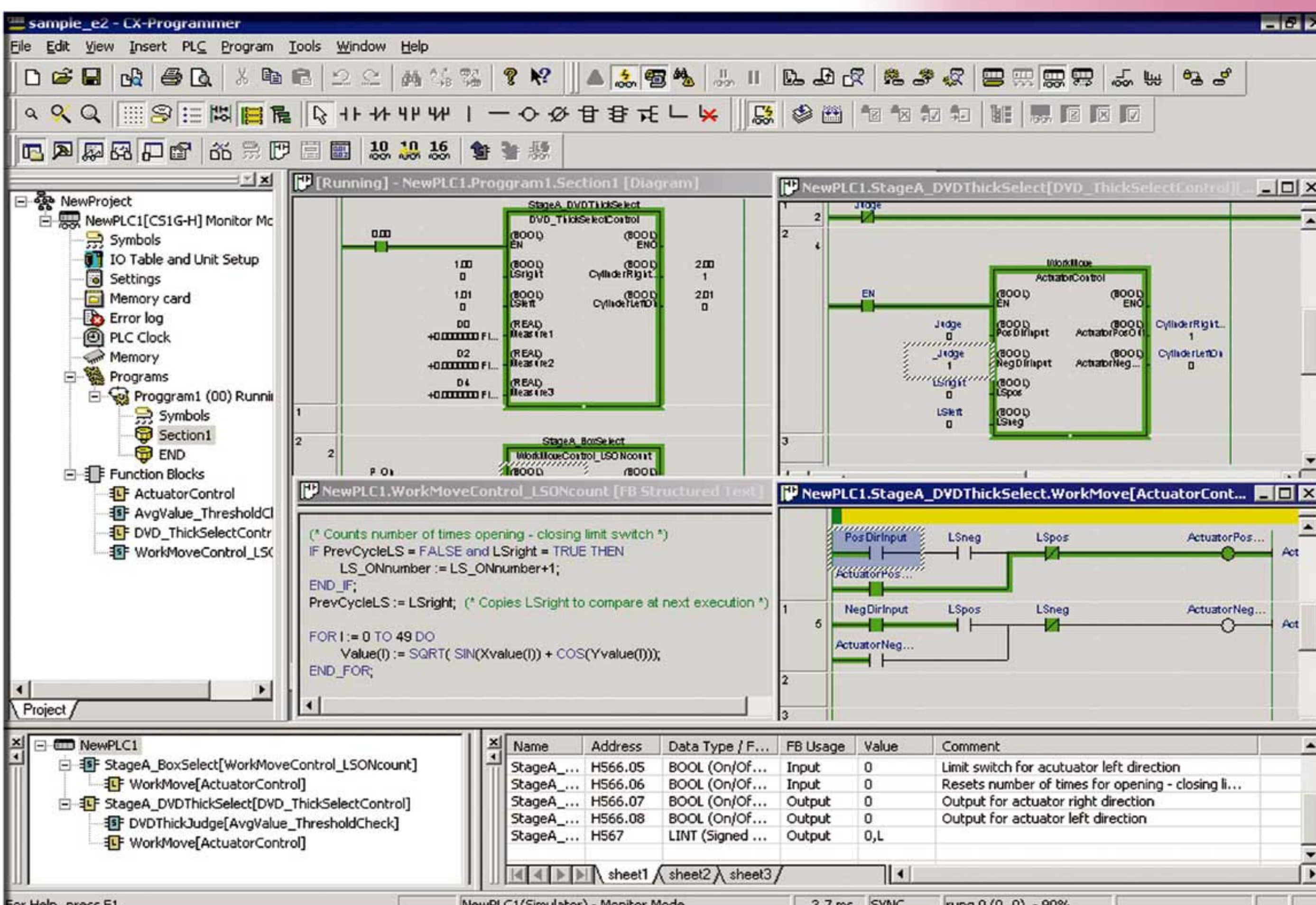
Function Block/ Structured Text Introduction Guide



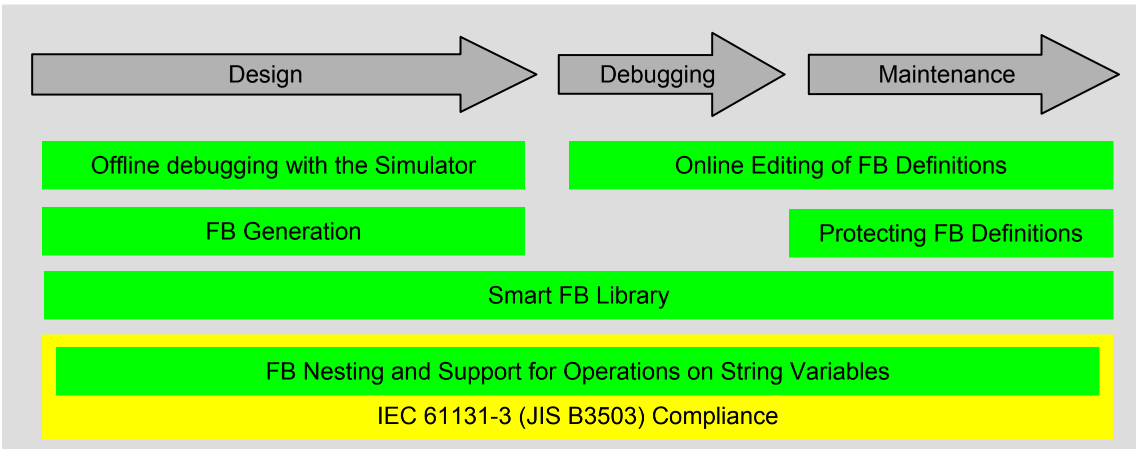
Introduction

- Please be sure to read and understand Precautions and Introductions in *CX-Programmer Operation Manual Function Block/Structured Text* and *CX-Programmer Operation Manual* before using the product.
- This guide describes the basic operation procedure of CX-Programmer. Refer to the Help or the Operation Manual of the PDF file for detailed descriptions.
- Acrobat Reader 5.0 or later is required to read the PDF files.
- You can display the PDF files from the [Start] menu on your desktop after installing the CX-Programmer.
- The screen views used in this guide may be different from the actual view, and be subject to change without notice.
- The product names, service names, function names, and logos described in this guide are trademarks or registered trademarks of their respective companies.
- The symbols (R) and TM are not marked with trademarks and registered trademarks in this guide respectively
- The product names of the other companies may be abbreviated in this guide.

Contents

Introduction
This section provides information that can be used when creating function blocks (FBs) and using the Smart FB Library with a SYSMAC CS1, CJ1-H, or CJ1M Series CPU Unit (unit version 3.0 or later) and CX-Programmer version 5.0 or higher.

Features of OMRON Function Blocks
OMRON function blocks can be written in ladder language or ST (structured text) language, and conform to the IEC 61131-3 standard. The function blocks provide functions for more efficient design and debugging of the user equipment, as well as easier maintenance.



PLC Program Development Steps and Corresponding Functions

Smart FB Library
The Smart FB Library is a set of function block elements that improve interoperability between OMRON PLC Units and FA components. If this library is used, it is not necessary to create a ladder program to use basic Unit and FA component functions. This enables the user to reduce the time spent on previous task, such as determining how to use the device's functions. (CS1/CJ1H unit version 3.0 or later and CX-Programmer version 5.0 or higher)

Online Editing of FB Definitions
FB definitions can be changed during operation, so FB definitions can be edited quickly during debugging. In addition, FBs can be used with confidence even in equipment that must operate 24 hours/day. (CS1/CJ1H unit version 4.0 or later and CX-Programmer version 7.0 or higher)

Nesting
Not only can programs be created with nested OMRON FBs, it is possible to make easy-to-understand, stress-free operations by switching windows depending on conditions and displaying structures in a directory-tree format. (CS1/CJ1H unit version 3.0 or later and CX-Programmer version 6.0 or higher)

Protecting FB Definitions
It is possible to prevent unintentional or unauthorized changes or disclosure of the program by setting passwords for the function block definitions allocated in the project file and protecting the definitions based on their purpose. (CS1/CJ1H unit version 3.0 or later and CX-Programmer version 6.1 or higher)

Offline Debugging with the Simulator
The PLC program's operation can be checked on the desktop, so program quality can be improved and verified early on. Both the ladder and ST can be executed in the computer application. (CX-Programmer version 6.1 or higher and CX-Simulator version 1.6 or higher)

String Operations for Variable Support
The functions that perform string data operations in ST language not only support string variables, they also strengthen the instructions (functions) used to communicate with string data I/O. (CS1/CJ1H unit version 4.0 or later and CX-Programmer version 7.0 or higher)

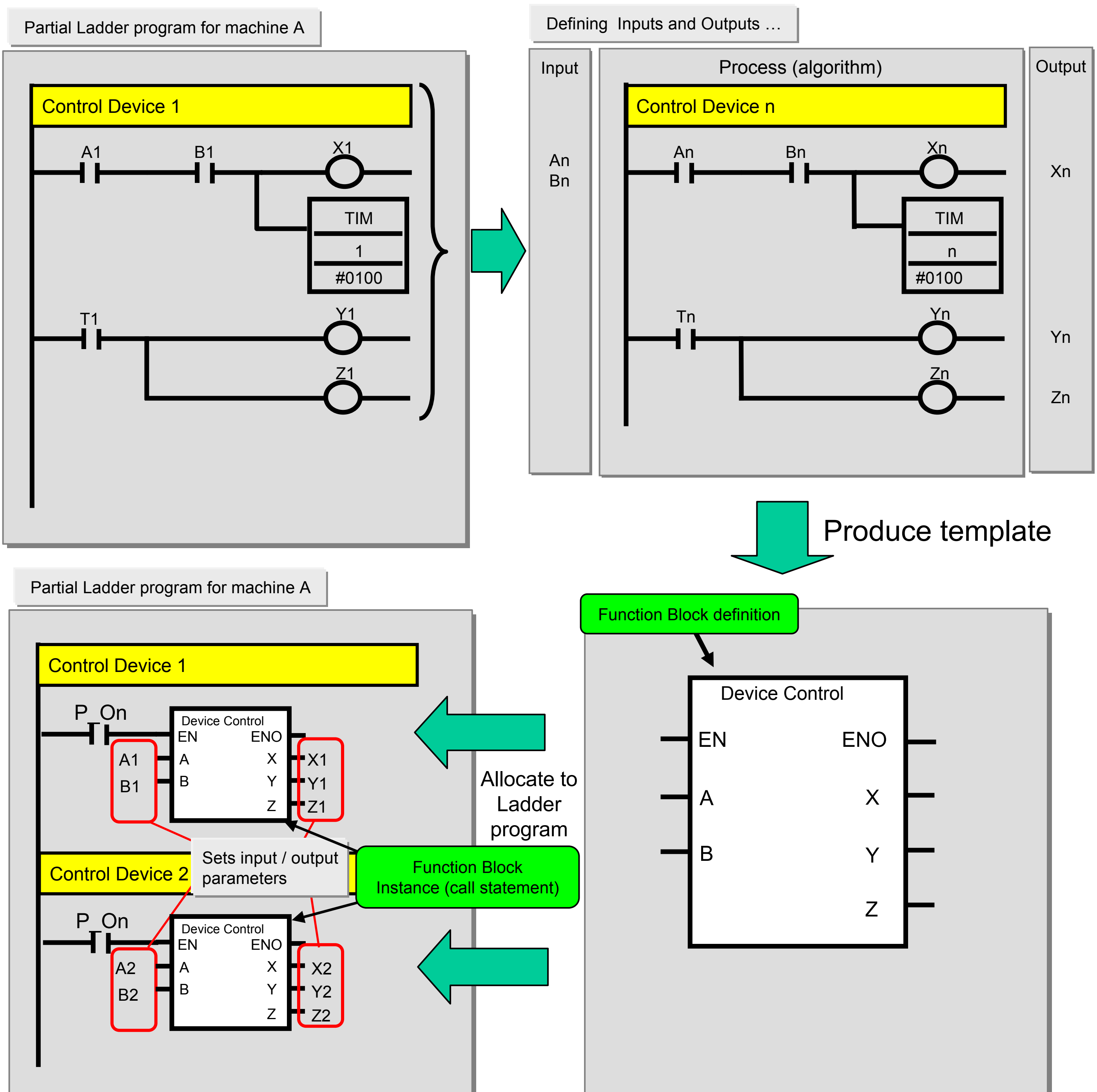
FB Generation Function
Existing PLC programs can be reused and easily converted to FBs. (CX-Programmer version 7.0 or higher)

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.

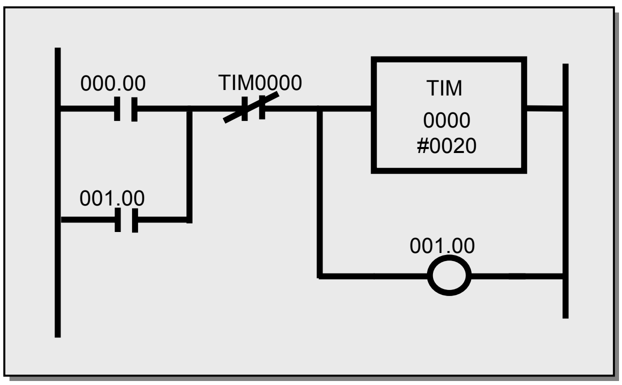


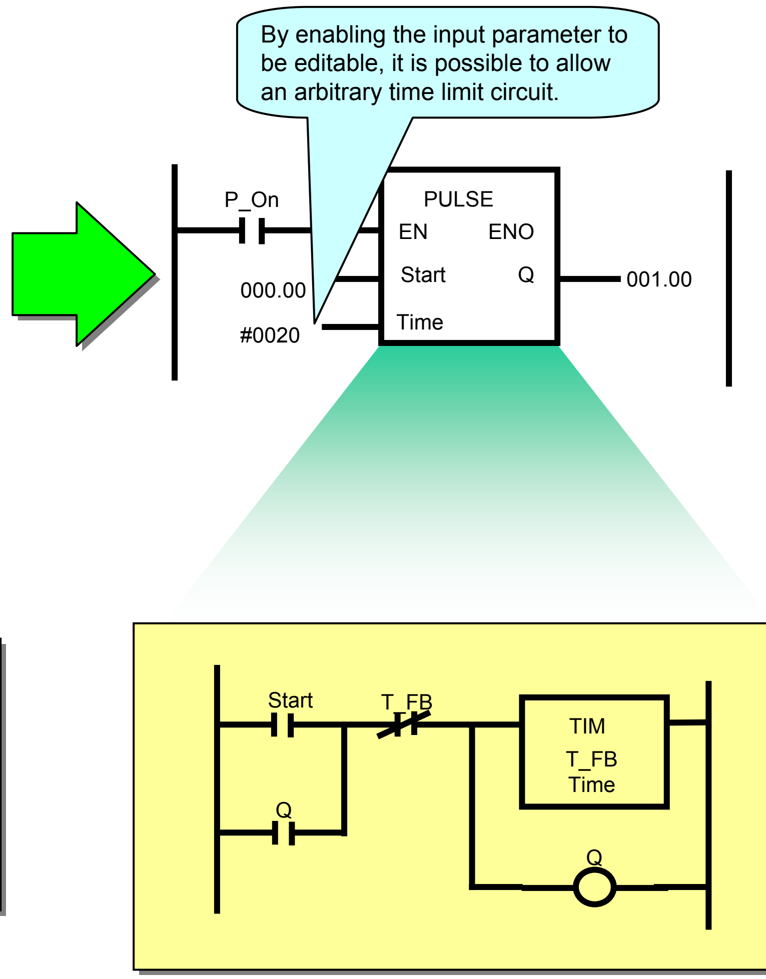
Function Block definition ··· This contains the defined logic (algorithm) and I/O interface. The memory addresses are not allocated in the Function Block Definition
Function Block instance (call statement) ··· This is the statement that will call the function block instance when used by the ladder program, using the memory allocated to the instance

2. An Example of a Function Block

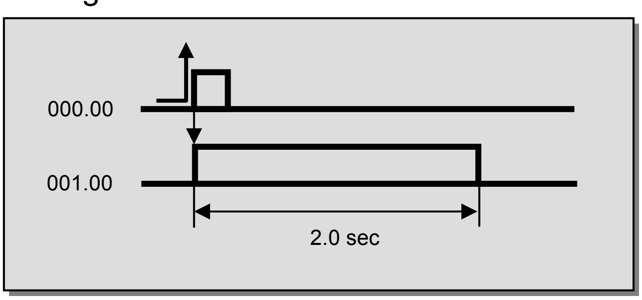
The following figures describe an example of a function block for a time limit circuit, to be used in the ladder. It is possible to edit the set point of the TIM instruction to reallocate the set time for turning off the output in the ladder rung. Using the function block as shown below, it is possible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

Ladder diagram





Timing chart



A function is also provided to generate function blocks based on existing ladder programs.
For details, refer to *Overview of Helpful Functions, Generating FBs Based on an Existing Ladder Program*.

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

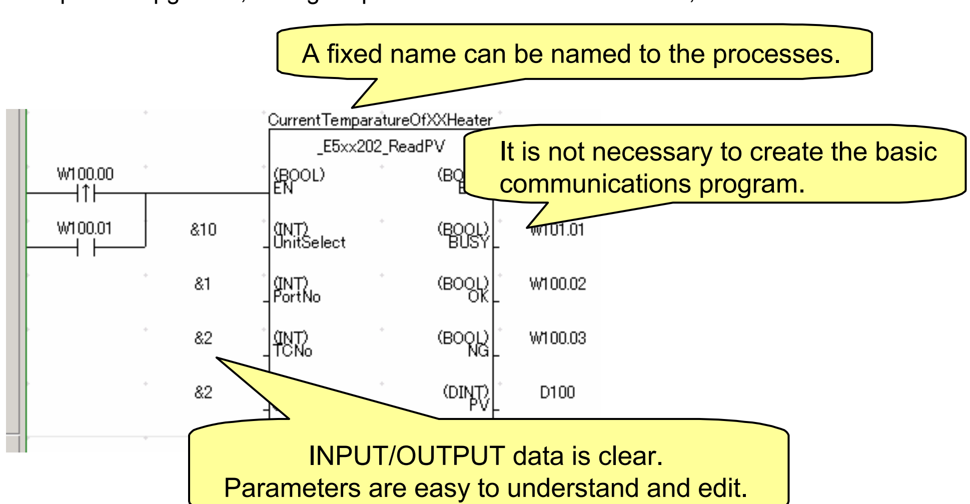
The function block has a clearly displayed name for its body and instances. A fixed name can be applied to the process.

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

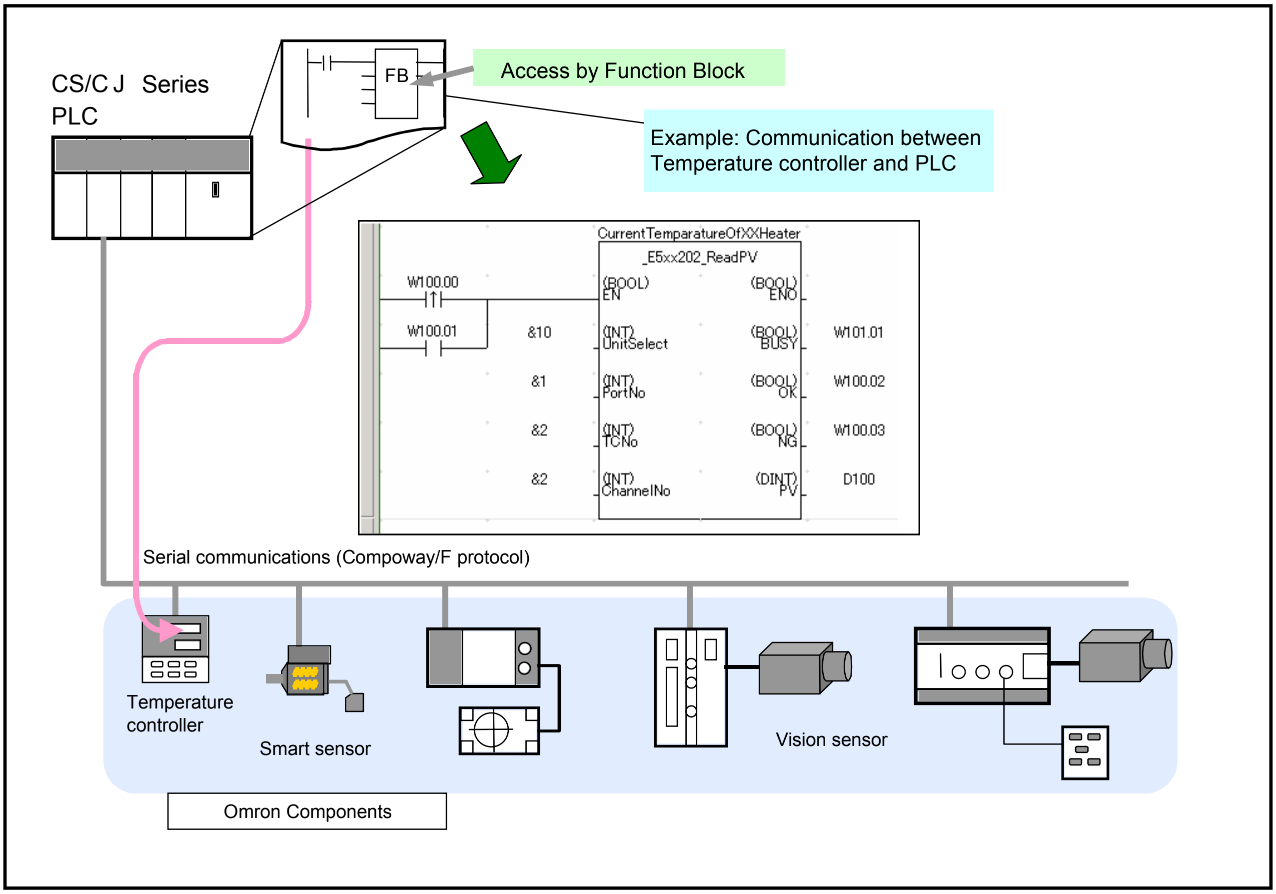
Omron will not change the interface between the ladder diagram and the function blocks. Units will operate by replacing the function block to the corresponding FB for the new unit in the event of PLC and the FA component upgrades, for higher performance or enhancements, in the future.



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.

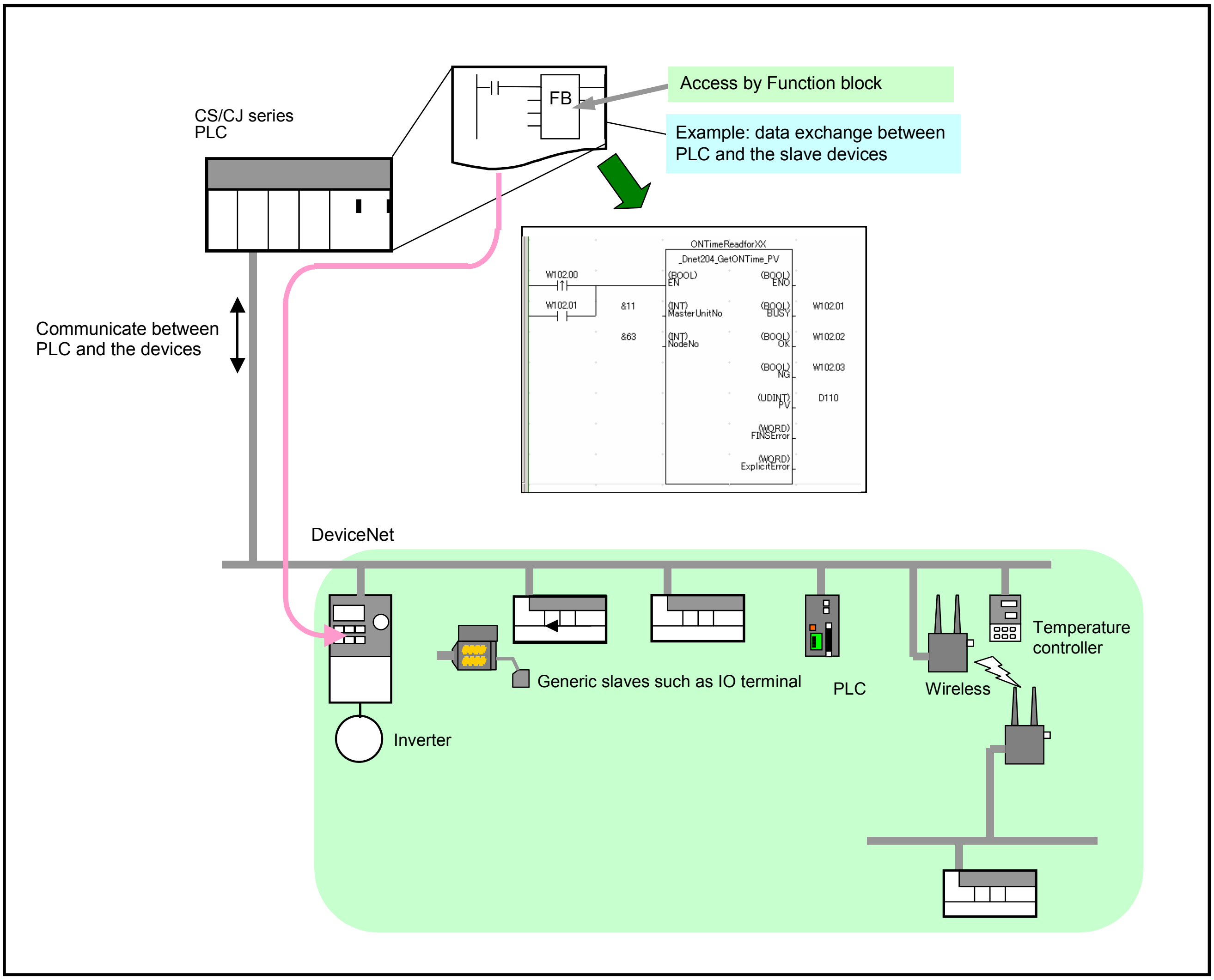
- Ability to configure low-cost communications (RS-232C/485)



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.

