_FILE(<i>P2,P3</i>)	Data Type	U
Function	Renames file <i>P2</i> with new name <i>P3</i> and saves the completion code in <i>P1</i> .		
<i>P1</i> (I)	The word to receive the completion code of the operation. If the completion code is 0, the operation succeeded; otherwise the operation failed.		
<i>P2</i> (I)	The byte array that contains the filename or the full pathname of the file to be renamed. The name is a null-terminated string and has only ASCII characters in it.		
<i>P3</i> (I)	The byte array that contains the new filename. The name is a null-terminated string and has only ASCII characters in it.		
Example 1	\$U10 = "test.txt" \$U50 = "new.txt" \$U200 = RENAME_FILE(\$U10, \$U50) /* Rename the file "test.txt" to "new.txt". */		

GET_VOL_INFO

Format	P1 = GET_VOL_INFO(P2,P3)		Data Type	U															
Function	Gets the information of volume P2 and saves the result in P3 . The completion code is saved in P1 .																		
P1 (I)	The word to receive the completion code of the operation. If the completion code is 0, the operation succeeded; otherwise the operation failed.																		
P2 (I/C)	<div>The drive ID.</div> <table><tr><th>ID</th><th>Drive</th></tr><tr><td>0</td><td>Current drive</td></tr><tr><td>3</td><td>Drive C</td></tr><tr><td>4</td><td>Drive D</td></tr><tr><td>5</td><td>Drive E</td></tr></table>				ID	Drive	0	Current drive	3	Drive C	4	Drive D	5	Drive E					
ID	Drive																		
0	Current drive																		
3	Drive C																		
4	Drive D																		
5	Drive E																		
P3 (I)	<div>The starting location of the memory block that is used as a Volume Information Block to receive the operation result. The structure of the Volume Information Block is shown below:</div> <table><tr><th>Data Item</th><th>Data Type/Size</th><th>Word No.</th></tr><tr><td>Volume name</td><td>Byte array with 32 elements</td><td>0 through 15</td></tr><tr><td>Volume size</td><td>32-bit Unsigned Integer</td><td>16 and 17</td></tr><tr><td>Free size</td><td>32-bit Unsigned Integer</td><td>18 and 19</td></tr><tr><td>Drive ID</td><td>16-bit Unsigned Integer</td><td>20</td></tr></table> <div>The volume name is a null-terminated character string. The maximum allowable size is 31 characters. Both the unit of volume size and the unit of free size are 1024 bytes. A Volume Information Block requires 21 words.</div>				Data Item	Data Type/Size	Word No.	Volume name	Byte array with 32 elements	0 through 15	Volume size	32-bit Unsigned Integer	16 and 17	Free size	32-bit Unsigned Integer	18 and 19	Drive ID	16-bit Unsigned Integer	20
Data Item	Data Type/Size	Word No.																	
Volume name	Byte array with 32 elements	0 through 15																	
Volume size	32-bit Unsigned Integer	16 and 17																	
Free size	32-bit Unsigned Integer	18 and 19																	
Drive ID	16-bit Unsigned Integer	20																	
Example 1	<div>\$U100 = GET_VOL_INFO(0, \$U0) /* Get the volume information of the current drive. The volume name will be stored in \$U0 through \$U15. The size of the drive will be stored in \$U16 and \$U17. The free size of the drive will be stored in \$U18 and \$U19. The ID of the current drive will be stored in \$U20. */</div>																		

READ_CSV

Format	$P1 = \text{READ_CSV}(P2, P3, P4)$	Data Type	S/U/SD/UD/F
Function	Reads the data in the field of row P3 and column P4 of the CSV file P2 and saves the result in P1 .		
P1 (I)	The word location to receive the value of the specified field. The data type selected for this command should be the same as the data type of the specified field, or the operation may fail. If the operation fails for any reason, no value will be written to P1 . To know if the operation failed or not, check the word \$S522. When the value of \$S522 is non-zero, the operation failed.		
P2 (I)	The file handle of the file to be read. The file must be a CSV file and is opened with the purpose of Read CSV File. The delimiter must be TAB.		
P3 (I/C)	The row number of the field to be read. The row counts from 0.		
P4 (I/C)	The column number of the field to be read. The column counts from 0.		
Example 1	<pre>\$U10 = "test.csv" \$U100 = OPEN_FILE(\$U10,5) /* Open the file "test.csv" for the READ CSV FILE operation. */ \$U200 = READ_CSV(\$U100,2,3) (F) /* Read the floating point number in the field of row 2 and column 3 and save the result in \$U200 and \$U201. */</pre>		

