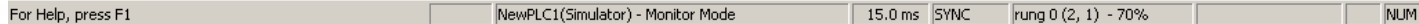


For CX-Programmer Ver.6.1 (CX-One Ver.1.1)



CX-One / CX-Programmer CD-ROM contains an operation manual PDF file.

Before using this product, you must read “Introduction”, “Safety Cautions”, and “Precautions for Safe Use”.

“Function Block Implementation Guide” describes basic operations to use OMRON FB Library and hints to create user program with Function Blocks.

Cautions and detailed explanation are available on Help and PDF Manual.

* Acrobat Reader 4.0 or later is required to read the PDF file.

Chapter 1 OMRON FB Library

1. What is a Function Block?	1-1
2. An Example of a Function Block	1-2
3. Overview of the OMRON FB Library	1-3
3-1. Benefits of the OMRON FB Library	1-3
3-2. Example of using the OMRON FB Library	1-4
3-3. Content of the OMRON FB Library	1-6
3-4. File Catalog and Where to Access the OMRON FB Library	1-7

Chapter 2 How to use the OMRON FB Library

1. Explanation of the target program	2-1
1-1. Application Specifications	2-1
1-2. Specifications of the OMRON FB Part file	2-1
1-3. Input program	2-2
2. Opening a new project and setting the Device Type	2-3
3. Main Window functions	2-4
4. Import the OMRON FB Part file	2-5
5. Program Creation	2-6
5-1. Enter a Normally Open Contact	2-6
5-2. Entering an Instance	2-7
5-3. Entering Parameters	2-7
6. Program Error Check (Compile)	2-9
7. Going Online	2-10
8. Monitoring - 1	2-11
9. Monitoring - 2 Change Parameter Current Value	2-12
10. Online Editing	2-13

Chapter 3 Customize the OMRON FB Part file

1. Explanation of target program	3-1
1-1. Changing File Specifications	3-1
1-2. Changing the contents of the OMRON FB Part file	3-1
2. Copy the OMRON FB Part file	3-2
3. Add a variable to the Function Block	3-3
4. Changing the Function Block Ladder	3-4
4-1. Entering a Contact	3-4
4-2. Checking Usage Status of Variables	3-5

Chapter 4 How to use the ST (Structured Text) language

1. What is the ST Language?	4-1
2. Explanation of the target program	4-1
3. Create a Function Block using ST	4-2
4. Entering Variables into Function Blocks	4-3
5. Entry of ST program	4-4
6. Entering the FB to the Ladder Program and error checking	4-5
7. Program Transfer	4-6
8. Monitoring the Function Block execution	4-7
Reference: Example of an ST program using IF-THEN-ELSE-END_IF	4-8

Chapter 5 Advanced (Componentizing a Program Using FB)

1. Overview	5-1
2. How to Proceed Program Development	5-1
3. Application Example	5-1
4. How to Proceed Program Development	5-2
5. Entering FB Definition	5-9
6. Creating FB Definition Library	5-20
7. Entering Main Program	5-21
8. Debugging Main Program	5-22

Supplemental Information

How to delete unused Function Block definitions

Memory allocation for Function Blocks

Useful Functions

Appendix. Examples of ST (Structured Text)	Appendix
--	----------

Introduction

This document summarizes tips for using OMRON FB Library and creating function block (FB), which are available for Omron's SYSMAC CS1/CJ1-H/CJ1M series CPU unit (unit version 3.0 or later) and CX-Programmer Ver.5.0 or later.

New functions available on CX-Programmer Ver. 6.0

Nesting of FB

Nesting of FBs is now available for easier use of FB as a method to structure and reuse user programs. FB can be called from a ladder program structured text (ST) program that is converted to FB. Following functions are also supported for this.

- Easy understanding of program structure . . . FB Instance Viewer
- Management as components including called FBs . . . Saving and loading of files including called FB
- Easy jump to called FB . . . Mouse double-click operation on FB Instance (call statement)

Monitoring of FB ladder

As with main ladder program, state of FB ladder program can be monitored.

Cross reference pop-up in FB ladder

As with main ladder program, cross reference pop-up is now available in FB ladder program.
Also supported now is jump to output coil from contact via space key.

Jump to ST help

You can jump Structured Text editor through pop-up menu to a help topic to easily check syntax for ST programming.

Jump to library reference of OMRON FB Library

You can easily view a PDF file of library cross reference that describes specifications from a OMRON FB Library registered in a project file.

Caution:

Although you can use a program that contains nested FB for a CS1/CJ1-H/CJ1M series CPU unit (unit version 3.0 or later), if you try to upload a program that contains nested FB by CX-Programmer (Ver.5.0 or older) that does not support nesting, either you will fail to do so or you will just get one under incomplete state.
If you save the file as it is, you will not be able to tell difference between the incomplete and correct programs.

[In case of CX-Programmer Ver.5.0]

Following messages will be shown after upload is finished:

- The PLC properties which are not supported by this version of CX-Programmer are set in the connecting target PLC.
- The PLC properties will not be displayed correctly. Do you wish to continue?

[In case of CX-Programmer Ver.4.0]

Following message will be shown after upload is finished:

- Function Block or other data besides Ladder are included in the programs.

[In case of CX-Programmer Ver.3.*]

When upload is finished, a message "Decompile error" is displayed and no program will be shown.

New Functions that can be used with CX-Programmer Ver.6.1

ST Monitoring, Execute Steps

In order to facilitate the sequential processing language ST debug, the following functions are supported:

- Display current value/change current value during ST Program execution.
- Possible to stop execution at the breakpoint, and execute steps using CX-Simulator

FB Protect Function

FB not wanted to be displayed can be concealed. This helps to protect against mistaken changes, know-how leaks, and improper program changes.

Chapter 1

OMRON FB Library

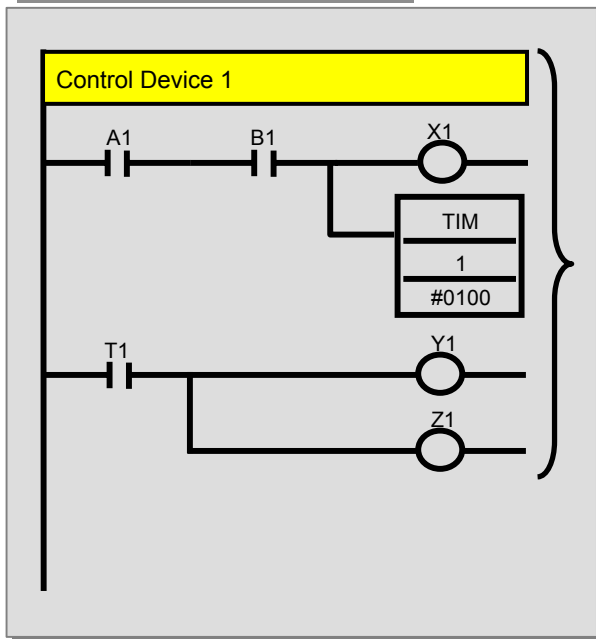
Function Block

1. What is a Function Block?

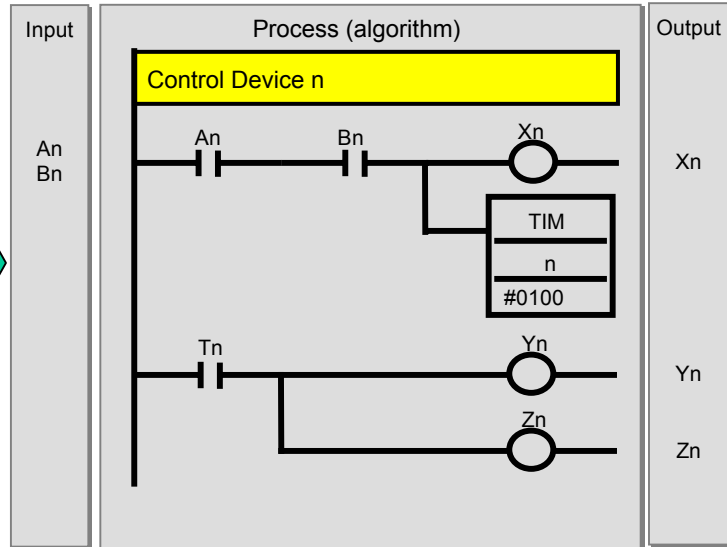
“Function Blocks” are predefined programs (or functions) contained within a single program element that may be used in the ladder diagram. A contact element is required to start the function, but inputs and outputs are editable through parameters used in the ladder arrangement.

The functions can be reused as the same element (same memory) or occur as a new element with its own memory assigned.

Partial Ladder program for machine A

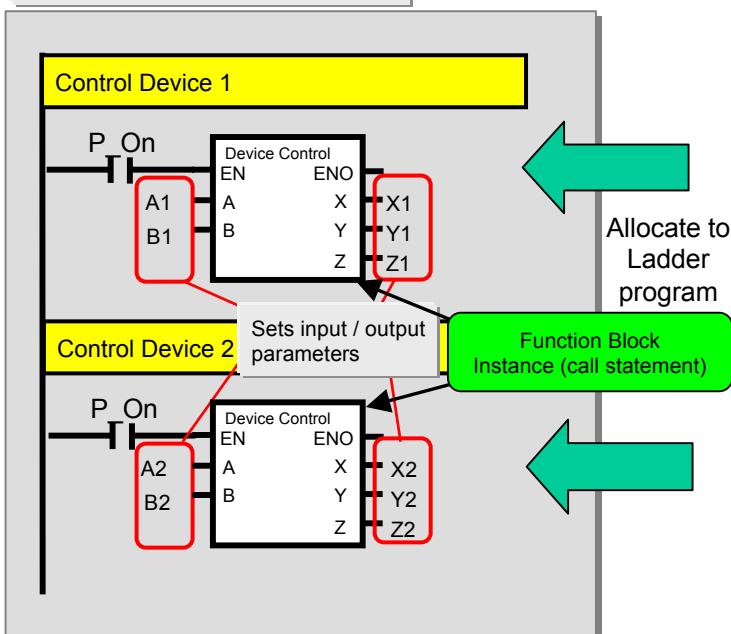


Defining Inputs and Outputs ...

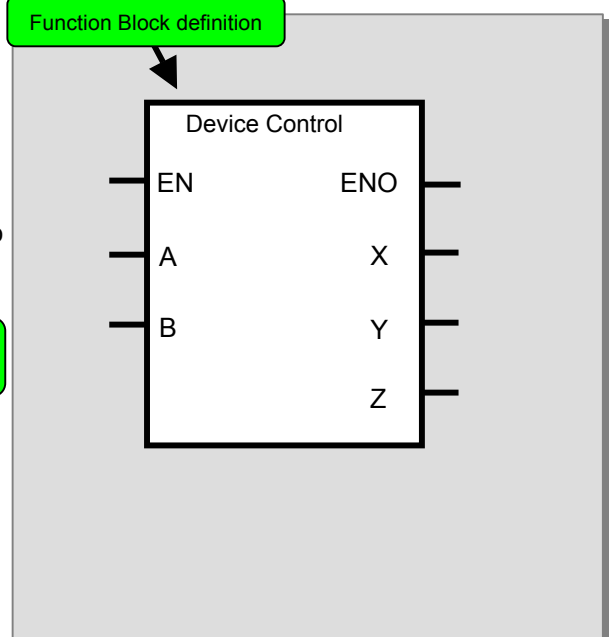


Produce template

Partial Ladder program for machine A



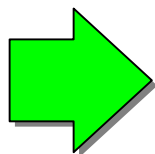
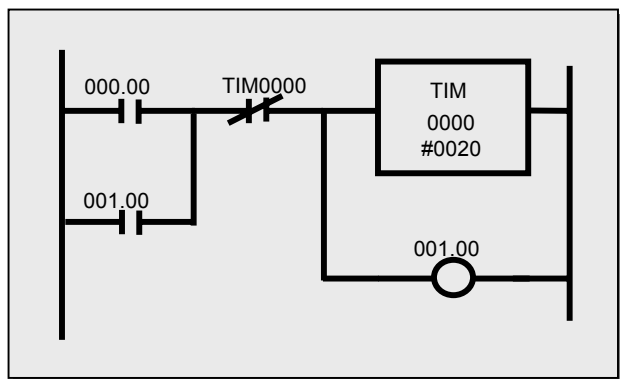
Function Block definition



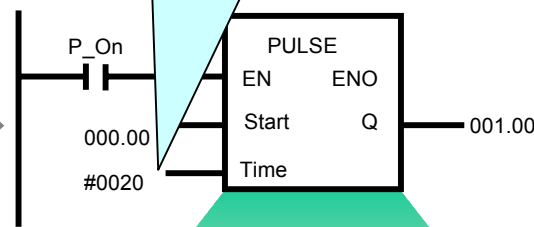
2. An Example of a Function Block

The following figures describe an example of a function block for a time limit circuit, to be used in the ladder. It is possible to edit the set point of the TIM instruction to reallocate the set time for turning off the output in the ladder rung. Using the function block as shown below, it is possible to make the time limit of the circuit arbitrary by only changing one specific parameter.

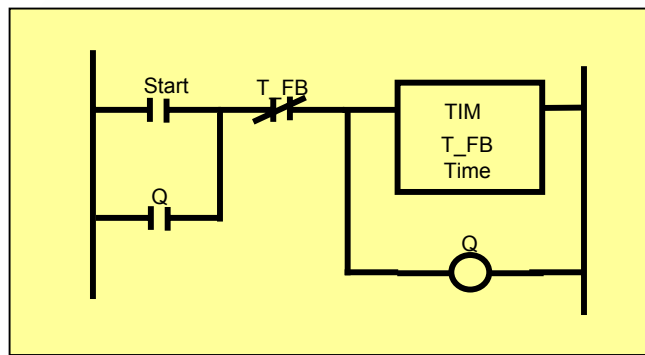
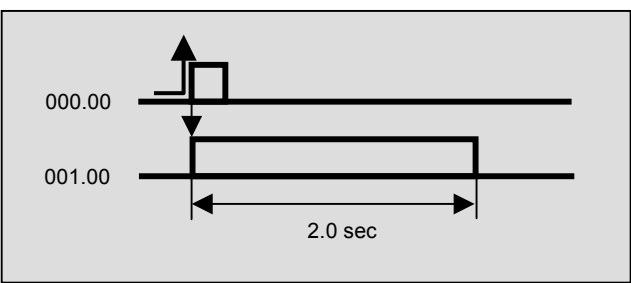
Ladder diagram



By enabling the input parameter to be editable, it is possible to allow an arbitrary time limit circuit.



Timing chart



3. Overview of the OMRON FB Library

The OMRON FB Library is a collection of predefined Function Block files provided by Omron. These files are intended to be used as an aid to simplify programs, containing standard functionality for programming PLCs and Omron FA component functions.

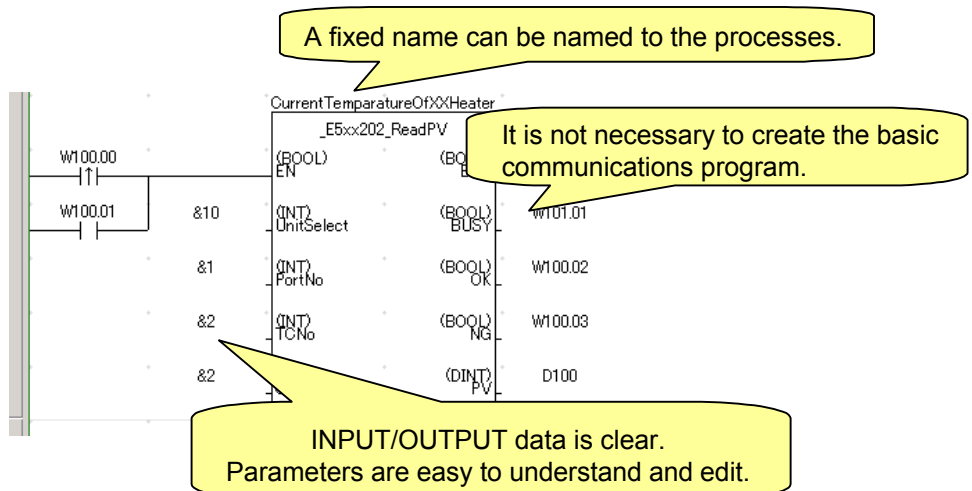
3-1. Benefits of the OMRON FB Library

The OMRON FB Library is a collection of function block examples that aim to improve the connectivity of the units for PLCs and FA components made by Omron. Here is a list of the benefits to be gained from using the OMRON FB Library:

- (1) No need to create ladder diagrams using basic functions of the PLC units and FA components
More time can be spent on bespoke programs for the external devices, rather than creating basic ladder diagrams, as these are already available.
- (2) Easy to use
A functioning program is achieved by loading the function block file to perform the target functionality, then by inputting an instance (function block call statement) to the ladder diagram program and setting addresses (parameters) for the inputs and outputs.
- (3) Testing of program operation is unnecessary
Omron has tested the Function Block library. Debugging the programs for operating the unit and FA components for the PLCs is unnecessary for the user.
- (4) Easy to understand
The function block has a clearly displayed name for its body and instances. A fixed name can be applied to the process.

The instance (function block call statement) has input and output parameters. As the temporary relay and processing data is not displayed, the values of the inputs and outputs are more visible. Furthermore, as the modification of the parameters is localised, fine control during debugging etc. is easier.

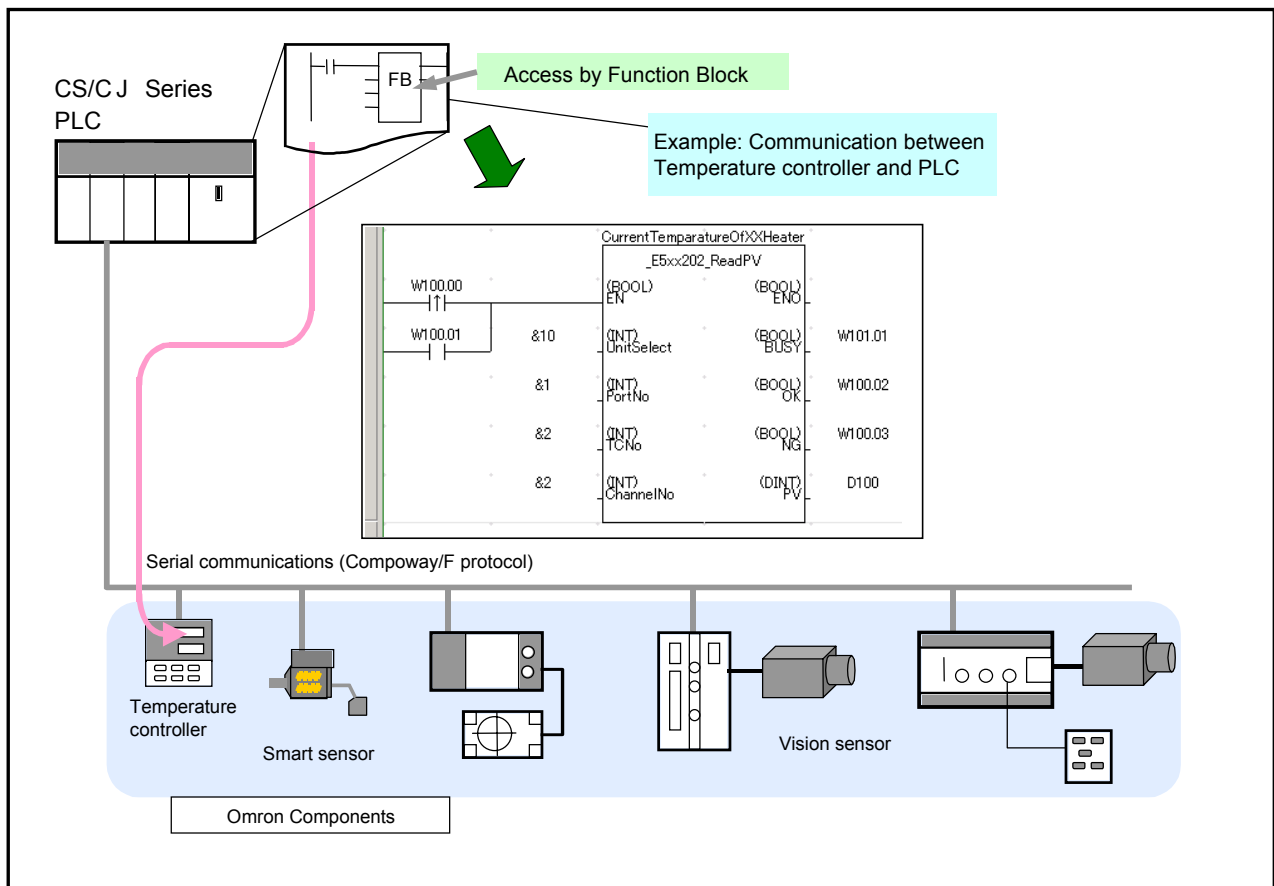
Finally, as the internal processing of the function block is not displayed when the instance is used in the ladder diagram, the ladder diagram program looks simpler to the end user.
- (5) Extendibility in the future
Omron will not change the interface between the ladder diagram and the function blocks. Units will operate by replacing the function block to the corresponding FB for the new unit in the event of PLC and the FA component upgrades, for higher performance or enhancements, in the future.



3-2-1. Example of using the OMRON FB Library - 1

Controlling the predefined components made by Omron can be easily achieved from the PLC ladder diagram.

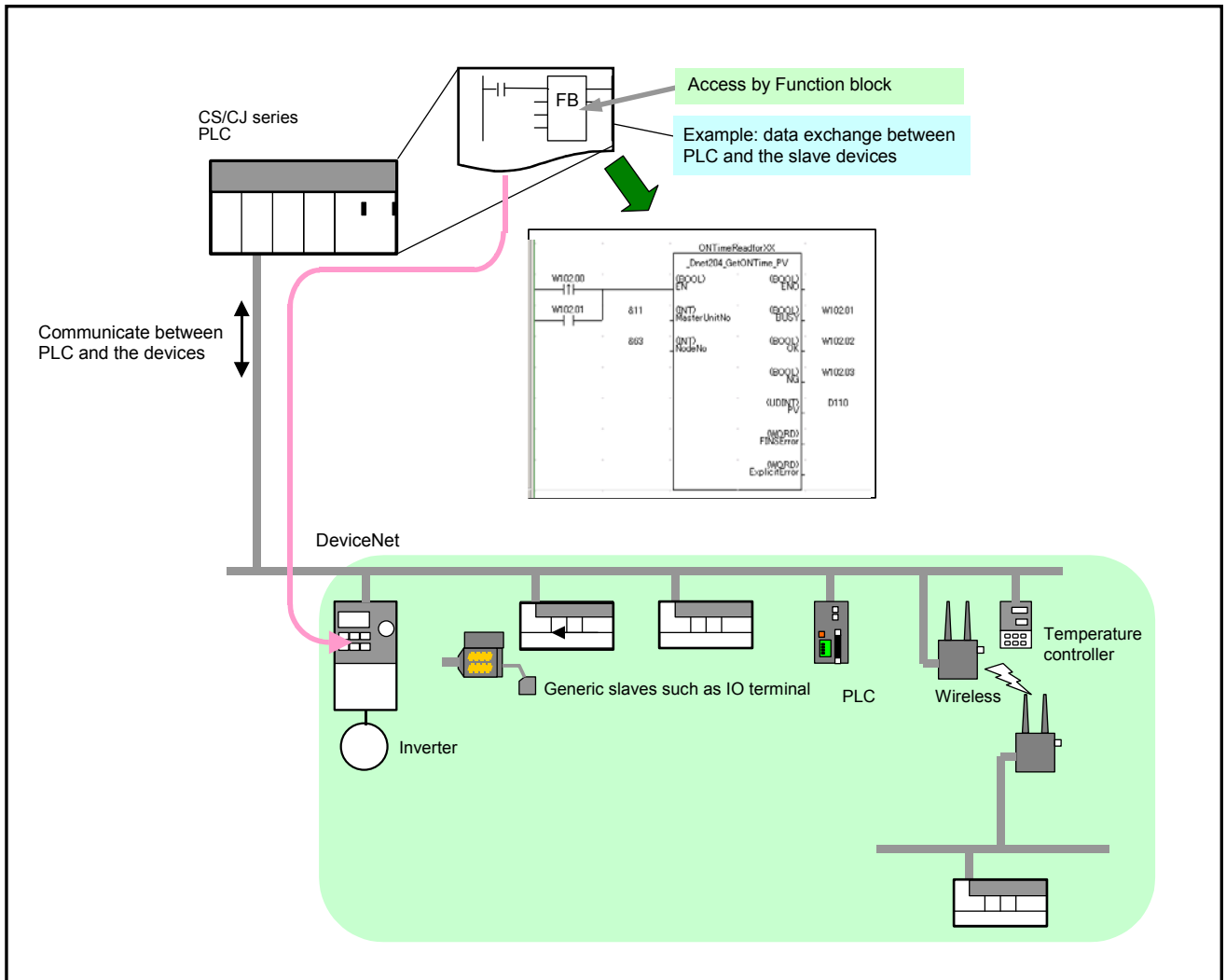
- Ability to configure low-cost communications (RS-232C/485)



3-2-2. Example of using the OMRON FB Library - 2

High performance communications can be made by DeviceNet level.

- Ability to communicate between PLC and DeviceNet slaves easily.



3-3. Content of the OMRON FB Library

The OMRON FB Library consist of the following:

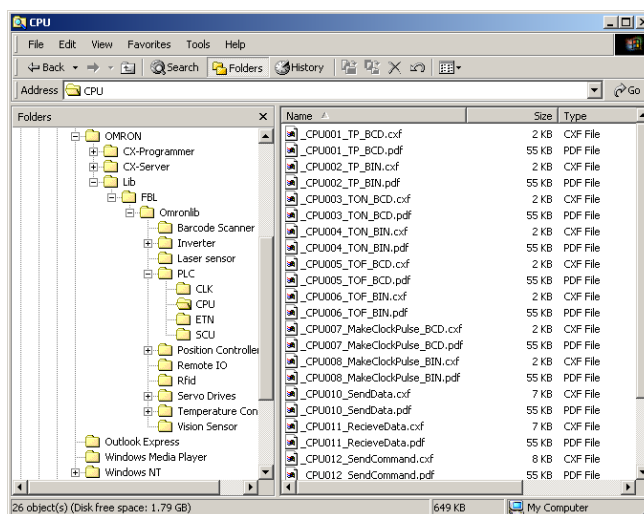
3-3-1. OMRON FB Part Files

The OMRON FB Part file is prepared using the ladder diagram function block, for defining each function of the PLC unit and the FA component.

The files contain a program written in ladder diagram and have the extension .CXF.

The file name of the OMRON FB Part file begins with '_' (under score).

When the OMRON FB Library is installed onto a personal computer, the OMRON FB Part files are classified in the folder appropriate to each PLC Unit and FA component in the Omron Installation directory.



3-3-2. Library reference

The library reference describes the operation specifications of the OMRON FB Part file, and the specifications of the input and the output parameters for each. The file format for this is PDF.

When the OMRON FB Library is used, the user should select the OMRON FB Part file, set the input / output parameters, and test the program operations referring to the library reference.