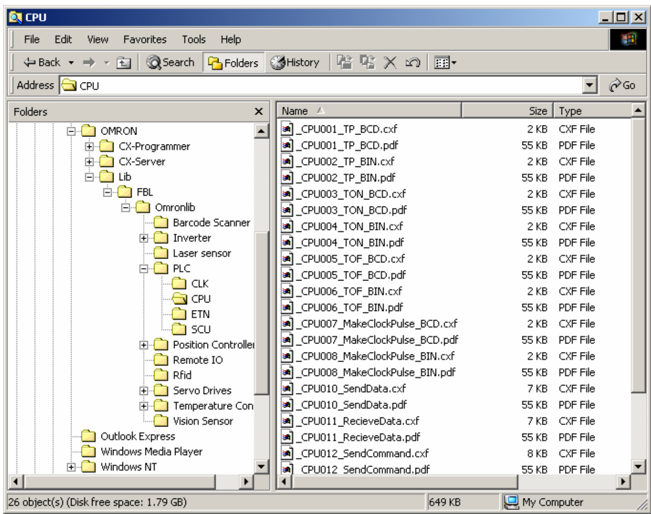


3-3. Content of the OMRON FB Library

The OMRON FB Library consist of the following:

3-3-1. OMRON FB Part Files

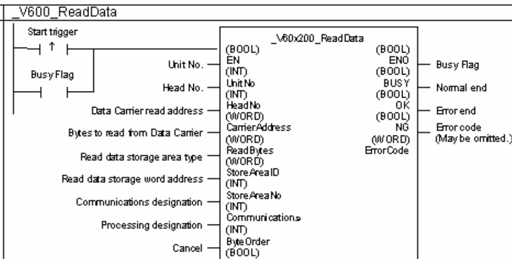
The OMRON FB Part file is prepared using the ladder diagram function block, for defining each function of the PLC unit and the FA component.
The files contain a program written in ladder diagram and have the extension .CXF.
The file name of the OMRON FB Part file begins with '_' (under score).
When the 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	_V600_ReadData
Symbol	



File name	¥Lib¥FBL¥English¥omronlib¥RFID¥V600¥_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 • 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	• 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).

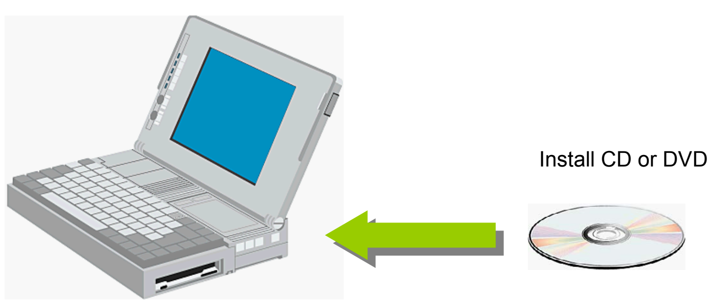
3-4. File Catalog and Where to Access the OMRON FB Library

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

Type	Target components
FA components	Temperature controller, Smart sensor, ID sensor, Vision sensor, 2 dimensions bar code reader, Wireless terminal
PLC	CPU unit, Memory card, Special CPU IO unit (Ethernet, Controller Link, DeviceNet unit, Temperature control unit)
Motion control components	Position control unit Inverter Servo motor driver

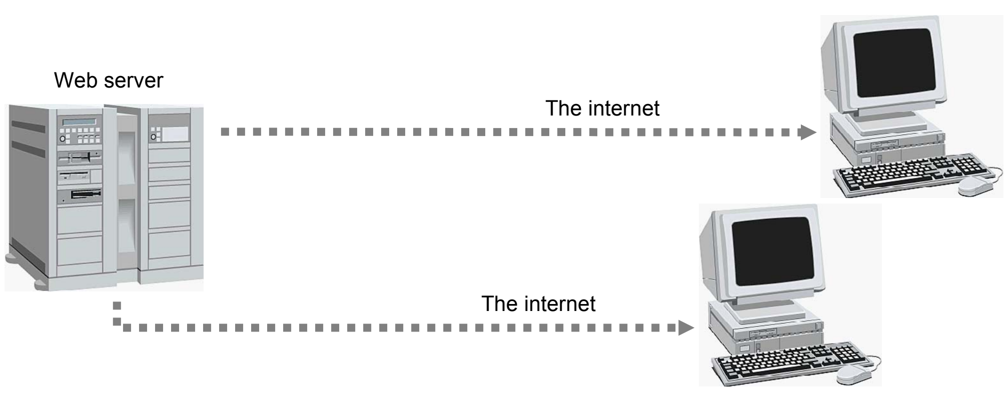
3-4-2. CX-One / CX-Programmer installation CD or DVD

OMRON FB Library is contained on the same install CD or DVD 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 on our Web site.



Chapter 2
How to use the OMRON FB Library

1. Explanation of the target program

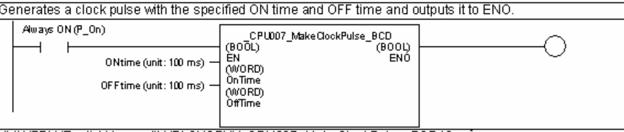
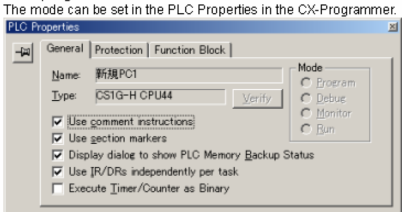
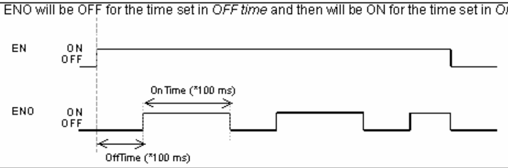
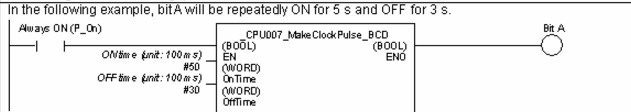
This chapter describes how to use OMRON FB Library using the OMRON FB Part file 'Make ON Time/OFF Time Clock Pulse in BCD'.

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.cxf
Applicable models	CS1-H, CS1-H, and CJ1M CPU Units
Conditions for usage	PLC Properties • 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.  Shared Resources • Timers
Function description	ENO will be OFF for the time set in *OFF time* and then will be ON for the time set in *ON time*. 
EN input condition	Connect the EN input to the Always ON Flag (P_On).
Restrictions Input variables	• 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	In the following example, bit A will be repeatedly ON for 5 s and OFF for 3 s. 
Related FBs	Use the correct FB for the timer/counter PV update mode set in the PLC Setup. Binary mode: Make ON Time/OFF Time Clock Pulse in Binary (_CPU008_MakeClockPulse_BIN) BCD mode: Make ON Time/OFF Time Clock Pulse in BCD (_CPU007_MakeClockPulse_BCD)

■ Variable Tables

Input Variables

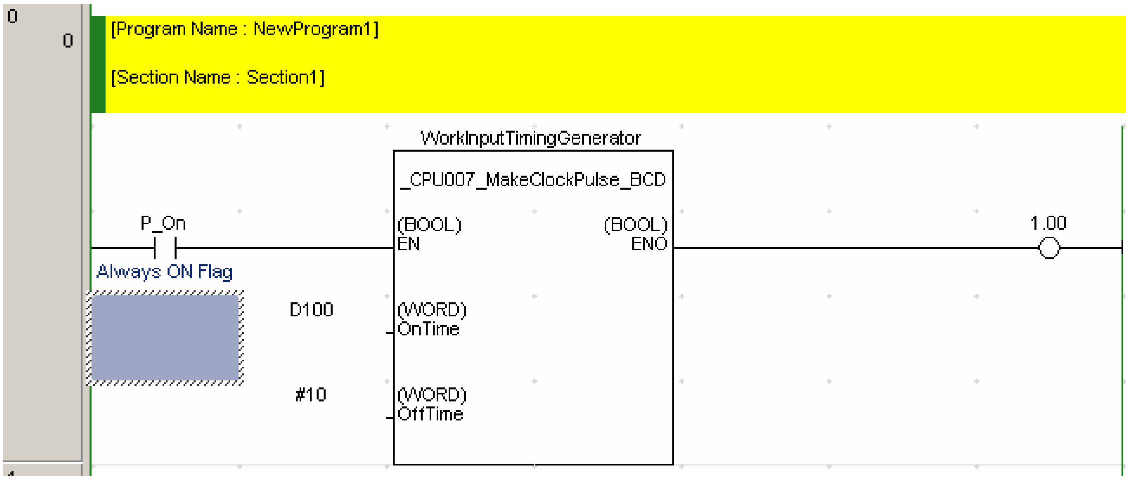
Name	Variable name	Data type	Default	Range	Description
EN	EN	BOOL			1 (ON): FB started 0 (OFF): FB not started.
ON time	OnTime	WORD		#0000 to #9999	Specify the ON time (unit: 100 ms). For example, #30 means 3 seconds.
OFF time	OffTime	WORD		#0000 to #9999	Specify the OFF time (unit: 100 ms). For example,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

Output Variables

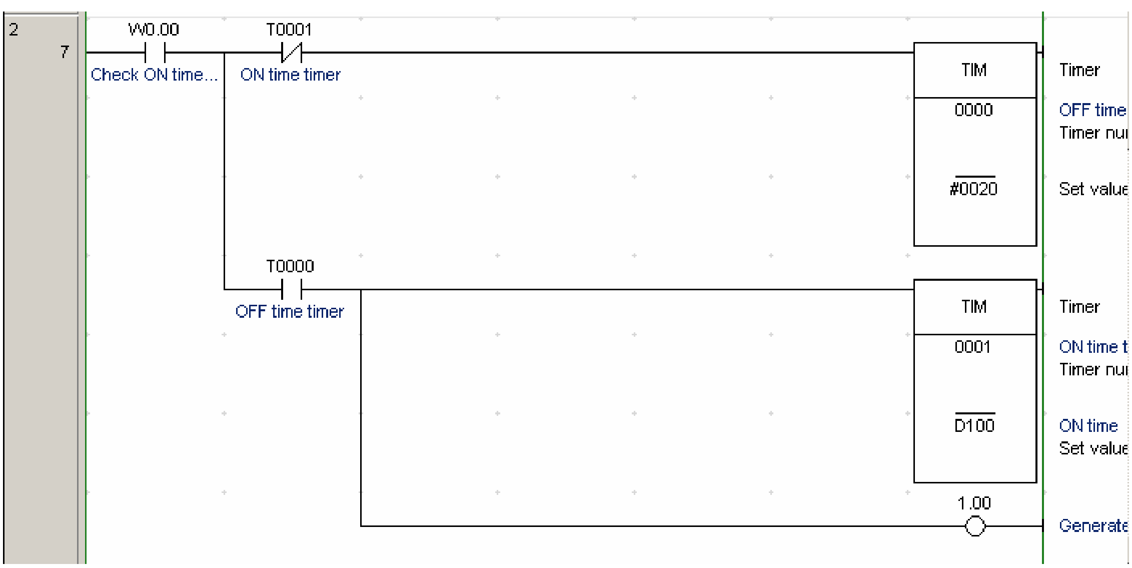
Name	Variable name	Data type	Range	Description
ENO	ENO	BOOL		Turns ON for the OnTime and OFF for the OffTime.

1-3. Input program

Create the following ladder program:-



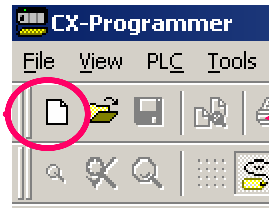
[Reference] If created as a straightforward ladder diagram, the program would be as below:-

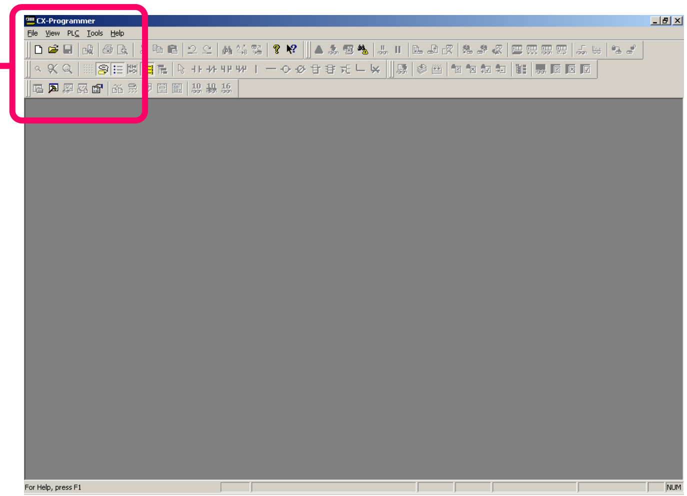


2. Opening a new project and setting the Device Type

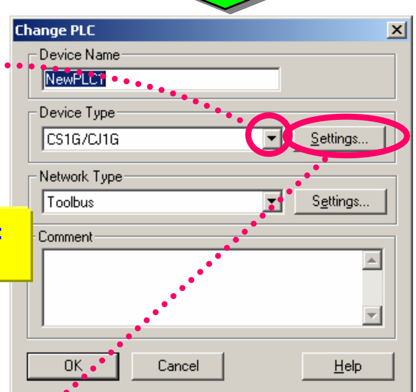
Click the toolbar button [New] in CX-Programmer.

Click





Click the left mouse button.

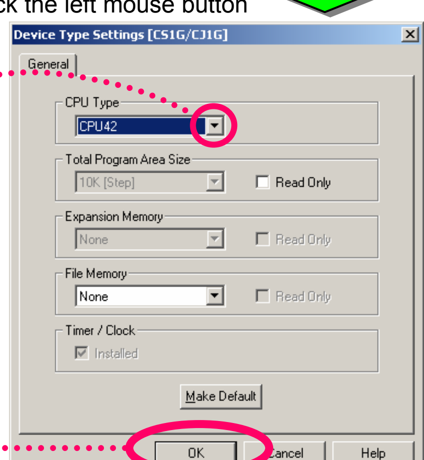


! To use Function Blocks, select the following PLCs:
CJ2H, CS1G-H, CS1H-H, CJ1G-H, CJ1H-H, CJ1M

Settings...

Click the left mouse button

Click the left mouse button to select CPU type.

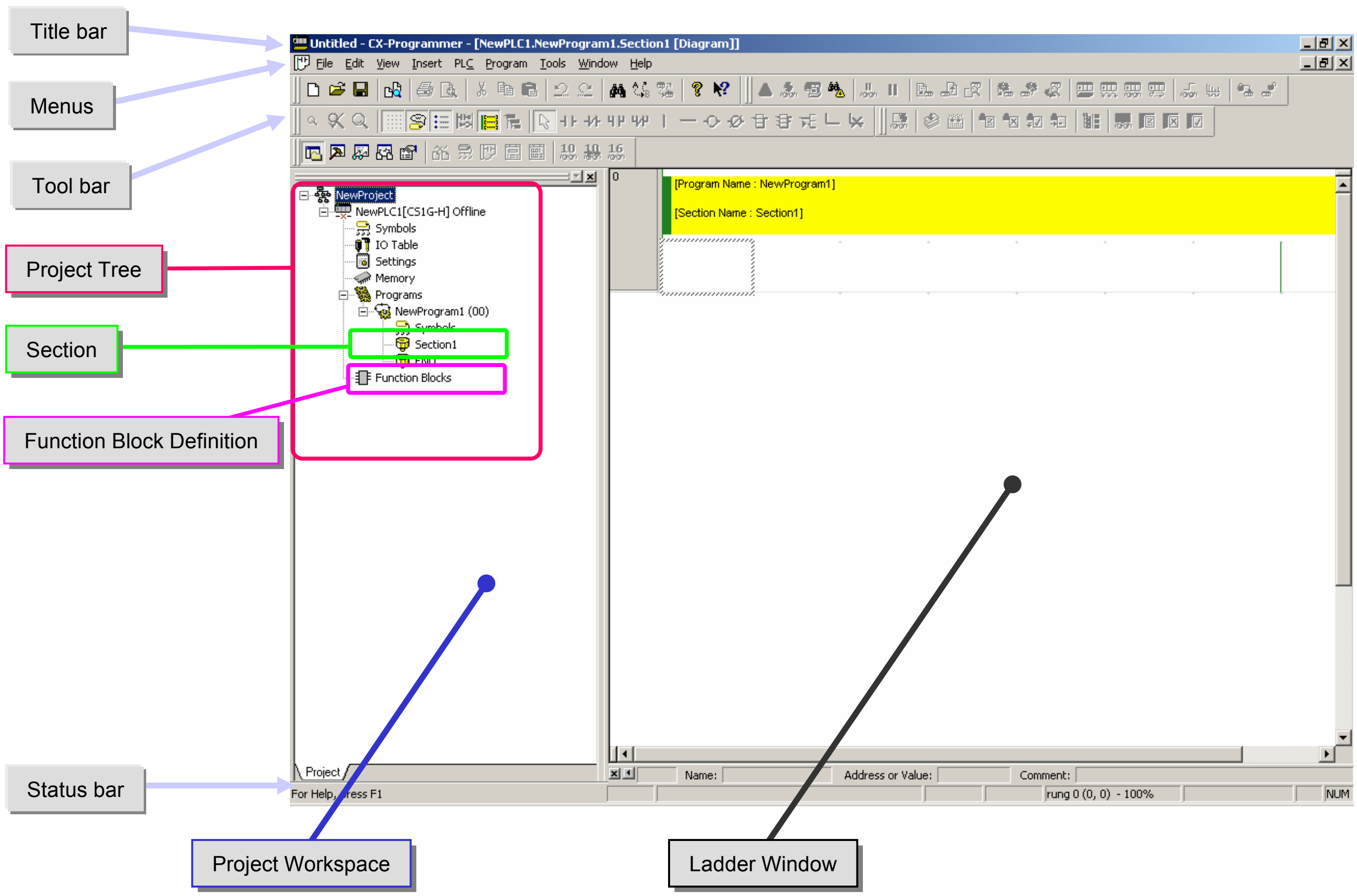


OK

Click [OK] to decide the selected CPU type.

3. Main Window functions

The main window functionality is explained here.

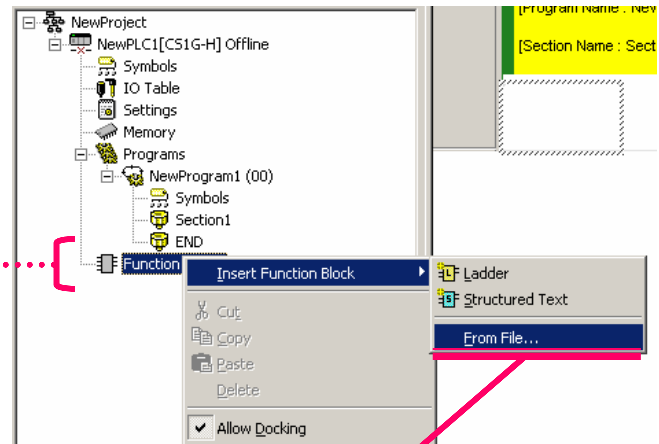


Name	Contents / Function
Title Bar	Shows the file name of saved data created in CX-Programmer.
Menus	Enables you to select menu items.
Toolbars	Enables you to select functions by clicking icons. Select [View] -> [Toolbars], display toolbars. Dragging toolbars enables you to change the display positions.
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.

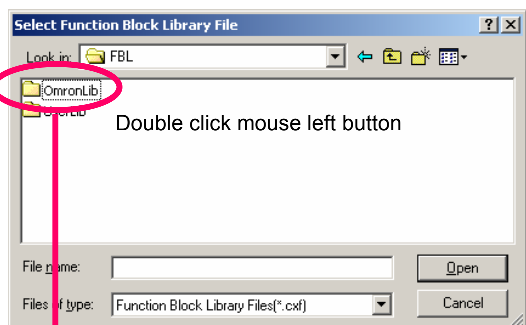
4. Import the OMRON FB Part file

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.

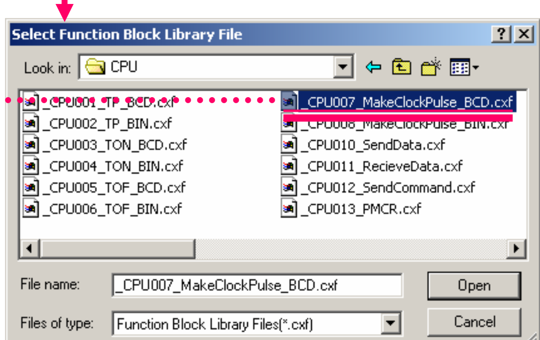


Select the necessary OMRON FB Part file in the 'Select Function Block Library' dialog.

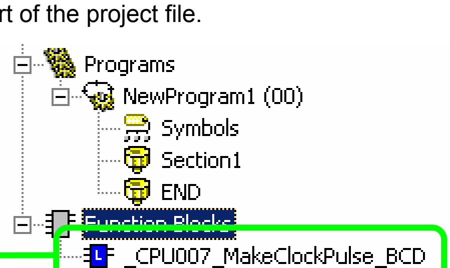
! The default path of the OMRON FB Library is C:\Program Files\Omron\CX-One\Lib\FBL.

Left Click '_CPU007_MakeClockPulse_BCD.cxf'

Left Click the [Open] button

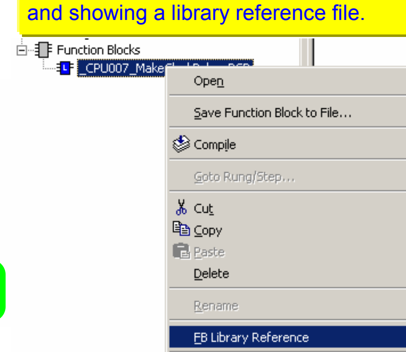


Function Block definition '_CPU007_MakeClockPulse_BCD' is registered as part of the project file.



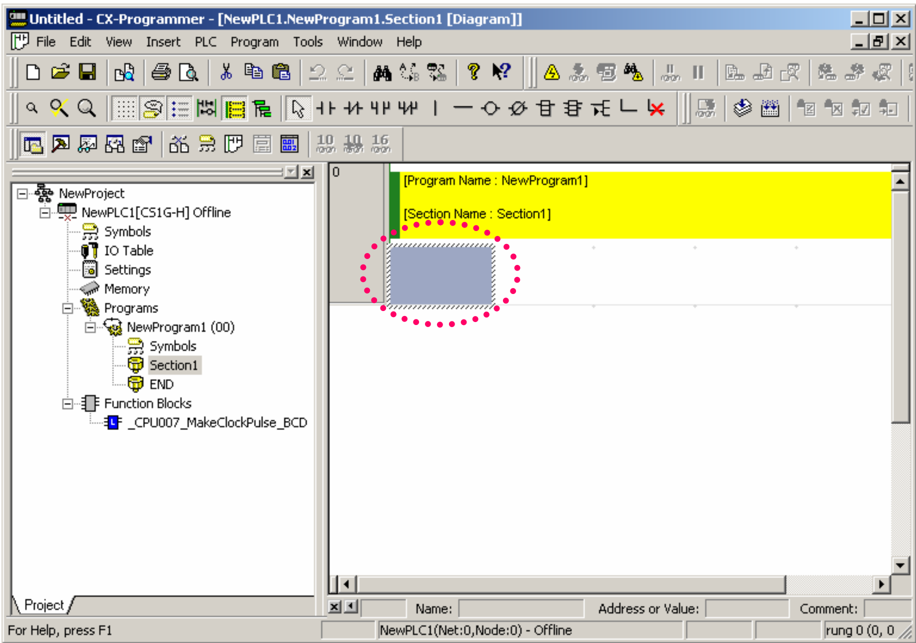
Function Block Definition

! 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.



5. Program Creation

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

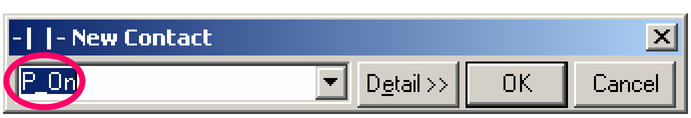


5-1. Enter a Normally Open Contact

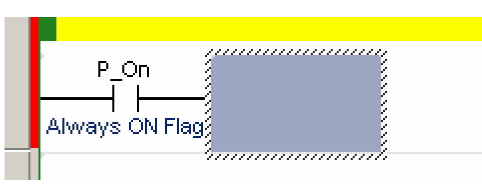
C

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

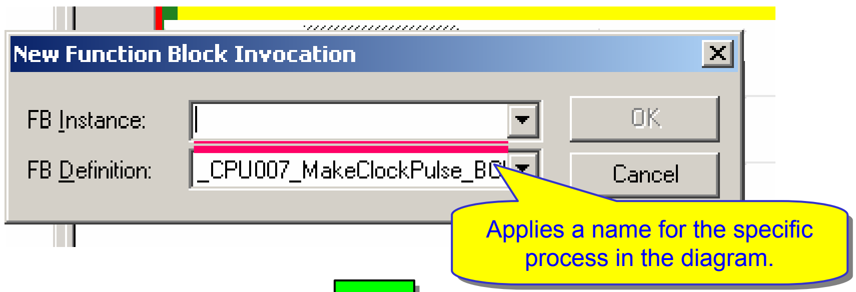
- Move the cursor to the command and then press the DEL key *or*
- Move the cursor to the right cell of the command and press the BS key.

“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.

5-2. Entering an Instance

F

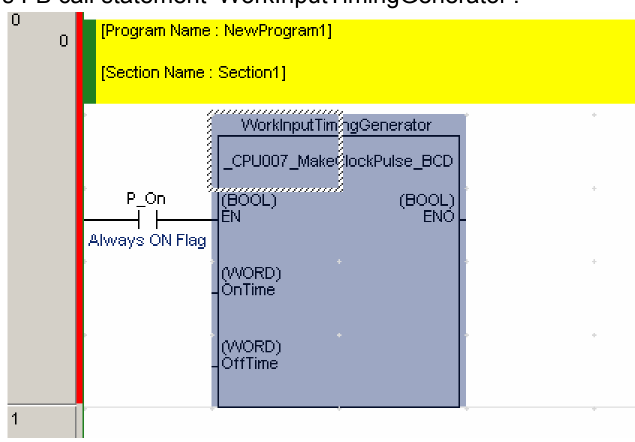
Press the [F] key on the keyboard to open the [New Function Block Invocation] dialog.



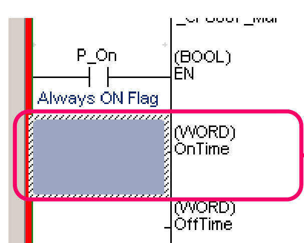
Enter text to create an FB instance name.
[WorkInputTimingGenerator]

ENT

Shows FB call statement ‘WorkInputTimingGenerator’.



5-3. Entering Parameters



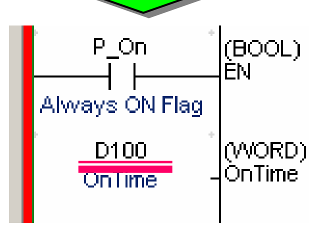
P or ENT

Move the cursor to the left of input parameter.

Enter the address.