READ_CSV_STR

Format	<i>P1</i> = READ_CSV_STR (<i>P2</i> , <i>P3</i> , <i>P4</i>)
Function	Reads the string in the field of row <i>P3</i> and column <i>P4</i> of the CSV file <i>P2</i> and saves the result in <i>P1</i> .
<i>P1</i> (I)	The byte array to receive the string in the specified field. The maximal string length that this command can handle is 128. If the operation fails for any reason, no value will be written to <i>P1</i> . To know if the operation failed or not, check the word \$S522. When the value of \$S522 is non-zero, the operation failed.
<i>P2</i> (I)	The file handle of the file to be read. The file must be a CSV file and is opened with the purpose of Read CSV File. The delimiter must be TAB.
<i>P3</i> (I/C)	The row number of the field to be read. The row counts from 0.
<i>P4</i> (I/C)	The column number of the field to be read. The column counts from 0.
Example 1	<pre>\$U10 = "test.csv" \$U100 = OPEN_FILE(\$U10,5) /* Open the file "test.csv" for the READ CSV FILE operation. */ \$U200 = READ_CSV_STR(\$U100,2,4) /* Read the string in the field of row 2 and column 4 and save the result in the byte array starting at \$U200. */</pre>

14.4.16. Comparison

==

Format	$P1 = P2 == P3$	Data Type	U/S/UD/SD/F/B
Function	Sets bit $P1$ to 1 if $P2$ is equal to $P3$, otherwise sets $P1$ to 0.		
$P1$ (I/E)	The bit location to save the result.		
$P2, P3$ (I/E/C/AE)	The operands.		
Example 1	$\$U3.3 = (\$U10 + \$U20) == 25.75$ (F)		

!=

Format	$P1 = P2 != P3$	Data Type	U/S/UD/SD/F/B
Function	Sets bit $P1$ to 1 if $P2$ is not equal to $P3$, otherwise sets $P1$ to 0.		
$P1$ (I/E)	The bit location to save the result.		
$P2, P3$ (I/E/C/AE)	The operands.		
Example 1	$\$U3.3 = (\$U10 + \$U20) != -700$ (S)		

>

Format	$P1 = P2 > P3$	Data Type	U/S/UD/SD/F
Function	Sets bit $P1$ to 1 if $P2$ is greater than $P3$, otherwise sets $P1$ to 0.		
$P1$ (I/E)	The bit location to save the result.		
$P2, P3$ (I/E/C/AE)	The operands.		
Example 1	$\$U3.3 = (\$U10 + \$U20) > \$U30$ (UD)		

>=

Format	$P1 = P2 >= P3$	Data Type	U/S/UD/SD/F
Function	Sets bit $P1$ to 1 if $P2$ is greater than or equal to $P3$, otherwise sets $P1$ to 0.		
$P1$ (I/E)	The bit location to save the result.		
$P2, P3$ (I/E/C/AE)	The operands.		
Example 1	$\$U3.3 = (\$U10 + \$U20) >= 25.75$ (F)		

<

Format	$P1 = P2 < P3$	Data Type	U/S/UD/SD/F
Function	Sets bit $P1$ to 1 if $P2$ is less than $P3$, otherwise sets $P1$ to 0.		
$P1$ (I/E)	The bit location to save the result.		
$P2, P3$ (I/E/C/AE)	The operands.		
Example 1	$\$U3.3 = (\$U10 + \$U20) < 25.75$ (F)		

<=

Format	$P1 = P2 <= P3$	Data Type	U/S/UD/SD/F
Function	Sets bit $P1$ to 1 if $P2$ is less than or equal to $P3$, otherwise sets $P1$ to 0.		
$P1$ (I/E)	The bit location to save the result.		
$P2, P3$ (I/E/C/AE)	The operands.		
Example 1	$\$U3.3 = (\$U10 + \$U20) <= 25.75$ (F)		