V60x 200	Read Data Carrier Data _V60x200_ReadData
FB name Symbol	<div> <div> <div>Start trigger</div> <div>Unit No.</div> <div>Head No.</div> <div>Data Carrier read address</div> <div>Bytes to read from Data Carrier</div> <div>Read data storage area type</div> <div>Read data storage word address</div> <div>Communications designation</div> <div>Processing designation</div> <div>Cancel</div> </div> <div> <div>(B00L)</div> <div>EN</div> <div>(INT)</div> <div>UnitNo</div> <div>(INT)</div> <div>HeadNo</div> <div>(W0RD)</div> <div>CarrierAddress</div> <div>(W0RD)</div> <div>ReadBytes</div> <div>(W0RD)</div> <div>StoreAreaID</div> <div>(INT)</div> <div>StoreAreaNo</div> <div>(INT)</div> <div>Communications</div> <div>(INT)</div> <div>ByteOrder</div> <div>(B00L)</div> <div>Cancel</div> </div> <div> <div>(B00L)</div> <div>ENO</div> <div>(B00L)</div> <div>BUSY</div> <div>(B00L)</div> <div>OK</div> <div>(B00L)</div> <div>NO</div> <div>(W0RD)</div> <div>ErrorCode</div> </div> <div> <div>Busy Flag</div> <div>Normal end</div> <div>Error end</div> <div>Error code (May be omitted.)</div> </div> </div>
File name	\\Lib\FBL\English\omronlib\FIDW600x_V60x200_ReadData10.cxf
Applicable models	CS1W-V600C11/V600C12 and CJ1W-V600C11/V600C12 ID Sensor Units
Basic function	Reads data from a Data Carrier.
Conditions for usage	Other <ul style="list-style-type: none"> This FB cannot be executed if the ID Sensor Unit is busy. The NG Flag will turn ON if an attempt is made.
Function description	Data is read from the specified area of the Data Carrier specified by the Unit No. and Vendor No. Up to 2048 bytes (1024 words) can be read at one time. The word designation for storing the data is specified using the area type and beginning word address. For example, for D1000, the area type is set to P_DM and the beginning word address is set to &1000.
EN input condition	Connect EN to an OR between an upwardly differentiated condition for the start trigger and the BUSY output from the FB.
Restrictions Input variables	<ul style="list-style-type: none"> Always use an upwardly differentiated condition for EN. If the input variables are out of range, the ENO Flag will turn OFF and the FB will not be processed. Always specify a head number of &1 for One-Head ID Sensor Units (CS1W-V600C11 and CJ1W-V600C11).

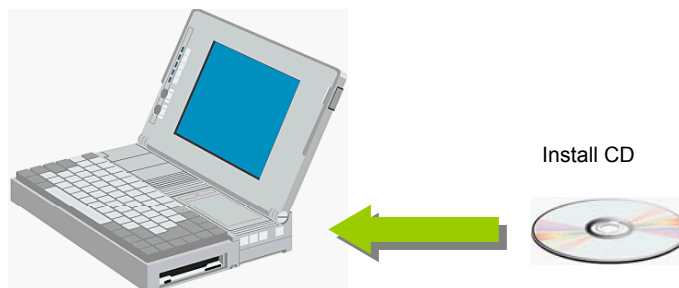
3-4. File Catalog and Where to Access the OMRON FB Library

3-4-1. Catalog of OMRON FB Library files

Type	Target components	Number of OMRON FB Part files (at the time of February '05)
FA components	Temperature controller, Smart sensor, ID sensor, Vision sensor, 2 dimensions bar code reader, Wireless terminal	approx. 80
PLC	CPU unit, Memory card, Special CPU IO unit (Ethernet, Controller Link, DeviceNet unit, Temperature control unit)	approx. 95
Motion control components	Position control unit Inverter Servo motor driver	approx. 70

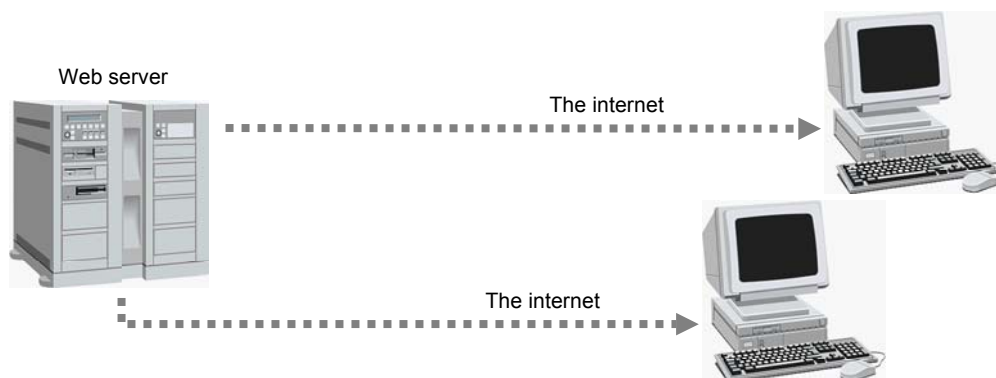
3-4-2. CX-One / CX-Programmer installation CD

OMRON FB Library is contained on the same install CD as CX-One / CX-Programmer. Installation can be selected during CX-One / CX-Programmer installation.



3-4-3. Accessing OMRON FB Library files from Web server

The latest version OMRON FB Library files are provided by Omron on the Web server. New files will be added to support new or enhanced PLC units and FA components. The download service of the OMRON FB Library is provided as a menu of Omron Web in each country.



Chapter 2

How to use the OMRON FB Library

Explanation of
target ProgramOpening a
new projectImport
FB LibraryCreating a
program

Program Check

1. Explanation of the target program

This chapter describes how to use OMRON FB Library using the OMRON FB Part file 'Make ON Time/OFF Time Clock Pulse in BCD'.

1-1. Application Specifications

The target application specifications are as follows :-

- Pulse is generated after PLC mode is changed to 'run' or 'monitor' mode.
- Output the pulse to address 1.00.
- On time of generated pulse is set at D100.
- Off time of generated pulse is 2 seconds.

1-2. Specifications of the OMRON FB Part file

The OMRON FB Part file 'Make ON Time/OFF Time Clock Pulse in BCD' has the following specifications:-

CPU 007	Make ON Time/OFF Time Clock Pulse in BCD CPU007_MakeClockPulse_BCD
Basic function	Generates a clock pulse with the specified ON time and OFF time and outputs it to ENO.
Symbol	
File name	\\Lib\FBL\English\omronlib\PLC\CPU\CPU007_MakeClockPulse_BCD10.cxl
Applicable models	CS1-H, CS1-H, and CJ1M CPU Units
Conditions for usage	<p>PLC Properties</p> <ul style="list-style-type: none"> The PV update method for timers and counters must be set to BCD in the PLC Setup. A compiling error will occur if BCD mode is not set. The mode can be set in the PLC Properties in the CX-Programmer. <p>Shared Resources</p> <ul style="list-style-type: none"> Timers
Function description	<p>ENO will be OFF for the time set in OFF time and then will be ON for the time set in ON time.</p>
EN input condition	Connect the EN input to the Always ON Flag (P_On).
Restrictions	<ul style="list-style-type: none"> If the input variables are out of range, the ENO Flag will turn OFF and the FB will not be processed. Set the ON time and OFF time input variables to between #0000 and #9999 in BCD (100 ms units). If a setting is not within range, ENO is turned OFF.
Application example	<p>In the following example, bit A will be repeatedly ON for 5 s and OFF for 3 s.</p>
Related FBs	<p>Use the correct FB for the timer/counter PV update mode set in the PLC Setup.</p> <p>Binary mode: Make ON Time/OFF Time Clock Pulse in Binary (_CPU008_MakeClockPulse_BIN)</p> <p>BCD mode: Make ON Time/OFF Time Clock Pulse in BCD (_CPU007_MakeClockPulse_BCD)</p>

Variable Tables

Input Variables

Name	Variable name	Data type	Default	Range	Description
EN	EN	BOOL			1 (ON): FB started 0 (OFF): FB not started.
ON time	OnTime	WORD		#0000 to #9999	Specify the ON time (unit: 100 ms). For example, #30 means 3 seconds.
OFF time	OffTime	WORD		#0000 to #9999	Specify the OFF time (unit: 100 ms). For example, #30 means 3 seconds.

Output Variables

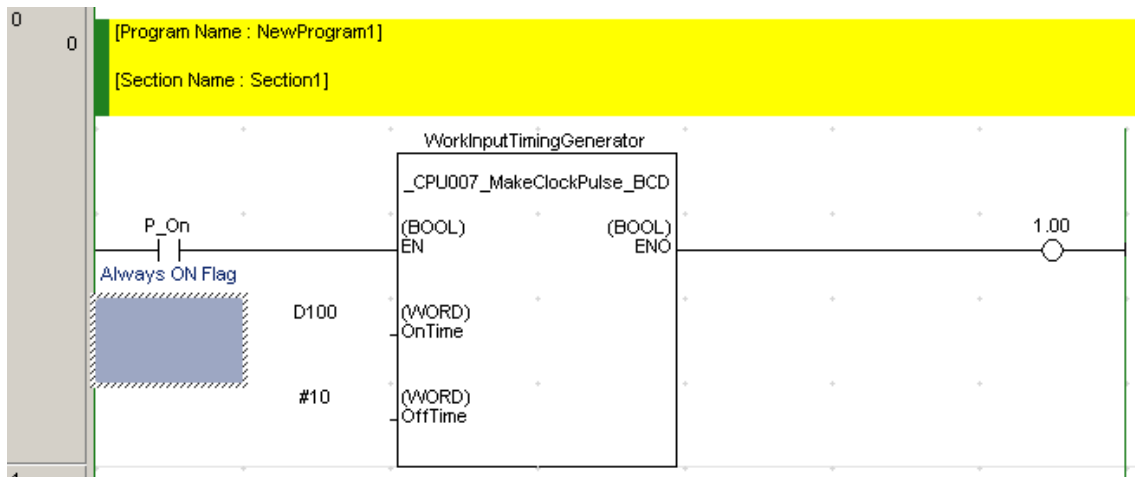
Name	Variable name	Data type	Range	Description
ENO	ENO	BOOL		Turns ON for the OnTime and OFF for the OffTime.

Explanation of
target ProgramOpening a
new projectImport
FB LibraryCreating a
program

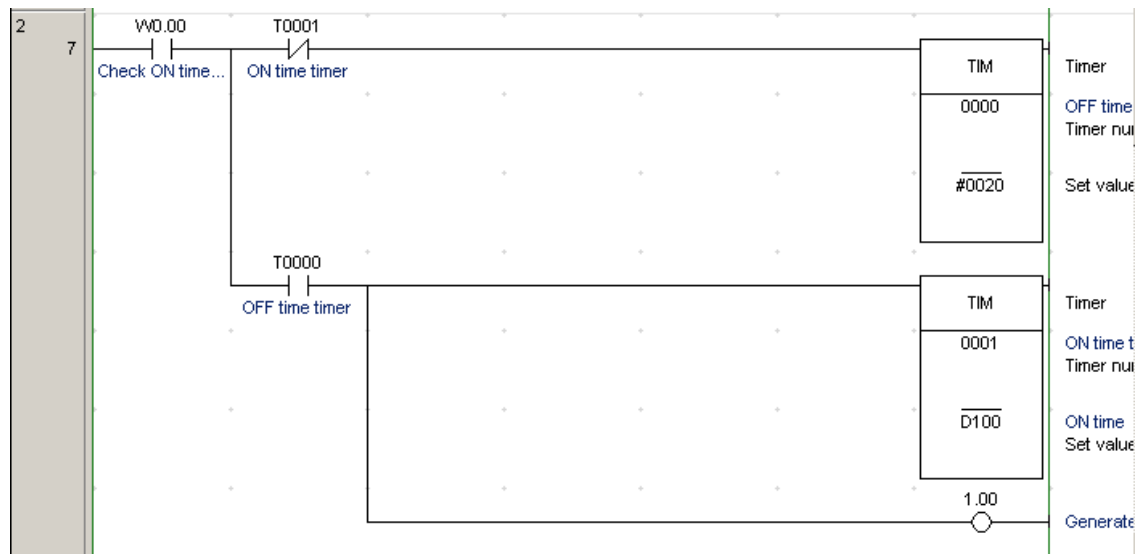
Program Check

1-3. Input program

Create the following ladder program:-



[Reference] If created as a straightforward ladder diagram, the program would be as below:-

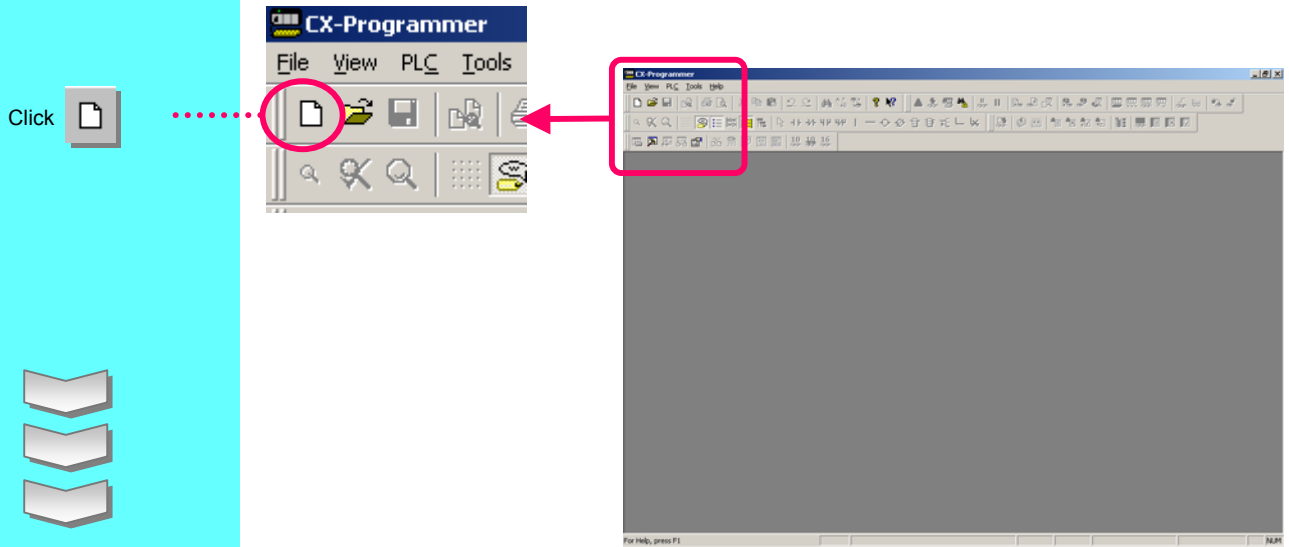


Explanation of
target ProgramOpening a
new projectImport
FB LibraryCreating a
program

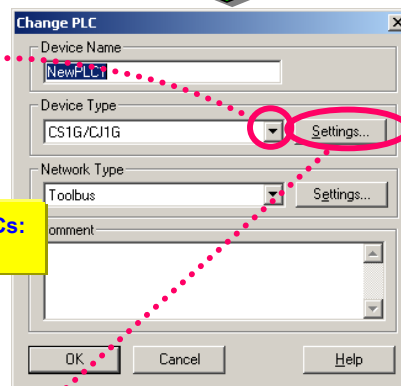
Program Check

2. Opening a new project and setting the Device Type

Click the toolbar button [New] in CX-Programmer.

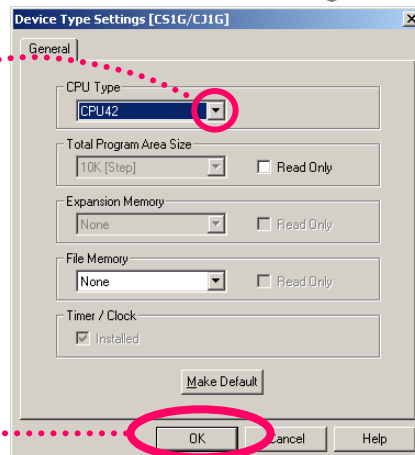


Click the left mouse button.



! To use Function Blocks, select the following PLCs:
CS1G-H, CS1H-H, CJ1G-H, CJ1H-H, CJ1M

Click the left mouse button

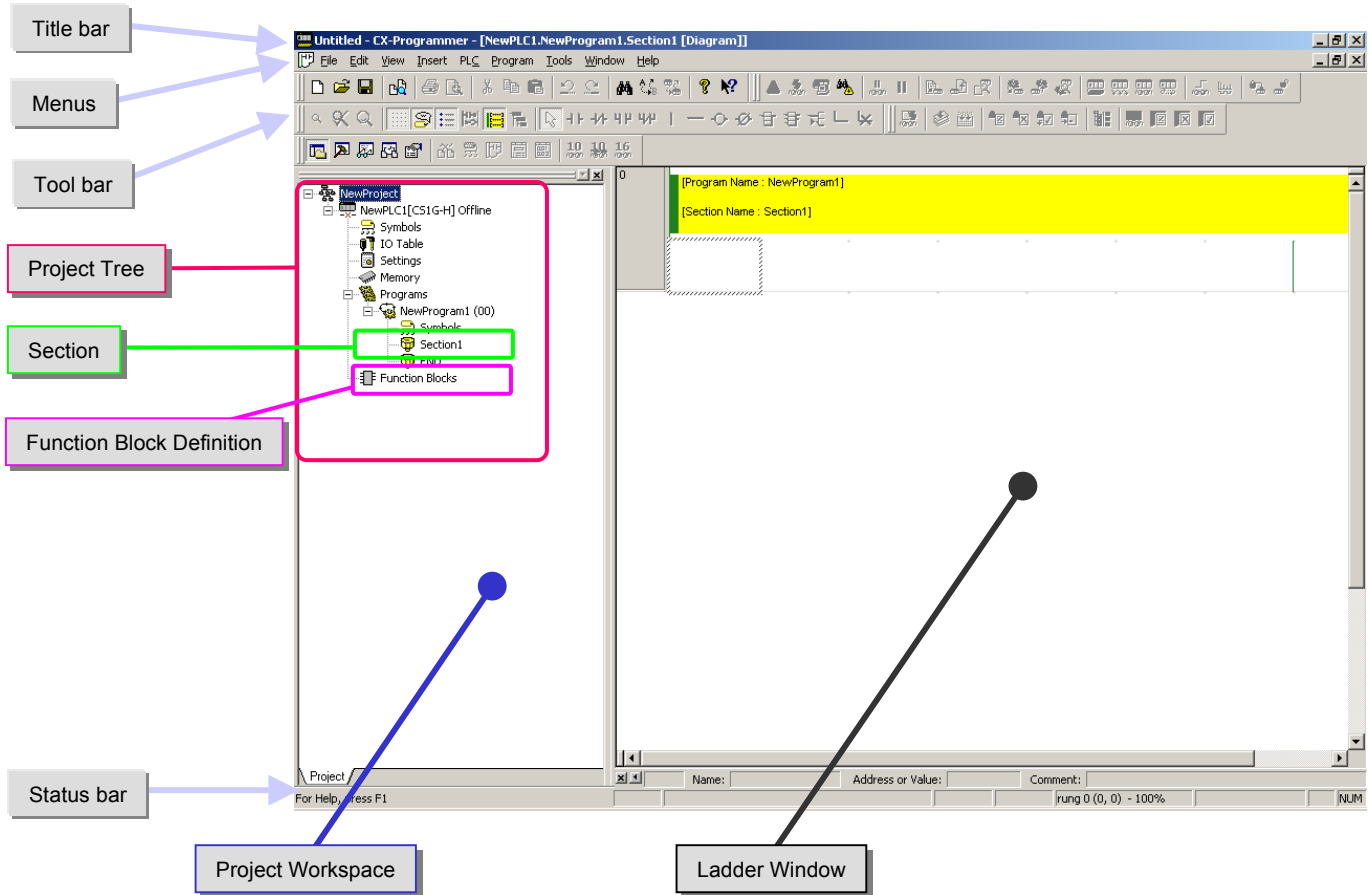





Click the left mouse button
to select CPU type.

Click [OK] to decide
the selected CPU type.

3. Main Window functions

The main window functionality is explained here.



Name	Contents / Function
Title Bar	Shows the file name of saved data created in CX-Programmer.
Menus	Enables you to select menu items.
Toolbars	Enables you to select functions by clicking icons. Select [View] -> [Toolbars], display toolbars. Dragging toolbars enables you to change the display positions.
Section	Enables you to divide a program into several blocks. Each can be created and displayed separately.
Project Workspace Project Tree	Controls programs and data. Enables you to copy element data by executing Drag and Drop between different projects or from within a project.
Ladder Window	A screen for creating and editing a ladder program.
Function Block Definition	Shows Function Block definition. By selecting the icons, you can copy or delete the selected Function Block definition. -  is shown if the file is a OMRON FB Part file. - In the case of a User-defined Function Block,  is shown if Ladder,  is shown if ST.
Status Bar	Shows information such as a PLC name, online/offline state, location of the active cell.

Explanation of
target ProgramOpening a
new projectImport
FB LibraryCreating a
program

Program Check

4. Import the OMRON FB Part file

Select Function Block definition icon from the project tree using the mouse cursor, right click. Select Insert Function Block, then select a Library file using mouse to navigate.

Click mouse right button
→ Insert Function Block
→ Library File

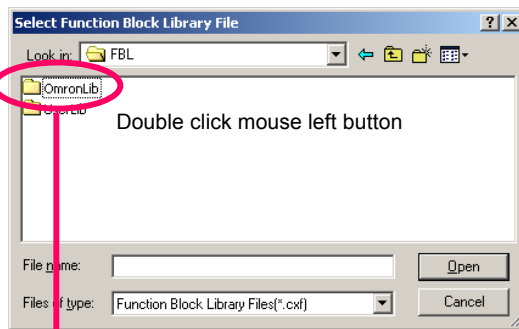
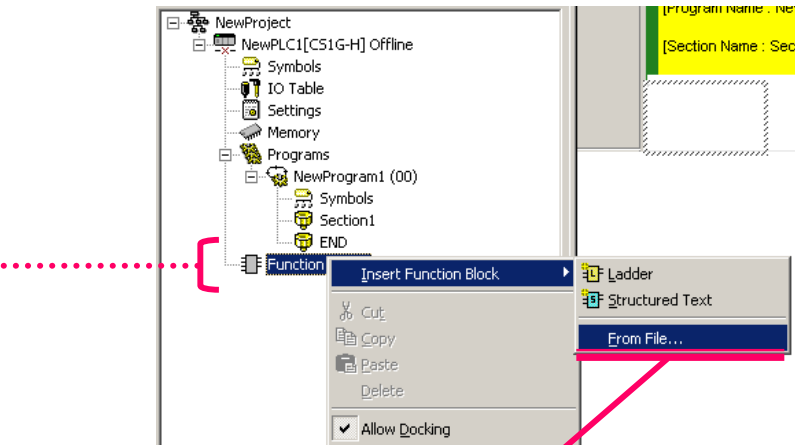


Double click mouse left button.
→ [OmronLib]
→ [Programmable Controller]
→ [CPU]
Select each of the above in
series.



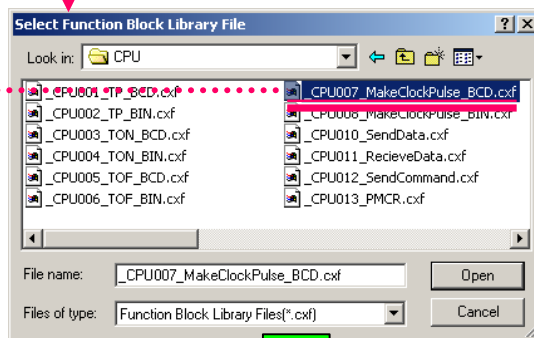
Left Click
'_CPU007_MakeClockPulse_BCD.cxf'

Left Click the [Open] button



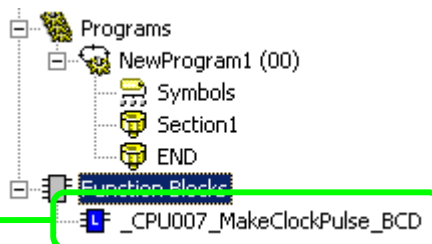
Select the necessary OMRON FB
Part file in the 'Select Function
Block Library' dialog.

! The default path of the OMRON FB Library is
C:\Program Files\Omron\CX-One\Lib\FBL.

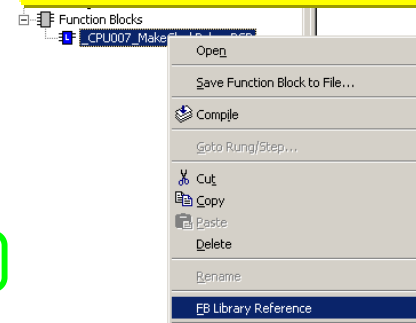


! You can easily check specifications of
OMRON FB part files by selecting
registered OMRON FB part files and [FB
Library Reference] from a pop-up menu
and showing a library reference file.

Function Block definition '_CPU007_MakeClockPulse_BCD'
is registered as part of the project file.



Function Block Definition

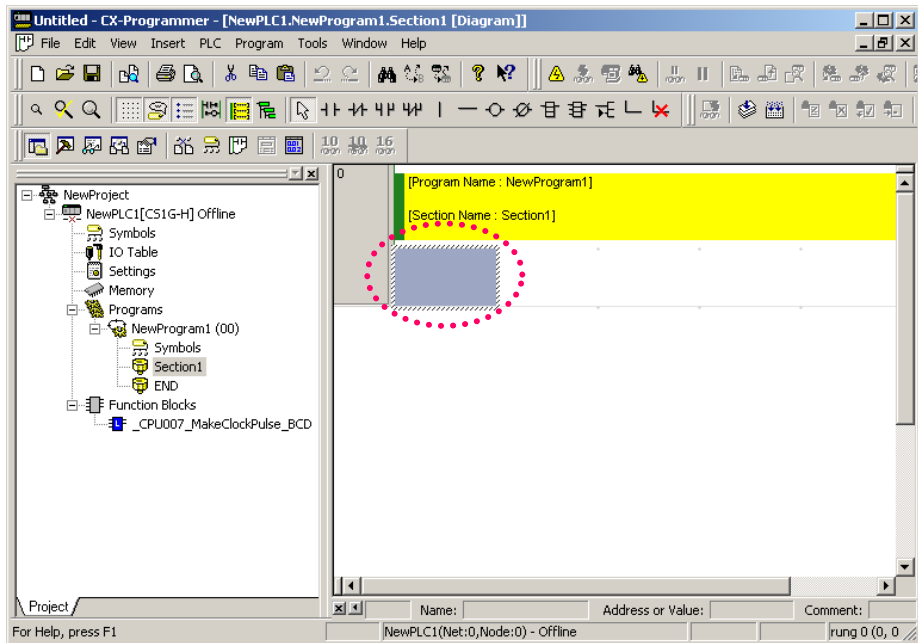


Explanation of
target ProgramOpening a
new projectImport
FB LibraryCreating a
program

Program Check

5. Program Creation

Confirm cursor position is at the upper left of Ladder Window to start programming.



5-1. Enter a Normally Open Contact

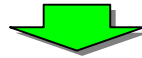
C

..... Press the [C] key on the keyboard to open the [New Contact] dialog.
Use the dropdownbox to select the "P_On" symbol.

P_On



ENT



Deleting commands

- Move the cursor to the command and then press the DEL key or
- Move the cursor to the right cell of the command and press the BS key.

"P_On" is a system defined symbol. Its state is always ON.

0 of the upper digit of an address is omitted when shown.

[.] (period) is displayed between a channel number and a relay number.

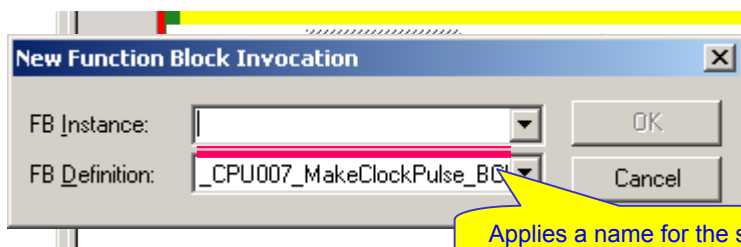
Explanation of
target ProgramOpening a
new projectImport
FB LibraryCreating a
program

Program Check

5-2. Entering an Instance

F

..... Press the [F] key on the keyboard to open the [New Function Block Invocation] dialog.