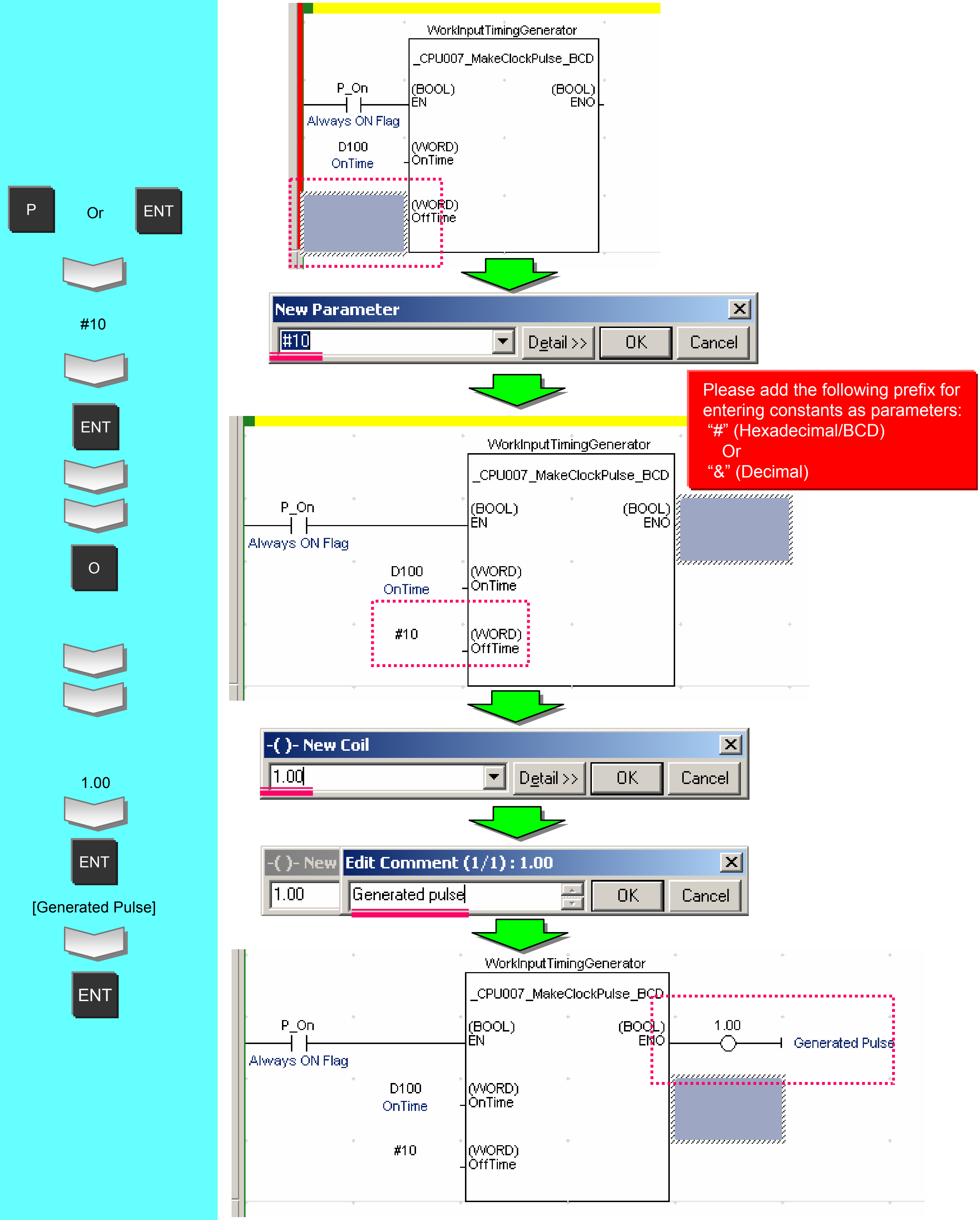
[d100]

ENT



Choose an address for the input parameter ‘OnTime’.

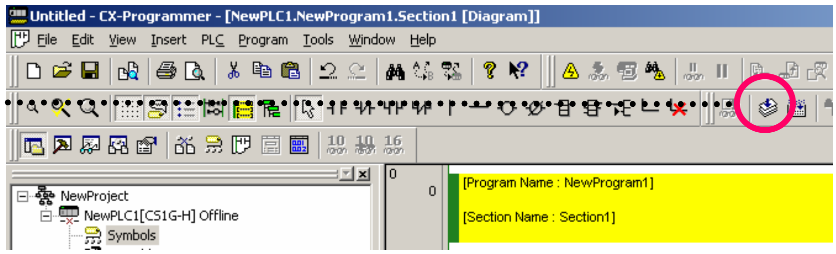
Enter the remaining parameters in the same way.

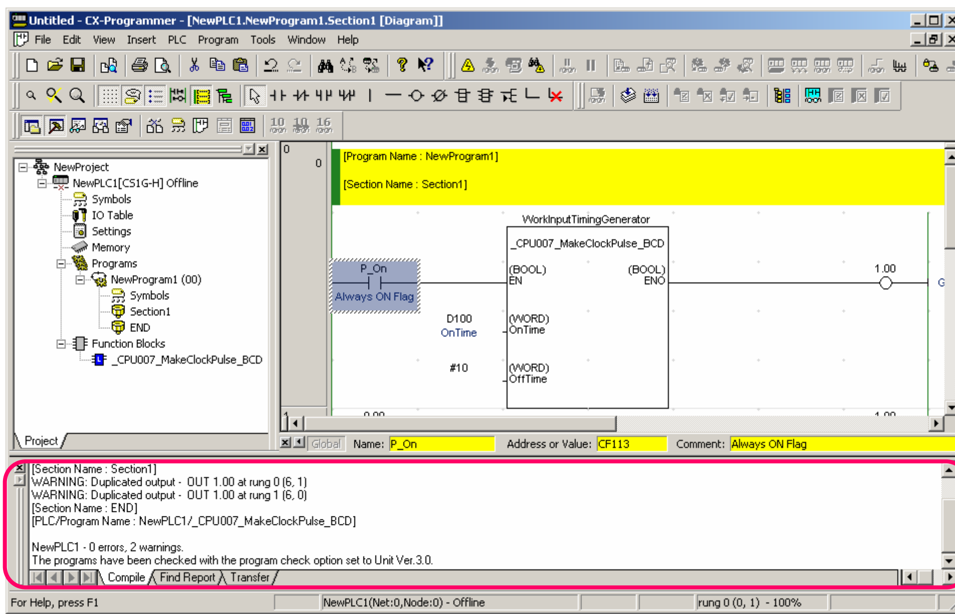


6. Program Error Check (Compile)

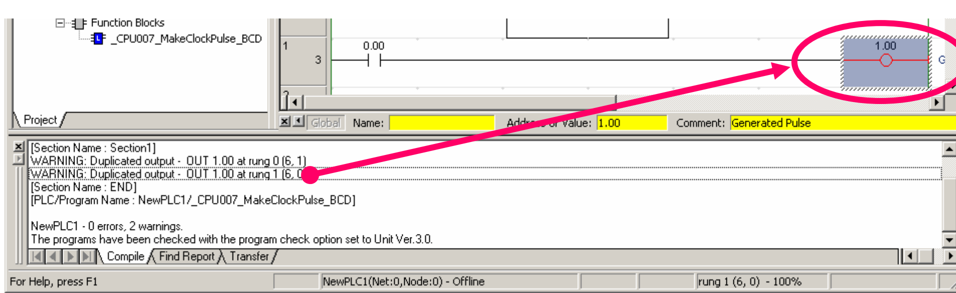
Before program transfer, check for errors using the program compile.

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.

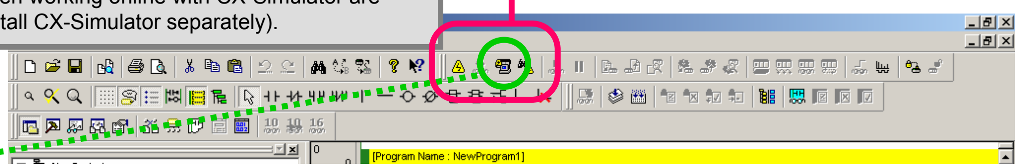
- Output Window automatically opens at program check.
- The cursor moves to an error location by pressing J or F4 key.
- Output Window closes by pressing the ESC key.

7. Going Online

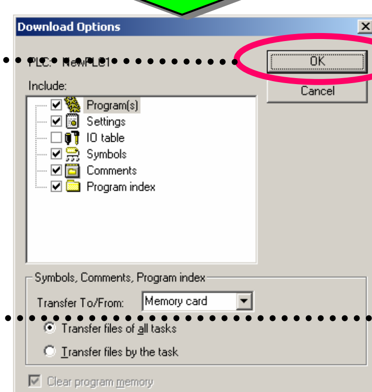
CX-Programmer provides three methods of connecting, depending on usage.

- 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).

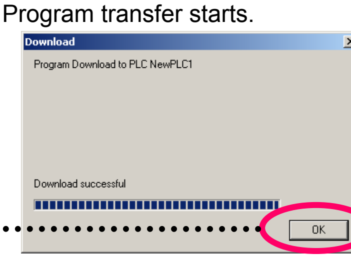


Click

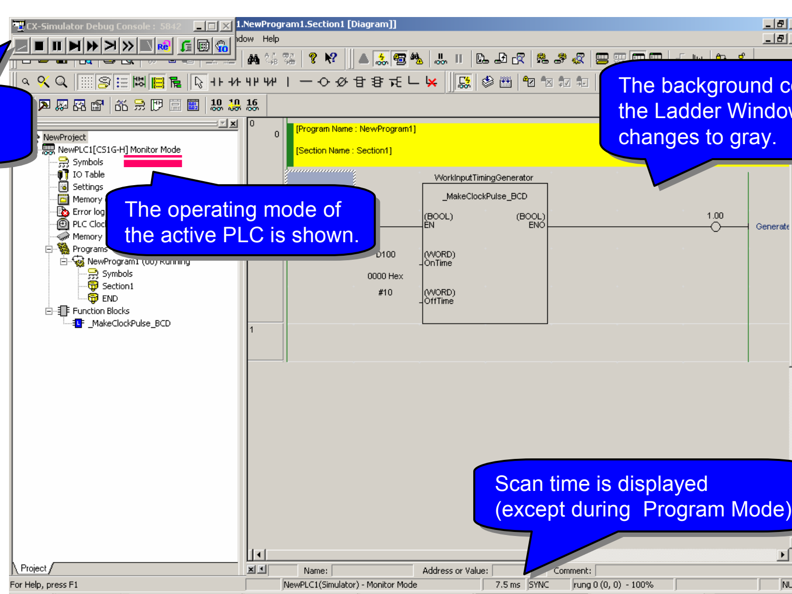


Click [OK]

Program transfer starts.



Click [OK]



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.

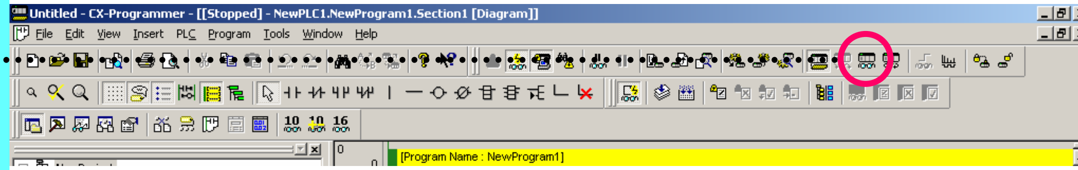
Scan time is displayed (except during Program Mode).

8. Monitoring - 1

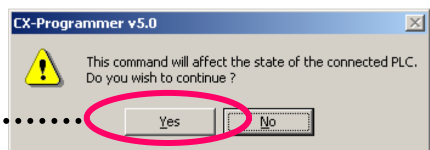
Change the PLC (Simulator) to Monitor mode.

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

Click 

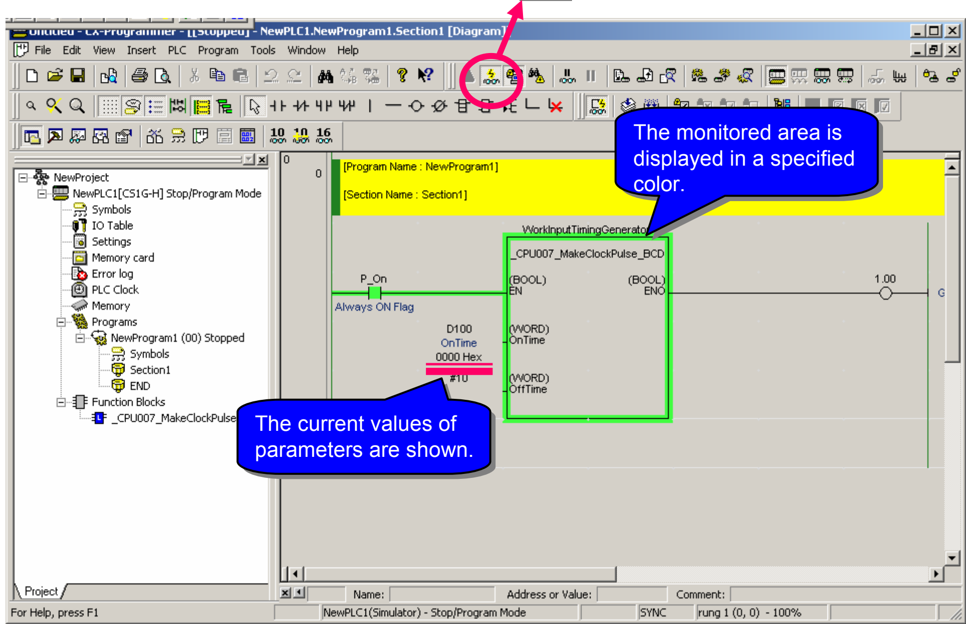


Click [Yes].



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.

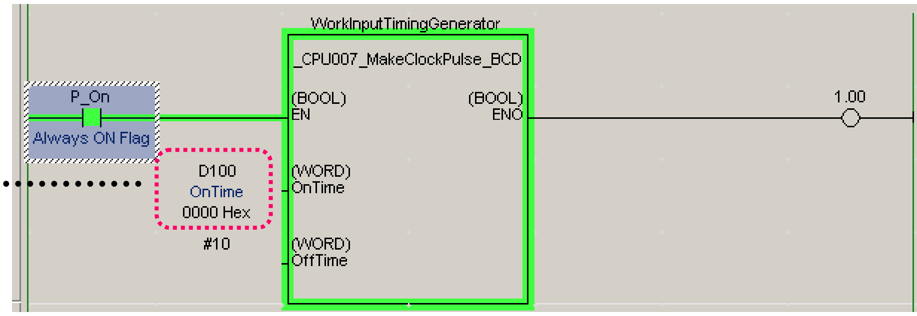
 ··· toggles PLC monitoring on/off



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’.



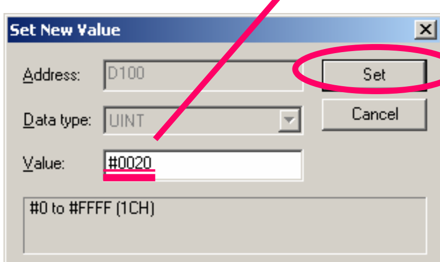
Click mouse right button and select the menu item [Set/Reset(S)] → [Setting Value (V)]

Or

Double click mouse left button.

Or

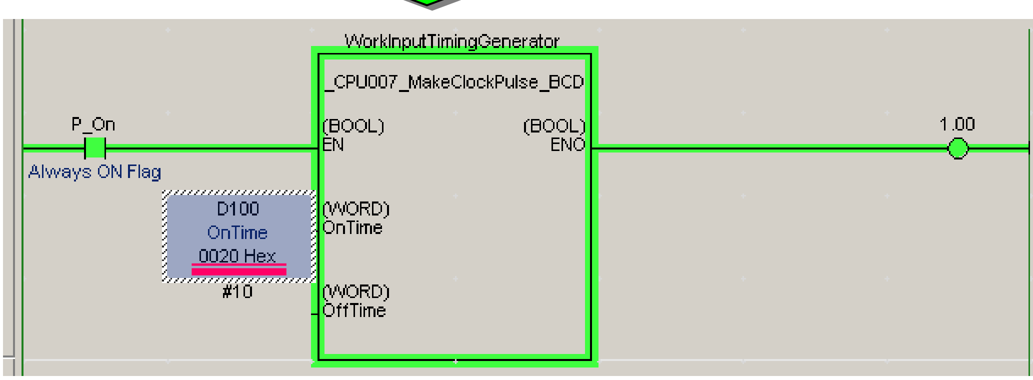
ENT



Change the current value of Input parameter.

Click [Set]

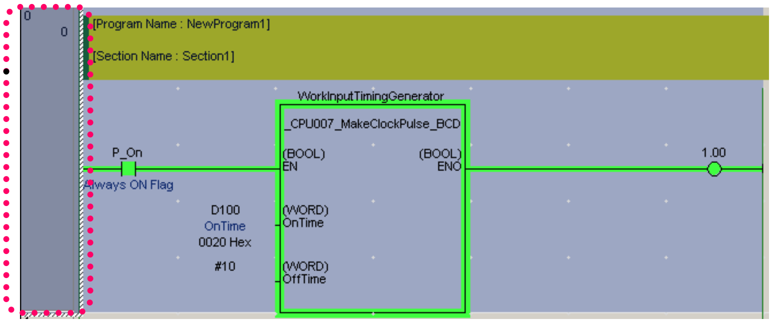
Please add the following prefix for entering constants as parameters:
“#” (Hexadecimal/BCD)
Or
“&” (Decimal)



10. Online Editing

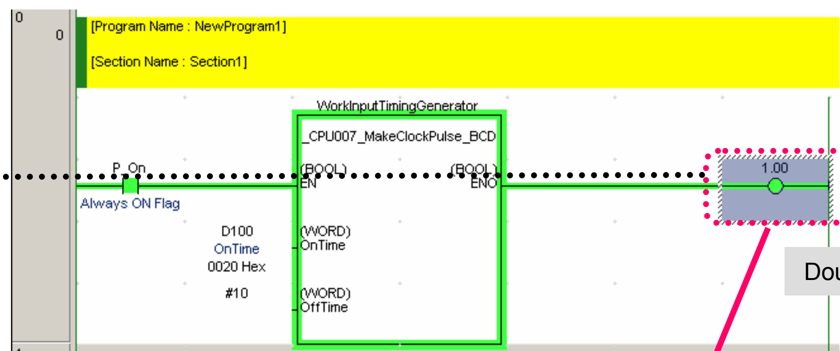
Move the cursor to the rung requiring modification.

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



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

Shortcut: [Ctrl]+[E]



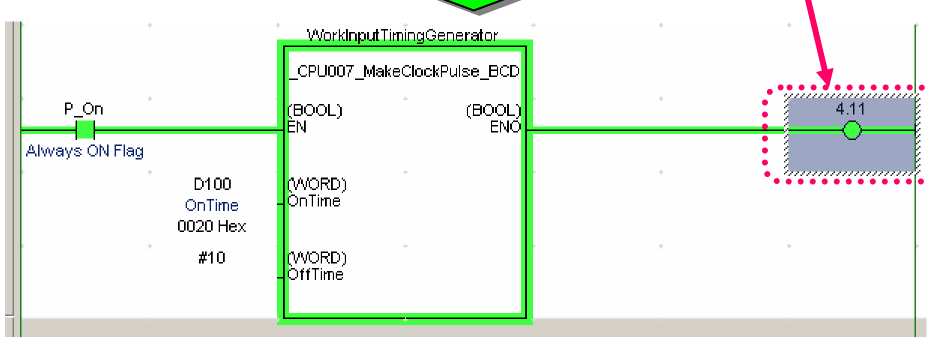
Move the cursor to the coil you want to modify. Double click the left mouse button.

Double click

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

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

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



End

Chapter 3

Customize the OMRON FB Part file

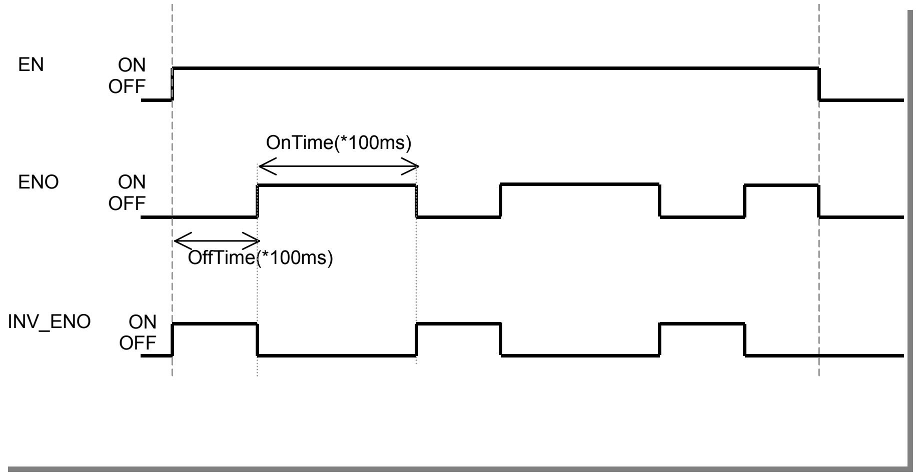
Function Block

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

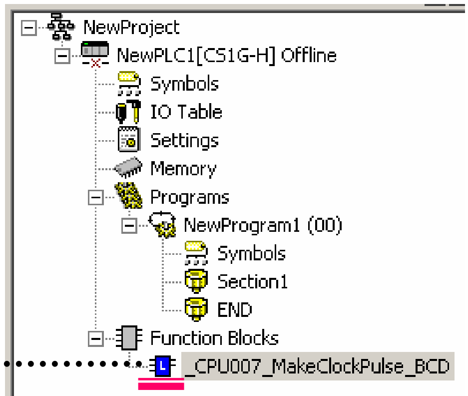
To satisfy the requirement described above, the following changes must be made to OMRON FB Part file 'Make ON Time/OFF Time Clock Pulse in BCD'

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

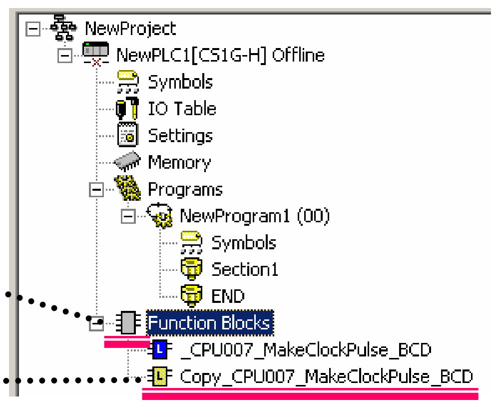
Caution
In particular, when you customize OMRON FB parts, read *CX-Programmer Operation Manual*: Function Blocks and Structured Text before customization to sufficiently understand the specifications of the FB function. After customization, further, please be sure to sufficiently verify the operation for the created FB definitions before proceeding with the actual operation. OMRON cannot guarantee the operation of customized OMRON FB parts. Please note that we cannot answer the questions about customized OMRON FB parts.

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)



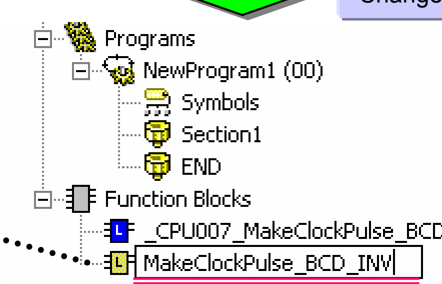
Select the OMRON FB Part icon then right click the mouse.
→ Copy



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

The OMRON FB Part file is pasted.

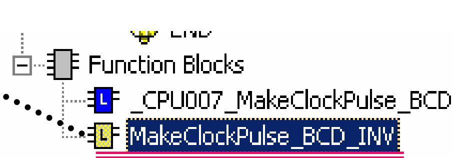
Change the FB definition name.



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 ‘_’.

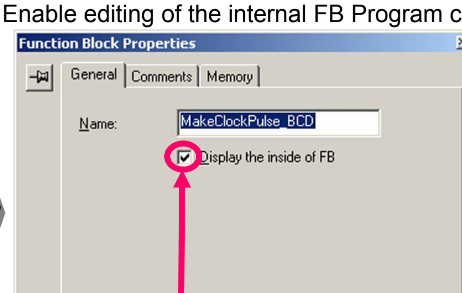
Enable editing of the internal FB Program code.



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

Or

ALT + ENT

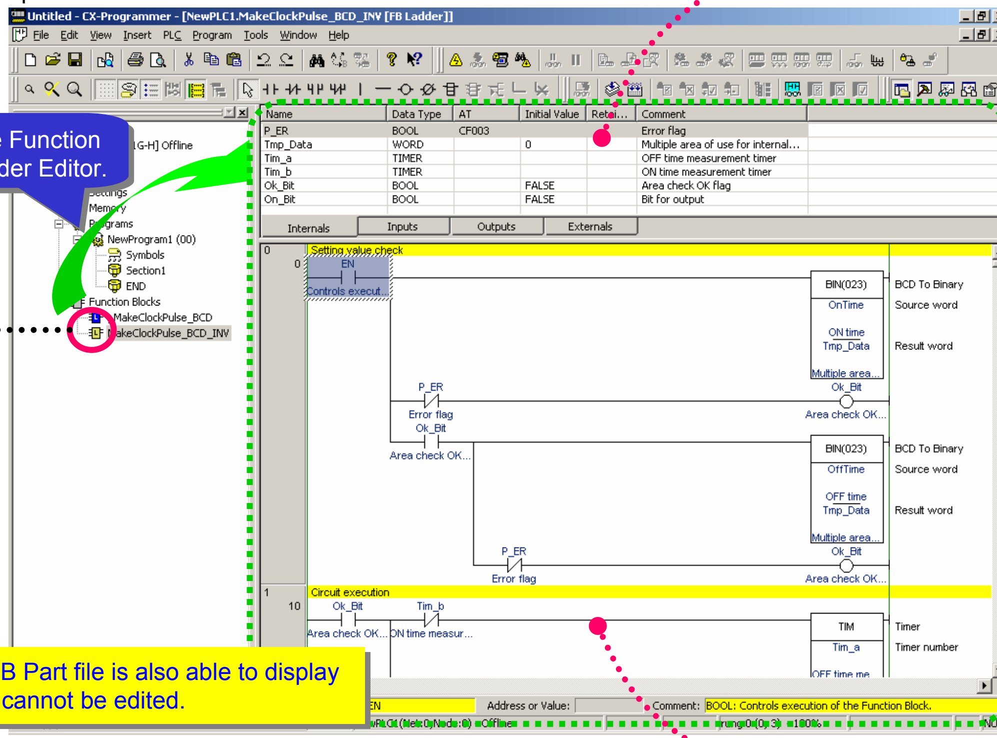


Tick the check box using the left mouse click.

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

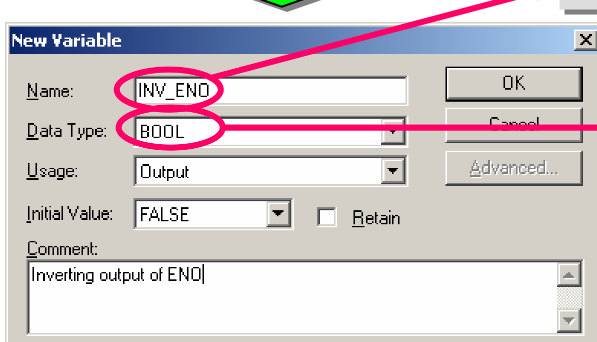
Name	Data Type	AT	Initial Value	Retai...	Comment
ENO	BOOL		FALSE		Indicates successful execution ...