14.4.17. String Operation

STRCPY

Format	STRCPY(<i>P1</i> , <i>P2</i>)												
Function	Copies the string in <i>P2</i> to <i>P1</i> .												
<i>P1</i> (l)	The byte array that receives a copy of the string in <i>P2</i> . The byte array must be large enough to hold the string and the null terminator.												
<i>P2</i> (l)	The source, i.e. the byte array that contains the null-terminated string to be copied.												
Example 1	<div><div>\$U10 = “ABCDE”</div><div>STRCPY(\$U20, \$U10)</div><div>After the command STRCPY is executed, the byte array \$U20 contains the string “ABCDE” and the memory content is like the following:</div><table><tr><th>Word</th><th>Low Byte</th><th>High Byte</th></tr><tr><td>\$U20</td><td>'A'</td><td>'B'</td></tr><tr><td>\$U21</td><td>'C'</td><td>'D'</td></tr><tr><td>\$U22</td><td>'E'</td><td>0</td></tr></table></div>	Word	Low Byte	High Byte	\$U20	'A'	'B'	\$U21	'C'	'D'	\$U22	'E'	0
Word	Low Byte	High Byte											
\$U20	'A'	'B'											
\$U21	'C'	'D'											
\$U22	'E'	0											
Example 2	<div><div>\$U10 = “12”</div><div>STRCPY(\$U20, \$U10)</div><div>After the command STRCPY is executed, the byte array \$U20 contains the string “12” and the memory content is like the following:</div><table><tr><th>Word</th><th>Low Byte</th><th>High Byte</th></tr><tr><td>\$U20</td><td>'1'</td><td>'2'</td></tr><tr><td>\$U21</td><td>0</td><td>Undefined</td></tr></table></div>	Word	Low Byte	High Byte	\$U20	'1'	'2'	\$U21	0	Undefined			
Word	Low Byte	High Byte											
\$U20	'1'	'2'											
\$U21	0	Undefined											

STRCAT

Format	STRCAT(<i>P1</i> , <i>P2</i>)
Function	Appends string in <i>P2</i> to string in <i>P1</i> .
<i>P1</i> (l)	The byte array that contains a null-terminated string to which the command appends <i>P2</i> . The byte array must be large enough to hold both strings and the null terminator.
<i>P2</i> (l)	The byte array that contains a null-terminated string to be appended to the string in <i>P1</i> .
Example 1	<div> <div>\$U10 = "ABC"</div> <div>\$U20 = "12345"</div> <div>STRCAT(\$U10, \$U20) /* After this command is executed, the byte array \$U10 contains "ABC12345" */</div> </div>
Example 2	<div> <div>\$U100 = "C:\MyFolder\"</div> <div>\$U130 = "Test"</div> <div>\$U140 = ".txt"</div> <div>STRCAT(\$U100, \$U130)</div> <div>STRCAT(\$U100, \$U140) /* After this command is executed, the byte array \$U100 contains "C:\MyFolder\Test.txt" */</div> </div>

STRLEN

Format	<i>P1</i> = STRLEN(<i>P2</i>)
Function	Gets the length of string <i>P2</i> and saves the result in <i>P1</i> .
<i>P1</i> (I)	The word to receive the result.
<i>P2</i> (I)	The byte array that stores the null-terminated string.
Example 1	<pre>\$U10 = "ABC" \$U20 = STRLEN(\$U10) /* After this command is executed, the value of \$U20 is 3. */</pre>

STRCMP

Format	<i>P1</i> = STRCMP(<i>P2</i>,<i>P3</i>)								
Function	Compares strings <i>P2</i> and <i>P3</i> lexicographically and saves a value indicating their relationship in <i>P1</i> .								
<i>P1</i> (I)	<div>The value of comparison result.</div> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>0</td><td><i>P2</i> and <i>P3</i> are identical.</td></tr> <tr> <td>1</td><td><i>P2</i> is greater than <i>P3</i>.</td></tr> <tr> <td>0xFFFF</td><td><i>P2</i> is less than <i>P3</i>.</td></tr> </table>	Value	Description	0	<i>P2</i> and <i>P3</i> are identical.	1	<i>P2</i> is greater than <i>P3</i> .	0xFFFF	<i>P2</i> is less than <i>P3</i> .
Value	Description								
0	<i>P2</i> and <i>P3</i> are identical.								
1	<i>P2</i> is greater than <i>P3</i> .								
0xFFFF	<i>P2</i> is less than <i>P3</i> .								
<i>P2</i>,<i>P3</i> (I)	The byte array that contains a null-terminated string to compare.								
Example 1	<pre>\$U10 = "ABC" \$U20 = "abc" \$U30 = STRCMP(\$U10, \$U20) /* After this command is executed, \$U30 is 0xFFFF*/</pre>								
Example 2	<pre>\$U10 = "XYZ" \$U20 = "ABC" \$U30 = STRCMP(\$U10, \$U20) /* After this command is executed, \$U30 is 1*/</pre>								
Example 3	<pre>\$U10 = "ABC" \$U20 = "ABC" \$U30 = STRCMP(\$U10, \$U20) /* After this command is executed, \$U30 is 0*/</pre>								