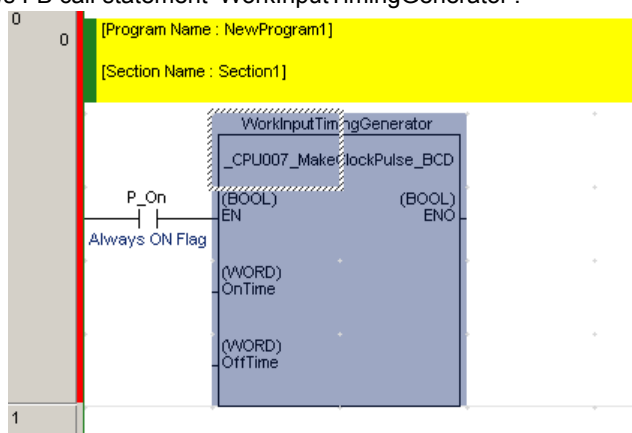


Applies a name for the specific process in the diagram.

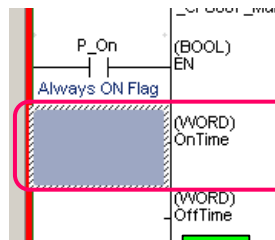
Enter text to create an FB instance name.
[WorkInputTimingGenerator]

ENT

..... Shows FB call statement 'WorkInputTimingGenerator'.



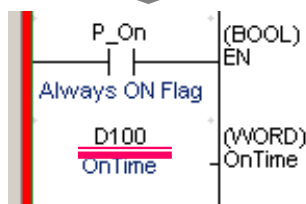
5-3. Entering Parameters



Move the cursor to the left of input parameter.

Enter the address.

[d100]

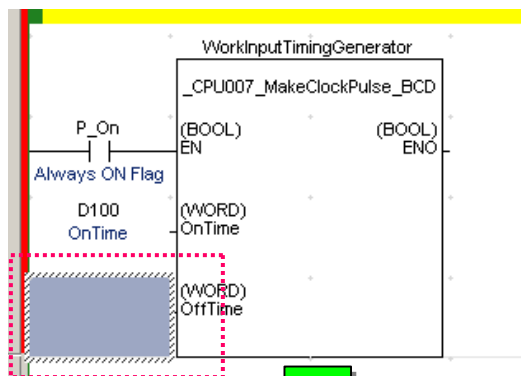


Choose an address for the input parameter 'OnTime'.

Explanation of
target ProgramOpening a
new projectImport
FB LibraryCreating a
program

Program Check

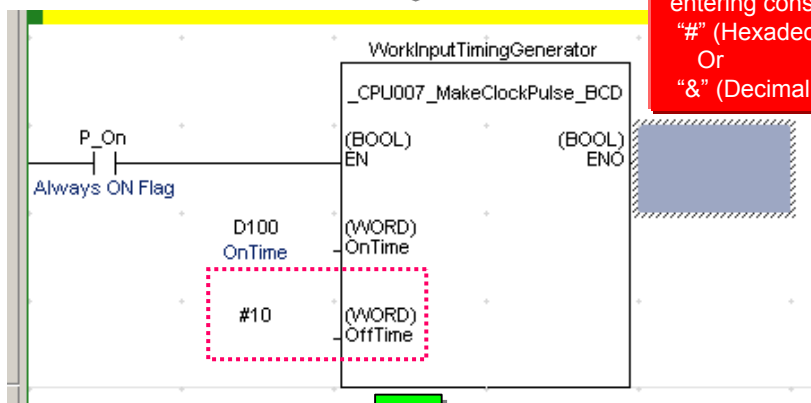
Enter the remaining parameters in the same way.



New Parameter

#10

Detail >> OK Cancel



Please add the following prefix for
entering constants as parameters:
"#" (Hexadecimal/BCD)
Or
"&" (Decimal)

-()- New Coil

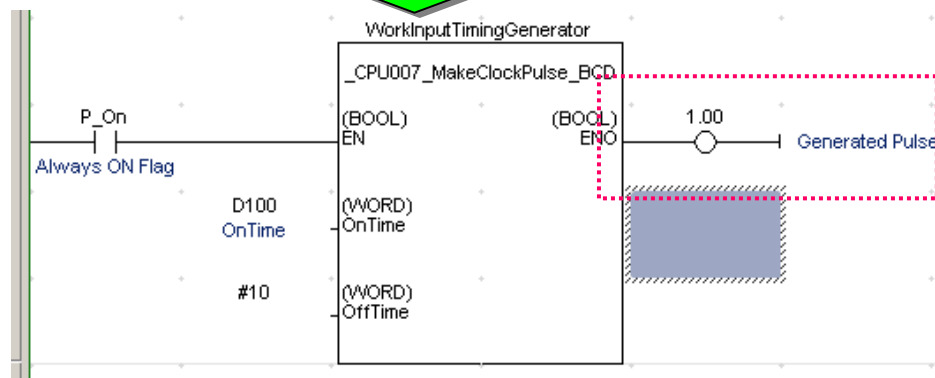
1.00

Detail >> OK Cancel

-()- New Edit Comment (1/1) : 1.00

1.00 Generated pulse

OK Cancel



P Or ENT

#10

ENT

O

1.00

ENT

[Generated Pulse]

ENT

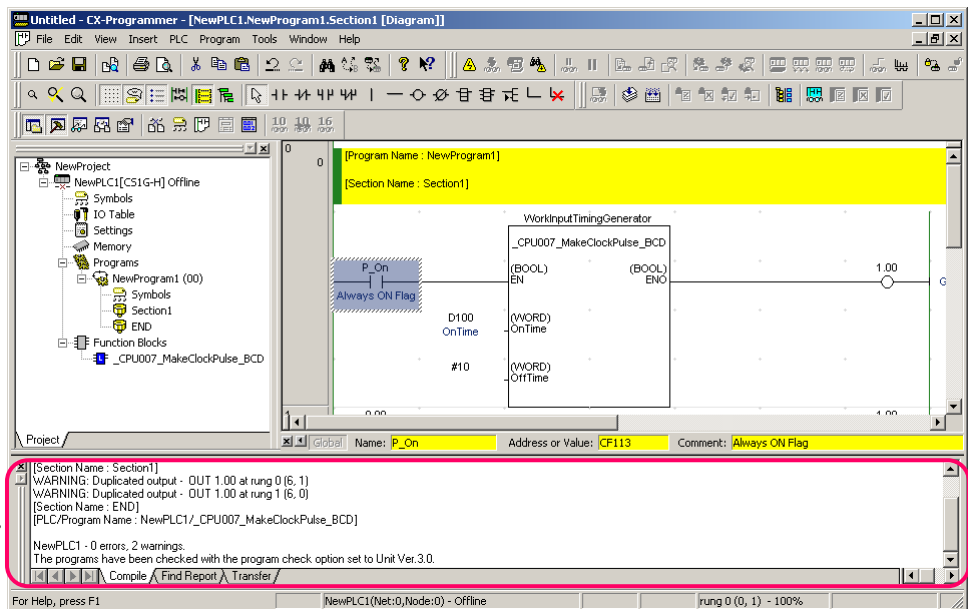
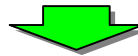
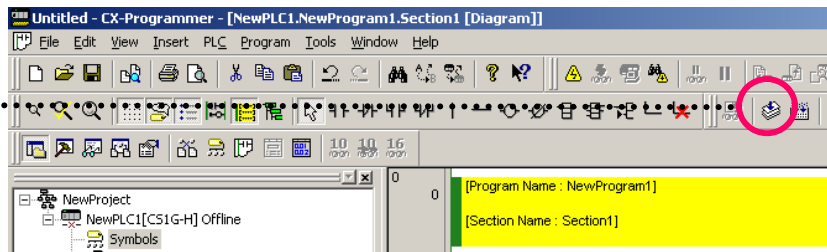
Explanation of
target ProgramOpening a
new projectImport
FB LibraryCreating a
program

Program Check

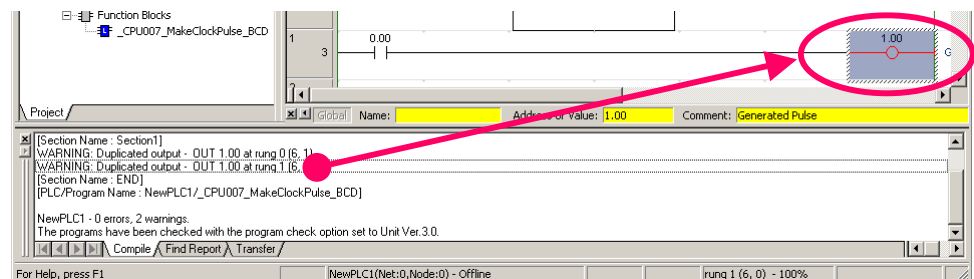
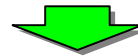
6. Program Error Check (Compile)

Before program transfer, check for errors using the program compile.

Click



Errors and addresses are
displayed in the Output Window.



Double-click on displayed errors,
and the Ladder Diagram cursor
will move to the corresponding
error location, displaying the error
rung in red.

Modify the error.

- Output Window automatically opens at program check.
- The cursor moves to an error location by pressing J or F4 key.
- Output Window closes by pressing the ESC key.

Online
to transfer

Monitoring

Online
Edit

7. Going Online

CX-Programmer provides three methods of connecting, depending on usage.



Normal online. Enables you to go online with a PLC of the device type and method specified when opening a project.



Auto online. Automatically recognizes the connected PLC and enables you to go online with a PLC with one button.
→ Uploads all data, such as programs, from the PLC.



Online with Simulator. Enables you to go online with CX-Simulator with one button (CX-Simulator must be installed.)

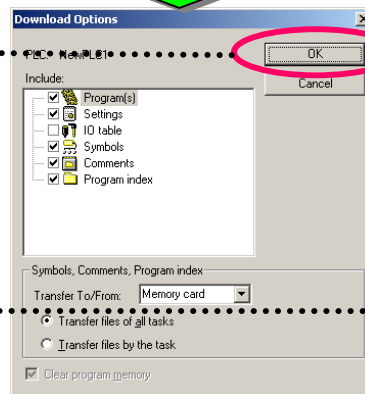
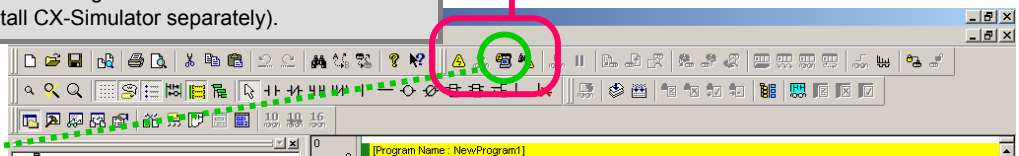
Online/debug functions when working online with CX-Simulator are explained in this guide (Install CX-Simulator separately).

Click

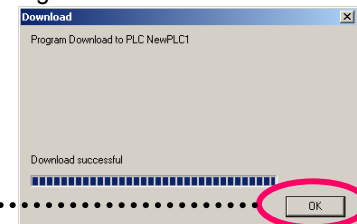


Click [OK]

Click [OK]



Program transfer starts.

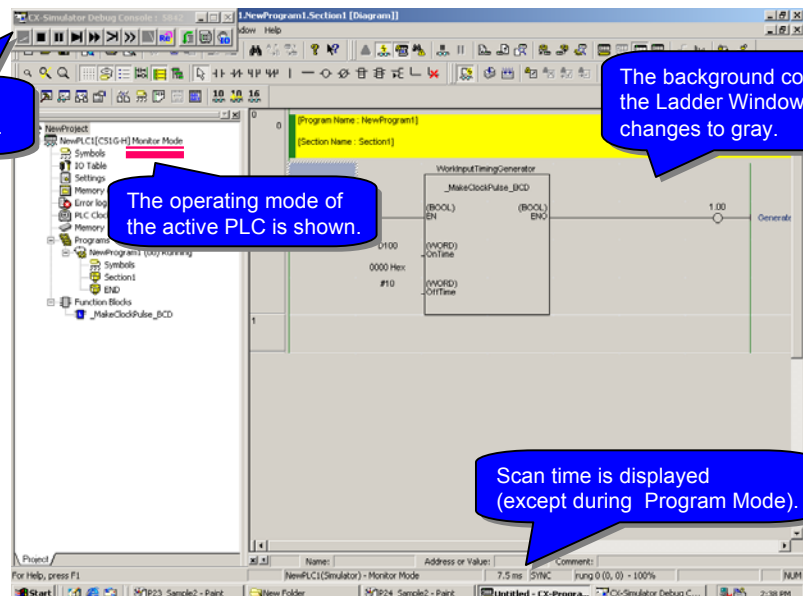


The CX-Simulator Console box is shown.

The operating mode of the active PLC is shown.

The background color of the Ladder Window changes to gray.

Scan time is displayed (except during Program Mode).



Online
to transfer

Monitoring

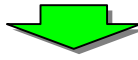
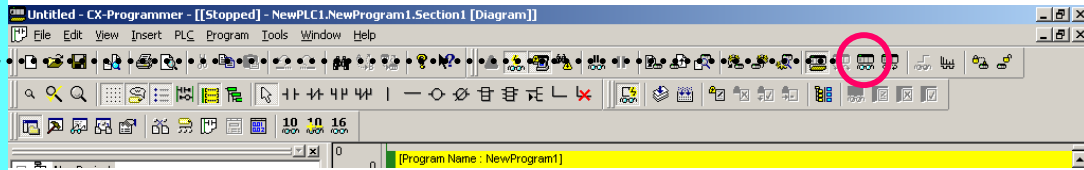
Online
Edit

8. Monitoring - 1

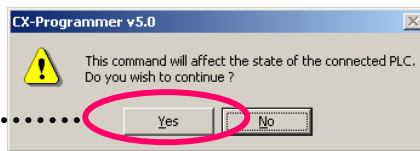
Change the PLC (Simulator)
to Monitor mode.

The on/off status of contacts and coils can be monitored.

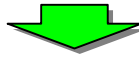
Click



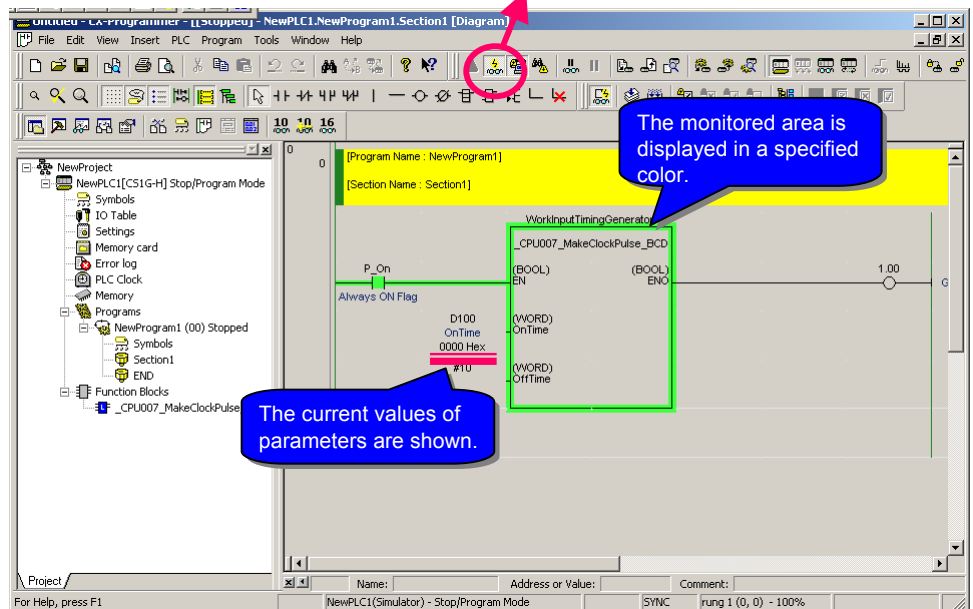
Click [Yes].



If your program has a large volume of data, the scroll speed of the screen may become slow when monitoring.
To resolve this, click the icon below to cancel monitoring, scroll to the address you want to monitor, then restart the monitor mode.



... toggles PLC monitoring on/off



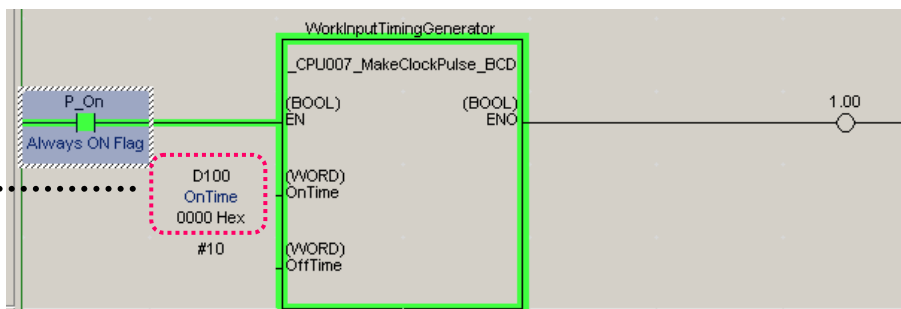
Online
to transfer

Monitoring

Online
Edit

9. Monitoring - 2 Change Parameter Current Value

Change the current value of contact/coils or word data in the Ladder Window.



Move the cursor to the input parameter 'D100'.

Click mouse right button and select the menu item [Set/Reset(S)]
→ [Setting Value (V)]

Or

Double click mouse left button.

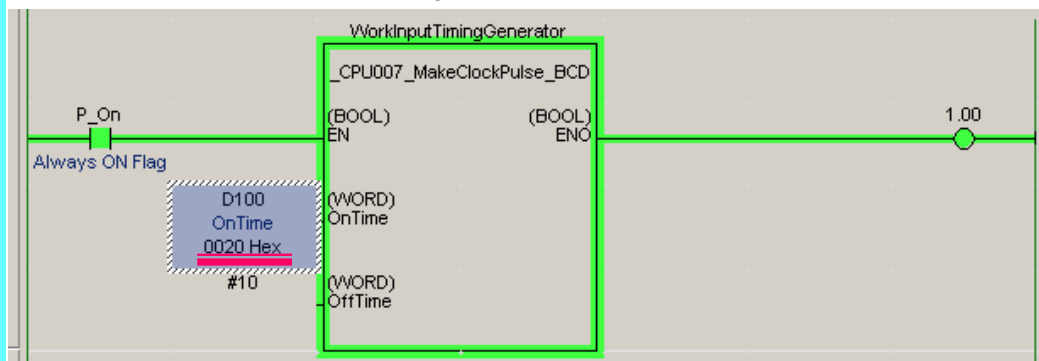
Or

ENT

Change the current value of Input parameter.

Click [Set]

Please add the following prefix for entering constants as parameters:
"#" (Hexadecimal/BCD)
Or
"&" (Decimal)

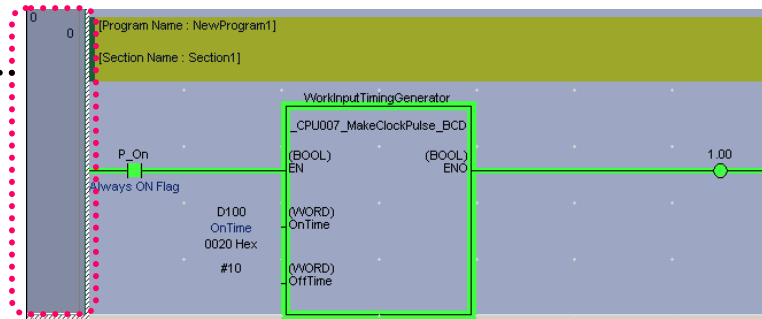


Online
to transfer

Monitoring

Online
Edit

10. Online Editing



Move the cursor to the rung requiring modification.

You can also select multiple rungs by using the Drag & Drop facility with the mouse.

Select [Program] → [Online Edit] → [Begin]

Shortcut: [Ctrl]+[E]

Move the cursor to the coil you want to modify. Double click the left mouse button.

Edit the address to the required bit number (4.11 in the example)

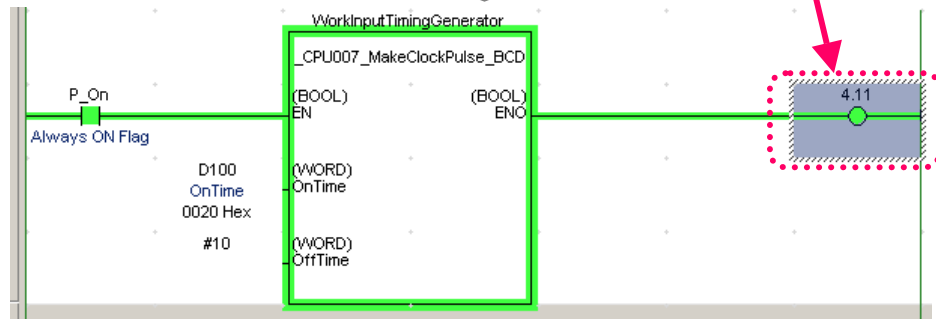
-()- Edit Coil

4.11

Detail > OK Cancel

Select [Program] → [Online Edit] → [Send Change]

Shortcut: [Ctrl]+[Shift]+[E]



End

Chapter 3

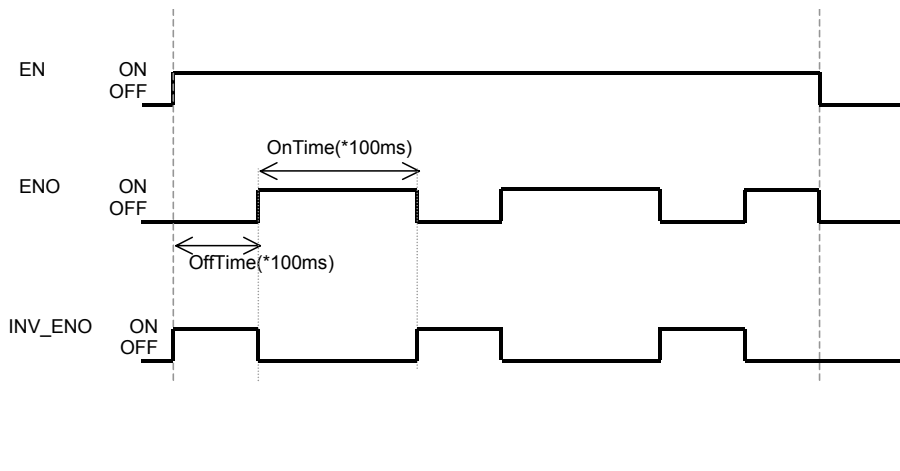
***Customize
the OMRON FB Part file***

1. Explanation of target program

This chapter describes how to customize the OMRON FB Library using the OMRON FB Part file 'Make ON Time/OFF Time Clock Pulse in BCD'.

1-1. Changing File Specifications

The OMRON FB Part file 'Make ON Time/OFF Time Clock Pulse in BCD' is designed to repeatedly turn off the ENO for the specified OffTime (unit: 100 msec) and on for the specified OnTime (unit: 100 msec). In this example, the OMRON FB Part file will be changed to output an invert signal by adding the output parameter 'INV_ENO'.



1-2. Changing the contents of the OMRON FB Part file

To satisfy the requirement described above, the following changes must be made to OMRON FB Part file 'Make ON Time/OFF Time Clock Pulse in BCD'

1. Add an output parameter 'INV_ENO'.
2. Add ladder program to output the ENO for inverting the signal.

Caution

OMRON cannot guarantee the operation of a customized OMRON FB parts. Please be sure to check the process of your FB part sufficiently before customization and confirm the operation of each FB parts thoroughly after that.

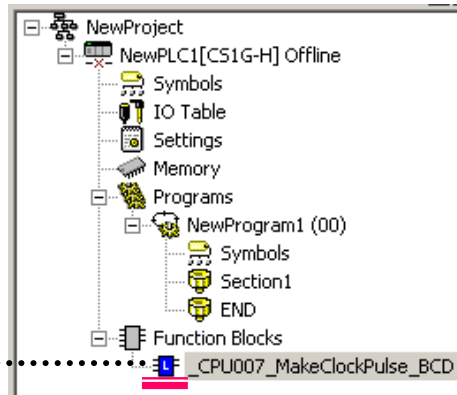
Explanation of
target Program

Copy of FB part

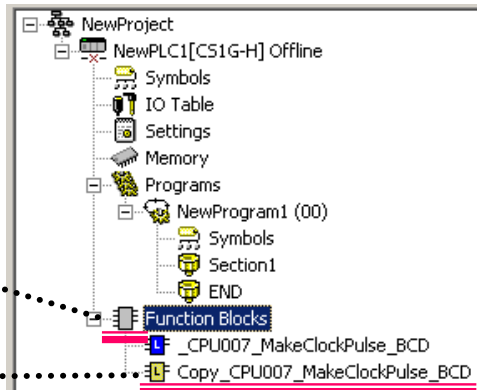
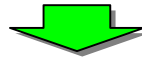
Change of
FB Definition

2. Copy the OMRON FB Part file

Import the 'Make ON Time/OFF Time Clock Pulse in BCD' Function Block Part file as explained in Chapter 1 (FB definition name: `_CPU007_MakeClockPulse_BCD`)

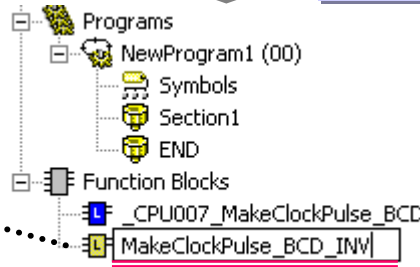
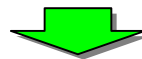


Select the OMRON FB Part icon
then right click the mouse.
→ Copy



Select Function Block Definition
icon and right click the
mouse.
→ Paste

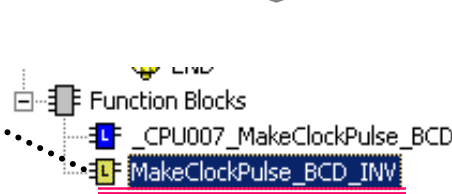
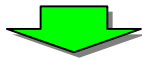
The OMRON FB Part file is
pasted.



Change the FB definition name.

Select pasted Function Block
icon and click mouse right
button.
→ Rename
[MakeClockPulse_BCD_INV]

Note:
The user can't create Function Block Definitions
With name starting '_' (underscore).
Please use names not starting with '_'.

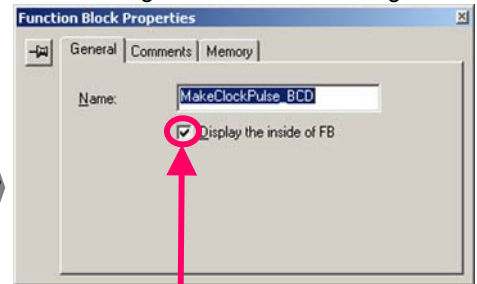


Select pasted Function Block
icon and right click the mouse
button.
→ Property

Or

ALT + ENT

Enable editing of the internal FB Program code.



Tick the check box using the left mouse click.

Explanation of
target Program

Copy of FB part


Change of
FB Definition

3. Add a variable to the Function Block

Variable Table

Open the Function Block Ladder Editor.

Opens the Function
Block Ladder Editor.

Select the Function Block icon  using the mouse cursor and double click the left mouse button.

The original OMRON FB Part file is also able to display its ladder program, but cannot be edited.

Ladder Editor

Variable table

Name	Data Type	AT	Initial Value	Retain...	Comment
ENO	BOOL		FALSE		Indicates successful execution ...

Select Output tab in Variable Table using the mouse cursor. And click the left mouse button.

Click the left mouse button and select Insert Variable(I).

Enter a new variable name.

Select BOOL for bit data.