Internals Inputs Outputs Externals

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.



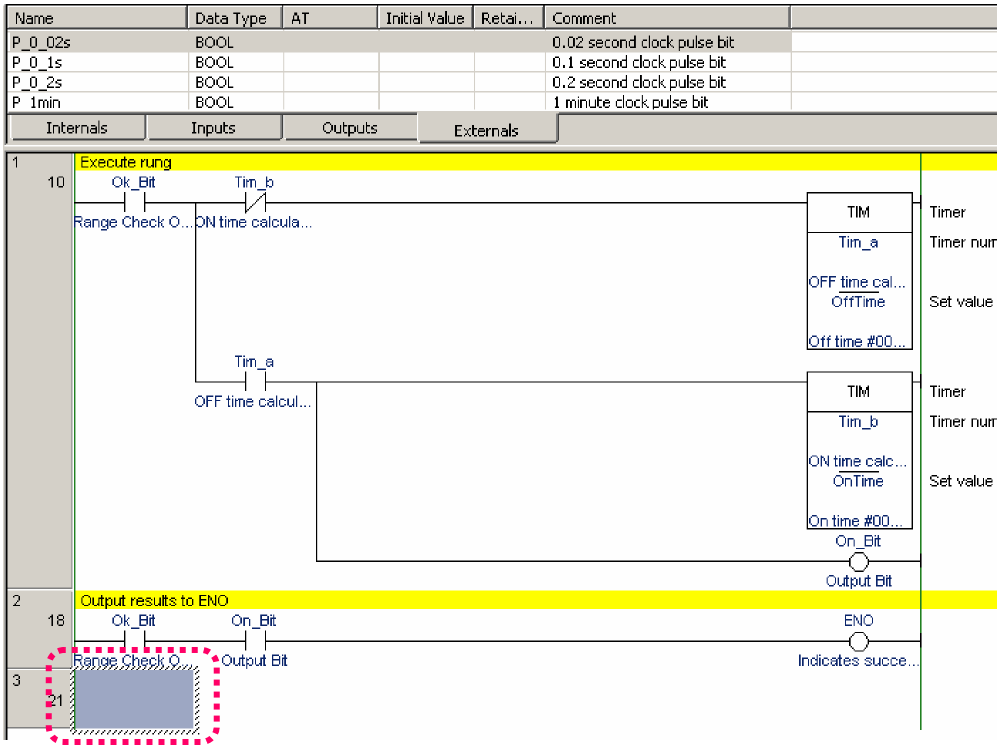
Select BOOL for bit data.

Confirm the entered variable is correct.

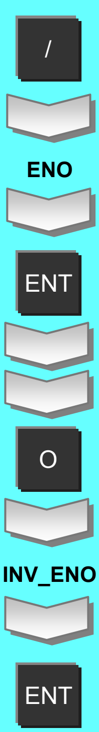
Name	Data Type	AT	Initial Value	Retai...	Comment
ENO	BOOL		FALSE		Indicates successful execution ...
INV_ENO	BOOL		FALSE		Inverting output of ENO

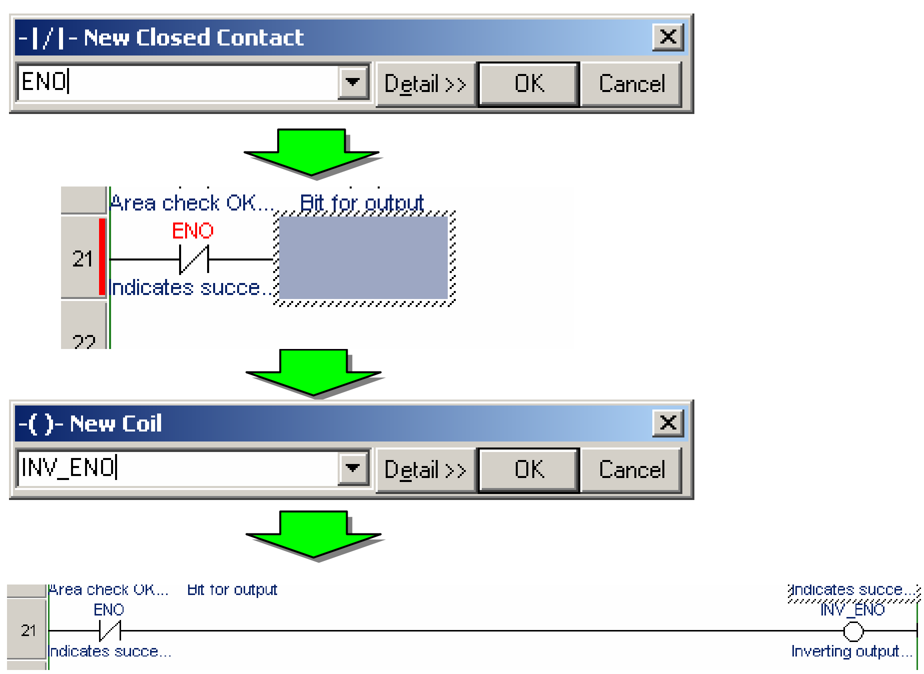
4. Changing the Function Block Ladder

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



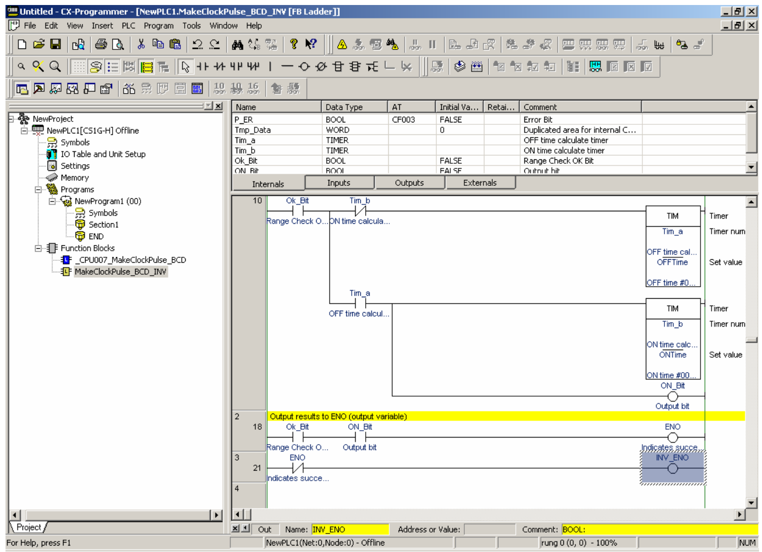
4-1. Entering a Contact





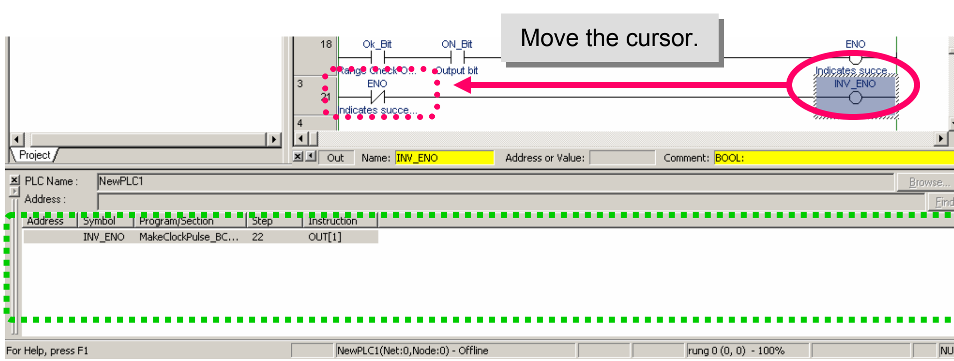
4-2. Checking Usage Status of Variables

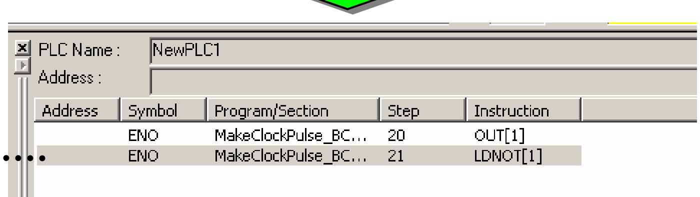
As with main ladder program, you can use cross reference pop-up to check usage conditions of variables.



Display cross reference pop-up.

Alt + 4

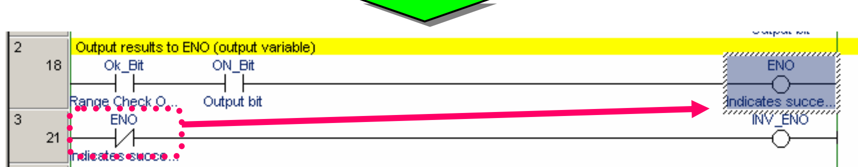




Select LDNOT from cross reference pop-up by the mouse cursor.

↑

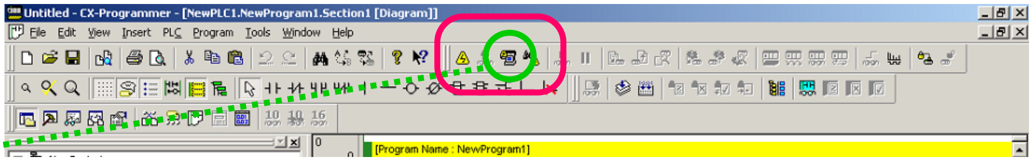
You can see that variable ENO is used in an output coil in the step No.20 as well.



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

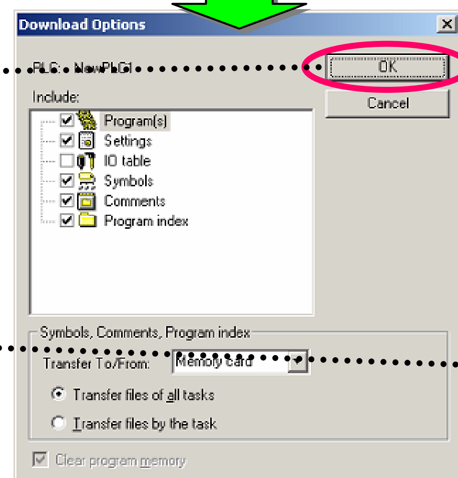
5. Transferring to the PLC

Transfer the program to the PLC after the function block definitions used in the program have been created by customizing the Smart FB Library versions.

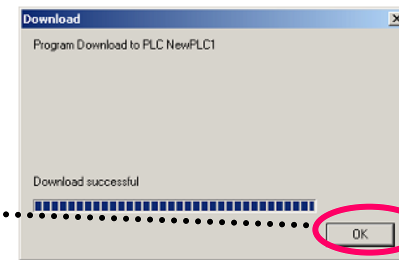


Click 

Click the **OK** Button.



Program transfer starts.



Click the **OK** Button.

6. Verifying Operation

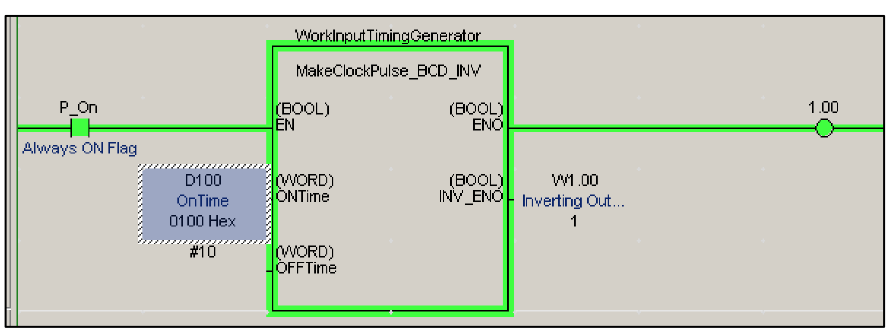
Program operation is verified and debugged while changing the value of D100 (ON time), which is specified in the function block's parameters.

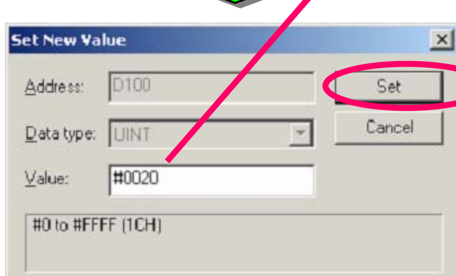
Right-click to display the pull-down menu and select ***Set/Reset – Set value***.

OR:
Double-click the left mouse button.

OR

ENT





Changes the input parameter's PV.

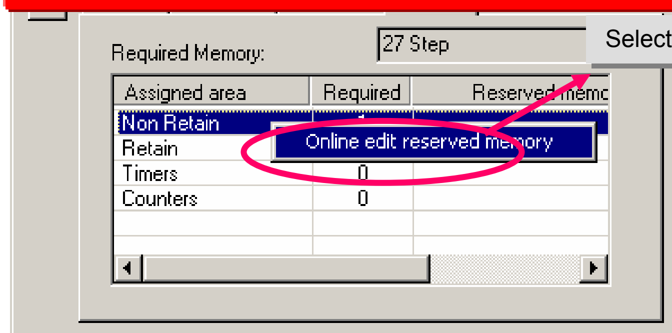
Click the **Set** Button.

When inputting a constant, always input the # prefix (for hexadecimal or BCD) or & prefix (for decimal) to the left of the number.

7. Online Editing of Function Blocks

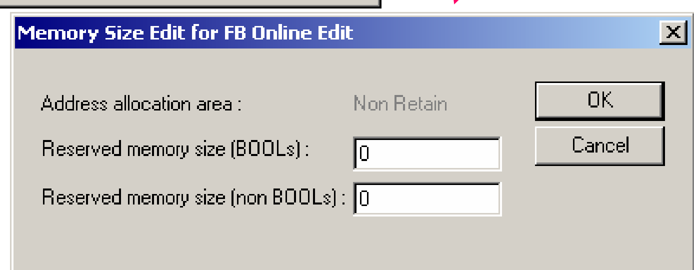
When adding a variable (internal variable) with FB online editing, memory must be allocated offline in advance in the Memory Tab of the Function Block Properties Window.

Select the function block definition that you want to edit online, right-click to display the pull-down menu, and select ***Properties***.

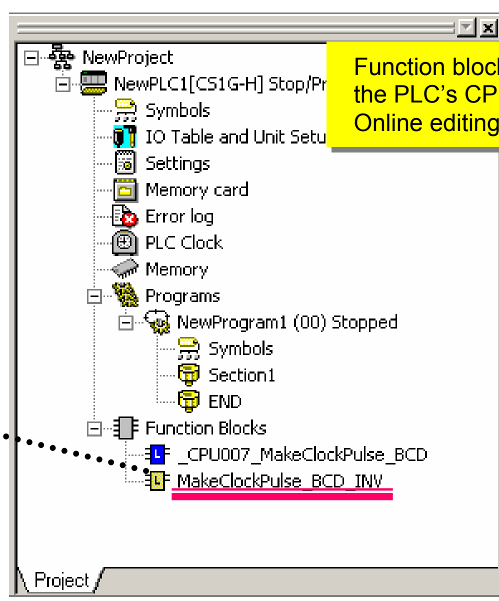


Select ***Online edit reserved memory***.

Select the variable area where you want to add a variable in online editing, right-click to display the pull-down menu, and select ***Online edit reserved memory***.

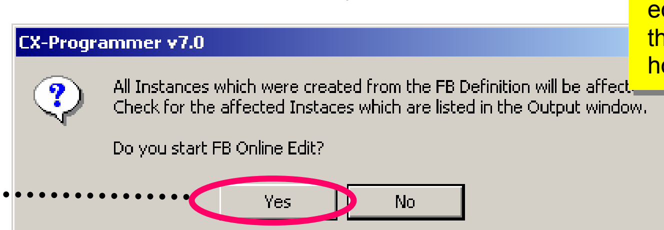


Edit the function block definition online.



Function block definitions can be edited online only if the PLC's CPU Unit is unit version 4.0 or later. Online editing cannot be used in CX-Simulator.

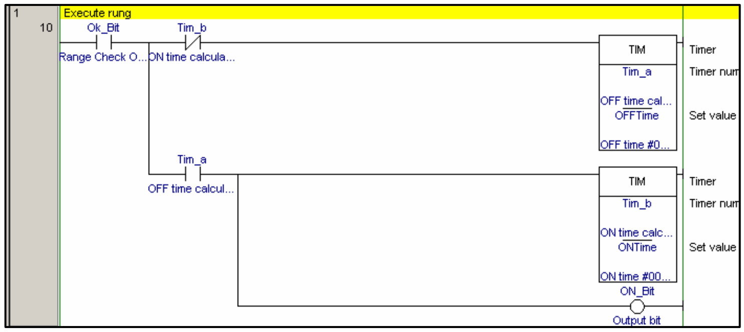
Select the function block definition that you want to edit online, right-click to display the pull-down menu, and select ***FB Online Edit – Begin***.



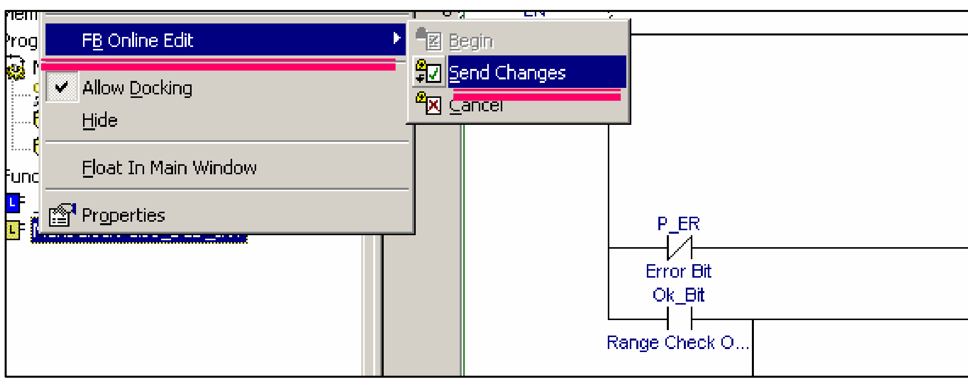
It is possible that the FB definition is called from more than one location, so start editing only after checking the output window to verify how the FB definition is used.

Click the **Yes** Button.

Edit the program section.

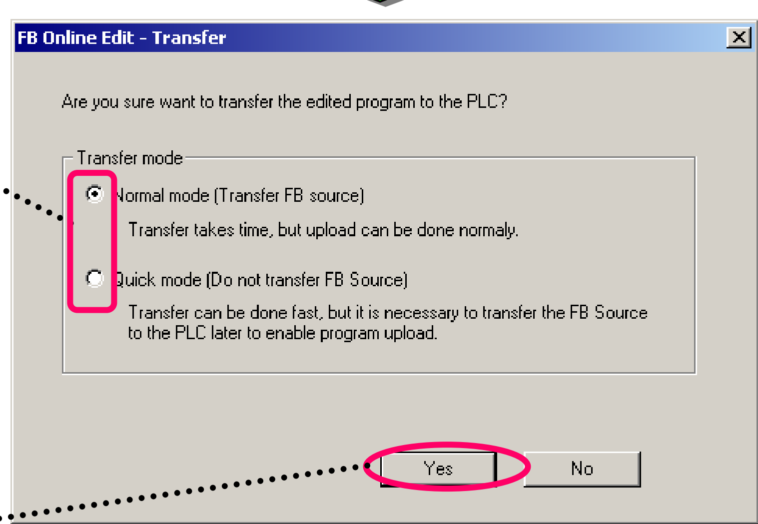


After editing the program section online, right-click to display the pull-down menu, and select ***FB Online Edit – Send Changes***.

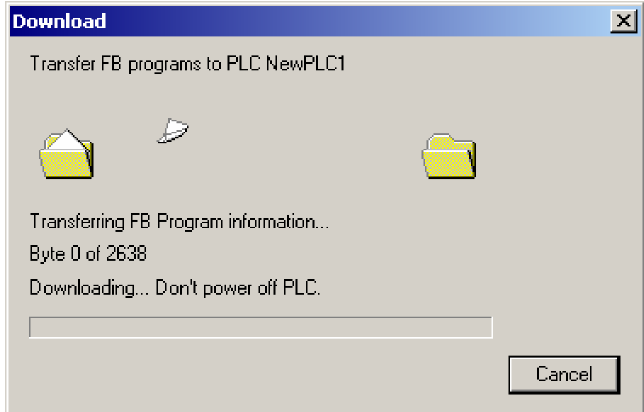


Select the online editing transfer mode.

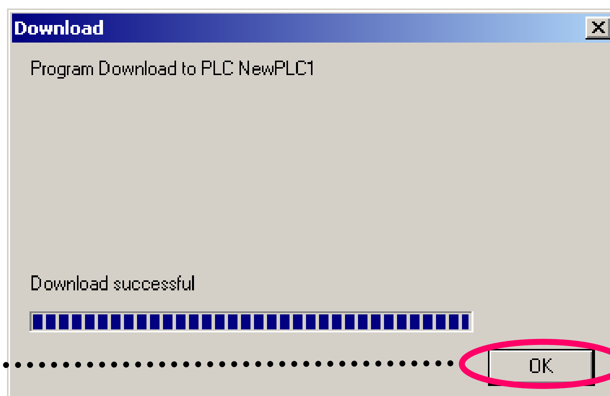
- Normal mode:
The FB source information is transferred.
- Quick mode:
The FB source information is not transferred.



Click the **Yes** Button.

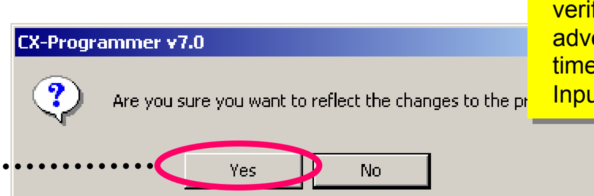


The FB definition information will be transferred.



Click the **OK** Button.

Click the **Yes** Button after verifying that there will be no adverse effects even if the cycle time is longer.
Input signals may be missed.



Click the **Yes** Button.

End

Chapter 4

How to use the ST(Structured Text)language

Function Block

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 Ver. 5 or higher is in conformance with IEC61131-3.
The arithmetic functions in CX-Programmer Ver. 5 or higher are as follows:

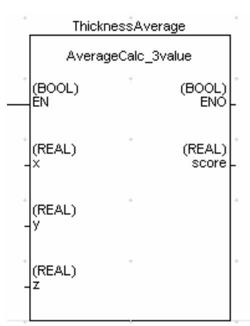
sine (SIN), cosine (COS), tangent (TAN), arc-sine (ASIN), arc-cosine (ACOS), arc-tangent (ATAN), square root (SQRT), absolute value (ABS), logarithm (LOG), natural-logarithm (LN), natural-exponential (EXP), exponentiation (EXPT)

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

(* CRC16 *)
CRCTMP := 16#FFFF;
FOR I := 1 TO N DO
    CRCTMP := CRCTMP XOR XMT[1];
    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;

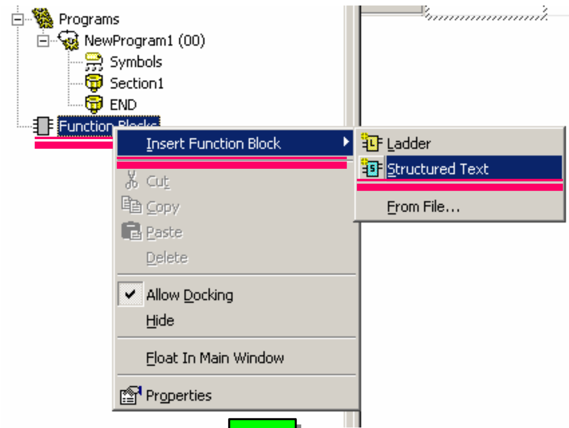
Substitute a value to a symbol is expressed by " := ".

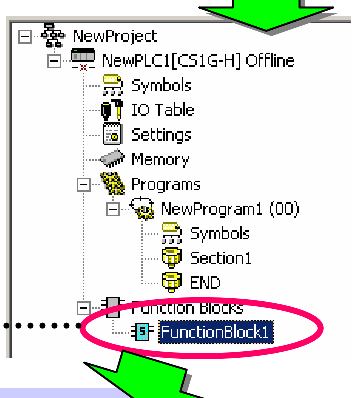
Enter " ; " (semicolon) to complete the code.

3. Create a Function Block using ST

Create a Function Block using Structured Text.

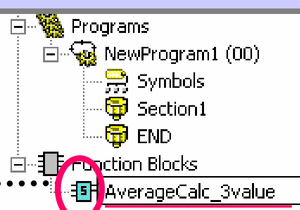
Select the Function Block icon using a mouse cursor, and click the right mouse button.
→ Insert Function Block(I)
→ Structured Text(S)





A New Function Block definition is created.

Change the Function Block definition name

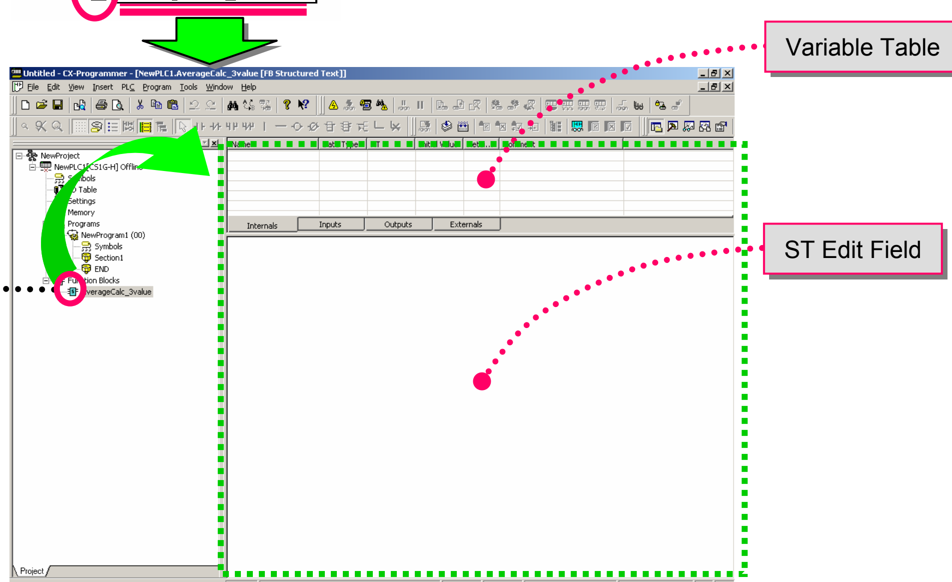


Select the Function Block definition icon using the mouse cursor and right click the mouse button.
Select Paste.
→ Rename
Enter
[AverageCalc_3value]

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

Open Function Block ST Editor

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



4. Entering Variables into Function Blocks

Select Variable Table.