STRICMP

Format	P1 = STRICMP(P2,P3)	
Function	Compares lowercase version of strings P2 and P3 lexicographically and saves a value indicating their relationship in P1 .	
P1 (I)	The value of comparison result.	
	Value	Description
	0	P2 and P3 are identical.
	1	P2 is greater than P3 .
	0xFFFF	P2 is less than P3 .
P2,P3 (I)	The byte array that contains a null-terminated string to compare.	
Example 1	\$U10 = "ABC" \$U20 = "abc" \$U30 = STRICMP(\$U10, \$U20) /* After this command is executed, \$U30 is 0*/	
Example 2	\$U10 = "XYZ" \$U20 = "ABC" \$U30 = STRICMP(\$U10, \$U20) /* After this command is executed, \$U30 is 1*/	
Example 3	\$U10 = "ABC" \$U20 = "ABC" \$U30 = STRCMP(\$U10, \$U20) /* After this command is executed, \$U30 is 0xFFFF*/	

STRNCMP

Format	P1 = STRNCMP(P2,P3,P4)	
Function	Lexicographically compares, at most, the first P4 characters in strings P2 and P3 and saves a value indicating the relationship between the substrings in P1 .	
P1 (I)	The value of comparison result.	
	Value	Description
	0	P2 's substring and P3 's substring are identical
	1	P2 's substring is greater than P3 's substring .
	0xFFFF	P2 's substring is less than P3 's substring .
	<p>Note: The comparison ends if a terminating null character is reached in either string before P4 characters are compared. If the strings are equal when a terminating null character is reached in either string before P4 characters are compared, the shorter string is less.</p> <p>The characters from 91 to 96 in the ASCII table ('[', '\', ']', '^', '_', and '^') will evaluate as less than any alphabetic character.</p>	
P2,P3 (I)	The byte array that contains a null-terminated string to compare.	
P4 (I/C)	The number of characters to compare.	
Example 1	<pre>\$U10 = "XYZ" \$U20 = "XYZAB" \$U30 = STRNCMP(\$U10, \$U20,4) /* After this command is executed, \$U30 is 0xFFFF*/</pre>	
Example 2	<pre>\$U10 = "ABZ" \$U20 = "ABC" \$U30 = STRNCMP(\$U10, \$U20,2) /* After this command is executed, \$U30 is 0*/</pre>	
Example 3	<pre>\$U10 = "AXC" \$U20 = "ABC" \$U30 = STRNCMP(\$U10, \$U20,3) /* After this command is executed, \$U30 is 1*/ \$U30 = STRCMP(\$U10, \$U20) /* After this command is executed, \$U30 is 0xFFFF*/</pre>	

STRCHR

Format	P1 = STRCHR(P2,P3)	
Function	Finds the first occurrence of a character P3 in a string P2 and saves a search result in value indicating the position of the found character in P1 .	
P1 (I)	The value of search result. If the character P3 is not found in P2 , the result value is 0xFFFF. Otherwise, the result value is the index to the first occurrence of character P3 in a string P2 .	
P2 (I)	The byte array that contains a null-terminated source string.	
P3 (I/C)	The byte that contains a character code to be located.	
Example 1	<pre>\$U10 = "The quick brown dog jumps over the lazy fox." \$U20 = 0x72 /* The ASCII code of character 'r' */ \$U30 = STRCHR(\$U10, \$U20) /* After this command is executed, \$U30 is 11*/</pre>	