New Variable

Name: INV_ENO

Data Type: BOOL

Usage: Output

Initial Value: FALSE ☐ Retain

Comment: Inverting output of ENO

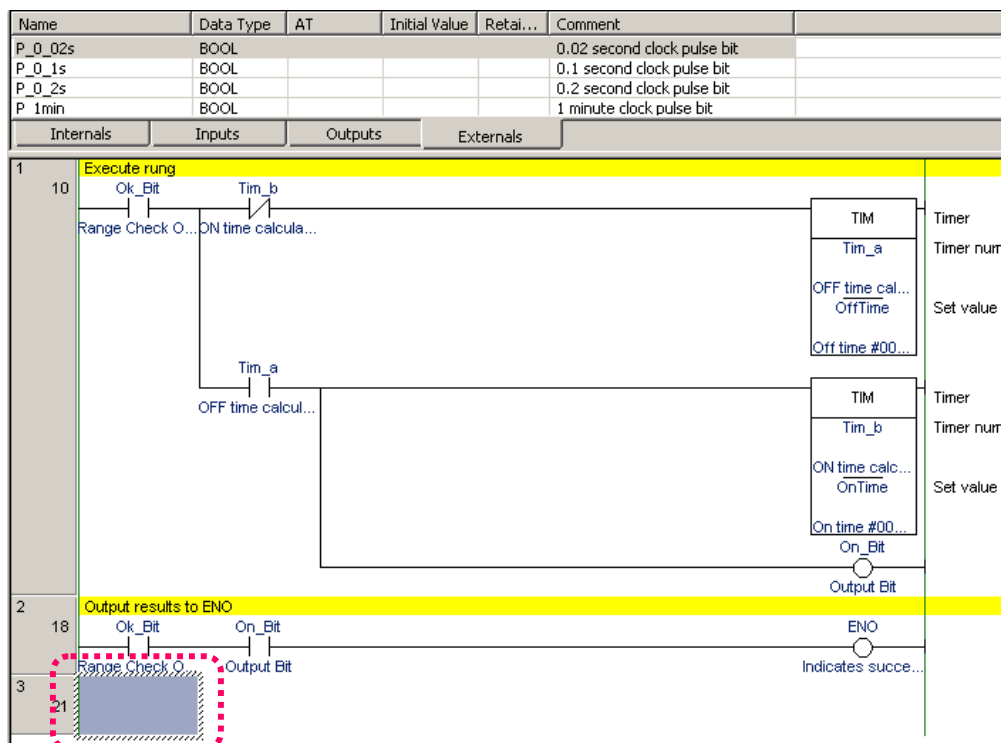
OK Cancel Advanced...

Confirm the entered variable is correct.

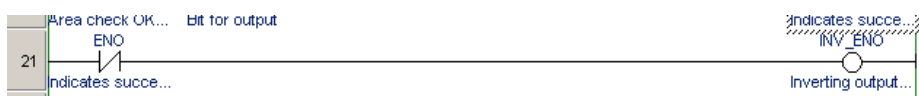
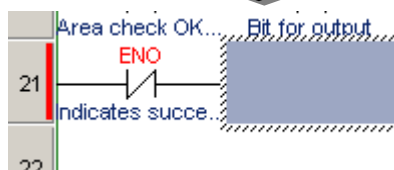
Name	Data Type	AT	Initial Value	Retain...	Comment
ENO	BOOL		FALSE		Indicates successful execution ...
INV_ENO	BOOL		FALSE		Inverting output of ENO

4. Changing the Function Block Ladder

Add the required ladder diagram on Function Block Ladder edit field.
Move the cursor to the left column of the next rung.

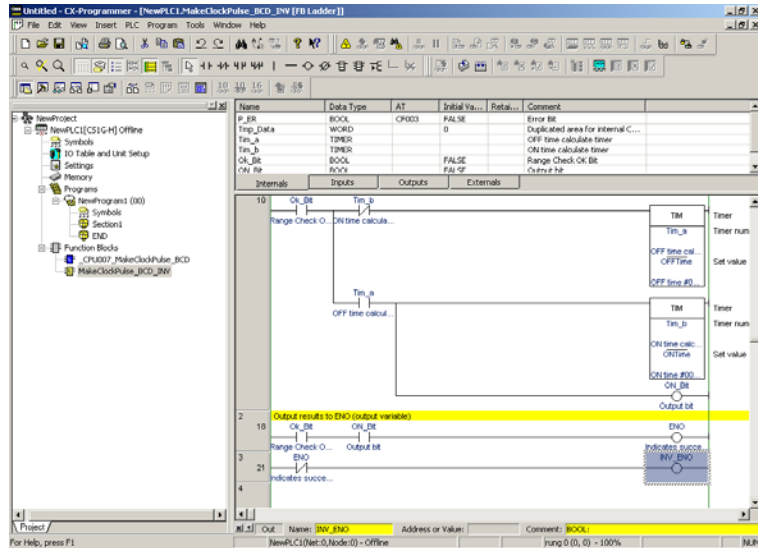


4-1. Entering a Contact



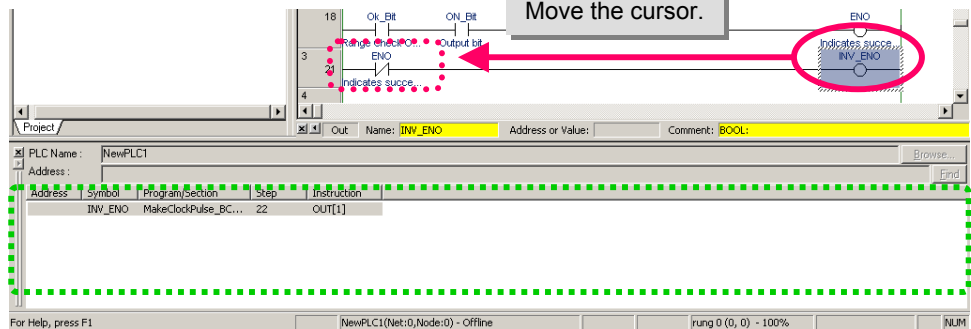
4-2. Checking Usage Status of Variables

As with main ladder program, you can use cross reference pop-up to check usage conditions of variables.



Display cross reference
pop-up.

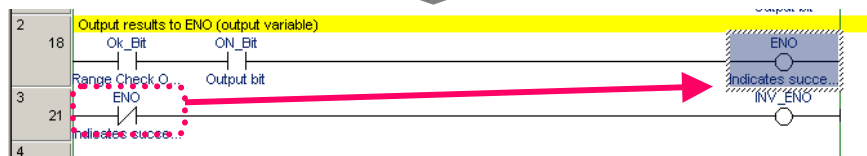
Alt + 4



Select LDNOT from cross
reference pop-up by the
mouse cursor.

PLC Name : NewPLC1					
Address :					
Address	Symbol	Program/Section	Step	Instruction	
ENO	MakeClockPulse_BC...	20	OUT[1]		
ENO	MakeClockPulse_BC...	21	LDNOT[1]		

You can see that variable ENO is used in an output coil in the step No.20 as well.



The cursor in the FB Ladder Editor moves to the output coil in the step No.20.

Chapter 4

**How to use
the ST(Structured Text)language**

Explanation of
target ProgramCreate new
FB DefinitionEntering
VariablesCreating
ST ProgramCreating Ladder
Program and check

1. What is the ST Language?

The ST (Structured Text) language is a high-level language code for industrial controls (mainly PLCs) defined by the IEC 61131-3 standard.

It has many control statements, including IF-THEN-ELSE-END_IF, FOR / WHILE loop, and many mathematical functions such as SIN / LOG. it is suitable for mathematical processing.

The ST language supported by CX-Programmer is in conformance with IEC 61131-3 standard.

The arithmetic functions in CX-Programmer Ver.5/6 are as follows:

sine (SIN), cosine (COS), tangent (TAN), arc-sine (ASIN), arc-cosine (ACOS), arc-tangent (ATAN), square root (SQRT), absolute value (ABS), logarithm (LOG), natural-logarithm (LN), natural-exponential (EXP), exponentiation (EXPT)

```

(* Initial Settings *)
XMT[1] := 2;
XMT[2] := 7;
N := 2;

(* CRC16 *)
CRCTMP := 16#FFFF;
FOR I = 1 TO N DO
  CRCTMP := CRCTMP XOR XMT[I];
  FOR J = 1 TO 8 DO
    CT := CRCTMP AND 1;
    IF CRCTMP < 0 THEN
      CH := 1;
      CRCTMP := CRCTMP AND 16#7FFF; (* CRCTMP & 0x7FFF *)
    ELSE
      CH := 0;
    END_IF;
    UINT_CRCTMP := WORD_TO_UINT(CRCTMP) / 2;
    CRCTMP := UINT_TO_WORD(UINT_CRCTMP);
    IF CH = 1 THEN
      CRCTMP := CRCTMP OR 16#4000; (* CRCTMP OR 0x4000 *)
    END_IF;
    IF CT = 1 THEN
      CRCTMP := CRCTMP XOR 16#A001; (* CRCTMP XOR 0xA001 *)
    END_IF;
  END_FOR;
END_FOR;

IF CRCTMP < 0 THEN
  CL := 1;
  CRCTMP := CRCTMP AND 16#7FFF; (* CRCTMP & 0x7FFF *)
ELSE
  CL := 0;
END_IF;

C_1 := CRCTMP AND 16#FF; (* CRCTMP & 0xFF *)
CRCTMP := CRCTMP AND 16#7F00; (* CRCTMP & 0x7F00 *)
UINT_CRCTMP := WORD_TO_UINT(CRCTMP) / 256;
C_2 := UINT_TO_WORD(UINT_CRCTMP);

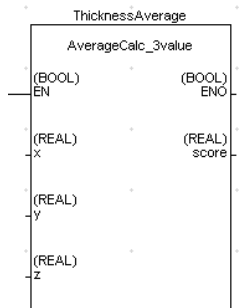
```

Reference: The IEC 61131 standard is an international standard for programming Programmable Logic Controllers (PLC), defined by the International Electro-technical Commission (IEC).

The standard consists of 7 parts, with part 3 defining the programming of PLCs.

2. Explanation of the target program

This example describes how to create an ST program in a Function Block to calculate the average value of a measured thickness.



The data type should be set to REAL to store the data.
REAL type allows values with 32 bits of length, see range below:-
 $-3.402823 \times 10^{38} \sim -1.175494 \times 10^{-38}$, 0,
 $+1.175494 \times 10^{-38} \sim +3.402823 \times 10^{38}$

FB definition name	AverageCalc_3Value
Input symbols	X (REAL type), Y (REAL type), Z (REAL type)
Output symbol	score (REAL type)
ST Program definition	score := (x + y + z) / 3.0;

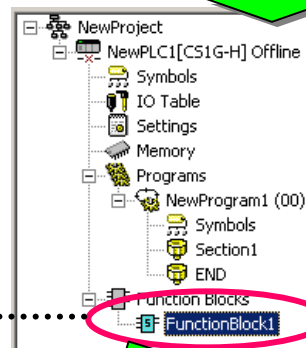
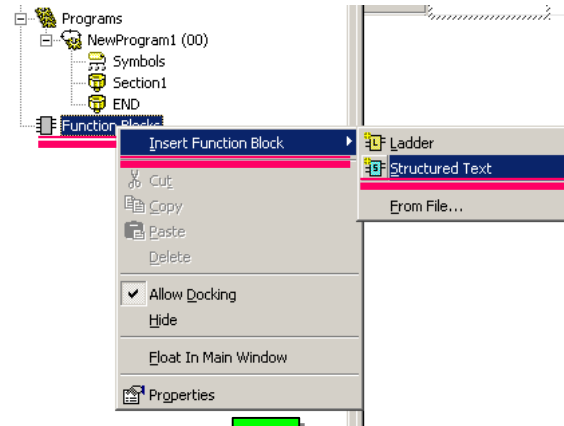
Substitute a value to a symbol is expressed by " := ".

Enter " ; " (semicolon) to complete the code.

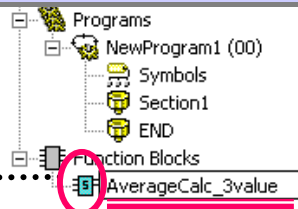
Explanation of
target ProgramCreate new
FB DefinitionEntering
VariablesCreating
ST ProgramCreating Ladder
Program and check

3. Create a Function Block using ST

Create a Function Block using
Structured Text.

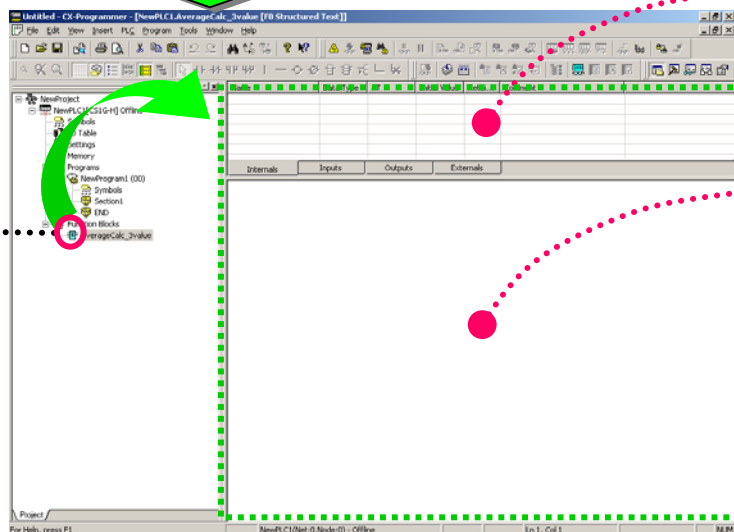


Change the Function Block definition name



Note:
The user can't create Function Block Definitions
with names starting with '_' (underscore).
Please use names not starting with '_'.

Variable Table



ST Edit Field

Select the Function Block icon
using a mouse cursor, and
click the right mouse button.
→ Insert Function Block(I)
→ Structured Text(S)

A New Function Block
definition is created.

Select the Function Block
definition icon using
the mouse cursor and right
click the mouse button.
Select Paste.
→ Rename
Enter
[AverageCalc_3value]

Open Function Block ST Editor

Select Function Block definition
icon by mouse cursor and
double click the left mouse
button.

Explanation of
target ProgramCreate new
FB DefinitionEntering
VariablesCreating
ST ProgramCreating Ladder
Program and check

4. Entering Variables into Function Blocks

Select Variable Table.

Name	Data Type	AT	Initial Value	Retain...	Comment
EN	BOOL		FALSE		Controls execution of the Func...

Select the Input tab using
the mouse cursor.Select Insert from the
Pop-up menu.Enter data for the following.
Name
Data type
Comment

New Variable

Name: x

Data Type: REAL

Usage: Input

Initial Value: 0 ☐ Retain

Comment: Input value

OK Cancel Advanced...

Enter a variable name

Select REAL

Enter and applicable
comment

Enter input symbol x, output symbols y,z by repeating the process above.

Name	Data Type	AT	Initial Value	Retain...	Comment
EN	BOOL		FALSE		Controls execution of the Func...
x	REAL		0.0		Input value 1
y	REAL		0.0		Input value 2
z	REAL		0.0		Input value 3

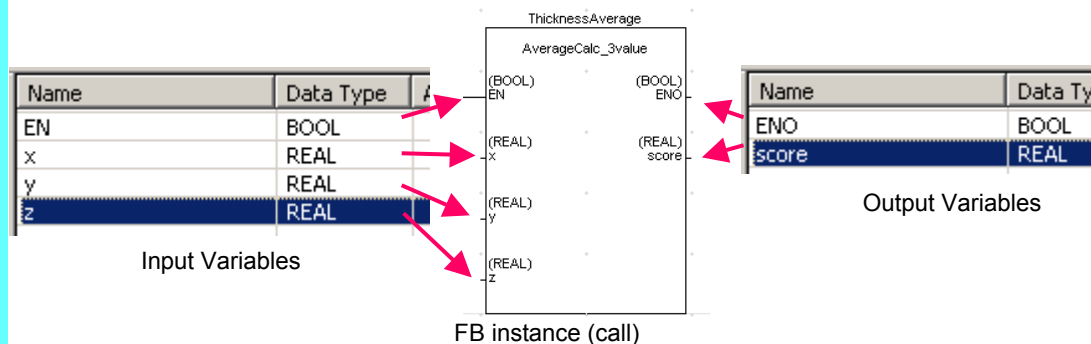
Input Variables

Name	Data Type	AT	Initial Value	Retain...	Comment
ENO	BOOL		FALSE		Indicates successful execution ...
score	REAL		0.0		Average

Output Variables

Reference: The copy and paste operation is available in FB Header.

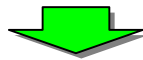
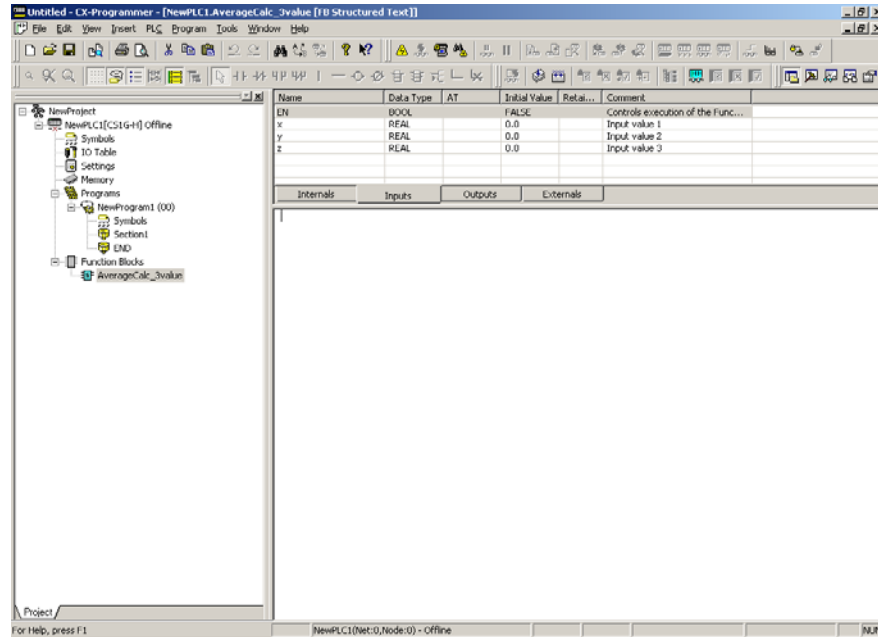
Reference: The order of the variables in the FB table becomes the order of parameters on FB instance (call statement) in the normal ladder view.
To change the order, it is possible to drag & drop variables within the table.



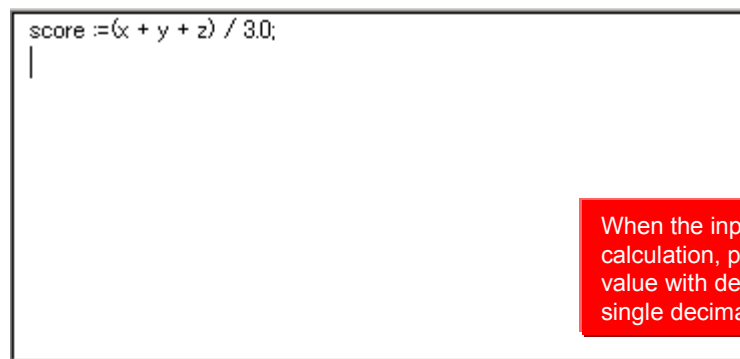
Explanation of
target ProgramCreate new
FB DefinitionEntering
VariablesCreating
ST ProgramCreating Ladder
Program and check

5. Entry of ST program

Select the ST Editor text field in the Function Block ST Editor window.

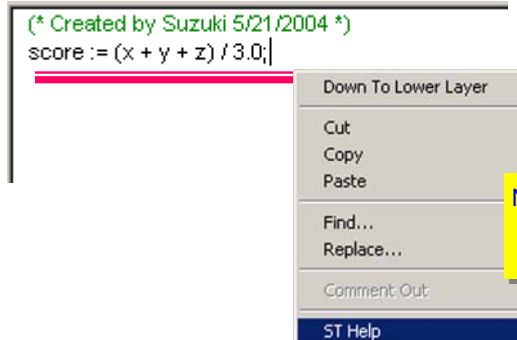


Enter text into the field: "score := (x + y + z) / 3.0;"



When the input expression is a real type calculation, please enter the constant value with decimal point and zero for single decimal places, e.g. '3.0'.

Reference: User may type Comments in the ST program.
Enter '(' and ')' both ends of comment strings, see below.
This is useful for recording change history, process expressions, etc.



Note: You can jump to a help topic that shows ST control syntax by selecting [ST Help] from a pop-up menu in the ST Editor.

Explanation of
target ProgramCreate new
FB DefinitionEntering
VariablesCreating
ST ProgramCreating Ladder
Program and check

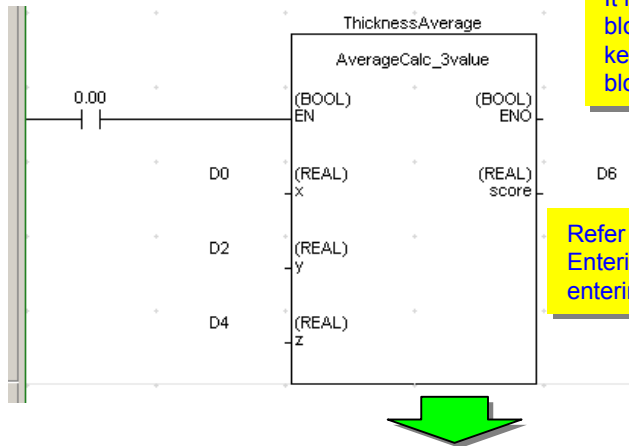
6. Entering the FB to the Ladder Program and error checking

Enter the following FB into the ladder program.

Instance name: ThicknessAverage

Input parameters: D0, D2, D4

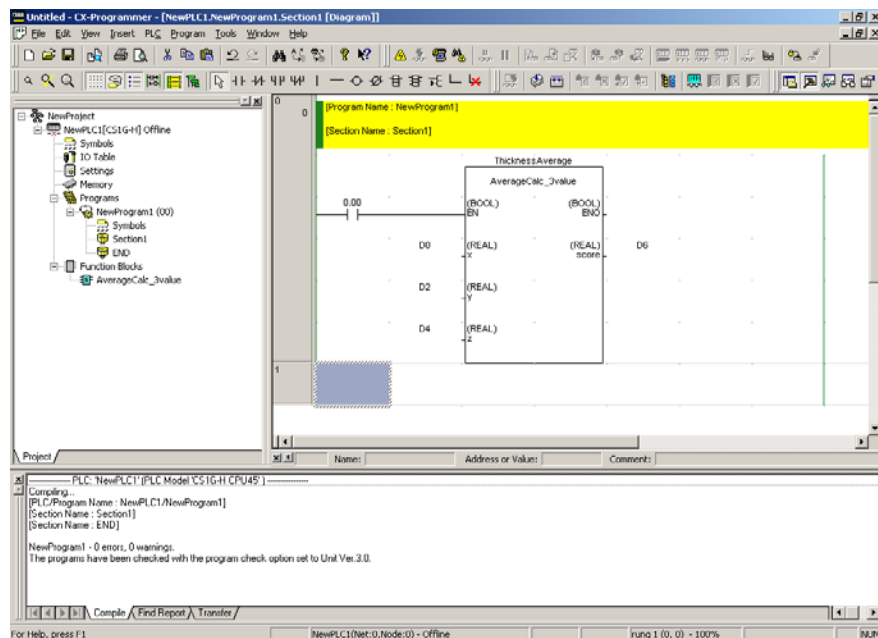
Output parameter: D6



It is able to jump the referred function block definition by entering [Shift]+[F] key when the cursor is in the function block instance.

Refer page 2-7 for entering FB instances. Entering ST FB instances is the same as entering FB Ladder instances.

Perform a programs check before transferring the program.



Refer page 2-9 for program checking. The functionality is the same as for Function Block Ladder instances.

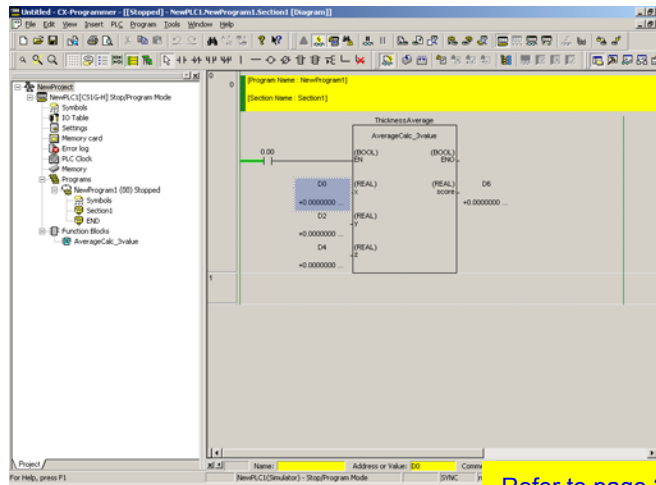
It is possible to change or add variables in the Function Block after inputting FB instance into the ladder editor. If modified, the Ladder editor changes the color of the left bus-bar of the rung containing the changed Function Block. When this occurs, please select the instance in the Ladder Editor using the mouse cursor, and select Update Function Block Instance (U) from the pop-up menu.

Transfer Program

Monitoring

7. Program Transfer

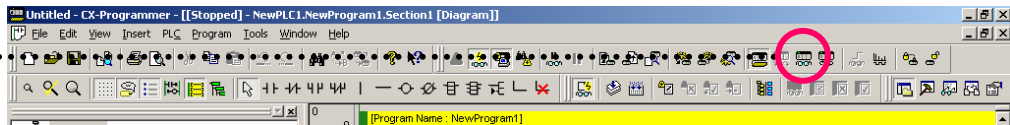
Go online to the PLC with CX-Simulator and transfer the program.



Change the PLC (Simulator) to Monitor mode.

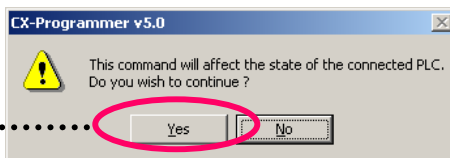
Refer to page 2-10 for steps to go online and transfer the program.

Click

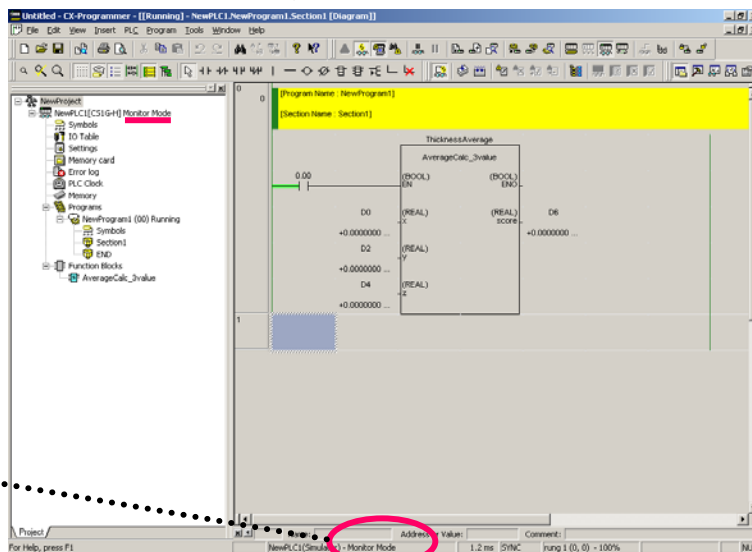


The on/off status of contacts and coils can be monitored.

Click [Yes]



Confirm that the PLC is Monitor mode.



Transfer Program

Monitoring

8. Monitoring the Function Block execution

Monitors the present value of parameters in the FB instance using the Watch Window.


Display the Watch Window.