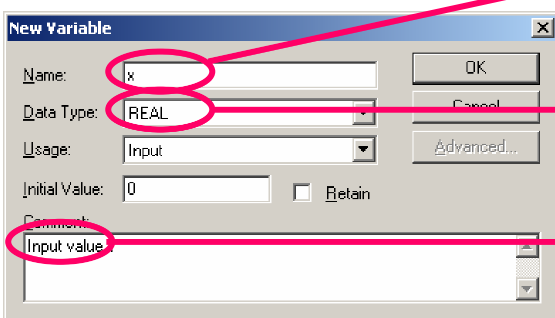
Name	Data Type	AT	Initial Value	Retai...	Comment
EN	BOOL		FALSE		Controls execution of the Func...

Internals | Inputs | Outputs | Externals

Select the Input tab using the mouse cursor.

Select Insert from the Pop-up menu.

Enter data for the following.
Name
Data type
Comment



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	Retai...	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	Retai...	Comment
ENO	BOOL		FALSE		Indicates successful execution ...
score	REAL		0.0		Avrage

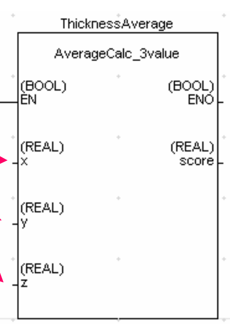
Output Variables

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.

Name	Data Type
EN	BOOL
x	REAL
y	REAL
z	REAL

Input Variables



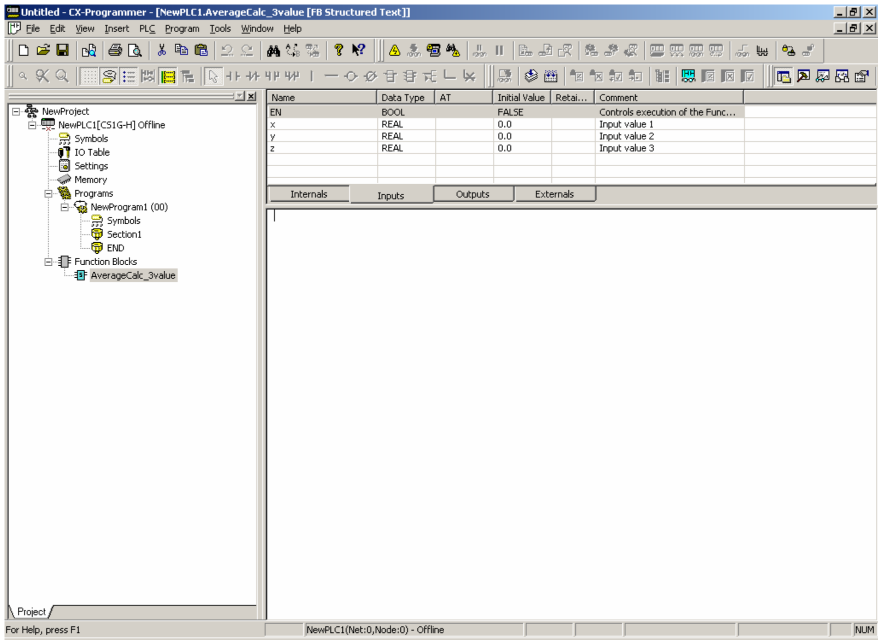
FB instance (call)

Name	Data Ty
ENO	BOOL
score	REAL

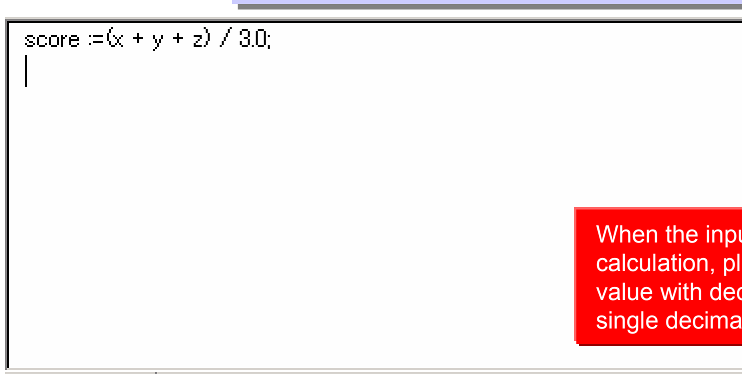
Output Variables

5. Entry of ST program

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

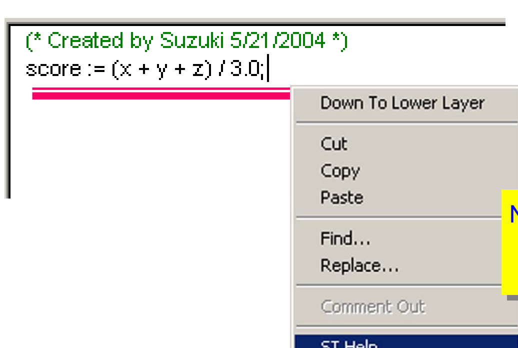


Enter text into the field: “score := (x + y + z) / 3.0;”.



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’.

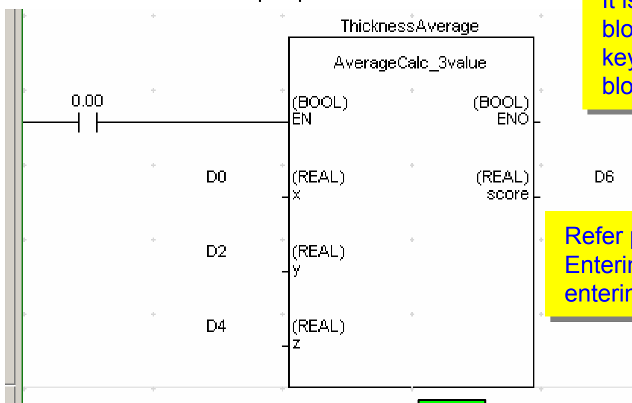
Reference: User may type Comments in the ST program.
Enter ‘(*’ and ‘*)’ both ends of comment strings, see below.
This is useful for recording change history, process expressions, etc.



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.

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

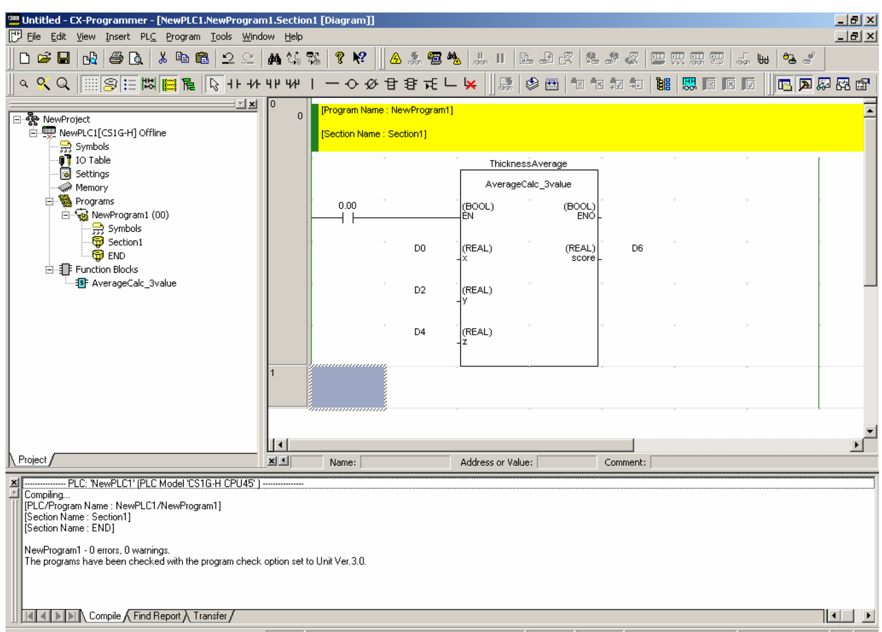
Enter the following FB into the ladder program.
Instance name: ThicknessAvarage
Input parameters: D0, D2, D4
Output parameter: D6



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.

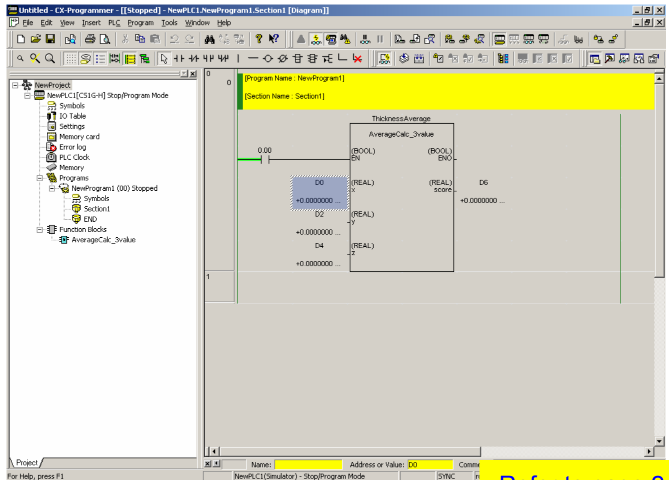


Refer page 2-9 for program checking.
The functionality is the same as for Function Block Ladder instances.

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.

7. Program Transfer

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



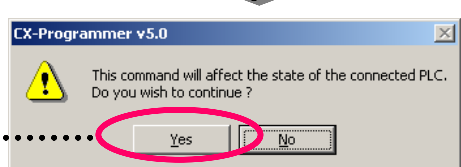
Refer to page 2-10 for steps to go online and transfer the program.

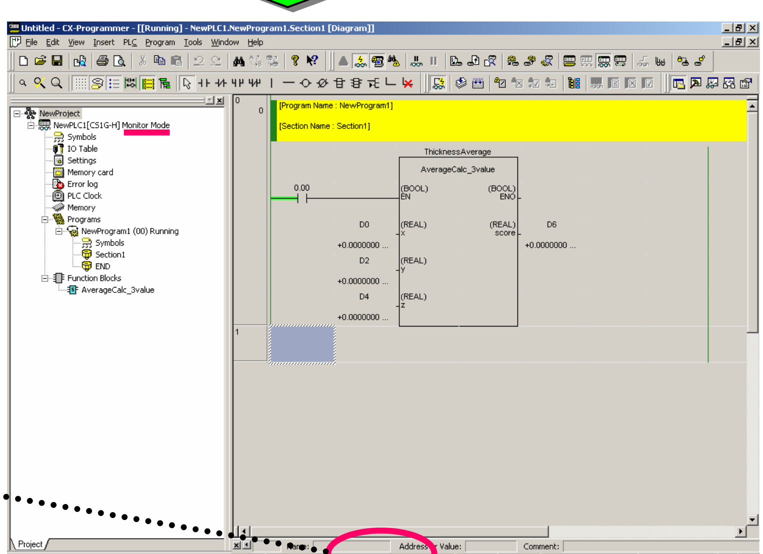
The on/off status of contacts and coils can be monitored.

Change the PLC (Simulator) to Monitor mode.

Click 

Click [Yes]





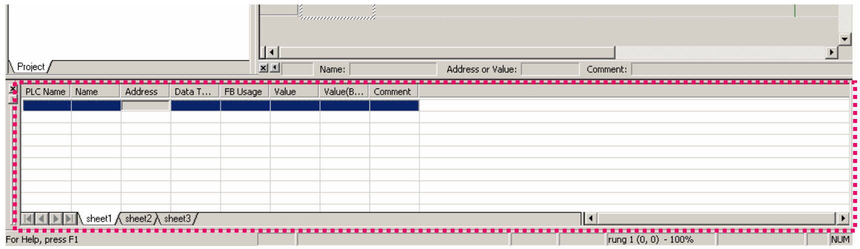
Confirm that the PLC is Monitor mode.

8. Monitoring the Function Block execution

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

Display the Watch Window.

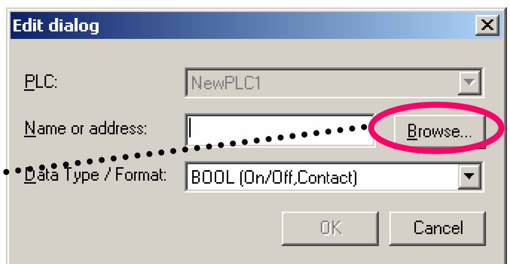
Alt + 3



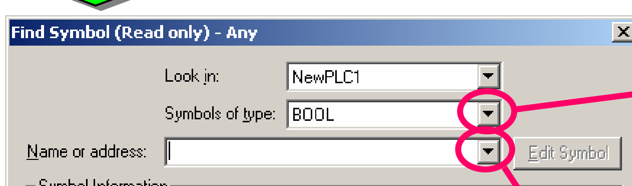
Open the Edit dialog.

ENT

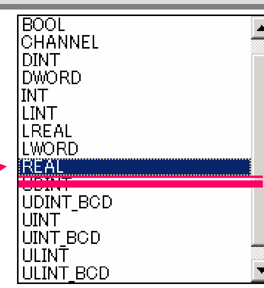
Click Browse… button using the mouse left button.



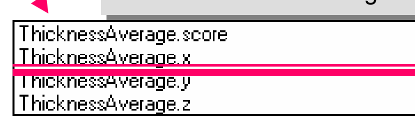
Click the button using the left mouse button, then select the following:
[Symbols of type]
[Name or address]

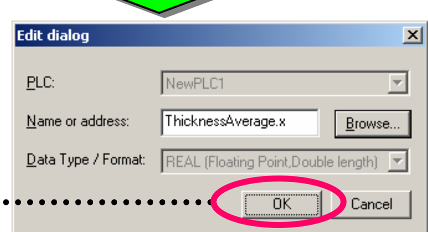


Select REAL(32bit floating point)



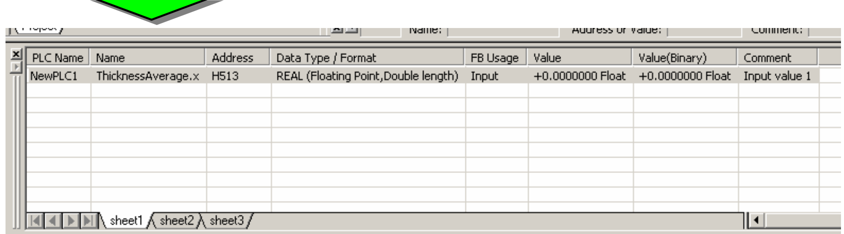
Select ThicknessAvarage.x





Click [OK] button using the left mouse button.

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"



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

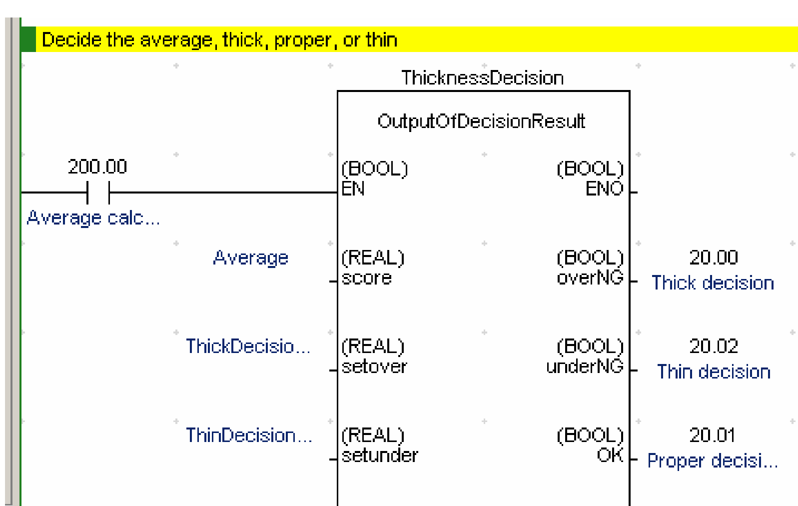
The following ST program checks the average value calculated by the example of page 4-7 against a range (upper limit or lower limit).

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*)
```

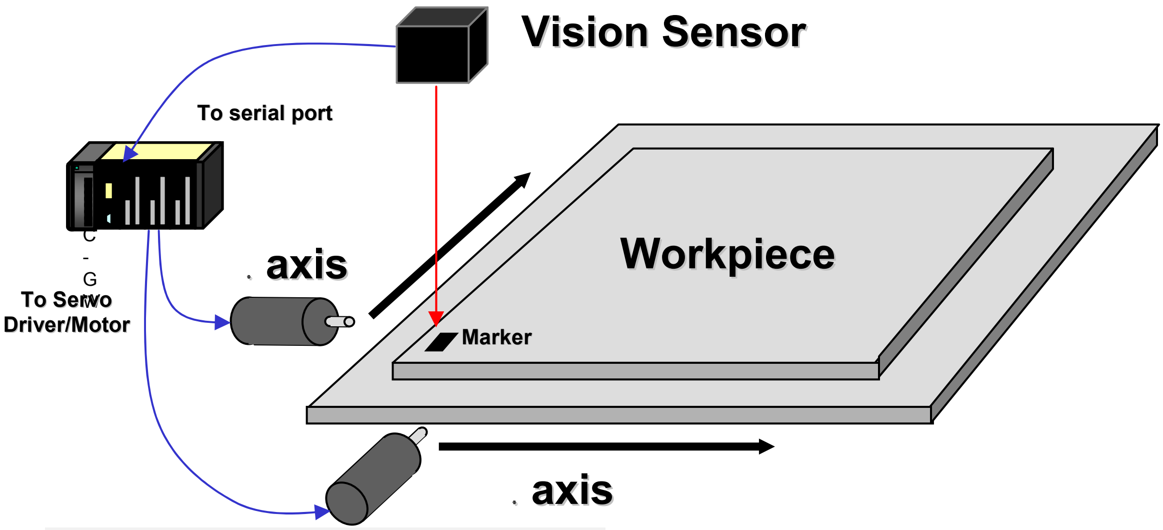
Example of an FB instance (the instance name is 'ThicknessDecision')



Reference: Example of an ST Program Using String Variables

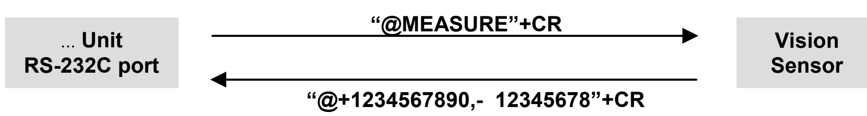
1. Application Example

In this example, a Vision Sensor is used to detect the workpiece's position and Servomotors are used to perform positioning on the X and Y axes.

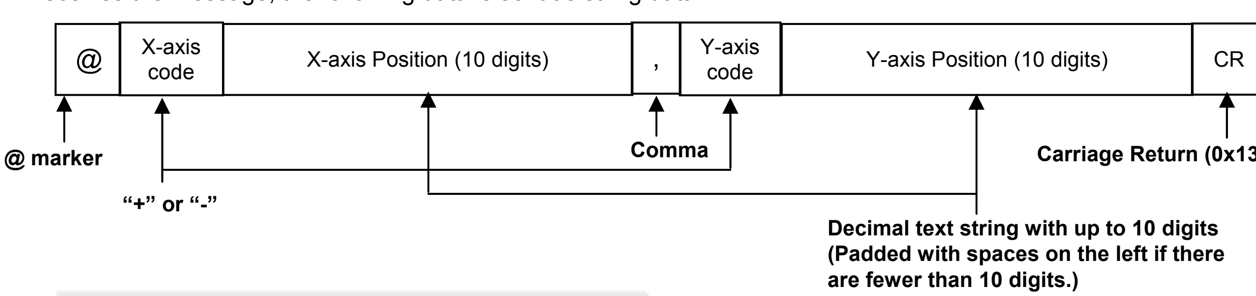


2. Interface with the Vision Sensor

The following messages are transferred between the Vision Sensor and the CPU Unit via the CPU Unit's RS-232C port.



When the CPU Unit sends the message "MEASURE"+CR(0x13) from its RS-232C port and the Vision Sensor receives the message, the following data is sent as string data.



3. Range of Programming in FB (ST)

The following range of processes are created in the FB.
Set two words of data each for the X and Y coordinate, as the NC Unit's command values.
Receive the workpiece's present position (X and Y coordinates) from the Vision Sensor through serial communications.
Analyze the data received from the Vision Sensor to get the workpiece's present position (X and Y coordinate).
Output the difference between the workpiece's target position and present position as the NC Unit's command values.

4. FB (ST) Program

The following ST program satisfies the application's requirements.

Variable Table

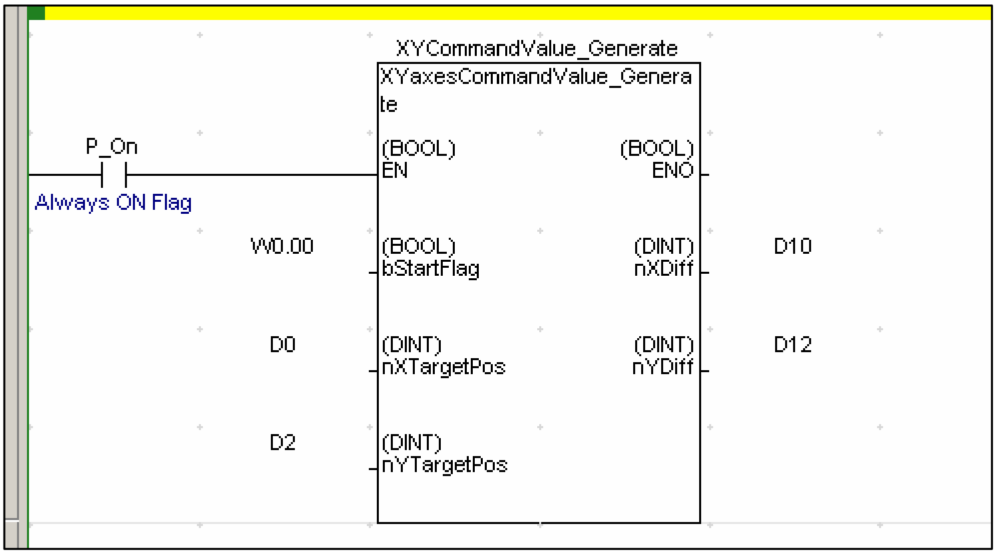
Variable type	Name	Data type	AT	Initial value	Held	Comment
Internal	bSending	BOOL		FALSE		Sending flag
Internal	nSendStatus	INT		0		Send status (1: Sending enabled, 2: Sending, 3 Send completed)
Internal	SndEnableCPUPort	BOOL	A392.05			Built-in host link port send ready flag
Internal	EndRecvCPUPort	BOOL	A392.06			Built-in host link port receive completed flag
Internal	strXYPosition	STRING(30)				String read from Vision Sensor
Internal	strXPos	STRING(15)				X-axis present value temporary variable
Internal	strYPos	STRING(15)				Y-axis present value temporary variable
Internal	nLen	INT		0		Temporary variable for reception data analysis
Internal	nCommaPos	INT		0		Comma position for reception data analysis
Input	bStartFlag	BOOL		FALSE		Send start flag
Input	nXTargetPos	DINT		100		X-axis target value
Input	nYTargetPos	DINT		100		Y-axis target value
Output	nXDiff	DINT		0		X-axis command value
Output	nYDiff	DINT		0		Y-axis command value

ST Program

```
(*Read position information from Vision Sensor and produce command value to the NC Unit.
String format read from Vision Sensor: '(X coordinate) (Delimiter character) (Y coordinate)'
                X coordinate: Sign + 10 digits max.
                Y coordinate: Sign + 10 digits max.
                Delimiter: Comma
Example: '+1234567890,–654321'  (The number of X and Y coordinate digits varies.) *)
(* Detect read start trigger *)
IF ( bStartFlag AND NOT(bBusy) ) THEN
                nStatus := 1;
                (*Not executed if data is already being read.*)
END_IF;