NUM2STR

Format	<i>P1</i> = NUM2STR(<i>P2</i> , <i>P3</i>)	Data Type	U/UD
Function	Converts the number in <i>P2</i> to a string with <i>P3</i> characters and saves the result in <i>P1</i> .		
<i>P1</i> (I)	The byte array that stores the result.		
<i>P2</i> (I/C)	The number or the location that holds the number to be converted.		
<i>P3</i> (I/C)	Specifies the exact number of characters that the result should have. If the number of digits of <i>P2</i> is less than <i>P3</i> , the result is padded on the left with zeros. If the number of digits of <i>P2</i> exceeds <i>P3</i> , the higher digits are truncated. If <i>P3</i> is 0, there is no limitation on the length of the result.		
Example 1	<i>\$U120</i> = 123 <i>\$U100</i> = NUM2STR(<i>\$U120</i> , 0) (U) /* After this command is executed, the byte array <i>\$U100</i> contains "123". */		
Example 2	<i>\$U120</i> = 1234567 (UD) <i>\$U100</i> = NUM2STR(<i>\$U120</i> , 10) (UD) /* After this command is executed, the byte array <i>\$U100</i> contains "0001234567". */		
Example 3	<i>\$U120</i> = 1234567 (UD) <i>\$U100</i> = NUM2STR(<i>\$U120</i> , 5) (UD) /* After this command is executed, the byte array <i>\$U100</i> contains "34567". */		

TIME2STR

Format	<i>P1</i> = TIME2STR(<i>P2</i>)		Data Type	U
Function	Converts the current system time to a string using the format specified by <i>P2</i> and saves the result in <i>P1</i> .			
<i>P1</i> (I)	The byte array that stores the result.			
<i>P2</i> (I/C)	Specifies the desired conversion format.			
	Format	<i>P2</i> Value	Remark	
	hhmmss	0	hh: hour(00-23); mm: minute(00-59); ss: second(00-59)	
	hhmm	1	hh, mm: same as above	
Example 1	\$U10 = TIME2STR(0) /* Assume that the current system time is 12:30:59. After this command is executed, the byte array \$U10 contains "123059". */			

DATE2STR

Format	<i>P1</i> = DATE2STR(<i>P2</i>)		Data Type	U
Function	Converts the current system date to a string using the format specified by <i>P2</i> and saves the result in <i>P1</i> .			
<i>P1</i> (I)	The byte array that stores the result.			
<i>P2</i> (I/C)	Specifies the desired conversion format.			
	Format	<i>P2</i> Value	Remark	
	YYMMDD	0	YY: year (00-99); MM: month(01-12); DD: day(01-31)	
	YYMM	1	YY, MM: same as above	
	YYMMMDD	2	YY: year (00-99); MMM: month(JAN-DEC); DD: day(01-31)	
	YYMMM	3	YY, MMM: same as above	
Example 1	\$U10 = DATE2STR(0) /* Assume that the current system date is December 7, 2008. After this command is executed, the byte array \$U10 contains “081207”. */			
Example 2	\$U20 = DATE2STR(3) /* Assume that the current system date is December 31, 2008. After this command is executed, the byte array \$U20 contains “08DEC”. */			

TD2STR

Format	$P1 = \text{TD2STR}(P2)$	Data Type	U
Function	Converts the current system time and date to a string using the format specified by $P2$ and saves the result in $P1$.		
$P1$ (I)	The byte array that stores the result.		
$P2$ (I/C)	Specifies the desired conversion format.		
	Format	$P2$ Value	Remark
	YYMMDD_hhmmss	0	YY: year (00-99); MM: month(01-12); DD: day(01-31) hh: hour(00-23); mm: minute(00-59) ; ss: second(00-59)
	YYMMDD_hhmmss	1	YY, DD, hh, mm, ss: same as above MMM: month(JAN-DEC)
	YYMMDD_hhmm	2	YY, DD, hh, mm: same as above; MM: month(01-12)
	YYMMDD_hhmm	3	YY, DD, hh, mm: same as above; MMM: month(JAN-DEC)
Example 1	$\$U10 = \text{TD2STR}(0)$ /* Assume that the current system date is December 7, 2008 and the current system time is 15:18:30. After this command is executed, the byte array \$U10 contains "081207_151830". */		
Example 2	$\$U20 = \text{TD2STR}(3)$ /* Assume that the current system date is December 31, 2008 and the current system time is 13:30:00. After this command is executed, the byte array \$U20 contains "08DEC31_1330". */		