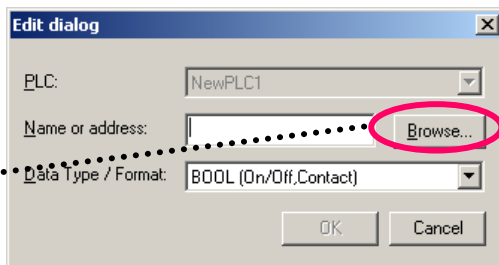
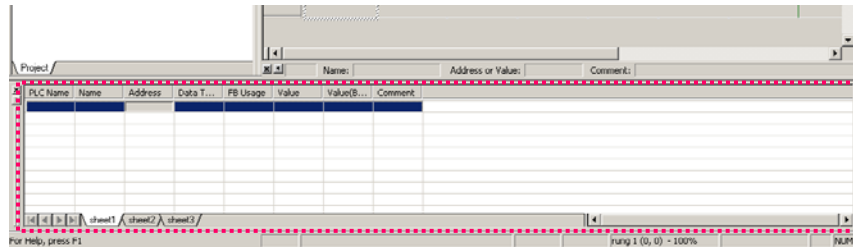
Alt + 3

Open the Edit dialog.

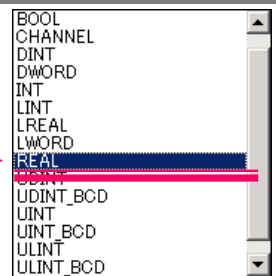
ENT

Click Browse... button using the mouse left button.

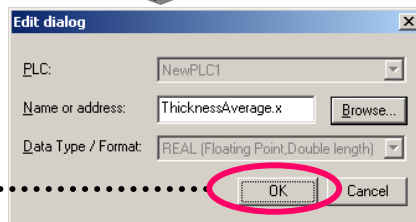
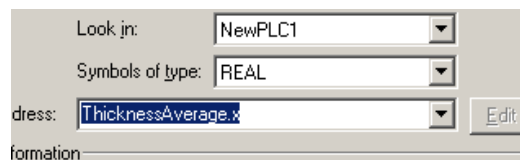
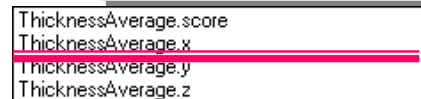
Click the  button using the left mouse button, then select the following:
[Symbols of type]
[Name or address]



Select REAL(32bit floating point)



Select ThicknessAverage.x



When monitoring internal variables at debug phase, collective registration is available in addition to the individual registration on the Watch Window through the operation shown here. For the details, refer "5-8 Batch Registration to Watch Window". When the function block is a ladder, conducting monitoring is available. For the details, refer "5-5 Operation Check- 1"

Click [OK] button using the left mouse button.

PLC Name	Name	Address	Data Type / Format	FB Usage	Value	Value(Binary)	Comment
NewPLC1	ThicknessAverage.x	H513	REAL (Floating Point,Double length)	Input	+0.0000000 Float	+0.0000000 Float	Input value 1

Reference: Example of an ST program using IF-THEN-ELSE-END_IF

The following ST program checks the average value calculated by the example of page 4-7 against a range (upper limit or lower limit).

FB Definition: OutputOfDecisionResult

Input symbols: score (REAL type), setover (REAL type), setunder (REAL type)

Output symbols: OK (BOOL type), overNG (BOOL type), underNG (BOOL type)

ST program:

```

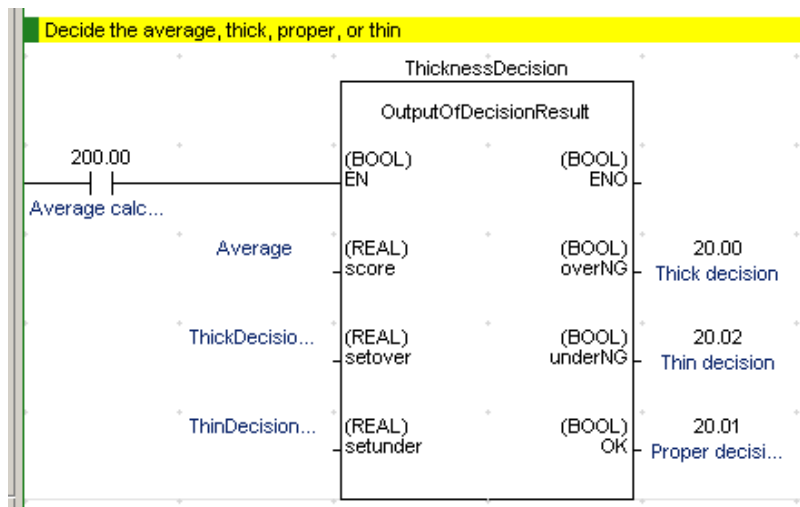
IF score > setover THEN      (* If score > setover, *)
    underNG := FALSE;      (* Turn off underNG *)
    OK := FALSE;           (* Turn off OK *)
    overNG := TRUE;        (* Turn on overNG *)

ELSIF score < setunder THEN  (* if score <= setover and score < setunder then *)
    overNG := FALSE;       (* Turn on overNG *)
    OK := FALSE;           (* Turn off OK *)
    underNG := TRUE;       (* Turn on underNG *)

ELSE                         (* if setover > score > setunder then *)
    underNG := FALSE;      (* Turn off underNG *)
    overNG := FALSE;       (* Turn off overNG *)
    OK := TRUE;            (* Turn off OK *)

END_IF;                     (* end of IF section*)
  
```

Example of an FB instance (the instance name is 'ThicknessDecision')



Chapter 5

Advanced

(Componentizing a Program Using FB)

Program
DesignEntering/Deb
ugging FB
DefinitionCreating FB
Definition
LibraryEntering Main
ProgramDebugging
Main Program

1. Overview

This chapter describes how to componentize a user program with an example using function blocks.

2. How to Proceed Program Development

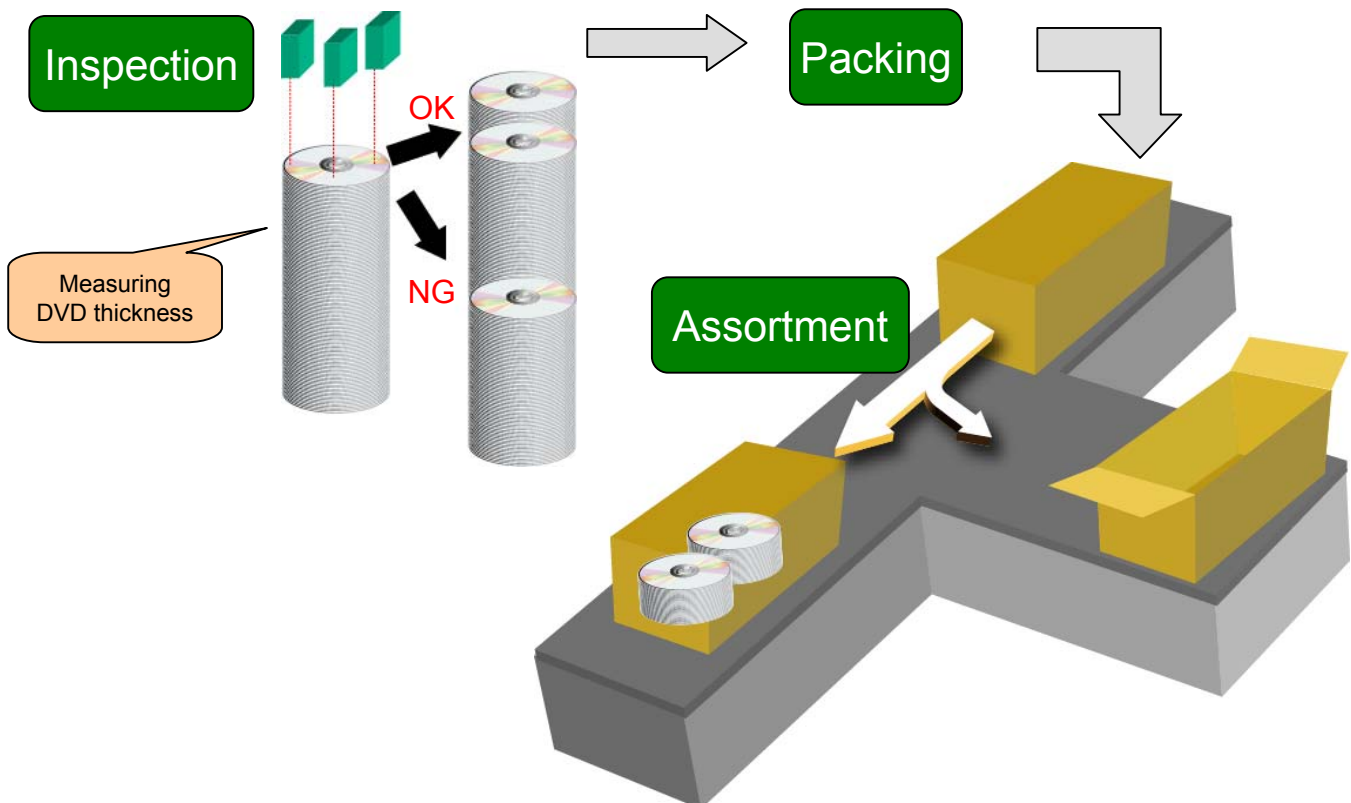
Generally shown below is a workflow to create a user program with componentization in the case of the application example below. Deliberate consideration is required especially in program design process.

- (1) Program Design
- (2) Creating Components
 - (2-1) Entering FB Component
 - (2-2) Debugging FB Component
 - (2-3) Creating FB Component Library (File Save)
- (3) Using Components in Application
 - (3-1) Importing Components
 - (3-2) Using Components for Program
 - (3-3) Debugging Program
- (4) Start-Up

3. Application Example

Shown here is a DVD inspection machine as an example for application.

Process can be primarily categorized into inspection, packing, and assortment.





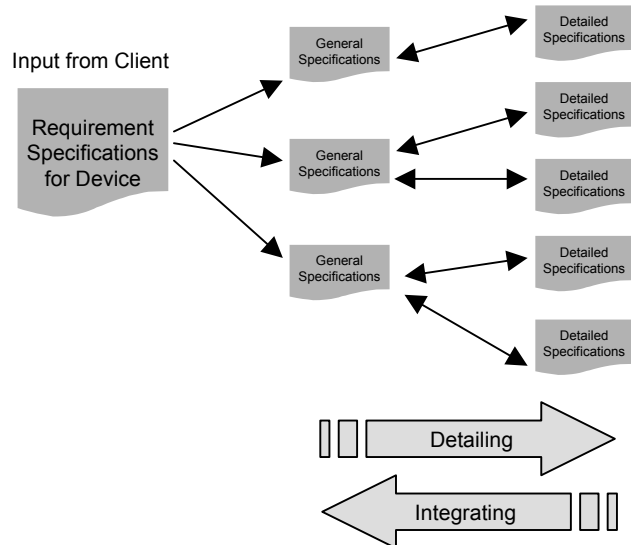
4. How to Proceed Program Development

Application can be materialized by using hardware and software (program) through combination of requirements.

Following sections describe how to proceed program design using an application example described before.

4-1 Overview of Design Process

Specifications should be repeatedly detailed and integrated to divide and classify them as shown in the right.



4-2 Extracting Requirement Specifications

Shown below are the extracted requirement specifications for this application.

Overview of DVD Inspection Machine (Requirement Specifications)

- Req. 1. DVD should be inserted from a loader.
- Req. 2. Thickness of DVD should be measured at 3 points. Average thickness of measurements should be calculated. If it is within its threshold range, DVD should be assorted into a stocker for good products, or a stocker for bad products if not.
- Req. 3. Good DVDs should be packed into the case.
- Req. 4. Packed DVDs should be packed into the paper box.
- Req. 5. Paper boxes should be classified into 2 types. Switching frequency should be counted to evaluate a life of limit switch adjacent to actuator of selection part.
- Req. 6. Other requirements

* To simplify the description, this document focuses on a part of device (underscored).


 Program Design

Entering/Debugging FB Definition

Creating FB Definition Library

Entering Main Program

Debugging Main Program

4-3 Detailing Specifications and Extracting Similar Processes

By detailing the specifications, there you will find similar processes or ones that can be used universally.

Actuator control (Example of similar process)

In this example, you can regard cylinder control for assortment of good and bad products and actuator control for paper box assortment as the same. Shown below are extracted requirements for these processes.

- The process has 2 actuators for bilateral movement which operate under input condition for each.
- Operation of each direction must be interlocked.
- The process has an input signal to reset its operation.

Average Threshold Check (Example of universal process)

A process should be extracted that will be used universally even if the process itself is used only once for this application. In this example, a process is extracted that calculates average of measured 3 thickness data of DVD and checks if it is within the threshold. Shown below are extracted requirements for this process.

- Average of 3 measurements must be calculated.
- Average value must be checked if it is within upper and lower limits of the threshold.

These requirements are used as the base for components. Names of components are defined as "ActuatorContro" FB and "AvgValue_ThresholdCheck" FB.

4-3-1 Creating Specifications for Components

Reuse of components can improve productivity of program development. To make reuse easily available, it is important to create specifications and insert comments for easier understanding specifications of input/output or operation without looking into the component.

It is advisable to describe library reference for OMRON FB Library.

Program
DesignEntering/Deb
ugging FB
DefinitionCreating FB
Definition
LibraryEntering Main
ProgramDebugging
Main Program

4-3-2 Example of FB Component Creation

“ActuatorControl” FB

It should be described in a ladder sequence because it is a process for sequence control.

[Input Variables]

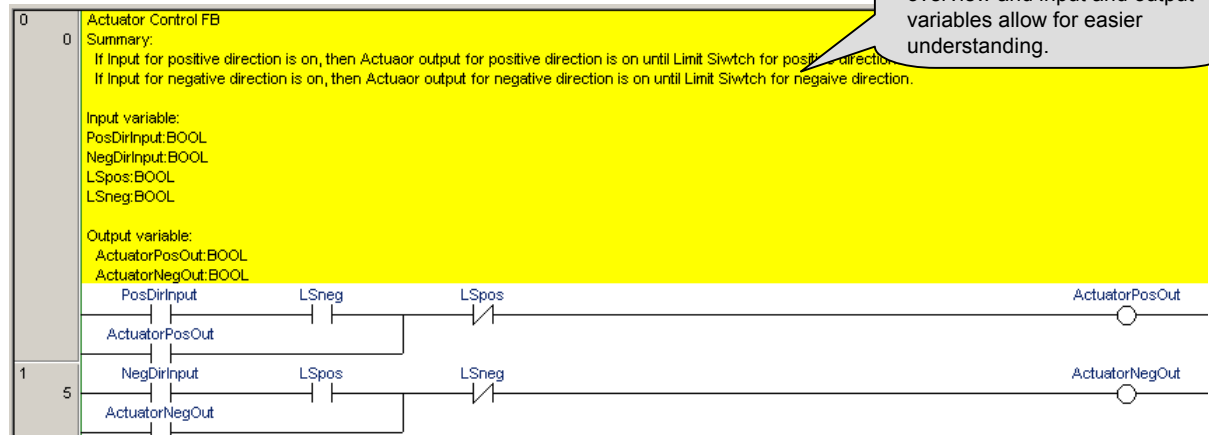
Name	Data Type	AT	Initial Value	Retained	Comment
EN	BOOL		FALSE		Controls execution of the Function Block.
PosDirInput	BOOL		FALSE		Input for positive direction
NegDirInput	BOOL		FALSE		Input for negative direction
LSpos	BOOL		FALSE		Limit switch for positive direction
LSneg	BOOL		FALSE		Limit switch for negative direction

[Output Variables]

Name	Data Type	AT	Initial Value	Retained	Comment
ENO	BOOL		FALSE		Indicates successful execution of the Function Block ...
ActuatorPosOut	BOOL		FALSE		Actuator output for positive direction
ActuatorNegOut	BOOL		FALSE		Actuator output for negative direction

[Internal Variables]

None.



“AvgValue_ThresholdCheck” FB

It should be described in ST because it is a process for numeric calculation and comparison.

[Input Variables]

Name	Data Type	AT	Initial Value	Retained	Comment
EN	BOOL		FALSE		Controls execution of the Function Block.
Input1	REAL		0.0		Input value 1
Input2	REAL		0.0		Input value 2
Input3	REAL		0.0		Input value 3
UpLimit	REAL		0.0		Upper limit value
LowLimit	REAL		0.0		Lower limit value

[Output Variables]

Name	Data Type	AT	Initial Value	Retained	Comment
ENO	BOOL		FALSE		Indicates successful execution of the Function Block...
Result	BOOL		FALSE		OK or NG judge flag

[Internal Variables]

Name	Data Type	AT	Initial Value	Retained	Comment
AvgValue	REAL		0.0		

```
(* Agarage value calculation and check of threshold for three values *)
```

```

AvgValue := ( Input1 + Input2 + Input3 ) / 3.0;          (* Divides Input 3 values by 3 *)
IF ((AvgValue <=UpLimit) AND (AvgValue >=LowLimit)) THEN (* Compare the agarage value if below of upper limit or above of lower limit *)
    Result := TRUE;
ELSE
    Result := FALSE;
END_IF;

```

Note: Use general names as long as possible for names of FB and variables in ladder diagram and ST, instead of specific names for the function at creation.

Program
Design

Entering/Deb
ugging FB
Definition

Creating FB
Definition
Library

Entering Main
Program

Debugging
Main Program

4-4. Integrating FBs

Detailed process components are extracted by now. Components for application will be created by combining them in the following sections.

4-4-1. Combining Existing Components - DVD_ThickSelectControl

Req. 2. "Thickness of DVD should be measured at 3 points. Average thickness of measurements should be calculated. If it is within its threshold range, DVD should be assorted into a stocker for good products, or a stocker for bad products if not." can be regarded as a process that combines "AvgValue_ThresholdCheck" and "ActuatorControl" investigated in the previous section. "Combining" these components allows creation of integrated component "DVD_ThickSelectControl" FB. Shown below is an example of an FB to be created.

[Input Variables]

Name	Data Type	AT	Initial Value	Retained	Comment
EN	BOOL		FALSE		Controls execution of the Function Block.
LSright	BOOL		FALSE		Limit switch for cylinder right direction
LSleft	BOOL		FALSE		Limit switch for cylinder left direction
Measure1	REAL		0.0		Measurement result 1 of DVD thickness (mm)
Measure2	REAL		0.0		Measurement result 2 of DVD thickness (mm)
Measure3	REAL		0.0		Measurement result 3 of DVD thickness (mm)

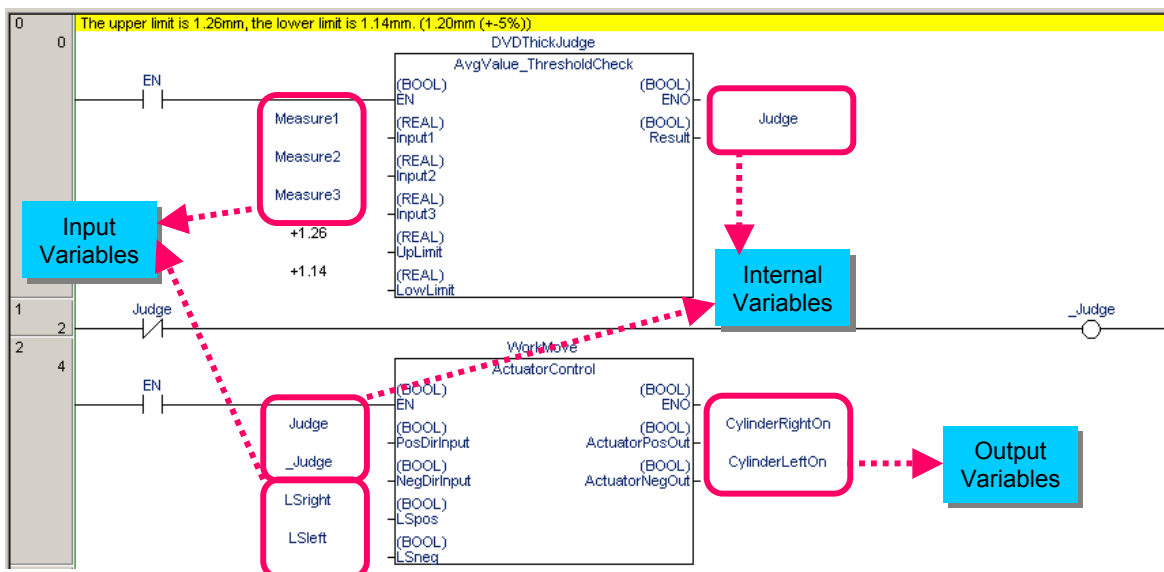
[Output Variables]

Name	Data Type	AT	Initial Value	Retained	Comment
ENO	BOOL		FALSE		Indicates successful execution of the Function Block ...
CylinderRightOn	BOOL		FALSE		Output for syylinder right direction
CylinderLeftOn	BOOL		FALSE		Output for syylinder left direction

[Internal Variables]

Name	Data Type	AT	Initial Value	Retained	Comment
WorkMove	FB [ActuatorControl]				
DVDThickJudge	FB [AvgValue_ThresholdCheck]				
Judge	BOOL		FALSE		
_Judge	BOOL		FALSE		

This FB has its specific name and variable names that include "DVD" or "Cylinder" because it is specifically created for application.



A function block can be called from within another function block. This is called "nesting".

To nest, declare a variable of FUNCTION BLOCK(FB) type as its internal variable to use the variable name as an instance.


 Program Design

Entering/Debugging FB Definition

Creating FB Definition Library

Entering Main Program

Debugging Main Program

4-4-2. Adding Functions to Existing Components - WorkMoveControl_LSONcount

Req. 5. "Paper boxes should be classified into 2 types. Switching frequency should be counted to evaluate a life of limit switch adjacent to actuator of selection part." can be materialized by counting OFF → ON switching of a limit switch as an input for "ActuatorControl". This component is called "WorkMoveControl_LSONcount" FB. Shown below is an example of an FB to be created.

[Input Variables]

Name	Data Type	AT	Initial Value	Retained	Comment
EN	BOOL		FALSE		Controls execution of the Function Block.
RightDirInput	BOOL		FALSE		Condition to move actuator to right direction
LeftDirInput	BOOL		FALSE		Condition to move actuator to left direction
LSright	BOOL		FALSE		Limit switch for actuator right direction
LSleft	BOOL		FALSE		Limit switch for actuator left direction
Reset	BOOL		FALSE		Resets number of times for opening - closing li...

[Output Variables]

Name	Data Type	AT	Initial Value	Retained	Comment
ENO	BOOL		FALSE		Indicates successful execution of the Function...
ActuatorRightOn	BOOL		FALSE		Output for actuator right direction
ActuatorLeftOn	BOOL		FALSE		Output for actuator left direction
LS_ONnumber	LINT		0		

[Internal Variables]

Name	Data Type	AT	Initial Value	Retained	Comment
PrevCycleLS	BOOL		FALSE		
WorkMove	FB [ActuatorControl]				

(* Work move control and count of number of times open - close of limit switch *)

(* Created by: machine development div. Yamada: 10-01-2005 *)

(* Resets number of times opening - closing limit switch *)

IF Reset = TRUE THEN

PrevCycleLS := FALSE;

END_IF;

(* Calls WorkMove (instance of ActuatorControl FB) *)

WorkMove(RightDirInput, LeftDirInput, LSright, LSleft, ActuatorRightOn, ActuatorLeftOn);

(* Counts number of times opening - closing limit switch *)

IF PrevCycleLS = FALSE and LSright = TRUE THEN

LS_ONnumber := LS_ONnumber+1;

END_IF;

PrevCycleLS := LSright; (* Copies LSright to compare at next execution *)


 Ladder FB is called from ST.

How to call FB (function block) from ST

FB to be called: MyFB

I/O variable of FB to be called:

Input: Input1, Input2

Output: Output1, Output2

Instance of MyFB declared in ST: MyInstance

I/O variable to be passed to FB in ST:

Input: STInput1, STInput2

Output: STOutput1, STOutput2

In this example, calling of FB instance from ST must be described as

MyInstance(Input1 := STInput1, Input2 := STInput2, Output1 => STOutput1, Output2 => STOutput2);

When all input/output variables are described, description of variables and assignment operators in one to be called can be omitted.

MyInstance(STInput1, STInput2, STOutput1, STOutput2);

By describing variables and assignment operators in one to be called, you can describe only a part of input/output variables.

MyInstance(Input1 := STInput1, Output2 => STOutput2);

Program
Design

Entering/Deb
ugging FB
Definition

Creating FB
Definition
Library

Entering Main
Program

Debugging
Main Program

4-5. Total Program Description

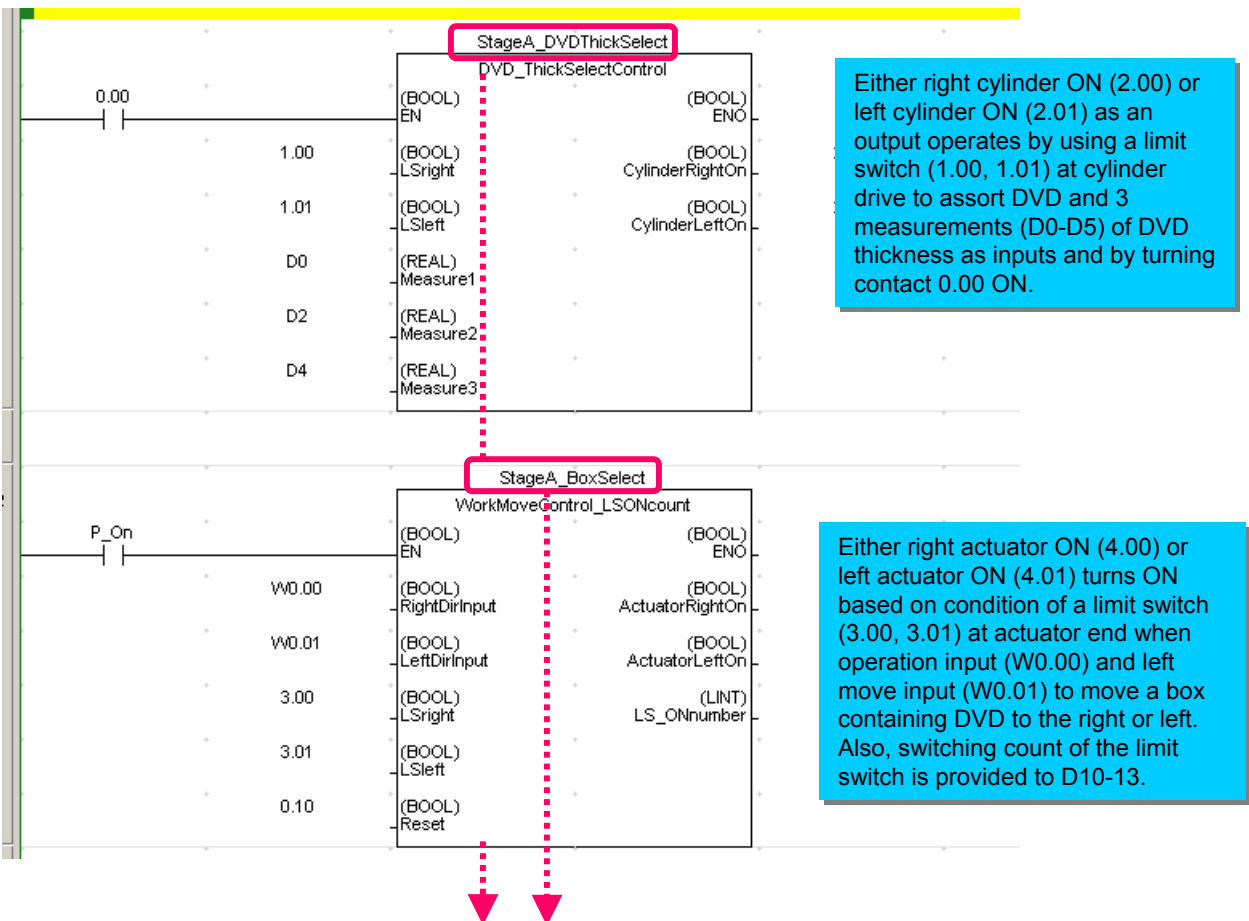
For components (FB) investigated here to work as a program, a circuit must be created that calls a component integrated from main ladder program.