(*Read processing*)
CASE nStatus OF
                1: (* Read command to bar code reader *)
                                IF SendEnableCPUPort = TRUE THEN
                                (* Send if RS-232C port can send data.  *)
                                                bBusy := TRUE; (* Turn ON Vision Sensor reading flag. *)
                                                TXD_CPU('MEASURE'); (* Send "Measure once" command. *)
                                                nStatus := 2;
                                END_IF;
                2: (* Get data read from bar code reader. *)
                                IF EndRecvCPUPort = TRUE THEN
                                                (* If the reception completed flag is ON *)
                                                RXD_CPU(strXYPosition, 25); (* Read reception data to strXYPosition. *)
                                                nStatus := 3;
                                END_IF;
                3: (* Processing after the read *)
                                (* Analyze the string from the Vision Sensor into X and Y coordinates. *)
                                nLen := LEN(strXYPosition);  (* String length *)
                                                nCommaPos := FIND(strXYPosition, ','); (* Delimiter position *)
                                                strXPos := LEFT(strXYPosition, nCommaPos - 1);
                                                (* Extract X-coordinate string. *)
                                                strYPos := MID(strXYPosition, nCommaPos + 1, nLen - nCommaPos);
                                                (* Extract Y-coordinate string. *)
                                                (* Convert strings to numbers and extract the command values. *)
                                                nXDiff := nXTargetPos - STRING_TO_DINT(strXPos);
                                                (* Command value := Target value – Present value *)
                                                nYDiff := nYTargetPos - STRING_TO_DINT(strYPos);
                                                 (* Command value := Target value – Present value *)
                                                nStatus := 0;
                                                bBusy := FALSE;  (* Turn OFF Vision Sensor reading flag. *)
END_CASE;
```

5. Example Application in a Ladder Program

The following example shows the FB used in a ladder program.
The X-axis and Y-axis target values are set in D0 and D2. If bit W0.0 is turned ON, the communications are performed in the FB and the command values are output to D10 and D12.



Chapter 5

Advanced

(Componentizing a Program Using FB)

Function Block

1. Overview

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

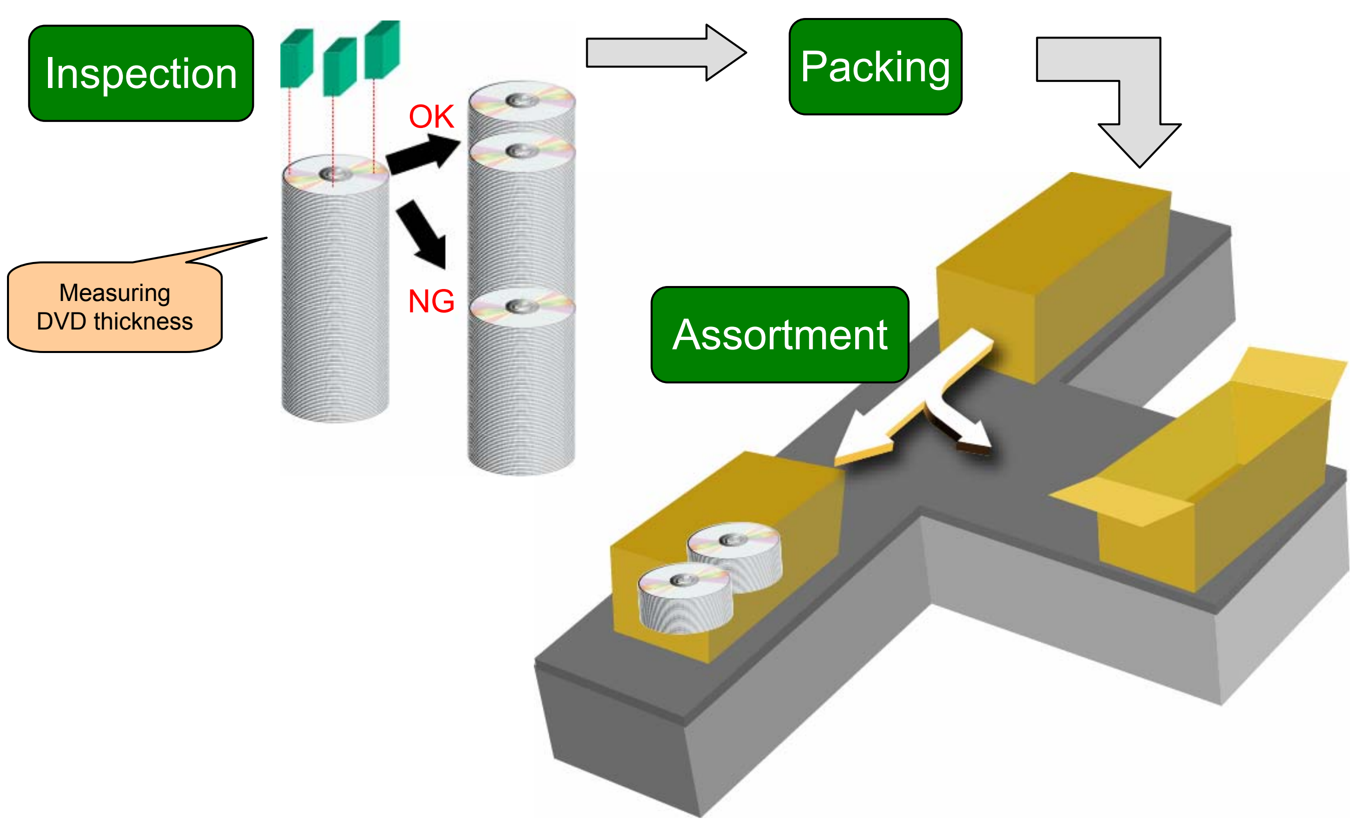
2. How to Proceed Program Development

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.

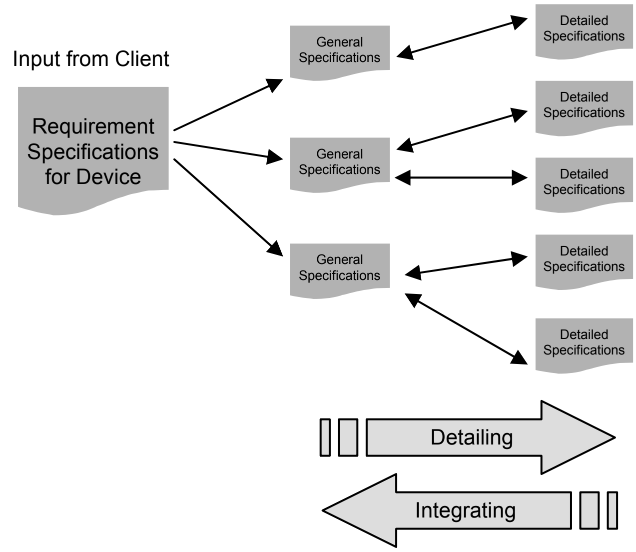


4. How to Proceed Program Development

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. <u>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.</u>

Req. 3. Good DVDs should be packed into the case.

Req. 4. Packed DVDs should be packed into the paper box.

Req. 5. <u>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.</u>

Req. 6. Other requirements

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

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.

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

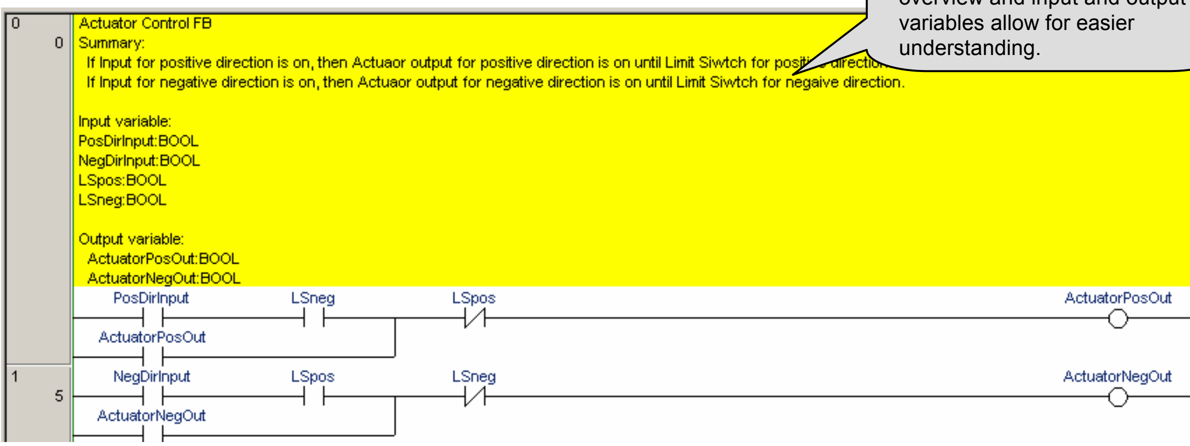
[Output Variables]

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.

Line comments for operational overview and input and output variables allow for easier understanding.



"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 threshould 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.

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)

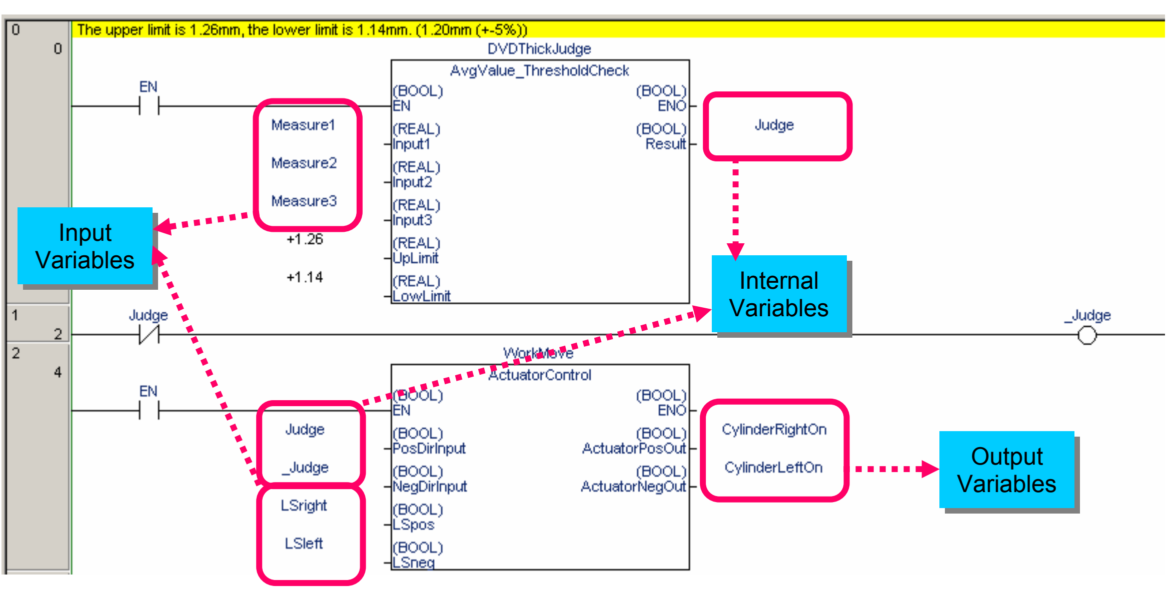
[Output Variables]

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

[Internal Variables]

Name	Data Type	AT	Initial Value	Retained	Com
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.

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 acutuator right direction
LSleft	BOOL		FALSE		Limit switch for acutuator 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 Functio…
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 siwtch *)
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);

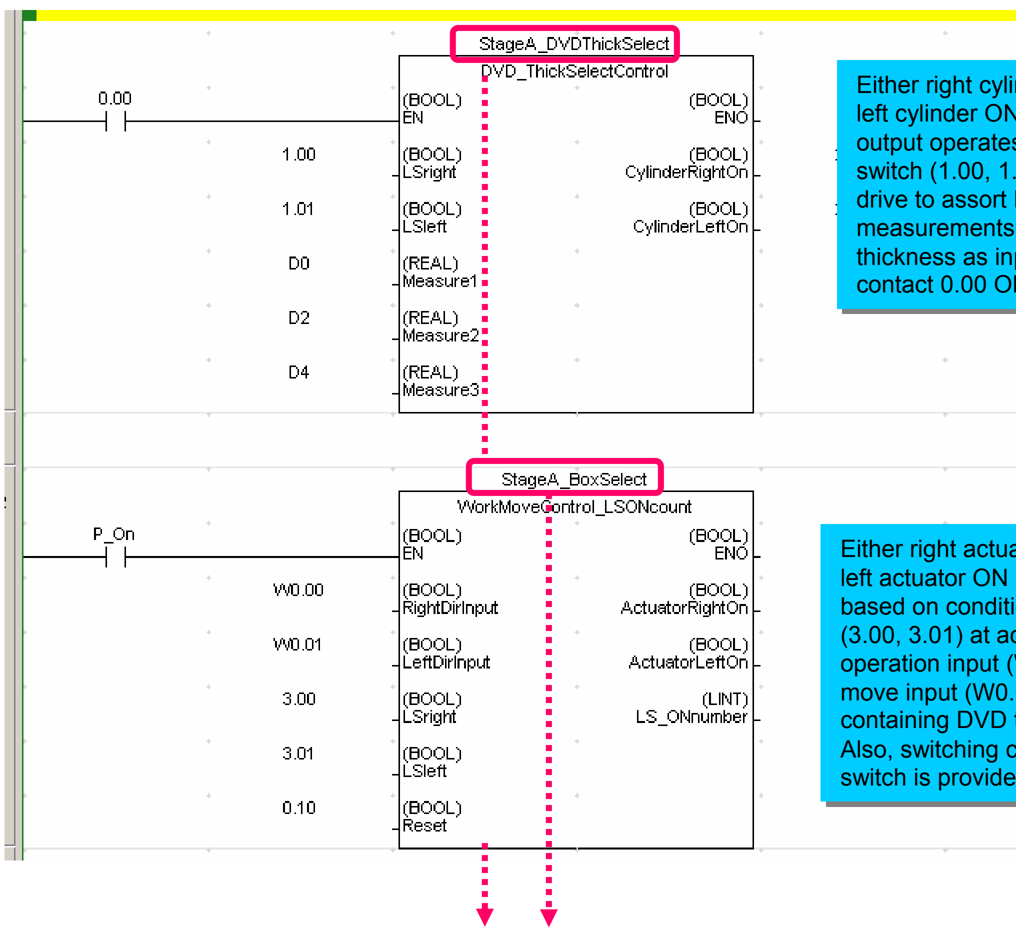
4-5. Total Program Description

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.



Either right cylinder ON (2.00) or left cylinder ON (2.01) as an output operates by using a limit switch (1.00, 1.01) at cylinder drive to assort DVD and 3 measurements (D0-D5) of DVD thickness as inputs and by turning contact 0.00 ON.

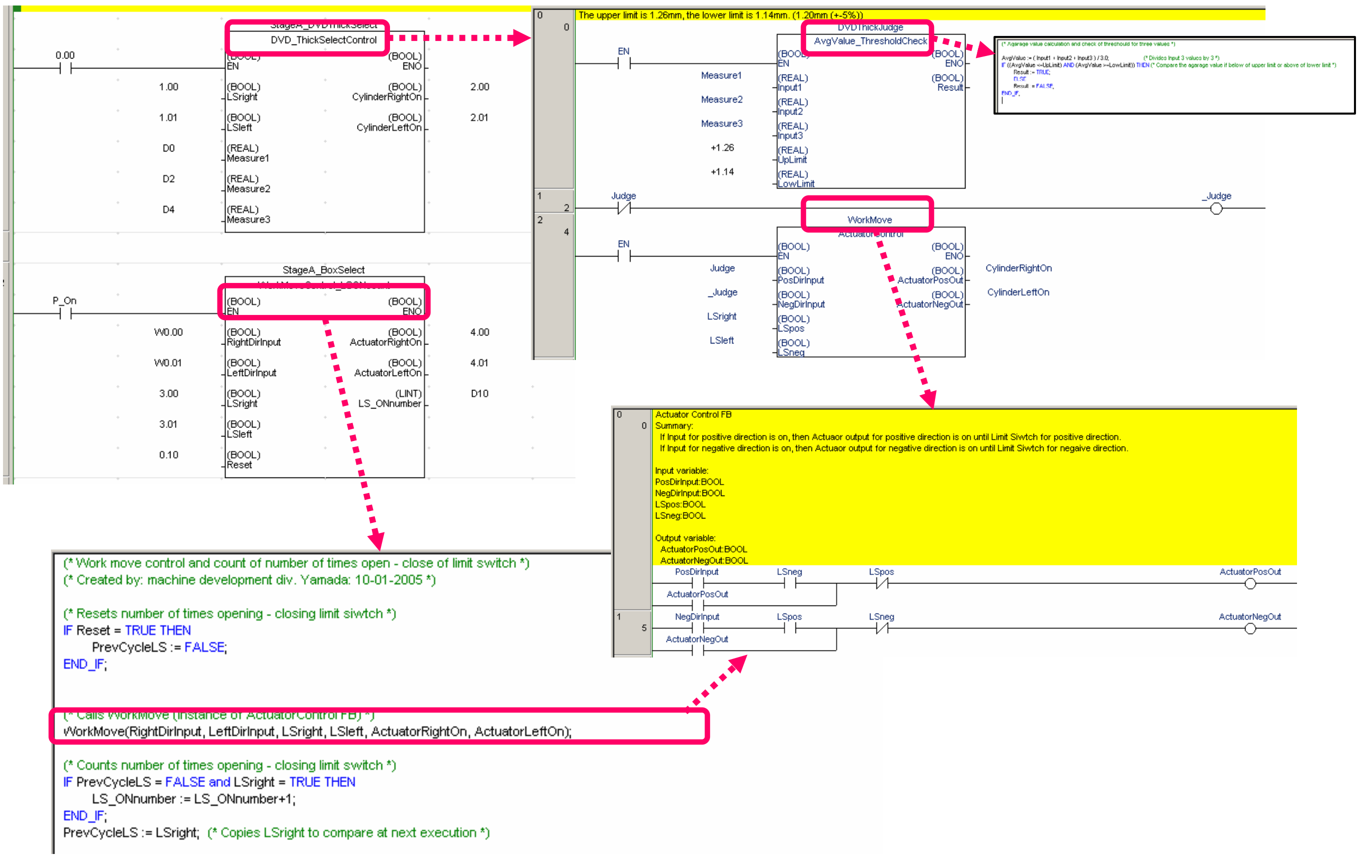
Either right actuator ON (4.00) or left actuator ON (4.01) turns ON based on condition of a limit switch (3.00, 3.01) at actuator end when operation input (W0.00) and left move input (W0.01) to move a box containing DVD to the right or left. Also, switching count of the limit switch is provided to D10-13.

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.

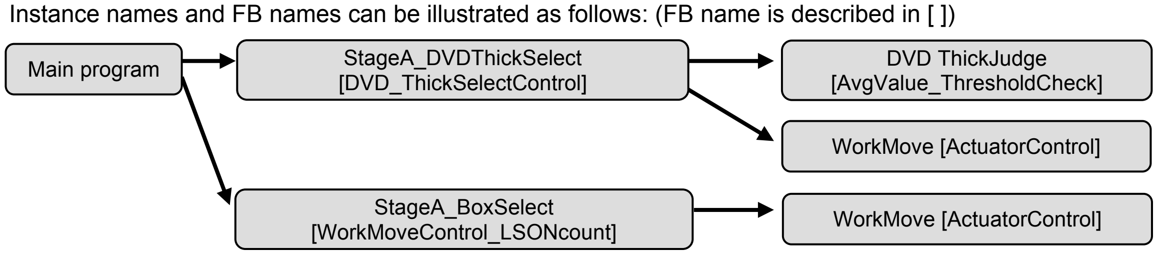
4-5-1. Total Program Structure

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

[Main Program]

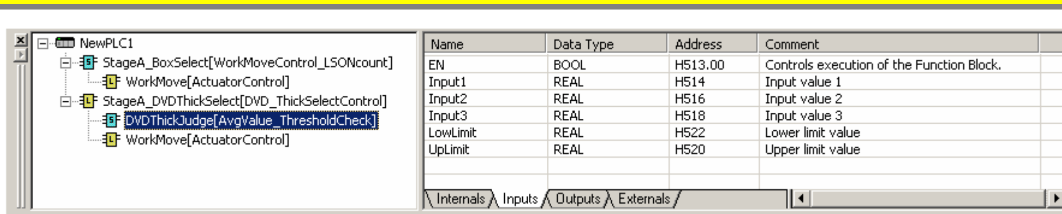


Instance names and FB names can be illustrated as follows: (FB name is described in [ ])



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 or higher 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

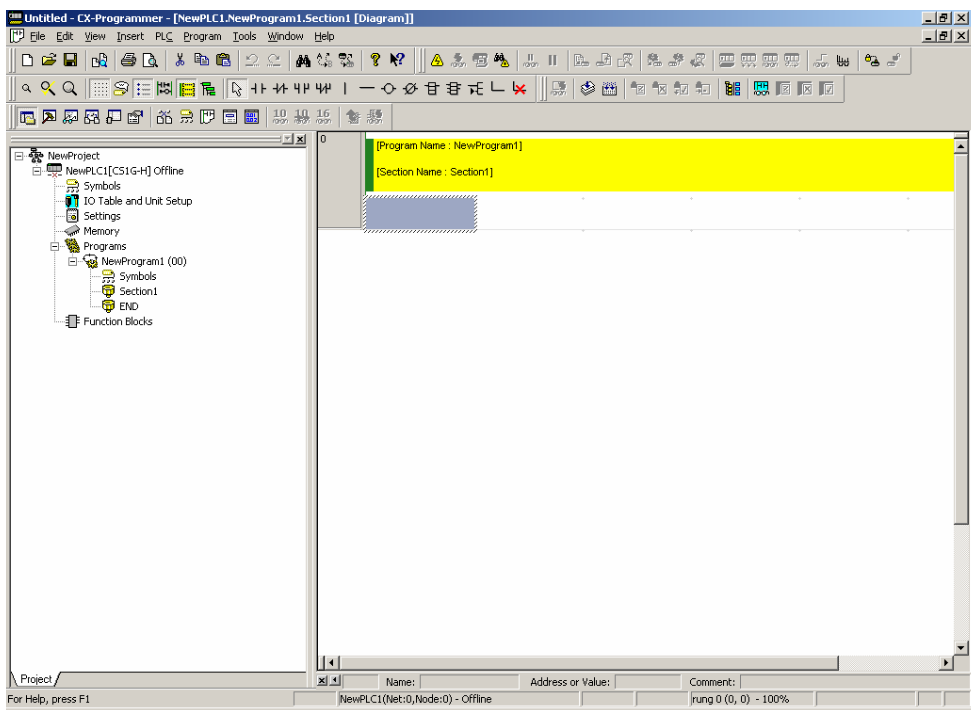
5. Entering FB Definition

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.

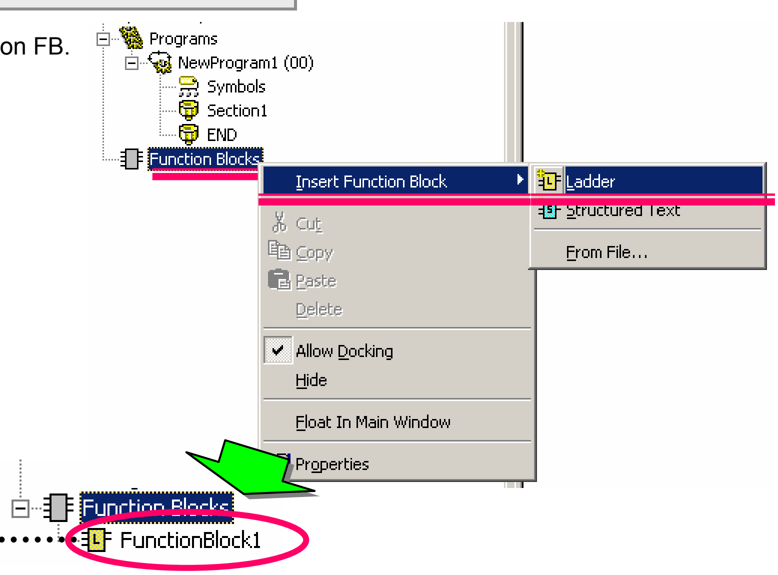
! Select a PC model from the followings to use function blocks.
CJ2H, CS1G-H, CS1H-H, CJ1G-H, CJ1H-H, CJ1M



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

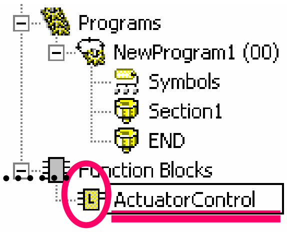


Now new FB is created.

5-3. Entering FB Ladder Program

Change FB definition name.

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



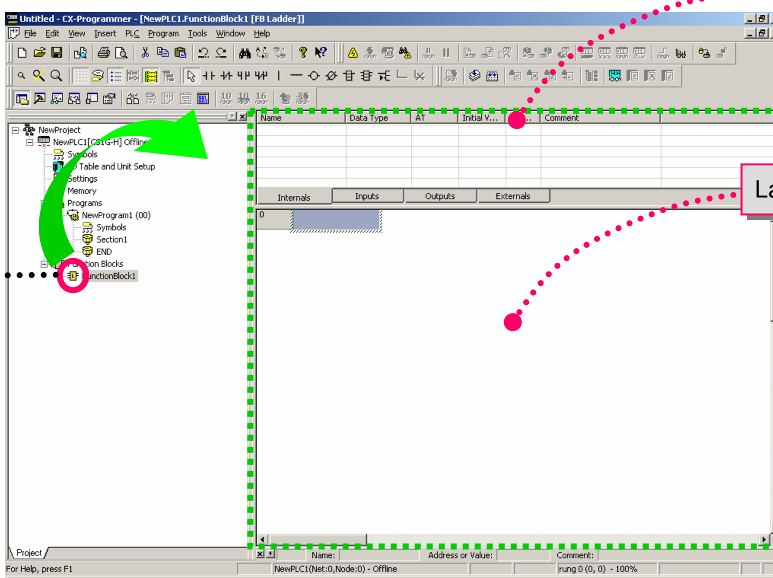
Caution:
A user cannot create function block definition name starting from "_".
The name must start from a character other than "_".

Open FB ladder editor.

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

Variables Table

Ladder Input Screen



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.

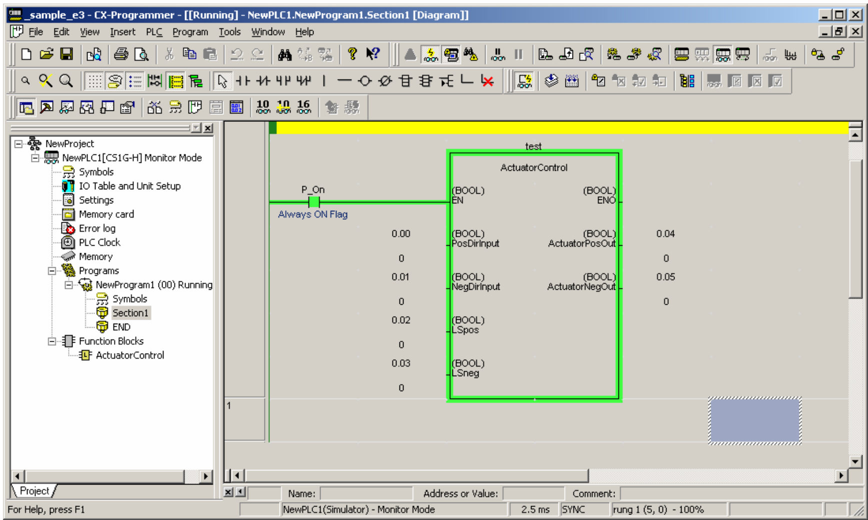
Note: Although you can enter a circuit in the FB ladder editor similar to the main ladder editor, entering of address in the FB is invalid.

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.

5-4. Transferring Program

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

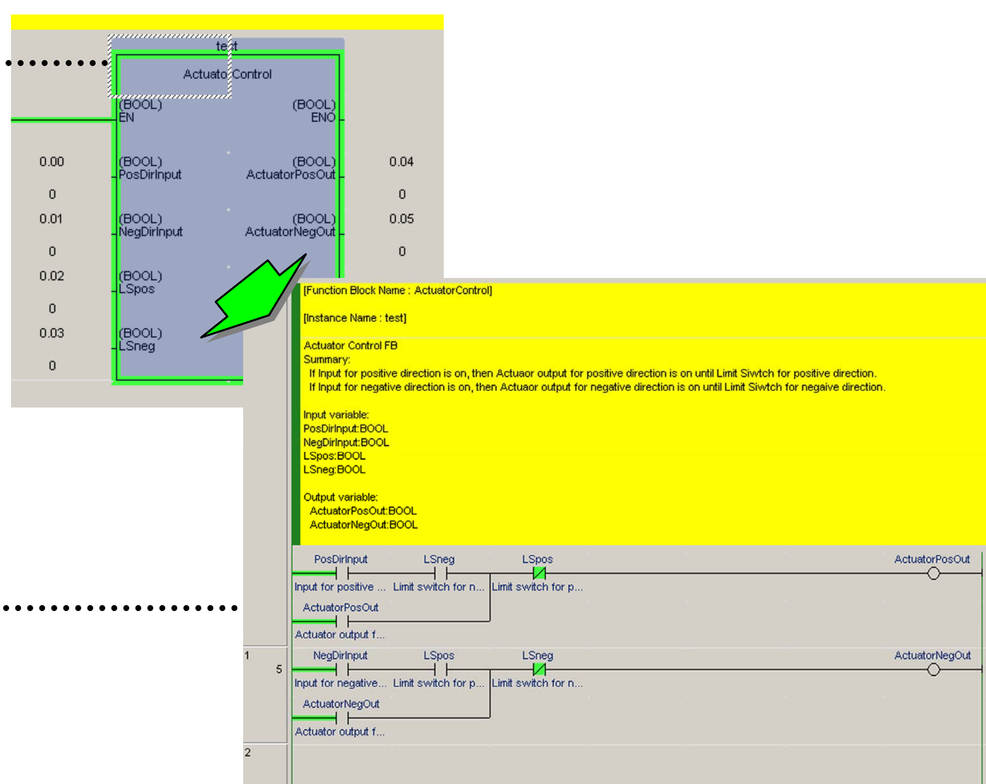
For how to connect online and transfer a program, see page 2-10.



5-5. Operation Check-1

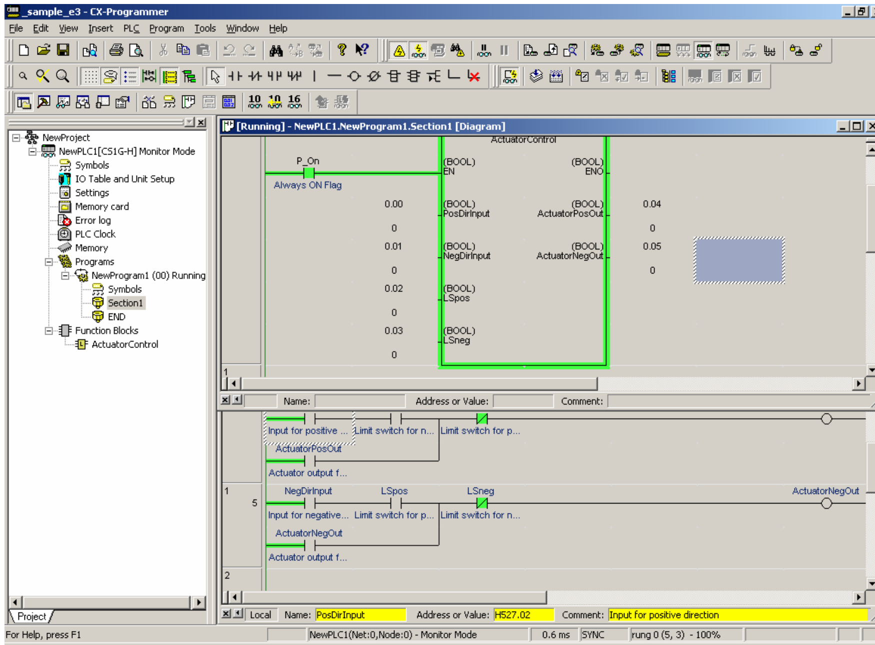
Change current parameter value of FB call statement on the main ladder, then check the operation of “ActuatorControl” FB.
Monitor the instance of ActuatorControl FB first.

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



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

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, <u>but all combination of conditions must be verified.</u>

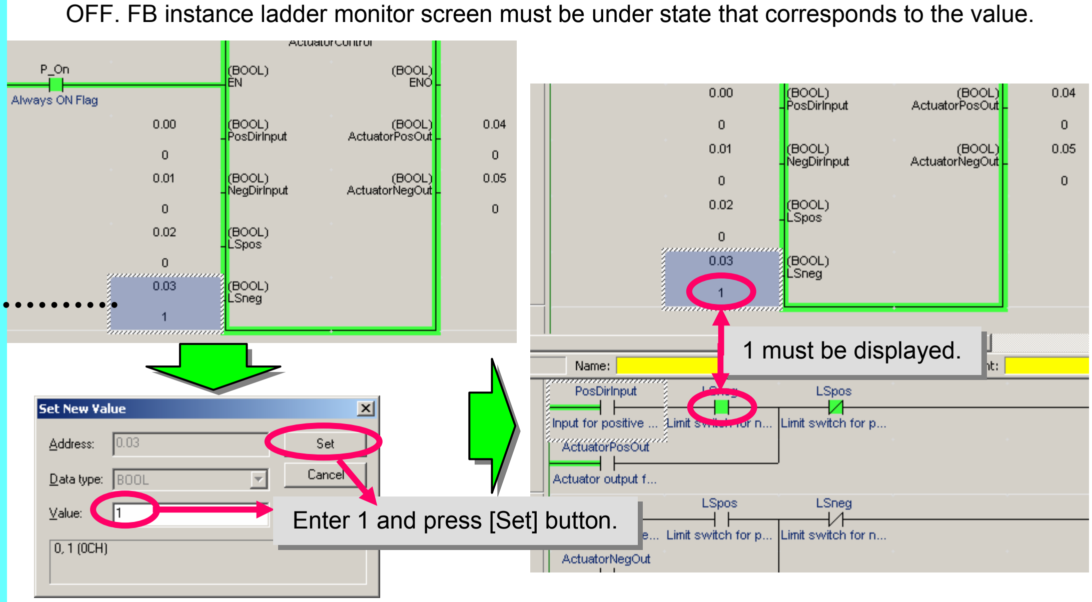
(1) Initial State: Turn 0.03 ON. => 0.04 and 0.05 must be OFF. FB instance ladder monitor screen must be under state that corresponds to the value.

(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.



5-7. Entering/Debugging Other FB Definition

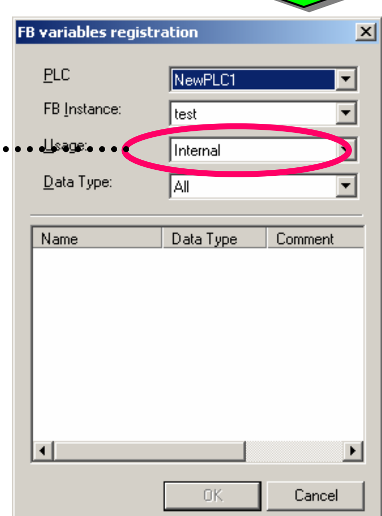
Thus far, entering and debugging for “ActuatorControl” FB are described. Other FB definition must be entered and debugged as well.

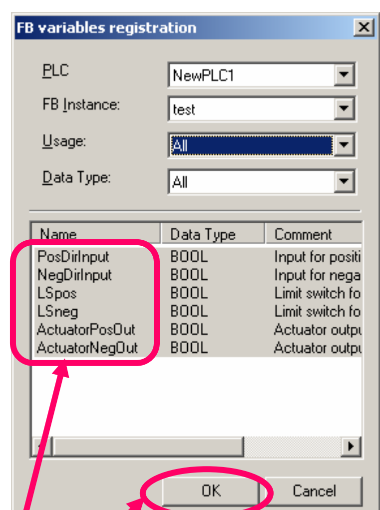
5-8. Batch Registration to Watch Window

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.

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

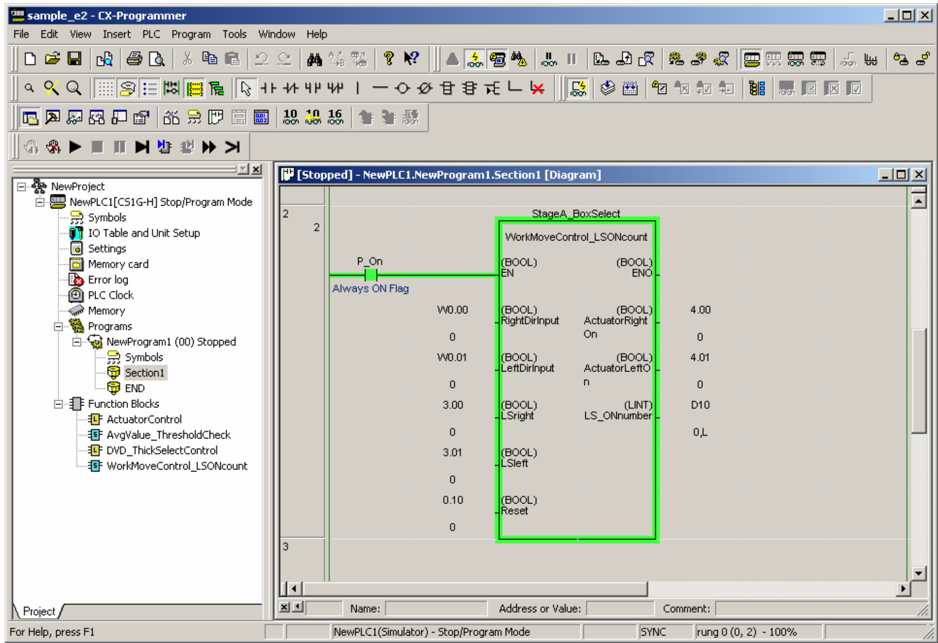
The toolbar buttons below are for use with the simulation function. The function of each button is described here.

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).

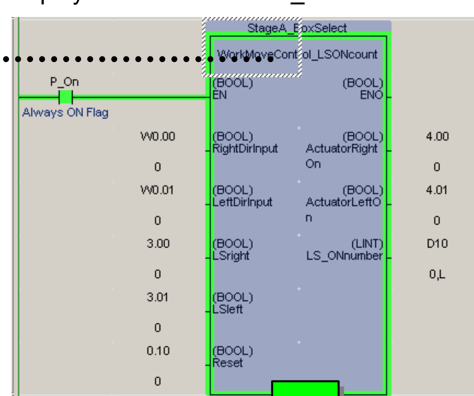
5-9-2. Setting Breakpoint and Executing Steps

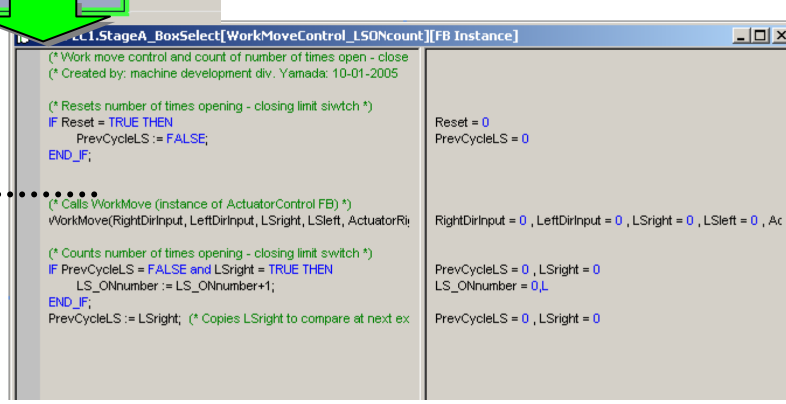