I2A

Format	$P1 = \text{I2A}(P2, P3)$	Data Type	U/S/UD/SD
Function	Converts the integer number in $P2$ to a string and saves the result in $P1$. The string is generated according to the format specified by $P3$ and $P4$.		
$P1$ (I)	The byte array that stores the result. The result is a null terminated string.		
$P2$ (I/C)	The integer number or the location that holds the integer number to be converted.		
$P3$ (I/C)	Specifies the maximum number of digits the string can have.		
$P4$ (I/C)	Specifies where to insert a decimal point in the string. A decimal point is inserted to the left of the nth digit when $P4$ is n. No decimal point is inserted when $P4$ is 0.		
Example 1	$\$U120 = 123$ $\$U100 = \text{I2A}(\$U120, 5, 0)$ /* After this command is executed, the byte array \$U100 contains "123". */		
Example 2	$\$U120 = 1234567$ (UD) $\$U100 = \text{I2A}(\$U120, 6, 2)$ (UD) /* After this command is executed, the byte array \$U100 contains "2345.67". */		
Example 3	$\$U120 = -12345$ (S) $\$U100 = \text{I2A}(\$U120, 5, 1)$ (UD) /* After this command is executed, the byte array \$U100 contains "-1234.5". */		

A2I

Format	$P1 = A2I(P2, P3, P4)$	Data Type	U/S/UD/SD
Function	Converts the string P2 to an integer value and saves the result in P1 .		
P1 (I)	The location that stores the result. The result is 0 when there is any conversion error.		
P2 (I)	The byte array that holds the string to be converted.		
P3 (I/C)	Specifies the length of the string. It is allowed to specify 0 for P3 . When P3 is 0, the string must be a null terminated string.		
P4 (I/C)	Specifies how many fractional digits in the string are to be converted.		
Example 1	$\$U120 = "123"$ $\$U100 = A2I(\$U120, 0, 0)$ /* After this command is executed, the value in word \$U100 is 123. */		
Example 2	$\$U120 = "1234567"$ $\$U100 = A2I(\$U120, 6, 0)$ (UD) /* After this command is executed, the value in double word \$U100 is 123456. */		
Example 3	$\$U120 = "-123.45"$ $\$U100 = A2I(\$U120, 0, 2)$ (S) /* After this command is executed, the value in word \$U100 is -12345. */		

F2A

Format	$P1 = F2A(P2, P3)$	Data Type	F
Function	Converts the floating point number in P2 to a string and saves the result in P1 . The string is generated according to the format specified by P3 and P4 .		
P1 (I)	The byte array that stores the result. The result is a null terminated string.		
P2 (I/C)	The floating point number or the location that holds the floating point number to be converted.		
P3 (I/C)	Specifies the number of integral digits the string can have.		
P4 (I/C)	Specifies the number of fractional digits the string can have.		
Example 1	$\$U120 = 123.45$ (F) $\$U100 = F2A(\$U120, 5, 2)$ /* After this command is executed, the byte array \$U100 contains "123.45". */		
Example 2	$\$U120 = 1234$ (F) $\$U100 = F2A(\$U120, 6, 2)$ (UD) /* After this command is executed, the byte array \$U100 contains "1234.00". */		
Example 3	$\$U120 = -1234.5$ (S) $\$U100 = F2A(\$U120, 5, 1)$ (UD) /* After this command is executed, the byte array \$U100 contains "-1234.5". */		

A2F

Format	<i>P1</i> = A2F(<i>P2</i> , <i>P3</i>)	Data Type	F
Function	Converts the string <i>P2</i> to a floating point number and saves the result in <i>P1</i> .		
<i>P1</i> (I)	The location that stores the result. The result is 0 when there is any conversion error.		
<i>P2</i> (I)	The byte array that holds the string to be converted.		
<i>P3</i> (I/C)	Specifies the length of the string. It is allowed to specify 0 for <i>P3</i> . When <i>P3</i> is 0, the string must be a null terminated string.		
Example 1	<pre>\$U120 = "123.4" \$U100 = A2F(\$U120, 0) /*The value of the floating point number in double word \$U100 is 123.4. */</pre>		
Example 2	<pre>\$U120 = "1234567" \$U100 = A2F(\$U120, 6) (UD) /* The value of the floating point number in double word \$U100 is 123456. */</pre>		
Example 3	<pre>\$U120 = "-123.45" \$U100 = A2F(\$U120, 0) (S) /* The value of the floating point number in double word \$U100 is -123.45. */</pre>		