* Example here limits to Req.2 and 5.

[Global Variables]

Name	Data Type	Address / Value	Rack Location	Usage
StageA_BoxSelect	FB [WorkMoveControl_LSONcount]	N/A [Auto]		
StageA_DVDThickSelect	FB [DVD_ThickSelectControl]	N/A [Auto]		

* Other instance variables than those to use FB are omitted.



Why the instance name is "StageA***"?

Although it is not explicitly described in the application example, a program for newly added stage B can be created only by describing an instance "StageB***" in the program and setting necessary parameters, without registering a new function block.

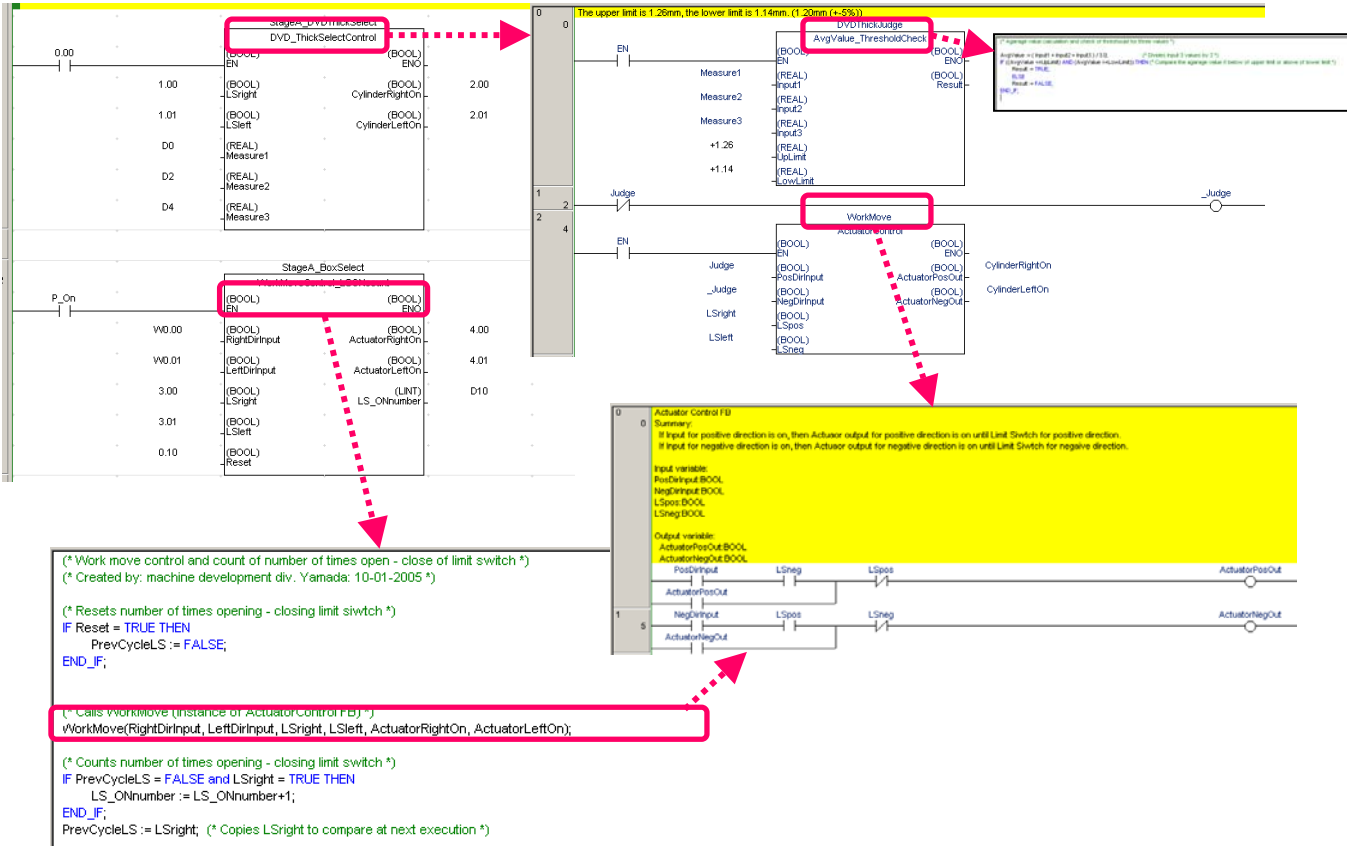
As a feature of Omron's function block, one FB can have more than one instance. By using operation-verified FB definition (algorithm), a program can be created only by assigning its address.

Program
DesignEntering/Deb
ugging FB
DefinitionCreating FB
Definition
LibraryEntering Main
ProgramDebugging
Main Program

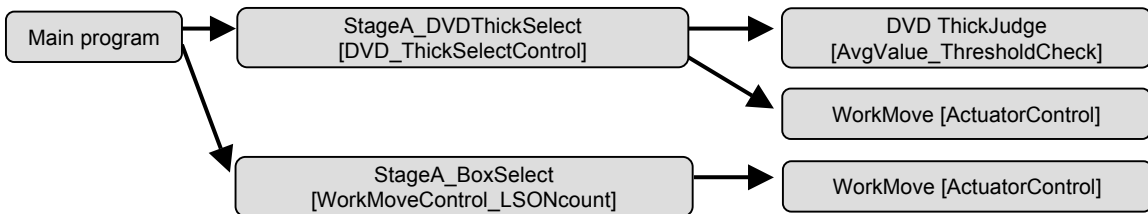
4-6-1. Total Program Structure

This section verifies total program structure including components (function blocks) created here.

[Main Program]



Instance names and FB names can be illustrated as follows: (FB name is described in [])



In a structured program, especially to change a lower level component (FB), it is important to understand parent/children relationship and components' sharing when process flow must be cleared in case of debugging, etc. It is advisable to create an understandable diagram of total program structure as design documentation.

CX-Programmer Ver.6.0 provides "FB instance viewer" when [Alt]+[5] key is pressed for easier understanding of software structure constructed by FBs. Also, address can be checked that is assigned to FB instance.

Name	Data Type	Address	Comment
EN	BOOL	H513.00	Controls execution of the Function Block.
Input1	REAL	H514	Input value 1
Input2	REAL	H516	Input value 2
Input3	REAL	H518	Input value 3
LowLimit	REAL	H522	Lower limit value
UpLimit	REAL	H520	Upper limit value

Program
Design

Entering/Deb
ugging FB
Definition

Creating FB
Definition
Library

Entering Main
Program

Debugging
Main Program

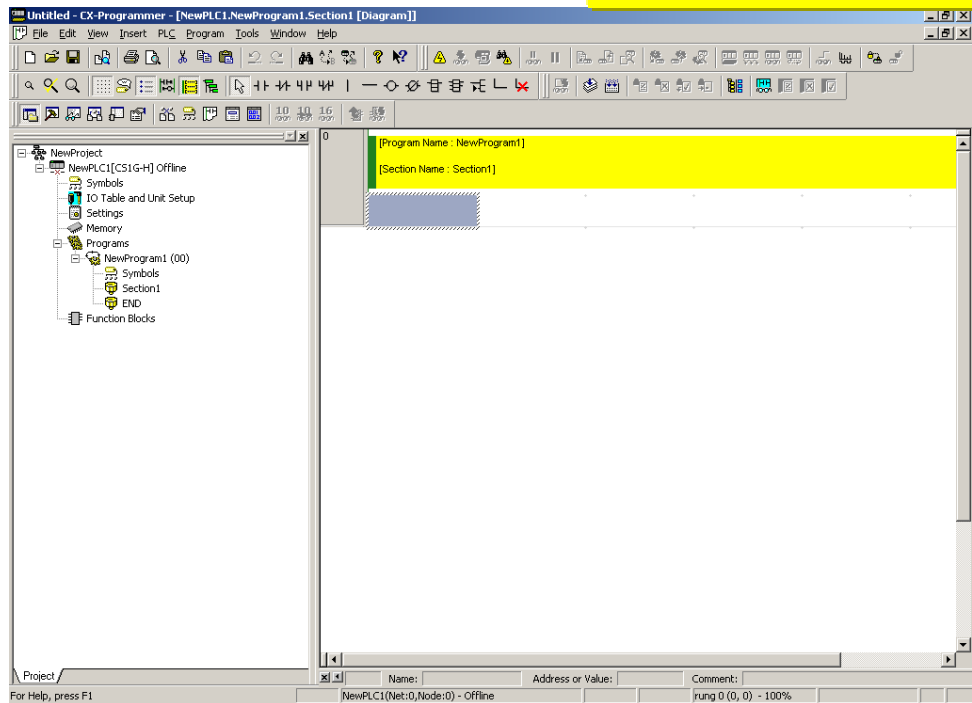
5. Entering FB Definition

This section describes how to enter an actually-designed program and debug it.
New project must be created and "ActuatorControl" FB of Page 5-4 must be entered.

5-1. New Project Creation and PLC Model/CPU Type Setting

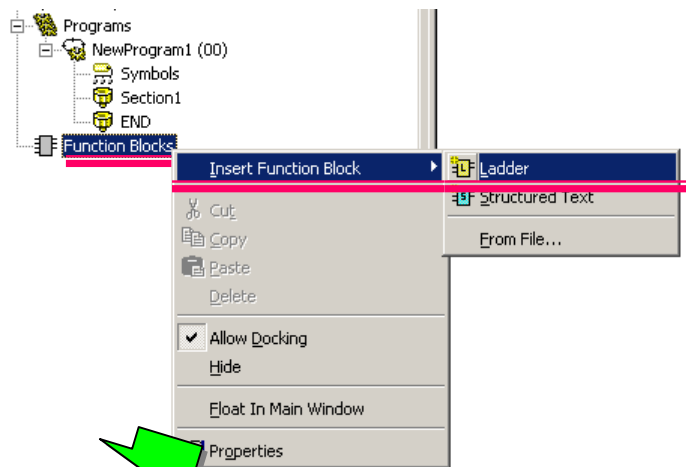
Refer to page 2-3 and create a new project.


! Select a PC model from the followings to use function blocks.
CS1G-H, CS1H-H, CJ1G-H, CJ1H-H, CJ1M



5-2. Creating Ladder Definition FB

Create Ladder definition FB.



Move the mouse cursor to a function block icon , then right-click. Select
→ Insert Function Block
→ Ladder

Now new FB is created.



Program
Design

Entering/Deb
ugging FB
Definition

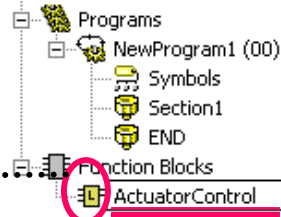
Creating FB
Definition
Library

Entering Main
Program

Debugging
Main Program


5-3. Entering FB Ladder Program

Change FB definition name.




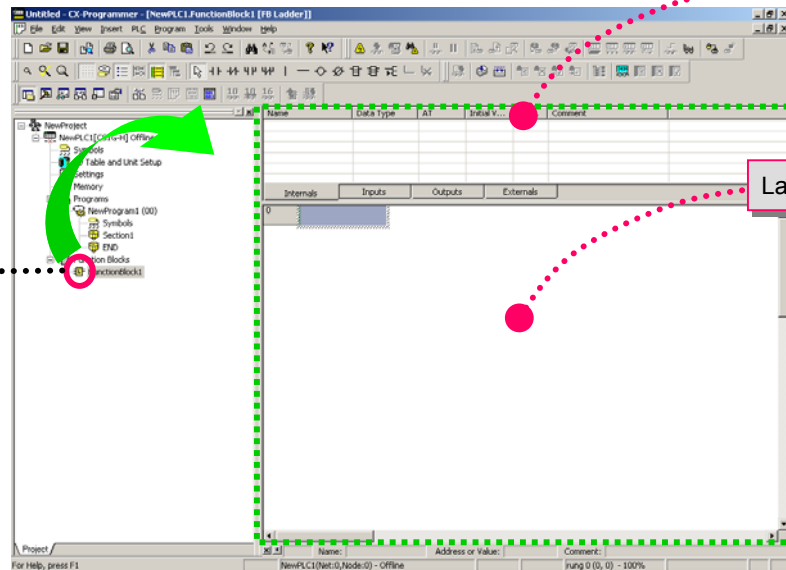
Caution:

A user cannot create function block definition name starting from "_".
The name must start from a character other than "_".

Move the mouse cursor to a copied function block icon , then right-click. Select → Rename
Enter [ActuatorControl].

Open FB ladder editor.

Move the mouse cursor to a function block icon , then double-click to open the function block ST editor.



Select the variables table and register variables in the function block.
All variables of "ActuatorControl" FB of page 5-4 must be registered.

Note: Order of variables must be the same as FB instance order.
To change order of variables, select a variable name then drag and drop it.

Select ladder input screen, then enter a ladder program.
All variables of "ActuatorControl" FB of page 5-4 must be registered.

Note: Although you can enter a circuit in the FB ladder editor similar to the main ladder editor, entering of address in the FB is invalid.

Note: To enter variable list in a line comment, you can select a variable from variables table then copy it. You can use it for more efficient input.

Program
Design

Entering/Deb
ugging FB
Definition

Creating FB
Definition
Library

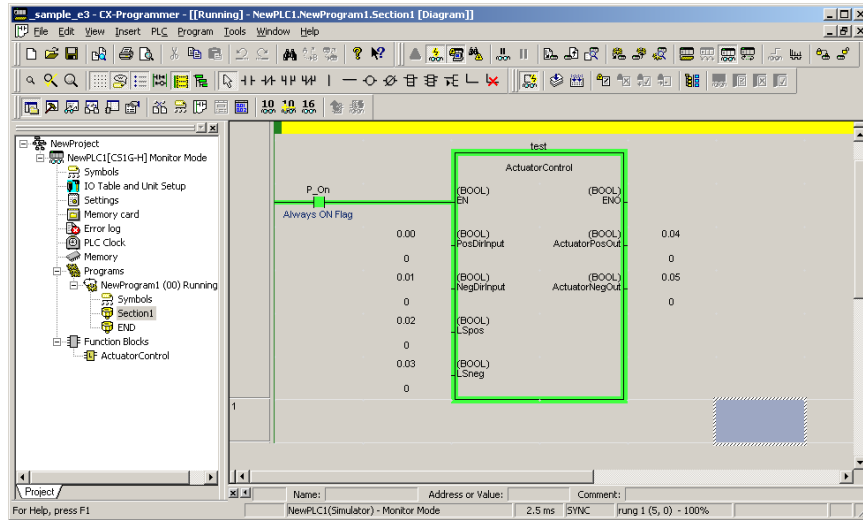
Entering Main
Program

Debugging
Main Program

5-4. Transferring Program

Connect to CX-Simulator online, transfer a program, then set PLC (simulator) to monitor mode.


For how to connect online and transfer a program, see page 2-10.

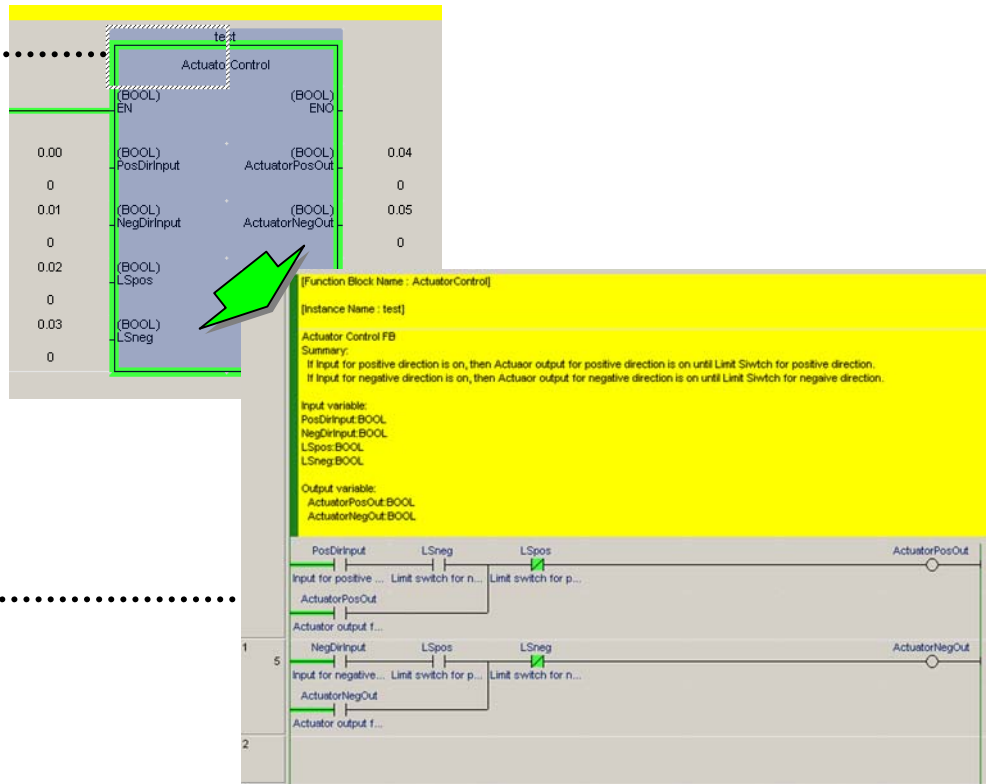


5-5. Operation Check-1

Change current parameter value of FB call statement on the main ladder, then check the operation of "ActuatorControl" FB.

Monitor the instance of ActuatorControl FB first.

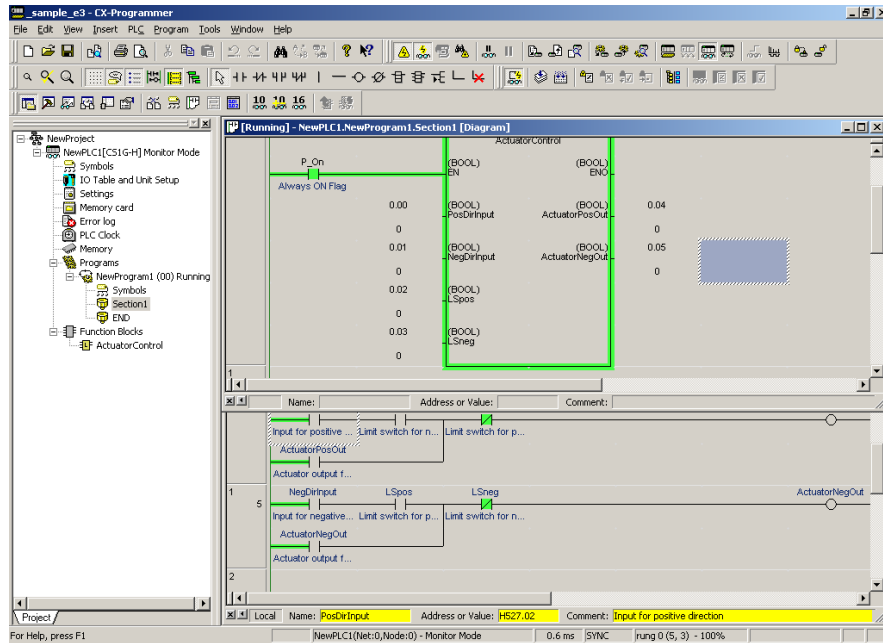
Move the cursor to FB call statement, then double-click or click  button.



FB ladder instance (under condition of address assigned) is monitored.

Program
DesignEntering/Deb
ugging FB
DefinitionCreating FB
Definition
LibraryEntering Main
ProgramDebugging
Main Program

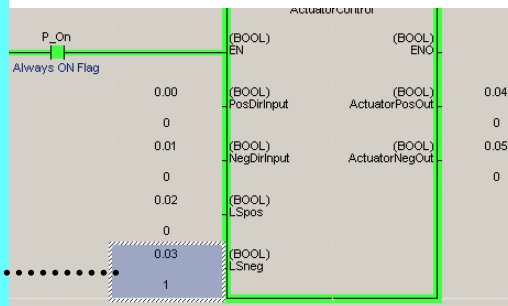
Display the main ladder and FB instance (FB ladder called by the main ladder) at the same time, then check the operation while changing current parameter value of FB call statement in the main ladder.



5-6. Operation Check-2

Enter following parameter values of FB call statement and check if expected output should be provided. In this example only (1) is shown, but all combination of conditions must be verified.

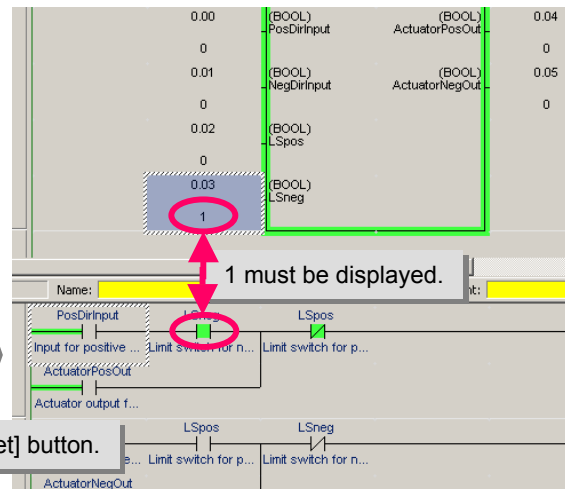
- (1) Initial State: Turn 0.03 ON. => 0.04 and 0.05 must be OFF. FB instance ladder monitor screen must be under state that corresponds to the value.
- (2) Actuator forward direction operation-1: Turn 0.00 ON => 0.04 must be turned ON. FB instance ladder monitor screen must be under state that corresponds to the value.
- (3) Actuator forward direction operation-2: Turn 0.03 OFF => 0.04 must be ON and 0.05 must be OFF. FB instance ladder monitor screen must be under state that corresponds to the value.
- (4) Actuator forward direction operation-3: Turn 0.02 ON => 0.04 must be OFF and 0.05 must be OFF. FB instance ladder monitor screen must be under state that corresponds to the value.



Move the cursor to 0.03 and press [ENT] key.

The 'Set New Value' dialog box is shown. The 'Address' field is set to '0.03', the 'Data type' is 'BOOL', and the 'Value' is '1'. The 'Set' button is highlighted.

Enter 1 and press [Set] button.



Program
Design

Entering/Deb
ugging FB
Definition

Creating FB
Definition
Library

Entering Main
Program

Debugging
Main Program

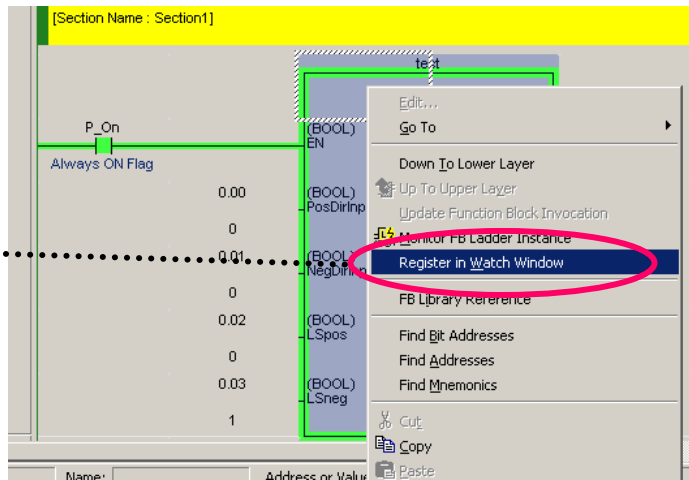
5-7. Entering/Debugging Other FB Definition

Thus far, entering and debugging for “ActuatorControl” FB are described. Other FB definition must be entered and debugged as well.

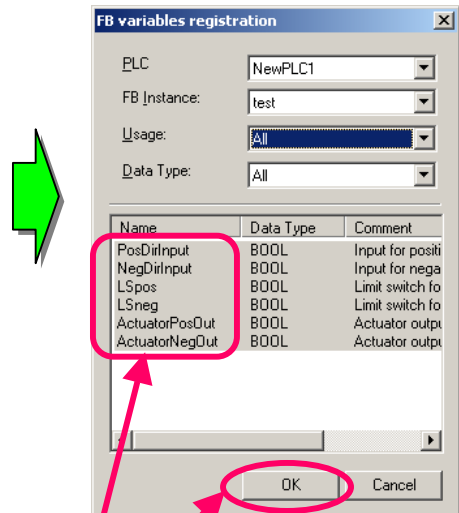
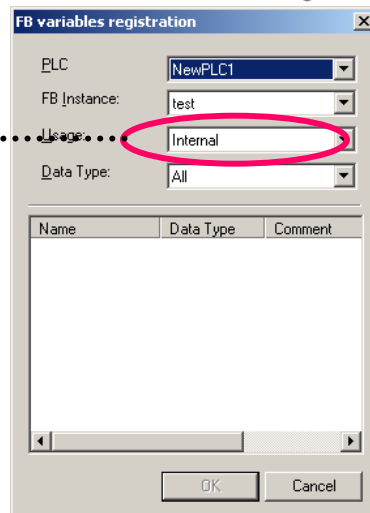
5-8. Batch Registration to Watch Window

For debugging, you can use batch registration of FB instance address to Watch Window instead of FB ladder monitor.

Move the cursor to FB call statement you want to register, right-click, then select [Register in Watch Window] in the menu.



Select Usage and Data type if necessary.



Select a name to register, then press [OK] button.

Program
Design

Entering/Deb
ugging FB
Definition

Creating FB
Definition
Library

Entering Main
Program

Debugging
Main Program

5-9. Executing Steps using the Simulation Function

Setting the simulation function breakpoint and using the Step Execution Function, you can stop the execution of the program and easily check the processing status during program execution.











This function can be used with CX-One Ver.1.1 and later (CX-Programmer Ver.6.1, CX-Simulator Ver.1.6 and later)

5-9-1. Explanation of the Simulation Buttons

The toolbar buttons below are for use with the simulation function. The function of each button is described here.



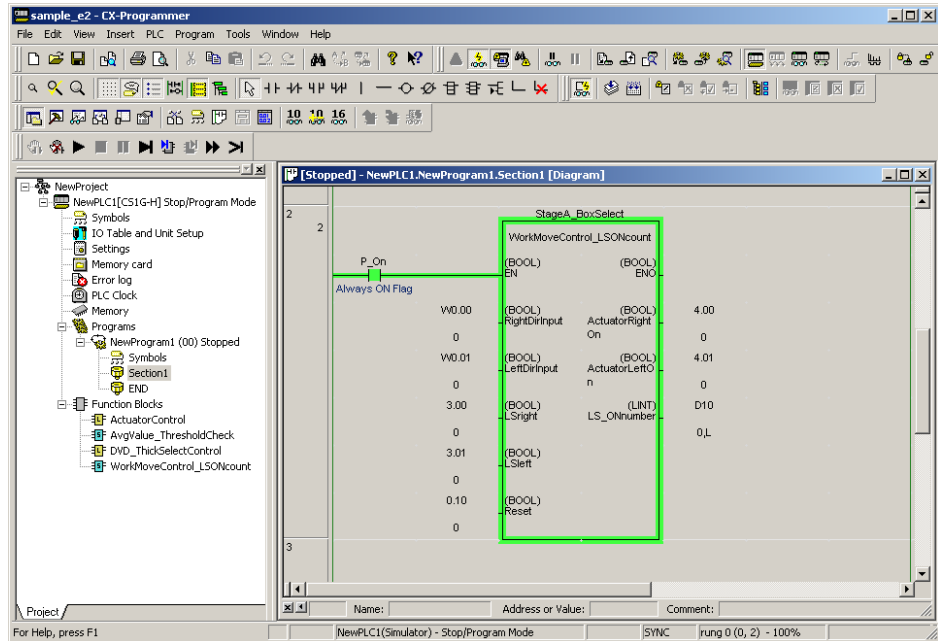
Simulation Buttons

	Set/Clear Breakpoint (F9 key)	Select locations (ladder, ST) where you want to stop while executing the simulation and a red mark will be displayed by pressing this button.
	Clear All Breakpoints	Delete a breakpoint (red mark) set using the Set Breakpoint button.
	Run(Monitor Mode) (F8 key)	Execute user program. Run mode becomes monitor mode.
	Stop(Program Mode)	Stop user program execution. Run mode becomes program mode.
	Pause	User program execution pauses at the cursor location.
	Step Run (F10 key)	Execute one user program step. In the case of a ladder, one instruction, and in the case of ST, one line.
	Step In (F11 key)	Execute one user program step. In cases where the cursor location calls the FB call statement, it transfers to the called FB instance (ladder or ST).
	Step Out (Shift+F11 key)	Execute one user program step. In cases where the cursor location is the FB instance, transfers to the base FB call statement.
	Continuous Step Run	Executes user program step, but automatically executes steps continuously after pausing for a certain amount of time.
	Scan Run	Execute one user program scan (one cycle).