Here is an explanation using Simulation Function “WorkMoveControl_LSONcount” FB Debug as an example.



Change from run mode to monitor mode.
Display “WorkMoveControl_LSONcount” FB instance.

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





ST Program

Variables and present values

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

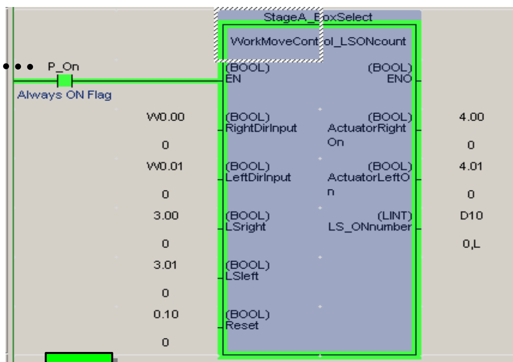
Set the current value in the FB call statement parameter and confirm execution condition.
Set the following cases:

RightDirInput: ON
LeftDirInput: OFF
LSright: OFF
LSleft: ON
Reset: OFF

In this case, the following outputs are expected:

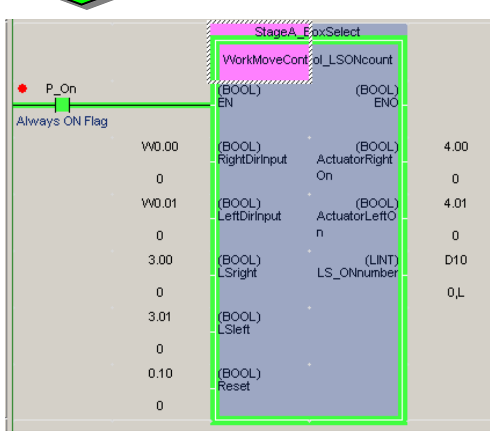
ActuatorRightOn: ON
ActuatorLeftOn: OFF
LS_ONnumber: 1

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



The programs stops at the breakpoint.

Click the button.



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

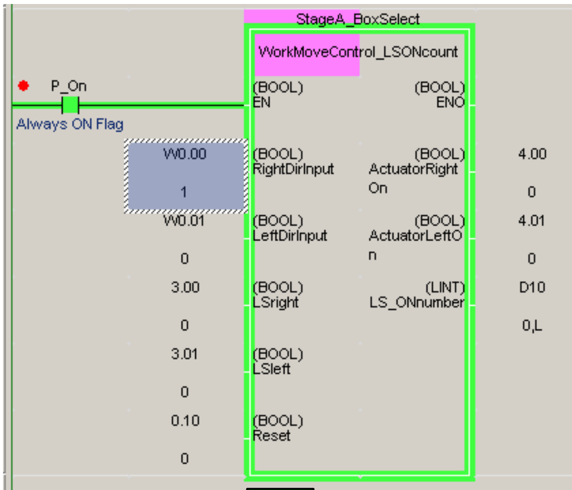
Turn input parameter "RightDirInput" and "LSleft" ON in the FB call statement.

Press [←] [↓][↓]

ENT

1

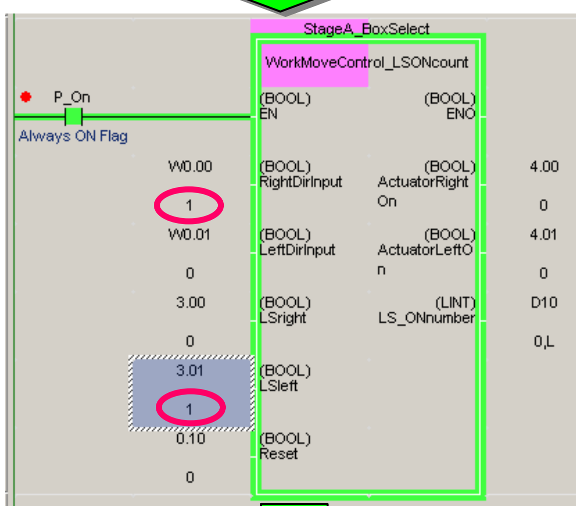
ENT



Press [↓][↓][↓]

1

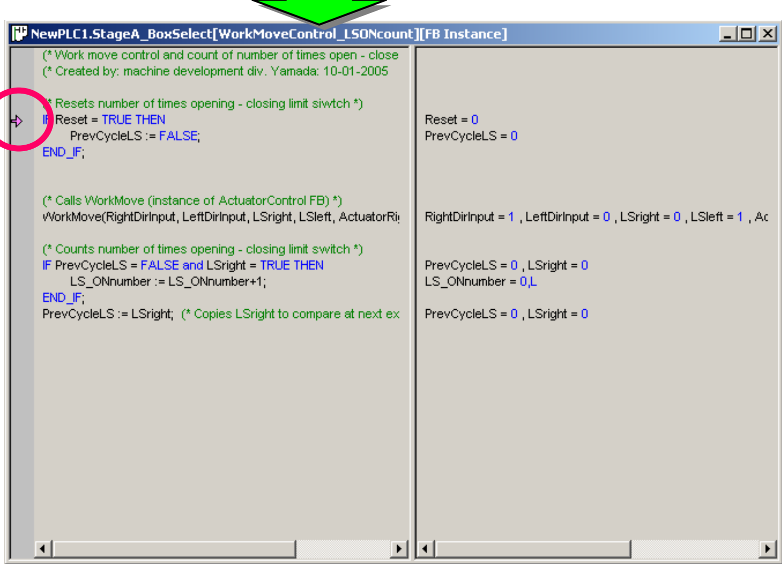
ENT



The necessary input parameters were set.

Click the button.

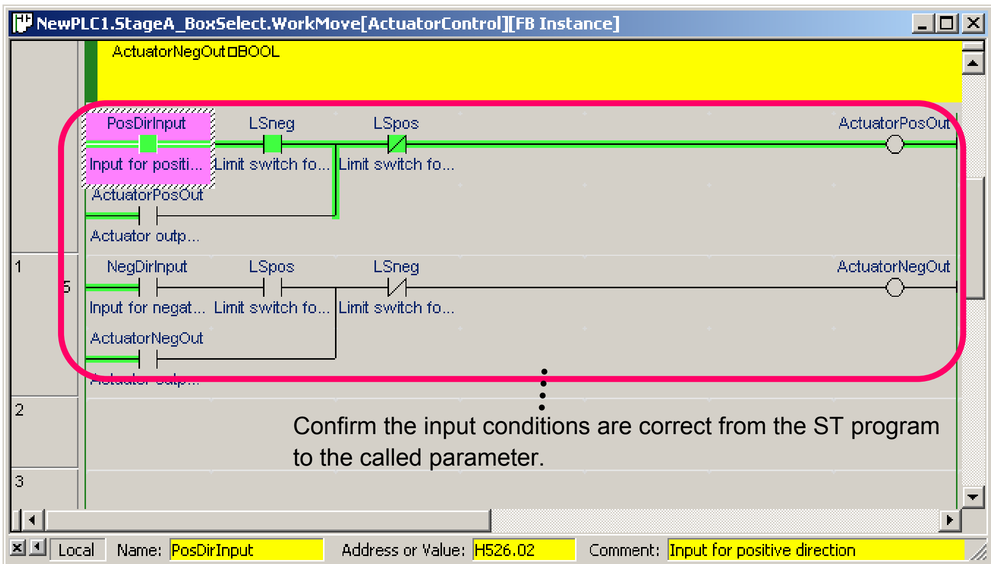
Position of ST Monitor execution



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

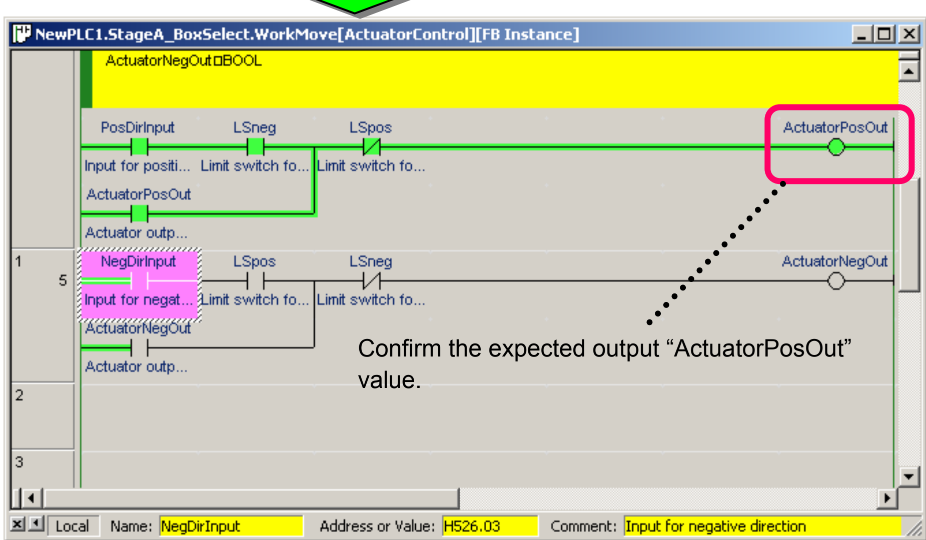
Click the button two times.

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



Confirm the input conditions are correct from the ST program to the called parameter.

Click the button five times.



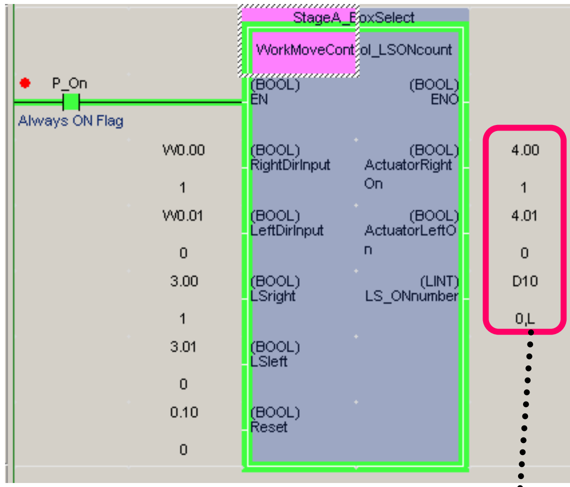
Confirm the expected output "ActuatorPosOut" value.

Click the button.

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

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

Click the button.

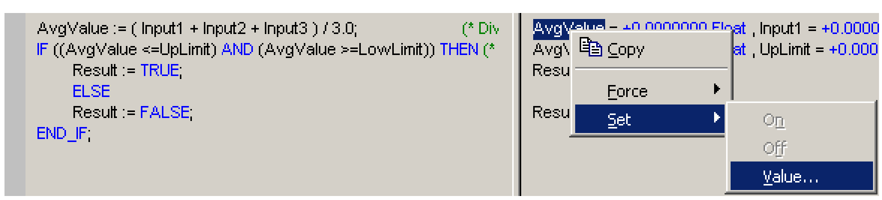


Transfer to the calling ladder program.

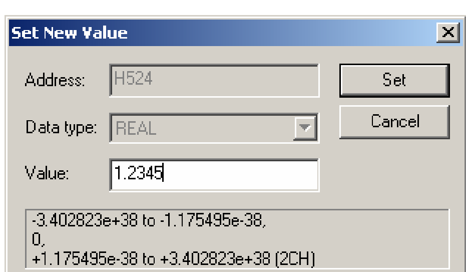
Confirm the output parameter is reflected correctly.

Hint

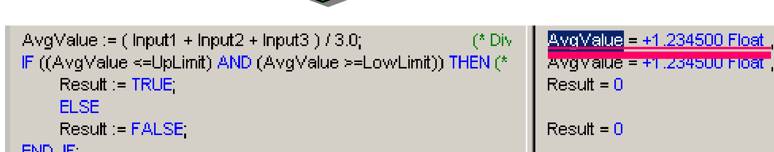
ST program change parameter current value can be performed with the following operation.



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



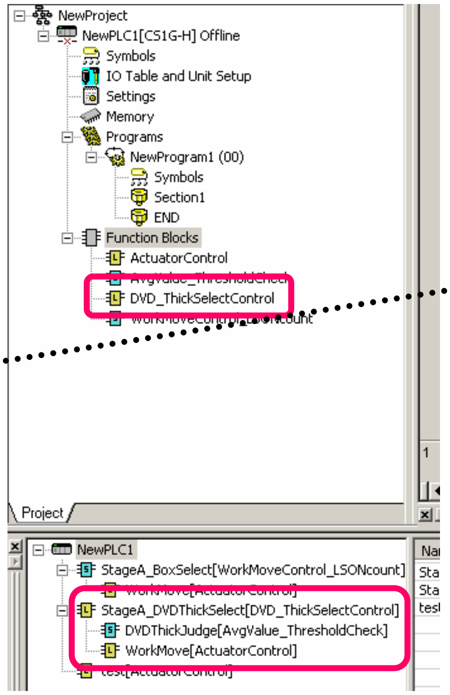
Set value and click the [Set] button.

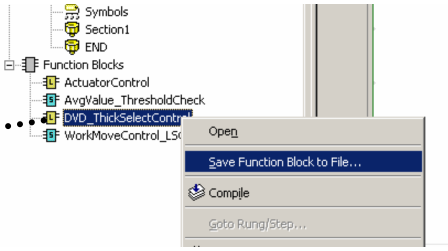


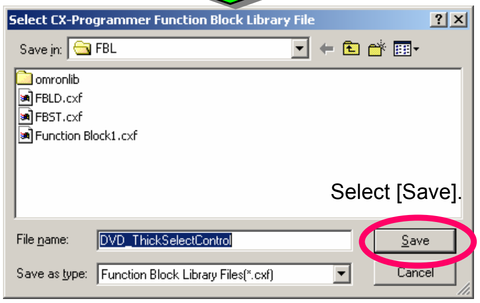
6. Creating FB Definition Library

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.

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







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.

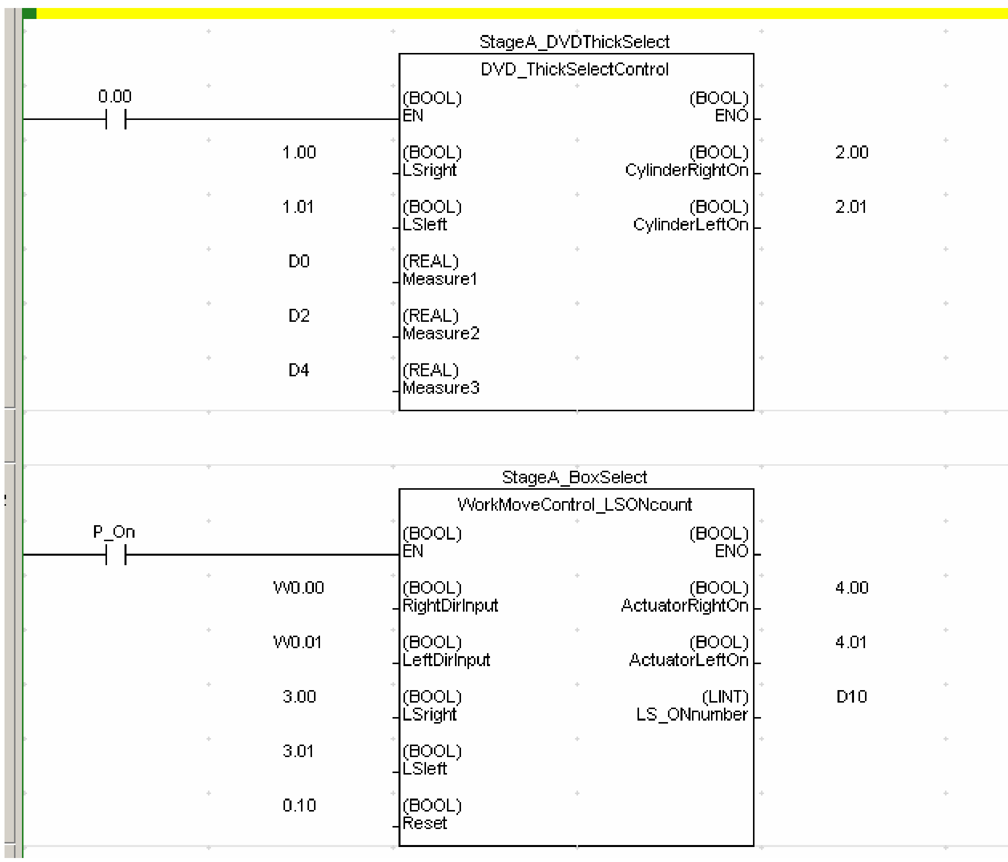
7. Entering Main Program

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.

8. Debugging Main Program

Main program must be debugged considering followings:

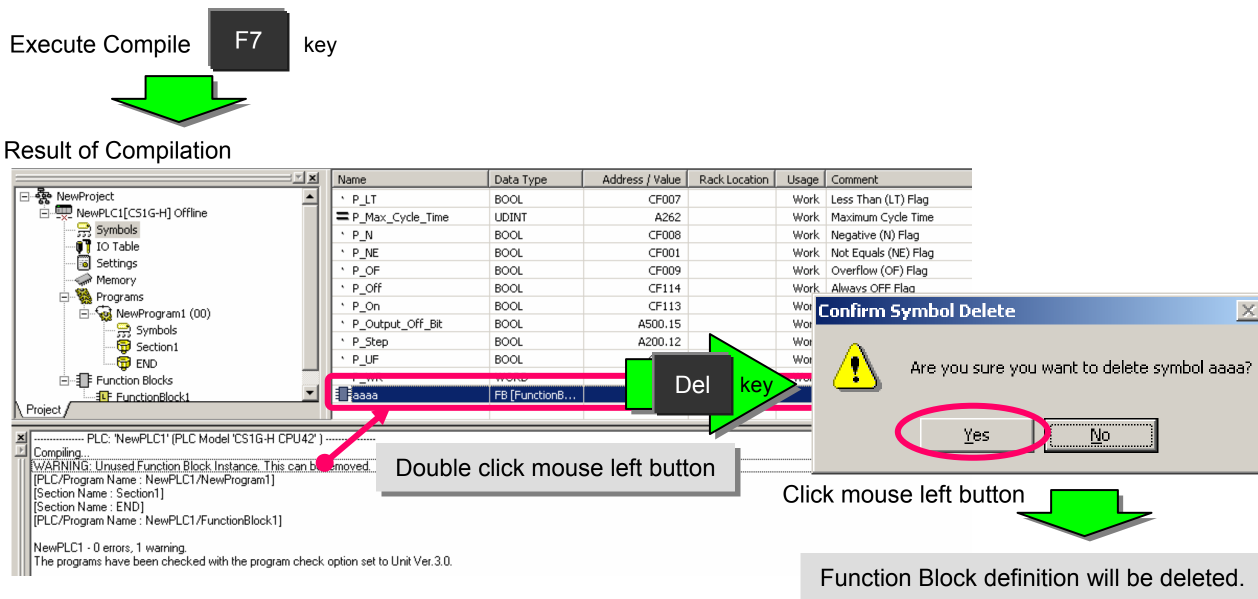
- Program areas that are irrelevant to FB
- Program areas that are relevant to an input parameter to FB
- Program areas that refer to an output parameter from FB

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.



Memory allocation for Function Blocks

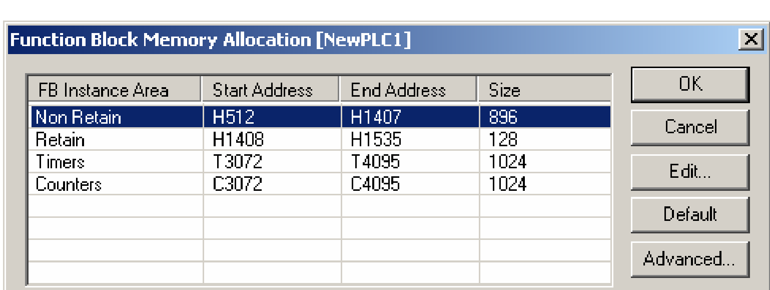
It is necessary to allocate required memory for each function block instances to execute Function Blocks.
CX-Programmer allocates the memory automatically based on the following setting dialog information.
(PLC menu → Function Block Memory → Function Block Memory Allocation)
There are 4 types of areas, ‘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.



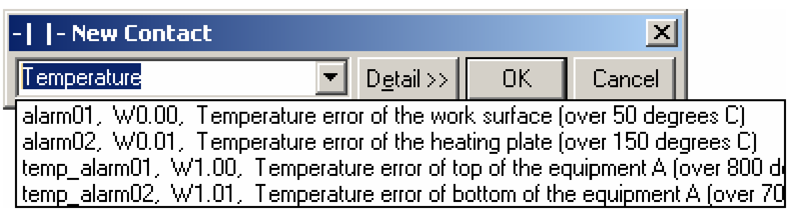
Useful Functions

Command Operand Input Automatic Search and List Display

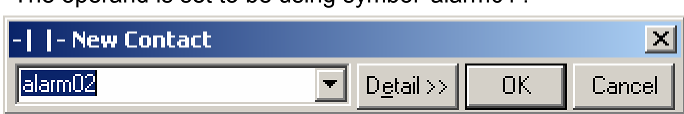
It is possible to automatically display a list of symbol names or IO comments when entering the operands of commands. When entering the operand for contact or output (or special instructions), enter a string, and the dropdown list is automatically updated to display in symbol names or IO Comments using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

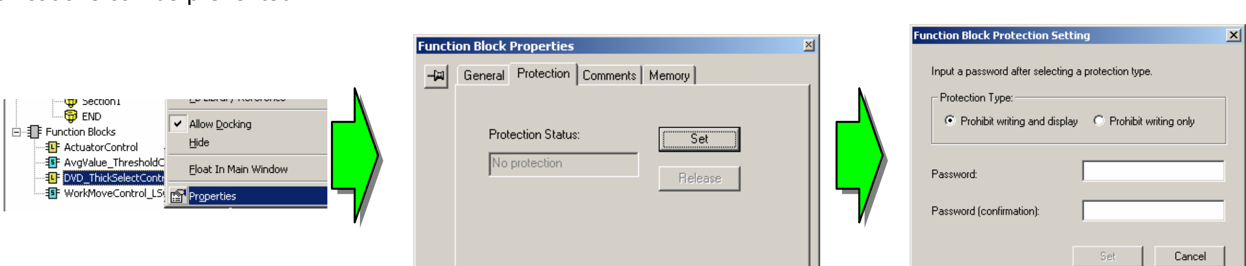


FB Protect Function

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.

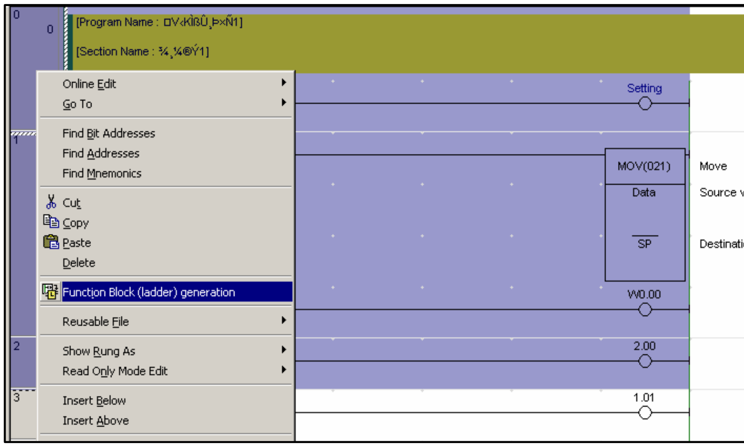


Overview of Helpful Functions

Generating FBs Based on an Existing Ladder Program

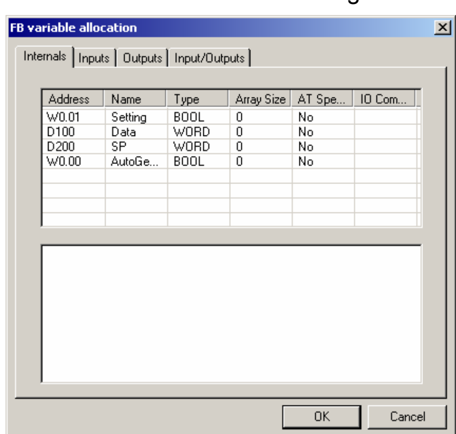
FBs can be generated easily based on programs with proven operating results.
This function can accelerate the conversion of program resources to FBs.

Select the program section that you want to convert to an FB and right-click the mouse.

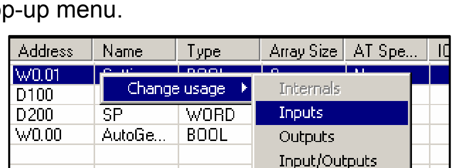


Select ***Function Block (ladder) generation***.

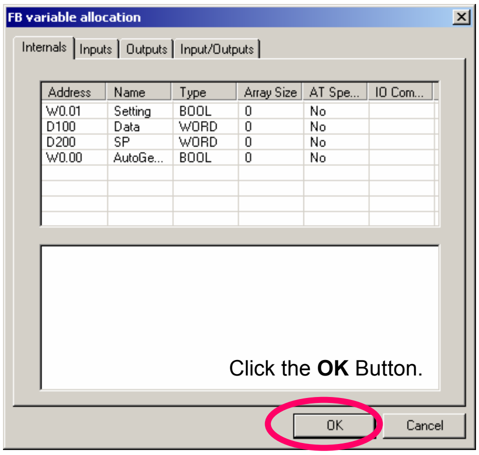
The FB Variable Allocation Dialog Box will be displayed.



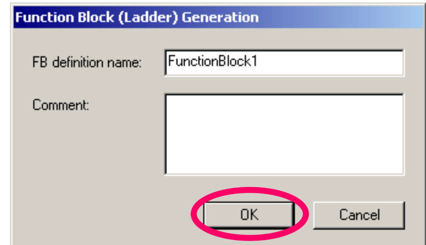
When necessary, change the usage of variables and addresses (internal variable, input variable, output variable, or input-output variable) used in the program section. Select the variable and select ***Change usage*** from the pop-up menu.



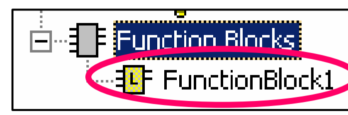
Note:
If a variable does not exist in an address being used in the program, a variable starting with “AutoGen” will be added automatically.
When the FB is called in the program, parameters are displayed as variable names, so at a minimum we recommend changing input, output, and input-output variables to easy-to-understand variable names.
To change the names, double-click the address that you want to change in the FB variable allocation Dialog Box to display a dialog box in which the name can be changed.



Click the **OK** Button.

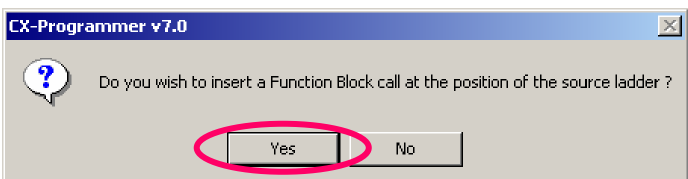


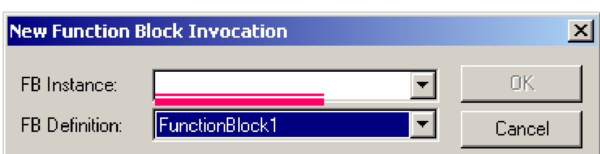
Input the FB definition name and comment, and click the **OK** Button.



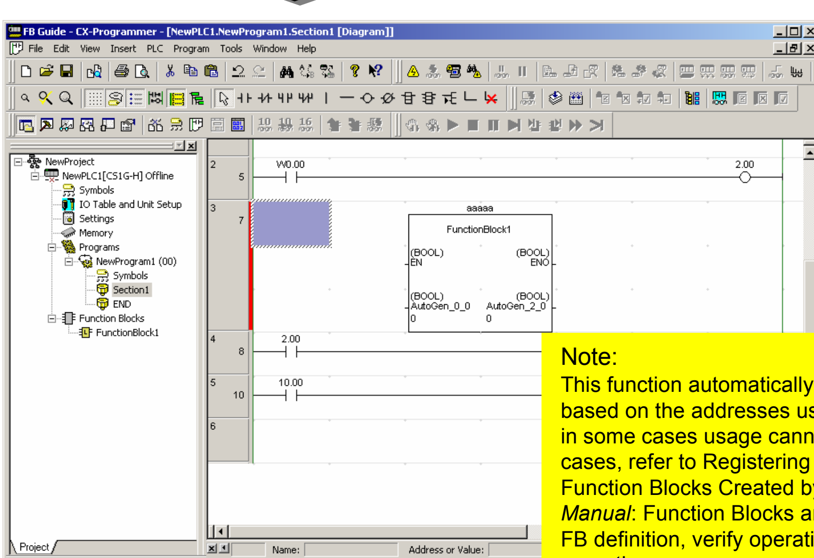
The FB definition will be created.

To insert an FB call instruction created in the ladder program, click the **Yes** Button.





Input the FB instance name and click the **OK** Button.



Note:
This function automatically determines the usage of variables based on the addresses used in the selected program section, but in some cases usage cannot be converted automatically. In these cases, refer to Registering Variables First in 3-2-3 Defining Function Blocks Created by User of the *CX-Programmer Operation Manual*: Function Blocks and Structured Text, check the created FB definition, verify operation sufficiently, and proceed with actual operation.

The FB call instruction will be inserted in the ladder program.

Chapter 6 Advanced: Creating a Task Program Using Structured Text

Task programs can be created using the structured text (ST) language with CX-Programmer. A wider choice of programming languages is now supported to enable optimizing the language to the control object. You can select from SFC, ladder diagrams, or structured text.
Structured text was standardized under IEC 61131-3 (JIS B3503) as a high-level textual programming language for industrial control. Starting with CX-Programmer version 7.2, structured text can be used in task programs, in addition to the previous support for use in function blocks.

Note: Refer to page 4-1 for information on using structured text in function blocks.

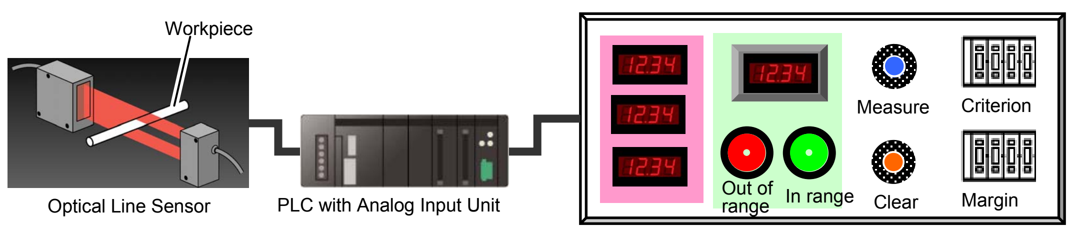
Controls using IF-THEN-ELSE or FOR/WHILE loops, or numeric calculations using SIN, COS, and other functions can be easily achieved using actual addresses.
Structured text can thus be used in tasks to easily program numeric calculations using actual addresses, while structured text can be used in function blocks to enable easily reusing programming.

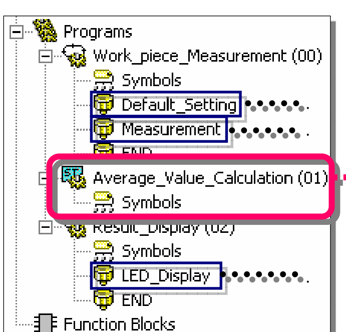
Note: A task is the smallest programming unit that can be executed in a SYSMAC CS1/CJ1-series CPU Unit. With controls separated into tasks, execution of non-active tasks is stopped to enable shortening the cycle time.

1. Description of Program

The procedure used to create a program that finds average values is described as an example.

The diameter of a workpiece is measured in three locations and then the average diameter is found. If the average value is within the allowable range, a green lamp is lit. If the average value is outside the allowable range, a red lamp is lit. Here, an ST program is created to average the workpiece diameters and determine if the average value is within the allowable range.

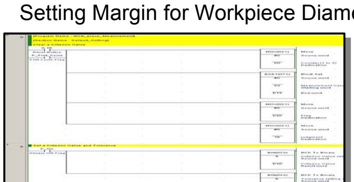




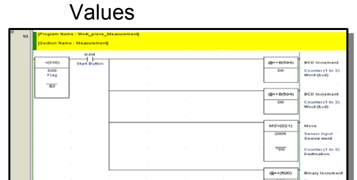