14.4.18. Run Operation

RUN

Format	RUN(P1)
Function	Runs the executable P1 which is on the same PC. This command is only available for the runtime software on the PC.
P1 (I/A)	The name of the executable to be run.
Example 1	RUN "ABC.exe" /* Run the program ABC */
Example 2	\$U10 = "XYZ.bat" RUN \$U10 /* Run the batch file XYZ */

RUNW

Format	P1 = RUNW(P2)
Function	Runs the executable P2 which is on the same PC and saves the result in P1 . Note that the macro command following this one will not be executed until the program is closed. This command is only available for the runtime software on the PC.
P1 (I)	The word to receive the result.
P2 (I/A)	The name of the executable to be run.
Example 1	\$U10 = RUNW "ABC.exe" /* Run the program ABC and use \$U10 to get the result. */ IF \$U10 == 0 /* If the result is 0 then run the batch file XYZ. */ \$U20 = "XYZ.bat" \$U11 = RUNW \$U20 /* Run the batch file XYZ. */ ENDIF

14.4.19. Print Operation

PRINT

Format	$P1 = \text{PRINT}(P2,P3)$	Data Type	U												
Function	Sends $P3$ bytes of data stored in byte array $P2$ to the printer and saves the completion code in $P1$.														
$P1$ (I)	<div>The word to receive the completion code of the operation. The following table describes the meanings of the completion codes.</div> <table><tr><th>Code</th><th>Description</th></tr><tr><td>0</td><td>Succeeded</td></tr><tr><td>1</td><td>Printer not ready</td></tr><tr><td>3</td><td>System error</td></tr><tr><td>4</td><td>Printer busy</td></tr><tr><td>7</td><td>No printer specified</td></tr></table>			Code	Description	0	Succeeded	1	Printer not ready	3	System error	4	Printer busy	7	No printer specified
Code	Description														
0	Succeeded														
1	Printer not ready														
3	System error														
4	Printer busy														
7	No printer specified														
$P2$ (I)	The starting location of the byte array that stores the data to be sent to the printer.														
$P3$ (I/C)	The length in byte of the data to be sent to the printer.														
Example 1	$\$U10 = \text{"This is a test."}$ $\$U20 = \text{PRINT}(\$U10, 15)$ /* Send the string "This is a test." to the printer. */ $\$U10 = 10$ $\$U20 = \text{PRINT}(\$U10, 1)$ /* Send the line-feed character to the printer */ $\$U10 = 12$ $\$U20 = \text{PRINT}(\$U10, 1)$ /* Send the form-feed character to the printer */														
Example 2	$\$U10 = 0x401b$ /* ESC, '@' */ $\$U20 = \text{PRINT}(\$U10, 2)$ /* Send the initialization command to the EPSON printer */														

PRINT_SCREEN

Format	<i>P1</i> = PRINT_SCREEN(<i>P2</i> , <i>P3</i>)	Data Type	U																		
Function	Prints screen <i>P2</i> and saves the result in <i>P1</i> .																				
<i>P1</i> (I)	The word to receive the completion code of the operation. The following table describes the meanings of the completion codes.																				
	<table><tr><th>Code</th><th>Description</th></tr><tr><td>0</td><td>Succeeded</td></tr><tr><td>1</td><td>Printer not ready</td></tr><tr><td>2</td><td>Invalid screen number</td></tr><tr><td>3</td><td>System error</td></tr><tr><td>4</td><td>Printer busy</td></tr><tr><td>5</td><td>System busy</td></tr><tr><td>6</td><td>Improper use of this command (See Note)</td></tr><tr><td>7</td><td>No printer specified</td></tr></table>			Code	Description	0	Succeeded	1	Printer not ready	2	Invalid screen number	3	System error	4	Printer busy	5	System busy	6	Improper use of this command (See Note)	7	No printer specified
	Code	Description																			
	0	Succeeded																			
	1	Printer not ready																			
	2	Invalid screen number																			
	3	System error																			
	4	Printer busy																			
	5	System busy																			
	6	Improper use of this command (See Note)																			
7	No printer specified																				
Note: This command can only be used in the following types of macros: Main Macro, Event Macro, Time Macro, and Cycle Macro.																					
<i>P2</i> (I/C)	The number of the screen to be printed. The printed area is specified in the Screen Properties dialog box.																				
<i>P3</i> (I/C)	Reserved for future use. Must be 0.																				
Example 1	\$U0 = PRINT_SCREEN(28, 0) /* Print screen #28*/																				