Program
DesignEntering/Deb
ugging FB
DefinitionCreating FB
Definition
LibraryEntering Main
ProgramDebugging
Main Program

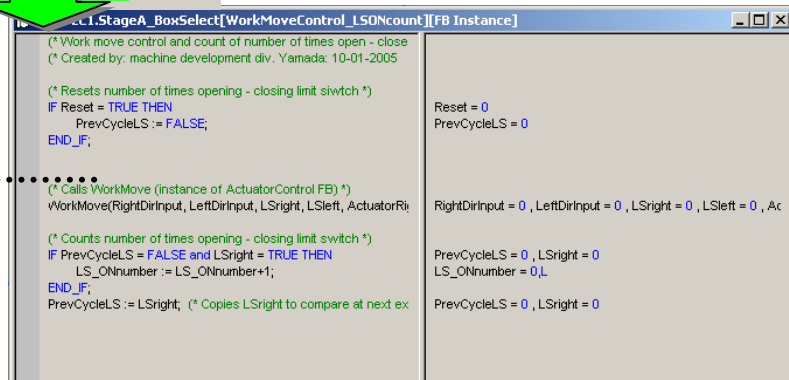
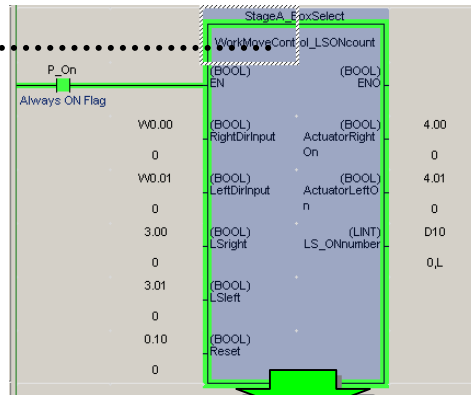
5-9-2. Setting Breakpoint and Executing Steps

Here is an explanation using Simulation Function “WorkMoveControl_LSONcount” FB Debug as an example.



Change from run mode to monitor mode.

Display “WorkMoveControl_LSONcount” FB instance.



ST Program

Variables and present values

Move the cursor inside the FB call statement and double-click the mouse or click the button.



The present values of the variables corresponding to the program are monitored in FB ST Instance (with assigned address).

Program
Design

Entering/Deb
ugging FB
Definition

Creating FB
Definition
Library

Entering Main
Program

Debugging
Main Program

Set the current value in the FB call statement parameter and confirm execution condition.

Set the following cases:

RightDirInput: ON

LeftDirInput: OFF

LSright: OFF

LSleft: ON


Reset: OFF

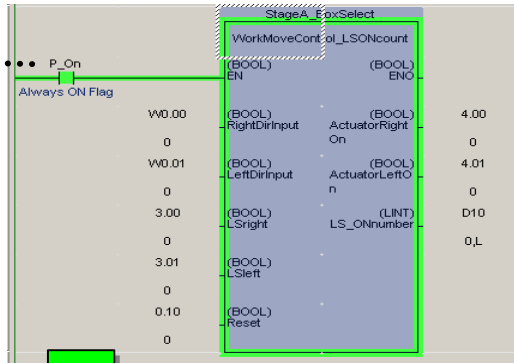
In this case, the following outputs are expected:

ActuatorRightOn: ON

ActuatorLeftOn: OFF

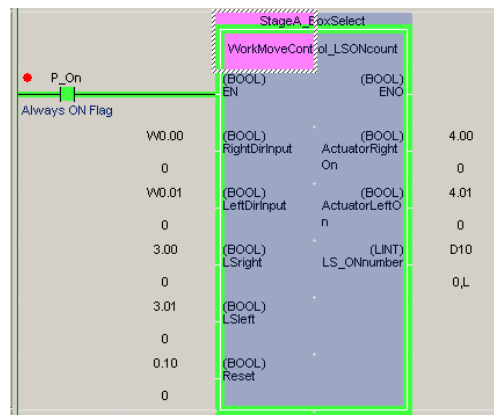
LS_ONnumber: 1

Move the cursor to the FB
call statement left input and
click the  button.

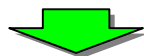


The program stops at the breakpoint.

Click the  button.



Perform breakpoint input contact. It stops at the following step of FB call statement.



Program
DesignEntering/Deb
ugging FB
DefinitionCreating FB
Definition
LibraryEntering Main
ProgramDebugging
Main Program

Press [←] [↓] [↓]

ENT

1

ENT

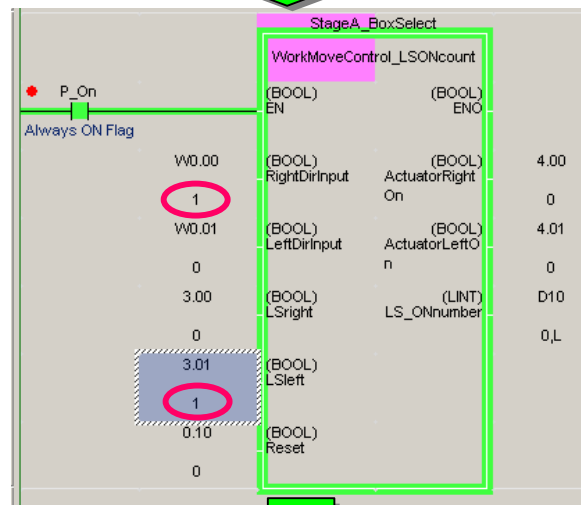
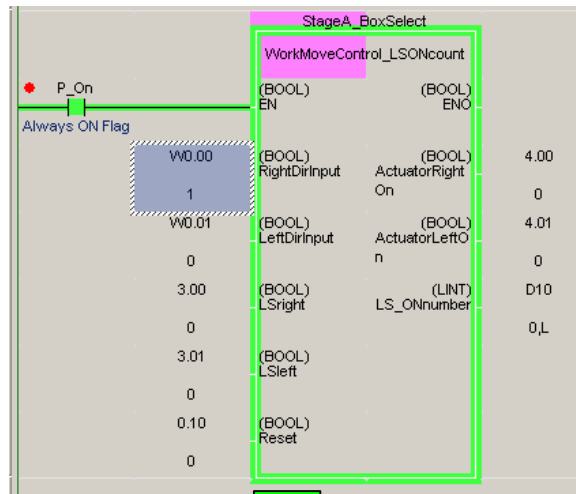
Press [↓] [↓] [↓]

1

ENT

Click the  button.

Turn input parameter "RightDirInput" and "LSleft" ON in the FB call statement.



The necessary input parameters were set.

```

(* Work move control and count of number of times open - close
(* Created by: machine development div. Yamada: 10-01-2005

(* Resets number of times opening - closing limit switch *)
Reset = TRUE THEN
  PrevCycleLS := FALSE;
END_IF;

(* Calls WorkMove (instance of ActuatorControl FB) *)
WorkMove(RightDirInput, LeftDirInput, LSright, LSleft, ActuatorRi

(* Counts number of times opening - closing limit switch *)
IF PrevCycleLS = FALSE and LSright = TRUE THEN
  LS_ONnumber := LS_ONnumber+1;
END_IF;
PrevCycleLS := LSright; (* Copies LSright to compare at next ex

Reset = 0
PrevCycleLS = 0

RightDirInput = 1 , LeftDirInput = 0 , LSright = 0 , LSleft = 1 , Ac

PrevCycleLS = 0 , LSright = 0
LS_ONnumber = 0,L

PrevCycleLS = 0 , LSright = 0
  
```

The cursor moves to the first line position of the called ST program.

Program Design



Entering/Debugging FB Definition



Creating FB Definition Library



Entering Main Program



Debugging Main Program



Click the button two times.



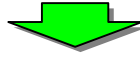
Click the button five times.



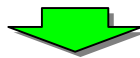
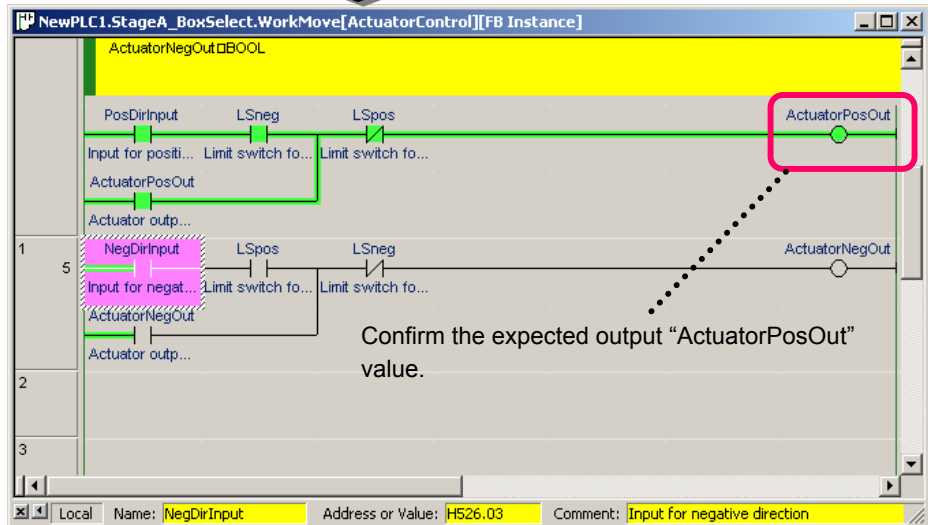
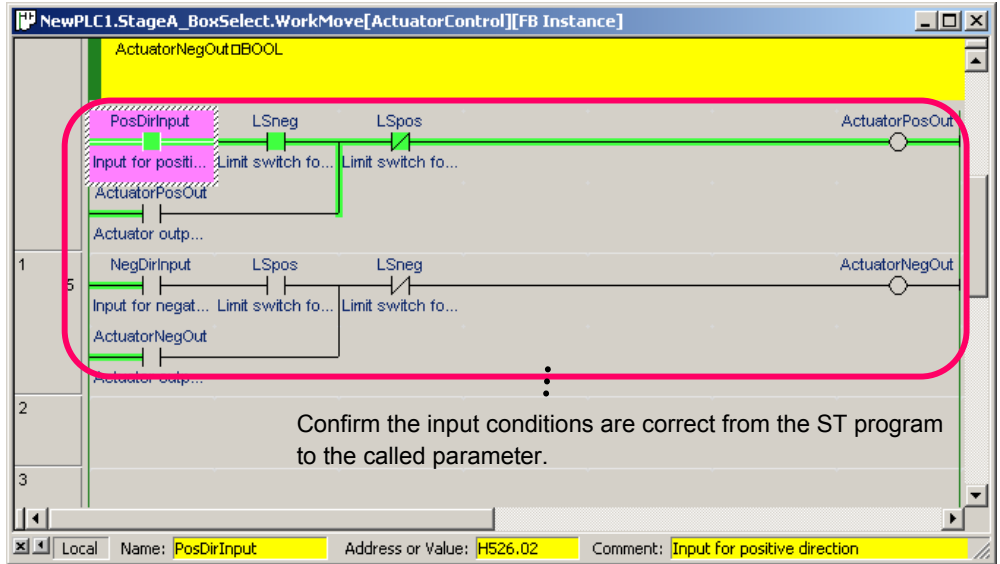
Click the button.



```
(* Calls WorkMove (instance of ActuatorControl FB) *)
WorkMove(RightDirInput, LeftDirInput, LSright, LSleft, ActuatorRightOn, ActuatorLeftOn, ActuatorPosOut, ActuatorNegOut, LSright, LSleft, ActuatorRightOn, ActuatorLeftOn, ActuatorPosOut, ActuatorNegOut)
```



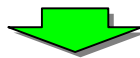
Transitions from the ST program to the called FB ladder program.



Confirmation has been completed. Return to the calling ST program.

```
(* Calls WorkMove (instance of ActuatorControl FB) *)
WorkMove(RightDirInput, LeftDirInput, LSright, LSleft, ActuatorRightOn, ActuatorLeftOn, ActuatorPosOut, ActuatorNegOut, LSright, LSleft, ActuatorRightOn, ActuatorLeftOn, ActuatorPosOut, ActuatorNegOut)
```

Confirm the previous circuit processing result is correctly reflected in the calling ST program monitor screen.



Program
Design

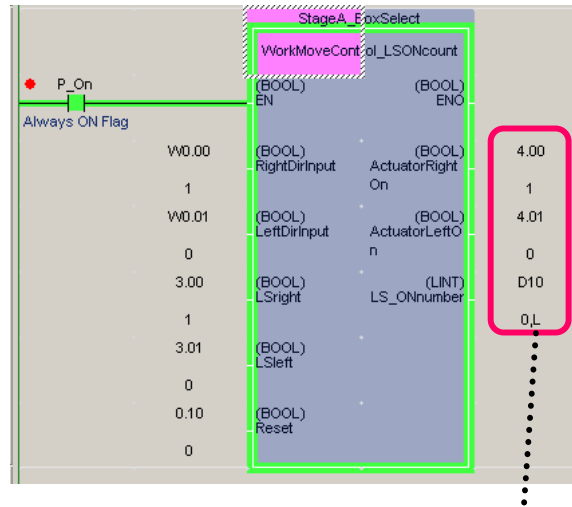
Entering/Deb
ugging FB
Definition

Creating FB
Definition
Library

Entering Main
Program

Debugging
Main Program

Click the  button.

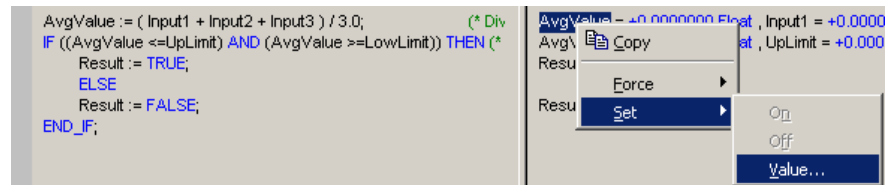


Transfer to the calling ladder program.

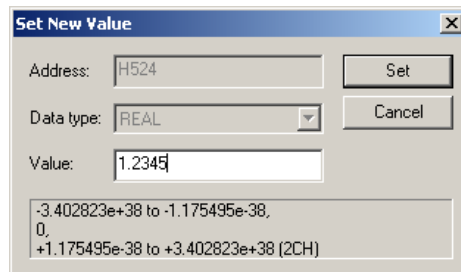
Confirm the output parameter is reflected correctly.

Hint

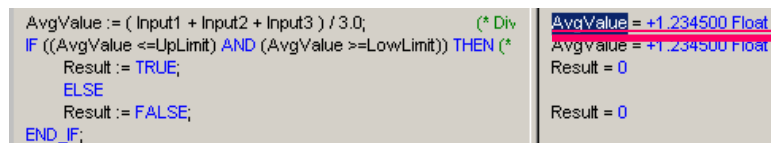
ST program change parameter current value can be performed with the following operation.



Select the parameter you want to change with the mouse cursor and click the right mouse button and select Set ⇒ Value



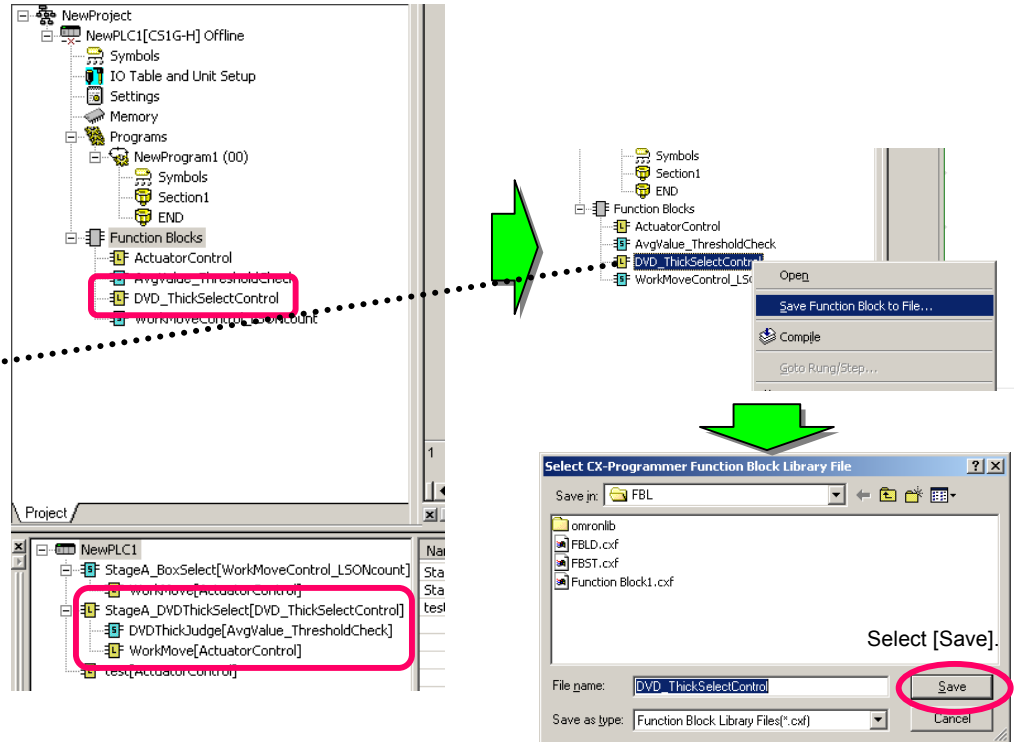
Set value and click the [Set] button.



Program
DesignEntering/Deb
ugging FB
DefinitionCreating FB
Definition
LibraryEntering Main
ProgramDebugging
Main Program

6. Creating FB Definition Library

To reuse operation-verified FB definition, it must be incorporated into library (file). Check the hierarchy using project workspace and FB instance viewer, then determine the FB definition you want to incorporate into library. In this case, it is "DVD_ThickSelectControl" FB.



Default folder for saving is C:\Program Files\Omron\CX-One\FBL.
It can be changed by CX-Programmer option setting "FB library storage folder".
OMRON FB Library is under omronlib folder.
Create a folder so that you should be able to classify it easily, such as Userlib\DVD.



When saving FB definition that calls another FB, both FB definition are saved.
When retrieving a project, calling-called relationship is maintained as saved.
It is easier to manage FB definition because saved FB definition is integrated.

Select
"DVD_ThickSelectControl" FB,
right-click and select [Save
Function Block to File] from
the context menu.

Program
DesignEntering/Deb
ugging FB
DefinitionCreating FB
Definition
LibraryEntering Main
ProgramDebugging
Main Program

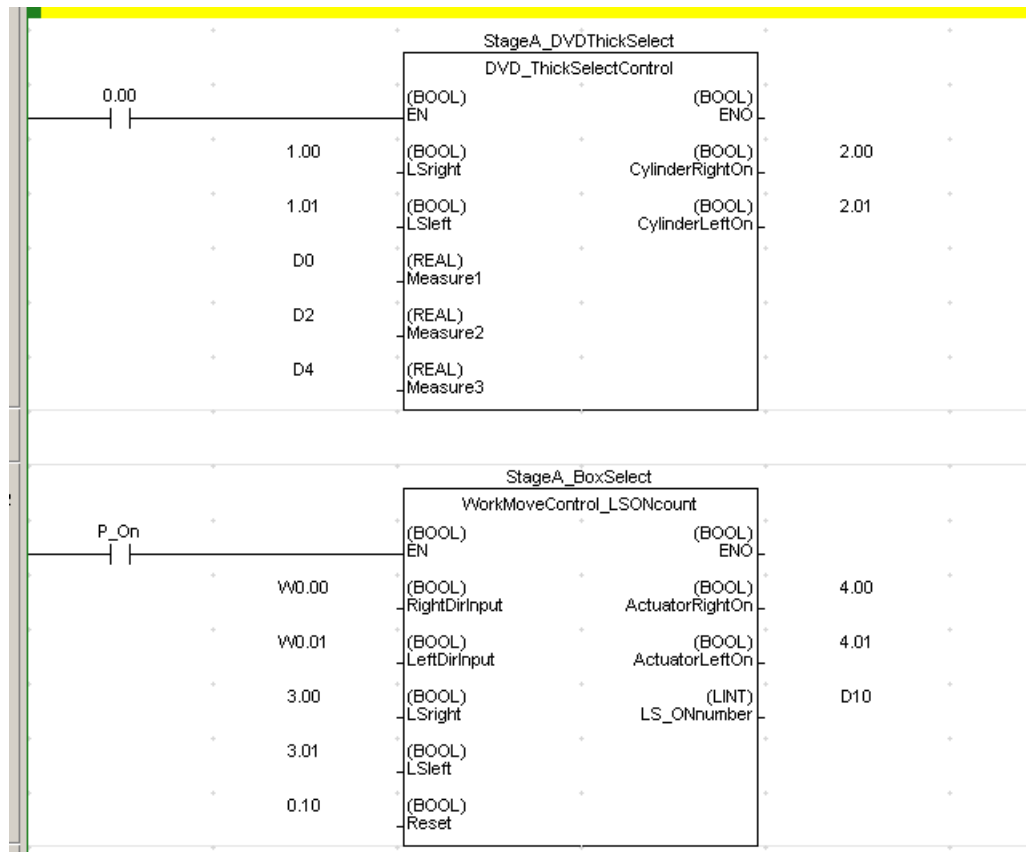
7. Entering Main Program

Add the main program to a project file that contains debugged FB definition. Program to be entered is one that is described in 4-5. Total Program Description in page 5-7.

[Global Variables]

Name	Data Type	Address / Value	Rack Location
StageA_BoxSelect	FB [WorkMoveControl_LSONcount]	N/A [Auto]	
StageA_DVDThickSelect	FB [DVD_ThickSelectControl]	N/A [Auto]	

* Other instance variables than those to use FB are omitted.



For how to enter a program, refer to pages from 2-6 to 2-9.



Program
Design

Entering/Debugging FB
Definition

Creating FB
Definition
Library

Entering Main
Program

Debugging
Main Program

8. Debugging Main Program

Main program must be debugged considering followings:

- Program areas that are irrelevant to FB
- Program areas that are relevant to an input parameter to FB
- Program areas that refer to an output parameter from FB

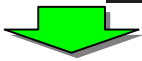
Main program in this example has no such area, thus explanation is omitted.

Supplemental Information

How to delete unused Function Block definitions

When you delete unused Function Block definitions, it is not enough just to delete the Function Block call statement. This is because the Function Block instance definitions are registered in the global symbol table. At this situation, when the compile (program check) is done, then the unused function block instances will be shown on the output window. You can identify the unused function block instance definitions and delete them easily. The Function Block definitions and Function Block instances are a part of user program in the CPU unit even if they are not called, so it is recommended to delete unused Function Block definitions and instances before transferring the program to the CPU unit.

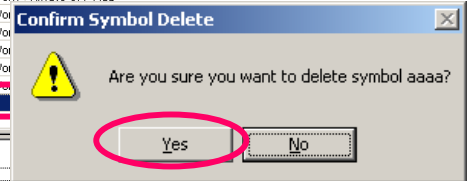
Execute Compile **F7** key



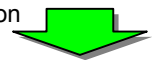
Result of Compilation

Name	Data Type	Address / Value	Rack Location	Usage	Comment
* P_LT	BOOL	CF007		Work	Less Than (LT) Flag
* P_Max_Cycle_Time	UDINT	A262		Work	Maximum Cycle Time
* P_N	BOOL	CF008		Work	Negative (N) Flag
* P_NE	BOOL	CF001		Work	Not Equals (NE) Flag
* P_OF	BOOL	CF009		Work	Overflow (OF) Flag
* P_Off	BOOL	CF114		Work	Always OFF Flag
* P_On	BOOL	CF113		Work	Always ON Flag
* P_Output_Off_Bit	BOOL	A500.15		Work	Output Off Bit
* P_Step	BOOL	A200.12		Work	Step
* P_UF	BOOL			Work	Underflow Flag
aaaa	FB (FunctionB...				

Double click mouse left button



Click mouse left button



Function Block definition will be deleted.

Memory allocation for Function Blocks

It is necessary to allocate required memory for each function block instances to execute Function Blocks.

CX-Programmer allocates the memory automatically based on the following setting dialog information.

(PLC menu → Function Block Memory → Function Block Memory Allocation)

There are 4 types of areas, 'Not retain', 'Retain', 'Timers', and 'Counters'. Please change the settings if requires.

- Notice when changing the settings

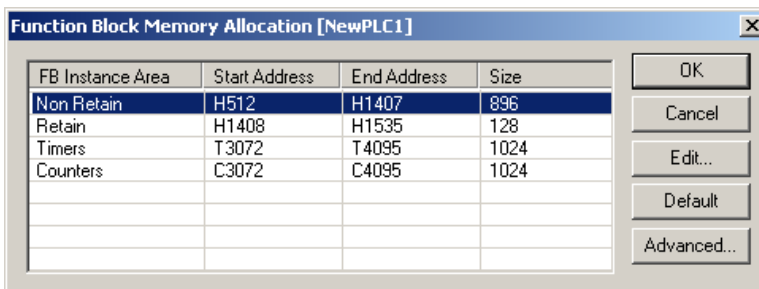
If you change the 'Not retain' or 'Retain' area, please consider the allocated memory areas for the special IO unit and CPU SIO unit.

- Special memory area for the Function Blocks

CS1/CJ1-H/CJ1M CPUs (unit version: 3.0 or higher) have a special memory area which is extended hold (H) relay area.

The address of the area is from H512 to H1535. CX-Programmer sets the area as a default.

Please note that the area cannot be used for the operands of ladder command.



Useful Functions


Command Operand Input Automatic Search and List Display

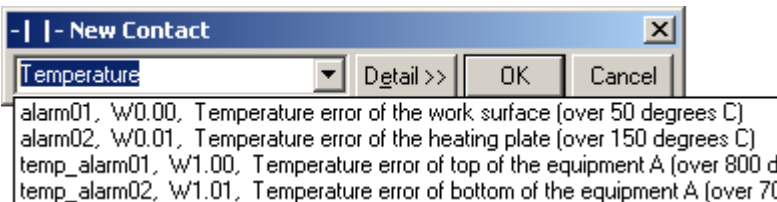
It is possible to automatically display a list of symbol names or IO comments when entering the operands of commands. When entering the operand for contact or output (or special instructions), enter a string, and the dropdown list is automatically updated to display in symbol names or IO Comments using the defined string. Selecting the item from the list defines the operand information.

This is an efficient way of entering registered symbol information into the ladder.

Example: Enter text "Temperature" to the edit field in the operand dialog.



Click  or push [F4] key; all symbols / address having IO comment containing the text 'temperature' are listed. See below:-



For instance, select 'temp_alarm01, W1.00, Temperature error of upper case of MachineA', from the list. The operand is set to be using symbol 'alarm01'.



FB Protect Function

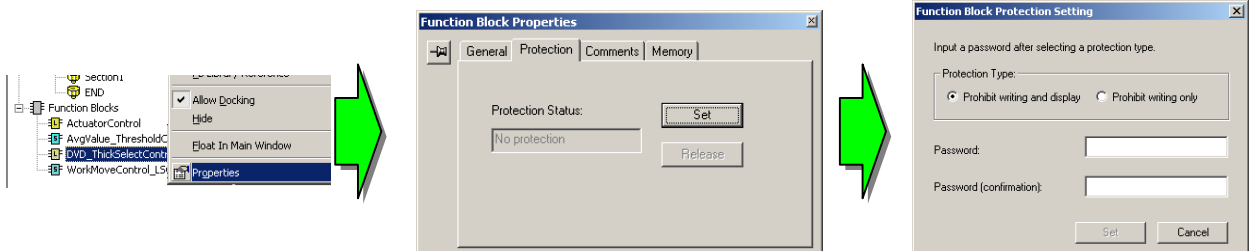
Preventative measures can be implemented by setting the password in the function block definition allocated on project file, protection corresponding to the use, program know-how leaks, improper changes, and alterations.

● Prohibit writing and display

By setting the protection classification "Prohibit writing and display," the corresponding function block definition contents cannot be displayed. By setting the password protection on the function block definition, program know-how leaks can be prevented.

● Prohibit writing only

By setting the protection classification "Prohibit writing only," the corresponding function block definition contents cannot be written or changed. By setting the password protection on the function block definition, improper program changes or modifications can be prevented.



IF Statement Examples

```
IF expression1 THEN statement-list1
[ ELSIF expression2 THEN statement-list2 ]
[ ELSE statement-list3 ]
END_IF;
```

The *expression1* and *expression2* expressions must each evaluate to a boolean value. The *statement-list* is a list of several simple statements e.g. a:=a+1; b:=3+c; etc.

The IF keyword executes *statement-list1* if *expression1* is true; if ELSIF is present and *expression1* is false and *expression2* is true, it executes *statement-list2*; if ELSE is present and *expression1* or *expression2* is false, it executes *statement-list3*. After executing *statement-list1*, *statement-list2* or *statement-list3*, control passes to the next statement after the END_IF.

There can be several ELSIF statements within an IF Statement, but only one ELSE statement.

IF statements can be nested within other IF statements (Refer to example 5).

Example 1

```
IF a > 0 THEN
  b := 0;
END_IF;
```

In this example, if the variable "a" is greater than zero, then the variable "b" will be assigned the value of zero.

If "a" is not greater than zero, then no action will be performed upon the variable "b", and control will pass to the program steps following the END_IF clause.

Example 2

```
IF a THEN
  b := 0;
END_IF;
```

In this example, if the variable "a" is true, then the variable "b" will be assigned the value of zero.

If "a" is false, then no action will be performed upon the variable "b", and control will pass to the program steps following the END_IF clause.

Example 3

```
IF a > 0 THEN
  b := TRUE;
ELSE
  b := FALSE;
END_IF;
```

In this example, if the variable "a" is greater than zero, then the variable "b" will be assigned the value of true (1), and control will be passed to the program steps following the END_IF clause.

If "a" is not greater than zero, then no action is performed upon the variable "b" and control is passed to the statement following the ELSE clause, and "b" will be assigned the value of false (0).

Control is then passed to the program steps following the END_IF clause.

Example 4

```
IF a < 10 THEN
  b := TRUE;
  c := 100;
ELSIF a > 20 THEN
  b := TRUE;
  c := 200;
ELSE
  b := FALSE;
  c := 300;
END_IF;
```

In this example, if the variable "a" is less than 10, then the variable "b" will be assigned the value of true (1), and the variable "c" will be assigned the value of 100. Control is then passed to the program steps following the END_IF clause.

If the variable "a" is equal to or greater than 10 then control is passed to the ELSE_IF clause, and if the variable "a" is greater than 20, variable "b" will be assigned the value of true (1), and the variable "c" will be assigned the value of 200. Control is then passed to the program steps following the END_IF clause.

If the variable "a" is between the values of 10 and 20 (i.e. both of the previous conditions IF and ELSE_IF were false) then control is passed to the ELSE clause, and the variable "b" will be assigned the value of false (0), and the variable "c" will be assigned the value of 300. Control is then passed to the program steps following the END_IF clause.

IF Statement Examples

Example 5

```
IF a THEN
  b := TRUE;
ELSE
  IF c>0 THEN
    d := 0;
  ELSE
    d := 100;
  END_IF;
  d := 400;
END_IF;
```

In this example (an example of a nested IF .. THEN statement), if the variable "a" is true (1), then the variable "b" will be assigned the value of true (1), and control will be passed to the program steps following the associated END_IF clause.

If "a" is false (0), then no action is performed upon the variable "b" and control is passed to the statement following the ELSE clause (in this example, another IF .. THEN statement, which is executed as described in Example 3, although it should be noted that any of the supported IEC61131-3 statements may be used).

After the described IF .. THEN statement is executed, the variable "d" will be assigned the value of 400.

Control is then passed to the program steps following the END_IF clause.

WHILE Statement Examples

```
WHILE expression DO
  statement-list;
END_WHILE;
```

The WHILE *expression* must evaluate to a boolean value. The *statement-list* is a list of several simple statements.

The WHILE keyword repeatedly executes the *statement-list* while the *expression* is true. When the *expression* becomes false, control passes to the next statement after the END_WHILE.

Example 1

```
WHILE a < 10 DO
  a := a + 1;
  b := b * 2.0;
END_WHILE;
```

In this example, the WHILE expression will be evaluated and if true (i.e. variable "a" is less than 10) then the statement-list (a:=a+1; and b:=b*2.0;) will be executed. After execution of the statement-list, control will pass back to the start of the WHILE expression. This process is repeated while variable "a" is less than 10. When the variable "a" is greater than or equal to 10, then the statement-list will not be executed and control will pass to the program steps following the END_WHILE clause.

Example 2

```
WHILE a DO
  b := b + 1;
  IF b > 10 THEN
    a:= FALSE;
  END_IF;
END_WHILE;
```

In this example, the WHILE expression will be evaluated and if true (i.e. variable "a" is true), then the statement-list (b:=b+1; and the IF .. THEN statement) will be executed. After execution of the statement-list, control will pass back to the start of the WHILE expression. This process is repeated while variable "a" is true. When variable "a" is false, the statement-list will not be executed and control will pass to the program steps following the END_WHILE clause.

Example 3

```
WHILE (a + 1) >= (b * 2) DO
  a := a + 1;
  b := b / c;
END_WHILE;
```

In this example, the WHILE expression will be evaluated and if true (i.e. variable "a" plus 1 is greater than or equal to variable "b" multiplied by 2) then the statement-list (a:=a+1; and b:=b/c;) will be executed. After execution of the statement-list, control will pass back to the start of the WHILE expression. This process is repeated while the WHILE expression equates to true. When the WHILE expression is false, then the statement-list will not be executed and control will pass to the program steps following the END_WHILE clause.

WHILE Statement Examples

Example 4

```
WHILE (a - b) <= (b + c) DO
  a := a + 1;
  b := b * a;
END_WHILE;
```

In this example, the WHILE expression will be evaluated and if true (i.e. variable "a" minus variable "b" is less than or equal to variable "b" plus variable "c") then the statement-list (a:=a+1; and b:=b*a;) will be executed. After execution of the statement-list, control will pass back to the start of the WHILE expression. This process is repeated while the WHILE expression is true. When the WHILE expression is false, then the statement-list will not be executed and control will pass to the program steps following the END_WHILE clause.

REPEAT Statement Examples

REPEAT

```
  statement-list;
UNTIL expression
END_REPEAT;
```

The REPEAT *expression* must evaluate to a boolean value. The *statement-list* is a list of several simple statements.

The REPEAT keyword repeatedly executes the *statement-list* while the *expression* is false. When the *expression* becomes true, control passes to the next statement after END_REPEAT.

Example 1

```
REPEAT
  a := a + 1;
  b := b * 2.0;
UNTIL a > 10
END_REPEAT;
```

In this example, the statement-list (a:=a+1; and b:=b*2.0;) will be executed. After execution of the statement-list the UNTIL expression is evaluated and if false (i.e. variable "a" is less than or equal to 10), then control will pass back to the start of the REPEAT expression and the statement-list will be executed again. This process is repeated while the UNTIL expression equates to false. When the UNTIL expression equates to true (i.e. variable "a" is greater than 10) then control will pass to the program steps following the END_REPEAT clause.

Example 2

```
REPEAT
  b := b + 1;
  IF b > 10 THEN
    a := FALSE;
  END_IF;
UNTIL a
END_REPEAT;
```

In this example, the statement-list (b:=b+1; and the IF .. THEN statement) will be executed. After execution of the statement-list the UNTIL expression is evaluated and if false (i.e. variable "a" is false), then control will pass back to the start of the REPEAT expression and the statement-list will be executed again. This process is repeated while the UNTIL expression equates to false. When the UNTIL expression equates to true (i.e. variable "a" is true) then control will pass to the program steps following the END_REPEAT clause.

Example 3

```
REPEAT
  a := a + 1;
  b := b / c;
UNTIL (a + 1) >= (b * 2)
END_REPEAT;
```

In this example, the statement-list (a:=a+1; and b:=b/c;) will be executed. After execution of the statement-list the UNTIL expression is evaluated and if false (i.e. variable "a" plus 1 is less than variable "b" multiplied by 2) then control will pass back to the start of the REPEAT expression and the statement-list will be executed again. This process is repeated while the UNTIL expression equates to false. When the UNTIL expression equates to true (i.e. variable "a" plus 1 is greater than or equal to variable "b" multiplied by 2) then control will pass to the program steps following the END_REPEAT clause.

Example 4

```
REPEAT
  a := a + 1;
  b := b * a;
UNTIL (a - b) <= (b + c)
END_REPEAT;
```

In this example, the statement-list (a:=a+1; and b:=b*a;) will be executed. After execution of the statement-list the UNTIL expression is evaluated and if false (i.e. variable "a" minus variable "b" is greater than variable "b" plus variable "c"), then control will pass back to the start of the REPEAT expression and the statement-list will be executed again. This process is repeated while the UNTIL expression equates to false. When the UNTIL expression equates to true (i.e. variable "a" minus variable "b" is less than or equal to variable "b" plus variable "c") then control will pass to the program steps following the END_REPEAT clause.

FOR Statement Examples

```
FOR control variable := integer expression1 TO integer expression2 [BY integer expression3] DO  
    statement-list;  
END_FOR;
```

The FOR *control variable* must be of an integer variable type. The FOR *integer expressions* must evaluate to the same integer variable type as the control variable. The *statement-list* is a list of several simple statements.

The FOR keyword repeatedly executes the *statement-list* while the *control variable* is within the range of *integer expression1* to *integer expression2*. If the BY is present then the *control variable* will be incremented by *integer expression3* otherwise by default it is incremented by one. The *control variable* is incremented after every executed call of the *statement-list*. When the *control variable* is no longer in the range *integer expression1* to *integer expression2*, control passes to the next statement after the END_FOR.

FOR statements can be nested within other FOR statements.

Example 1

```
FOR a := 1 TO 10 DO  
    b := b + a;  
END_FOR;
```

In this example, the FOR expression will initially be evaluated and variable "a" will be initialized with the value 1. The value of variable "a" will then be compared with the 'TO' value of the FOR statement and if it is less than or equal to 10 then the statement-list (i.e. b:=b+a;) will be executed. Variable "a" will then be incremented by 1 and control will pass back to the start of the FOR statement. Variable "a" will again be compared with the 'TO' value and if it is less than or equal to 10 then the statement-list will be executed again. This process is repeated until the value of variable "a" is greater than 10, and then control will pass to the program steps following the END_FOR clause.

Example 2

```
FOR a := 1 TO 10 BY 2 DO  
    b := b + a;  
    c := c + 1.0;  
END_FOR;
```

In this example, the FOR expression will initially be evaluated and variable "a" will be initialized with the value 1. The value of variable "a" will then be compared with the 'TO' value of the FOR statement and if it is less than or equal to 10 then the statement-list (i.e. b:=b+a; and c:=c+1.0;) will be executed. Variable "a" will then be incremented by 2 and control will pass back to the start of the FOR statement. Variable "a" will again be compared with the 'TO' value and if it is less than or equal to 10 then the statement-list will be executed again. This process is repeated until the value of variable "a" is greater than 10, and then control will pass to the program steps following the END_FOR clause.

Example 3

```
FOR a := 10 TO 1 BY -1 DO  
    b := b + a;  
    c := c + 1.0;  
END_FOR;
```

In this example, the FOR expression will initially be evaluated and variable "a" will be initialized with the value 10. The value of variable "a" will then be compared with the 'TO' value of the FOR statement and if it is greater than or equal to 1 then the statement-list (i.e. b:=b+a; and c:=c+1.0;) will be executed. Variable "a" will then be decremented by 1 and control will pass back to the start of the FOR statement. Variable "a" will again be compared with the 'TO' value and if it is greater than or equal to 1 then the statement-list will be executed again. This process is repeated until the value of variable "a" is less than 1, and then control will pass to the program steps following the END_FOR clause.

Example 4

```
FOR a := b + 1 TO c + 2 DO  
    d := d + a;  
    e := e + 1;  
END_FOR;
```

In this example, the FOR expression will initially be evaluated and variable "a" will be initialized with the value of variable "b" plus 1. The 'TO' value of the FOR statement will be evaluated to the value of variable "c" plus 2. The value of variable "a" will then be compared with the 'TO' value and if it is less than or equal to it then the statement-list (i.e. d:=d+a; and e:=e+1;) will be executed. Variable "a" will then be incremented by 1 and control will pass back to the start of the FOR statement. Variable "a" will again be compared with the 'TO' value and if it is less than or equal to it then the statement-list will be executed again. This process is repeated until the value of variable "a" is greater than the 'TO' value, and then control will pass to the program steps following the END_FOR clause.

FOR Statement Examples

Example 5

```
FOR a := b + c TO d - e BY f DO
  g := g + a;
  h := h + 1.0;
END_FOR;
```

In this example, the FOR expression will initially be evaluated and variable "a" will be initialized with the value of variable "b" plus variable "c". The 'TO' value of the FOR statement will be evaluated to the value of variable "d" minus variable "e". The value of variable "a" will then be compared with the 'TO' value. If the value of variable "f" is positive and the value of variable "a" is less than or equal to the 'TO' value then the statement-list (i.e. g:=g+a; and h:=h+1.0;) will be executed. If the value variable "f" is negative and the value of variable "a" is greater than or equal to the 'TO' value then the statement-list (i.e. g:=g+a; and h:=h+1.0;) will also be executed. Variable "a" will then be incremented or decremented by the value of variable "f" and control will pass back to the start of the FOR statement. Variable "a" will again be compared with the 'TO' value and the statement-list executed if appropriate (as described above).

This process is repeated until the value of variable "a" is greater than the 'TO' value (if the value of variable "f" is positive) or until the value of variable "a" is less than the 'TO' value (if the value of variable "f" is negative), and then control will pass to the program steps following the END_FOR clause.

CASE Statement Examples

CASE expression OF

```
case label1 [ , case label2 ] [ .. case label3 ] : statement-list1;
[ ELSE
  statement-list2 ]
END_CASE;
```

The CASE expression must evaluate to an integer value. The statement-list is a list of several simple statements. The case labels must be valid literal integer values e.g. 0, 1, +100, -2 etc..

The CASE keyword evaluates the expression and executes the relevant statement-list associated with a case label whose value matches the initial expression. Control then passes to the next statement after the END_CASE. If no match occurs within the previous case labels and an ELSE command is present the statement-list associated with the ELSE keyword is executed. If the ELSE keyword is not present, control passes to the next statement after the END_CASE.

There can be several different case labels statements (and associated statement-list) within a CASE statement but only one ELSE statement.

The "," operator is used to list multiple case labels associated with the same statement-list.

The ".." operator denotes a range case label. If the CASE expression is within that range then the associated statement-list is executed, e.g. case label of 1..10 : a:=a+1; would execute the a:=a+1 if the CASE expression is greater or equal to 1 and less than 10.

Example 1

```
CASE a OF
  2 : b := 1;
  5 : c := 1.0;
END_CASE;
```

In this example, the CASE statement will be evaluated and then compared with each of the CASE statement comparison values (i.e. 2 and 5 in this example).

If the value of variable "a" is 2 then that statement-list will be executed (i.e. b:=1;). Control will then pass to the program steps following the END_CASE clause.

If the value of variable "a" is 5 then that statement-list will be executed (i.e. c:=1.0;). Control will then pass to the program steps following the END_CASE clause.

If the value of variable "a" does not match any of the CASE statement comparison values then control will pass to the program steps following the END_CASE clause.

Example 2

```
CASE a + 2 OF
  -2 : b := 1;
  5 : c := 1.0;
ELSE
  d := 1.0;
END_CASE;
```

In this example, the CASE statement will be evaluated and then compared with each of the CASE statement comparison values (i.e. -2 and 5 in this example).

If the value of variable "a" plus 2 is -2 then that statement-list will be executed (i.e. b:=1;). Control will then pass to the program steps following the END_CASE clause.

If the value of variable "a" plus 2 is 5 then that statement-list will be executed (i.e. c:=1.0;). Control will then pass to the program steps following the END_CASE clause. If the value of variable "a" plus 2 is not -2 or 5, then the statement-list in the ELSE condition (i.e. d:=1.0;) will be executed. Control will then pass to the program steps following the END_CASE clause.

CASE Statement Examples

Example 3

```
CASE a + 3 * b OF
  1, 3 : b := 2;
  7, 11 : c := 3.0;
ELSE
  d := 4.0;
END_CASE;
```

In this example, the CASE statement will be evaluated and then compared with each of the CASE statement comparison values (i.e. 1 or 3 and 7 or 11 in this example).

If the value of variable "a" plus 3 multiplied by variable "b" is 1 or 3, then that statement-list will be executed (i.e. b:=2;). Control will then pass to the program steps following the END_CASE clause.

If the value of variable "a" plus 3 multiplied by variable "b" is 7 or 11, then that statement-list will be executed (i.e. c:=3.0;). Control will then pass to the program steps following the END_CASE clause.

If the value of variable "a" plus 3 multiplied by variable "b" is not 1, 3, 7 or 11, then the statement-list in the ELSE condition (i.e. d:=4.0;) will be executed. Control will then pass to the program steps following the END_CASE clause.

Example 4

```
CASE a OF
-2, 2, 4 : b := 2;
           c := 1.0;
6..11, 13 : c := 2.0;
1, 3, 5 : c := 3.0;
ELSE
  b := 1;
  c := 4.0;
END_CASE;
```

In this example, the CASE statement will be evaluated and then compared with each of the CASE statement comparison values, i.e. (-2, 2 or 4) and (6 to 11 or 13) and (1, 3 or 5) in this example.

If the value of variable "a" equals -2, 2 or 4, then that statement-list will be executed (i.e. b:=2; and c:=1.0;). Control will then pass to the program steps following the END_CASE clause.

If the value of variable "a" equals 6, 7, 8, 9, 10, 11 or 13 then, that statement-list will be executed (i.e. c:=2.0;). Control will then pass to the program steps following the END_CASE clause.

If the value of variable "a" is 1, 3 or 5, then that statement-list will be executed (i.e. c:=3.0;). Control will then pass to the program steps following the END_CASE clause.

If the value of variable "a" is none of those above, then the statement-list in the ELSE condition (i.e. b:=1; and c:=4.0;) will be executed. Control will then pass to the program steps following the END_CASE clause.

EXIT Statement Examples

```
WHILE expression DO
    statement-list1;
EXIT;
END_WHILE;
statement-list2;
```

```
REPEAT
    statement-list1;
EXIT;
UNTIL expression
END_REPEAT;
statement-list2;
```

```
FOR control variable := integer expression1 TO integer expression2 [ BY integer expression3 ] DO
    statement-list1;
EXIT;
END_FOR;
statement-list2;
```

The *statement-list* is a list of several simple statements.

The **EXIT** keyword discontinues the repetitive loop execution to go to the next statement, and can only be used in repetitive statements (**WHILE**, **REPEAT**, **FOR** statements). When the **EXIT** keyword is executed after *statement-list1* in the repetitive loop, the control passes to *statement-list2* immediately.

Example 1

```
WHILE a DO
IF c = TRUE THEN
    b:=0;EXIT;
END_IF;
IF b > 10 THEN
    a:= FALSE;
END_IF;
END_WHILE;
d:=1;
```

If the first IF expression is true (i.e. variable "c" is true), the statement-list (b:=0; and EXIT;) is executed during the execution of the WHILE loop. After the execution of the EXIT keyword, the WHILE loop is discontinued and the control passes to the next statement (d:=1;) after the END_WHILE clause.

Example 2

```
a:=FALSE;
FOR i:=1 TO 20 DO
    FOR j:=0 TO 9 DO
        IF i>=10 THEN
            n:=i*10+j;
            a:=TRUE;EXIT;
        END_IF;
    END_FOR;
    IF a THEN EXIT; END_IF;
END_FOR;
d:=1;
```

If the first IF expression is true (i.e. i>=10 is true) in the inside FOR loop, the statement-list (n:=i*10+j; and a:=TRUE; and EXIT;) is executed during the execution of the FOR loop. After the execution of the EXIT keyword, the inside FOR loop is discontinued and the control passes to the next IF statement after the END_FOR clause. If this IF expression is true (i.e. the variable "a" is true), EXIT keyword is executed, the outside FOR loop is discontinued after END_FOR clause, and the control passes to the next statement (d:=1;).

RETURN Statement Examples

statement-list1;
RETURN;
statement-list2;

The statement-list is a list of several simple statements.

The **RETURN** keyword breaks off the execution of the inside of the Function Block after *statement-list1*, and then the control returns to the program which calls the Function Block without executing *statement-list2*.

Example 1

```
IF a_1*b>100 THEN
  c:=TRUE;RETURN;
END_IF;
IF a_2*(b+10)>100 THEN
  c:=TRUE;RETURN;
END_IF;
IF a_3*(b+20)>100 THEN
  c:=TRUE;
END_IF;
```

If the first or second IF statement is true (i.e. "a_1*b" is larger than 100, or "a_2*(b+10)" is larger than 100), the statement (c:=TRUE; and RETURN;) is executed. The execution of the RETURN keyword breaks off the execution of the inside of the Function Block and the control returns to the program which calls the Function Block.

Array Examples

variable name [*subscript index*]

An array is a collection of like variables. The size of an array can be defined in the Function Block variable table.

An individual variable can be accessed using the array subscript operator [].

The *subscript index* allows a specific variable within an array to be accessed. The subscript index must be either a positive literal value, an integer expression or an integer variable. The subscript index is zero based. A subscript index value of zero would access the first variable, a subscript index value of one would access the second variable and so on.

Warning

If the subscript index is either an integer expression or integer variable, you must ensure that the resulting subscript index value is within the valid index range of the array. Accessing an array with an invalid index must be avoided. Refer to Example 5 for details of how to write safer code when using variable array offsets.

Example 1

```
a[0] := 1;
a[1] := -2;
a[2] := 1+2;
a[3] := b;
a[4] := b+1;
```

In this example variable "a" is an array of 5 elements and has an INT data type. Variable "b" also has an INT data type. When executed, the first element in the array will be set to the value 1, the second element will be set to -2, the third element will be set to 3 (i.e. 1+2), the forth element will be set to the value of variable "b" and the fifth element will be set to the value of variable "b" plus 1.

Example 2

```
c[0] := FALSE;
c[1] := 2>3;
```

In this example variable "c" is an array of 2 elements and has a BOOL data type. When executed, the first element in the array will be set to false and the second element will be set to false (i.e. 2 is greater than 3 evaluates to false).

Array Examples

Example 3

```
d[9] := 2.0;
```

In this example, variable "d" is an array of 10 elements and has a REAL data type. When executed, the last element in the array (the 10th element) will be set to 2.0.

Example 4

```
a[1] := b[2];
```

In this example, variable "a" and variable "b" are arrays of the same data type. When executed, the value of the second element in variable "a" will be set to the value of the third element in variable "b".

Example 5

```
a[b] := 1;  
a[b+1] := 1;  
a[(b+c) * (d-e)] := 1;
```

Note: As the integer variables and expressions are being used to access the array, the actual index value will not be known until run time, so the user must ensure that the index is within the valid range of the array a. For example, a safer way would be to check the array index is valid:

```
f := (b+c) * (d-e);  
IF (f > 0) AND (f < 5) THEN  
  a[f] := 1;  
END_IF;
```

Where variable "f" has an INT data type.

Example 6

```
a[b[1]] := c;  
a[b[2] + 3] := c;
```

This example shows how an array element expression can be used within another array element expression.

Numerical Functions and Arithmetic Functions

Function	Name	Argument data type	Return value type	Operation	Example
ABS(argument)	Absolute value	INT, DINT, LINT, UINT, UDINT, ULINT, REAL, LREAL	NT, DINT, LINT, UINT, UDINT, ULINT, REAL, LREAL	argument	a:=ABS(b)
SQRT(argument)	Square root	REAL, LREAL	REAL, LREAL		a:=SQRT(b)
LN(argument)	Natural logarithm	REAL, LREAL	REAL, LREAL		a:=LN(b)
LOG(argument)	Common logarithm	REAL, LREAL	REAL, LREAL		a:=LOG(b)
EXP(argument)	Natural exponential	REAL, LREAL	REAL, LREAL		a:=EXP(b)
SIN(argument)	Sine	REAL, LREAL	REAL, LREAL	SIN(argument)	a:=SIN(b)
COS(argument)	Cosine	REAL, LREAL	REAL, LREAL	COS(argument)	a:=COS(b)
TAN(argument)	Tangent	REAL, LREAL	REAL, LREAL	TAN(argument)	a:=TAN(b)
ASIN(argument)	Arc sine	REAL, LREAL	REAL, LREAL		a:=ASIN(b)
ACOS(argument)	Arc cosine	REAL, LREAL	REAL, LREAL		a:=ACOS(b)
ATAN(argument)	Arc tangent	REAL, LREAL	REAL, LREAL		a:=ATAN(b)
EXPT(base, exponent)	Exponential	Base: REAL, LREAL Exponent: INT, DINT, LINT, UINT, UDINT, ULINT	REAL, LREAL		a:=EXPT(b, c)



OMRON Corporation
Industrial Automation Company
Control Devices Division H.Q.
Shiokoji Horikawa, Shimogyo-ku,
Kyoto, 600-8530
Japan
Tel: (81)75-344-7109
Fax: (81)75-344-7149

Regional Headquarters

OMRON EUROPE B.V.
Wegalaan 67-69, NL-2132 JD Hoofddorp
The Netherlands
Tel: (31)2356-81-300/
Fax: (31)2356-81-388

OMRON ELECTRONICS LLC
1 East Commerce Drive, Schaumburg,
IL 60173 U.S.A.
Tel: (1)847-843-7900/Fax: (1)847-843-8568

OMRON ASIA PACIFIC PTE. LTD.
83 Clemenceau Avenue,
#11-01, UE Square,
Singapore 239920
Tel: (65)6835-3011/Fax: (65)6835-2711

OMRON (CHINA) CO., LTD.
Room 2211, Bank of China Tower,
200 Yin Cheng Zhong Road,
PuDong New Area, Shanghai, 200120 China
Tel: (86)21-5037-2222/Fax: (86)21-5037-2200

Authorized Distributor:

Note: Specifications subject to change without notice.
1633014-5B

Cat. No. R133-E01-02
Printed in Japan