BLANK

Format	<i>P1</i> = BLANK (<i>P2</i>)	Data Type	U
Function	Blanks the print buffer <i>P1</i> , i.e. makes the print buffer <i>P1</i> contain only blank characters.		
<i>P1</i> (I)	The print buffer to be blanked. The print buffer is a byte array. You should always blank a print buffer before printing strings to it.		
<i>P2</i> (I/C)	The size of the print buffer. The unit is byte (one-byte character). For example, if the size of a print buffer is 40, it has 20 words and can contain up to 40 one-byte characters.		
Example 1	BLANK(\$U100, 80) /* Blank the print buffer starting at \$U100 with a length of 40 words. */		

P2B

Format	<i>P1</i> = P2B (<i>P2</i> , <i>P3</i>)	Data Type	U
Function	Prints the null-terminated string <i>P1</i> to the print buffer <i>P2</i> at the specified byte position <i>P3</i> .		
<i>P1</i> (I)	The byte array that holds the string to be printed.		
<i>P2</i> (I)	The byte array that is used as a print buffer to accept the string <i>P1</i> .		
<i>P3</i> (I/C)	The byte position in the print buffer to put the string. The byte position counts from 0. For example, to print a string at the beginning of the print buffer, set <i>P3</i> to 0.		
Example 1	BLANK(\$U100, 20) /* Blank the print buffer. */ \$U10 = "Weight:" P2B(\$U100, \$U10, 0) /* Print the string "Weight:" at the position of byte 0 of the print buffer. */ \$U10 = I2A(1234, 2) /* The byte array \$U10 will hold the string "12.34" after this command is executed. */ P2B(\$U100, \$U10, 8) /* Print the string "12.34" at the position of byte 8 of the print buffer. */ \$U10 = "kg" P2B(\$U100, \$U10, 14) /* Print the string "kg" at the position of byte 14 of the print buffer. */ PRINT(\$U100, 20) /* Print string "Weight: 12.34 kg" in the print buffer to the real printer. */		

P2B_R

Format	<i>P1</i> = P2B_R (<i>P2</i> , <i>P3</i>)	Data Type	U
Function	Prints the null-terminated string <i>P1</i> to the print buffer <i>P2</i> . The string is right aligned with the byte position <i>P3</i> .		
<i>P1</i> (I)	The byte array that holds the string to be printed.		
<i>P2</i> (I)	The byte array that is used as a print buffer to accept the string <i>P1</i> .		
<i>P3</i> (I/C)	The byte position in the print buffer that the last characters of the string is placed. The byte position counts from 0. For example, to print a string with 6 characters at the beginning of the print buffer, set <i>P3</i> to 5 as the last character of the string should be placed at the position of byte 5.		
Example 1	BLANK(\$U100, 20) /* Blank the print buffer. */ \$U10 = "Weight:" P2B_R(\$U100, \$U10, 6) /* Print the string "Weight:" to the print buffer and align the string right with the position of byte 6. */ \$U10 = I2A(1234, 2) /* The byte array \$U10 will hold the string "12.34" after this command is executed. */ P2B_R(\$U100, \$U10, 12) /* Print the string "12.34" to the print buffer and align the string right with the position of byte 12. */ \$U10 = "kg" P2B_R(\$U100, \$U10, 15) /* Print the string "kg" to the print buffer and align the string right with the position of byte 15. */ PRINT(\$U100, 20) /* Print string "Weight: 12.34 kg" in the print buffer to the real printer. */		

14.4.20. Sound Operation

SOUND

Format	SOUND (<i>P1</i> , <i>P2</i> , <i>P3</i>)	Data Type	U
Function	Plays the sound <i>P1</i> .		
<i>P1</i> (I/C)	The identifier of the sound to be played. Note: The sounds and their identifiers are defined in the sound table of the panel application.		
<i>P2</i> (I/C)	The number of times you want the sound to be played. If you want the specified sound to be played just once, set <i>P2</i> to 1.		
<i>P3</i> (I/C)	The break time between two consecutive plays. The time unit is 100 ms (0.1 second). If you do not want any break between two plays, set <i>P3</i> to 0.		
Example 1	SOUND(10, 5, 3) /* Play the sound #10 5 times with a break of 0.3 second between two consecutive plays. */		

STOP_SOUND

Format	STOP_SOUND
Function	Stops playing the current sound.
Example 1	STOP_SOUND /* Stop playing the current sound.*/