```
(* Average Value Calculation *)
average:=(thickness1+thickness2+thickness3)/3.0;    (* average =Average Value   Athickness1 to 3 =Measurement Value 1 to 3 *)

(* Judgement *)
IF flag = 3 THEN                              (* flag =Three Times Measurement Flag *)
    IF average < criterion-margin THEN        (* criterion =Criterion Value   Amargin =Tolerance *)
        red_lamp:=TRUE;                       (* red_lamp =Red Lamp (without tolerance) *)
    ELSIF average > criterion+margin THEN
        red_lamp:=TRUE;
    ELSE
        green_lamp:=TRUE;                     (* green_lamp =Green Lamp (within tolerance) *)
    END_IF;
END_IF;
```

All other programming is done with ladder diagrams.

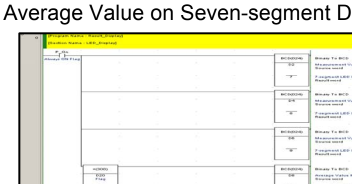
(1) Initializing Measurement Values and Setting Margin for Workpiece Diameter



(2) Setting Measurement Values



(3) Displaying Measurement Values and Average Value on Seven-segment Display

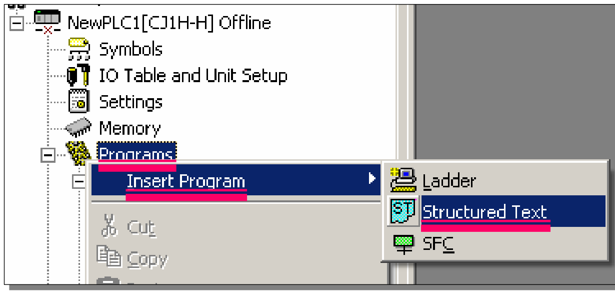


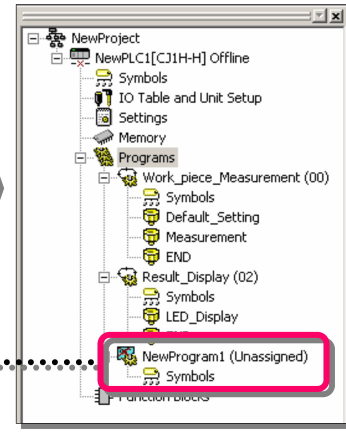
Description of Program → **Creating an ST Task** → Registering Symbols → Entering the ST Program

2. Creating an ST Task

Creating an ST Task

Right-click the *Programs* Icon and select ***Insert Program – Structured Text.***

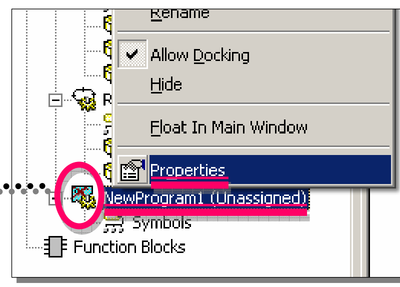




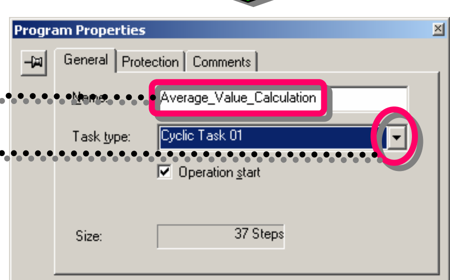
A new ST program will be created.

Change the name of the ST program and assigned it to a task.

Right-click the icon for the new program that was created and select ***Properties.*** A dialog box will be displayed.



Enter the name of the program: Average_Value_Calculation
Also, select the task type from the pull-down menu: Cyclic Task 01

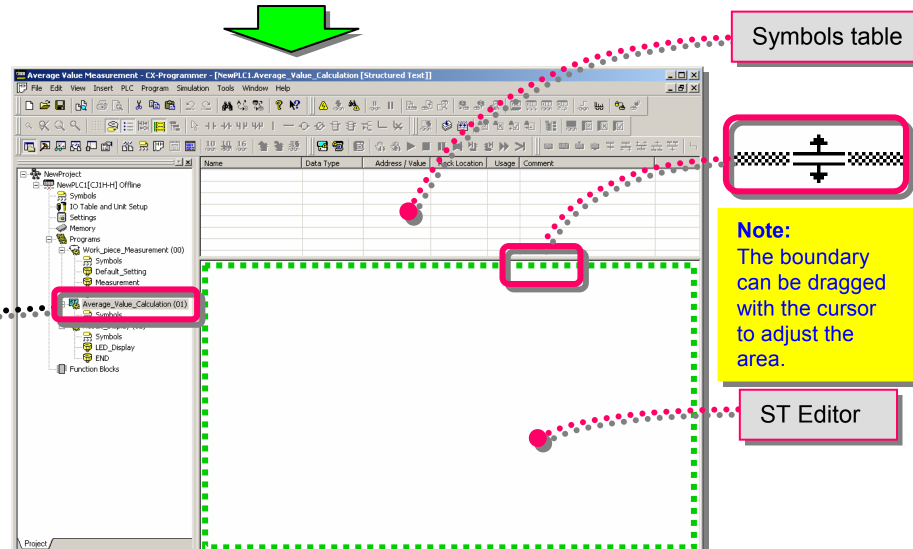


Note: Cyclic tasks are executed each cycle.

ENT

Open the ST Editor.

Double-click the ST program icon. The ST Editor will open.



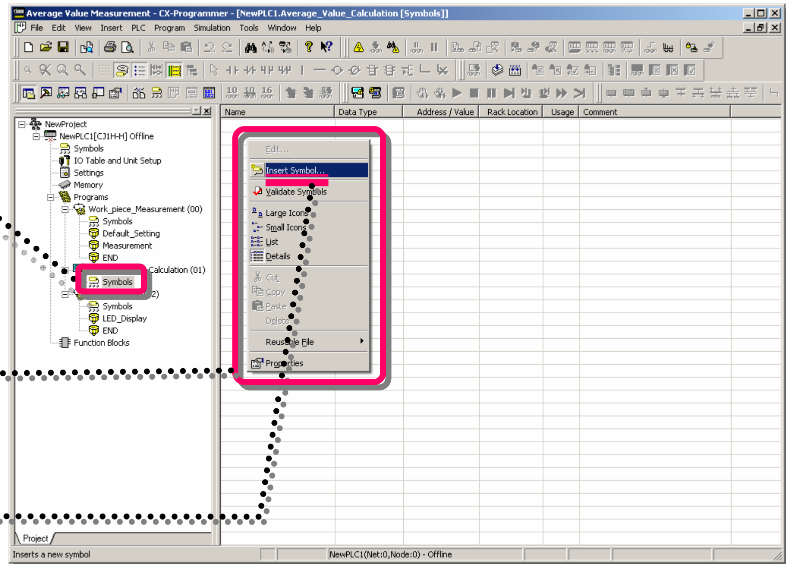
Note: The boundary can be dragged with the cursor to adjust the area.

3. Registering ST Program Symbols

The symbols used in the ST program must be registered.

Open the symbols table.

Double-click the *Symbols* Icon. The symbols table will open.

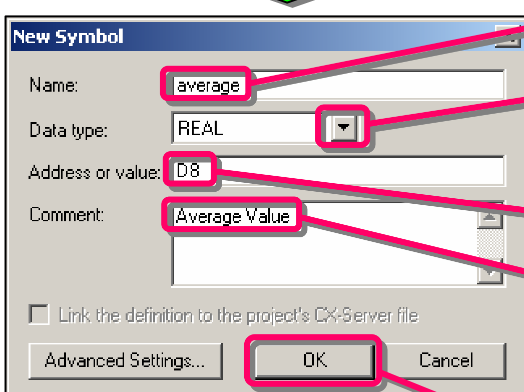


Register new symbols.

Right-click anywhere on the symbols table.

Select ***Insert Symbol*** from the pop-up menu. The New Symbol Dialog Box will be displayed.

Enter the name, data type, address or value, and comment for the symbol.



Enter the name of the symbol.

Select the data type:
- REAL
- BOOL
- INT

Enter the address or value.

Enter a comment to describe the symbol.

When finished, click the OK Button.

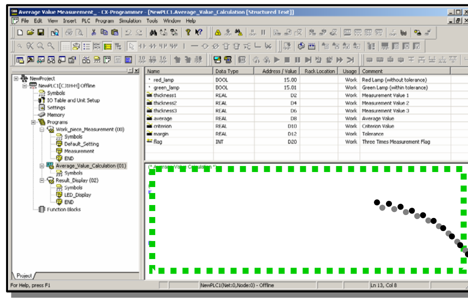
Repeat the above procedure to enter all symbols.

Name	Data Type	Address / Value	Rack Location	Usage	Comment
red_lamp	BOOL	15.00		Work	Red Lamp (without tolerance)
green_lamp	BOOL	15.01		Work	Green Lamp (within tolerance)
thickness1	REAL	D2		Work	Measurement Value 1
thickness2	REAL	D4		Work	Measurement Value 2
thickness3	REAL	D6		Work	Measurement Value 3
average	REAL	D8		Work	Average Value
criterion	REAL	D10		Work	Criterion Value
margin	REAL	D12		Work	Tolerance
flag	INT	D20		Work	Three Times Measurement Flag

Note:
A function to automatically assign address can be used when registering symbols to enable registering symbols without worrying about actual addresses, just as is possible for symbols used in function blocks. Refer to the *CX-Programmer Operation Manual* for details.

4. Entering the ST Program

Open the ST Editor again.



Enter the program.

```
(* Average Value Calculation *)
average:=(thickness1+thickness2+thickness3)/3.0;    (* average =Average Value   Athickness1 to 3 =Measurement Value 1 to 3 *)

(* Judgement *)
IF flag = 3 THEN                                     (* flag =Three Times Measurement Flag *)
    IF average < criterion-margin  THEN              (* criterion =Criterion Value   Amargin =Tolerance *)
        red_lamp:=TRUE;                              (* red_lamp =Red Lamp (without tolerance) *)
    ELSIF average > criterion+margin THEN
        red_lamp:=TRUE;
    ELSE
        green_lamp:=TRUE;                            green_lamp =Green Lamp (within tolerance) *)
    END_IF;
END_IF;
```

Note:
Comments can be added to an ST program to make it easier to understand: (* *).

In a substitution statement, the value on the right (formula, symbol, or constant) is substituted for the symbol on the left.
This statement calculates the average value. Three measurements are added together, divided by 3, and then the result is assigned to the *average* symbol. Here, the constant 3 is entered as “3.0” so that it is in the same data type as the *average* symbol.

```
average := ( thickness1 + thickness2 + thickness3 )/3.0;
```

In an IF statement, the IF line is executed if the condition is true. If the condition is false, the lines from ELSEIF on will be executed. If both conditions are false, the lines from ELSE on are executed.

Here, the average value is evaluated after three measurements are taken. If the average value is not in range, the red lamp is lit. If the average value is in range, the green lamp is lit.

```
IF flag = 3 THEN
    IF average < criterion-margin THEN
        red_lamp := TRUE;
    ELSEIF average > criterion+margin
THEN
        red_lamp := TRUE;
    ELSE
        green_lamp := TRUE;
    END_IF;
END_IF;
```

F7

This completes entering the ST program. The remaining processing is programmed in ladder diagrams and then the F7 Key is pressed to compile and run an error check.
When the entire program has been completed, an online connection is made with the PLC and the normal program transfer operation is performed.

Appendix. Examples of ST (Structured Text)

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)

Standard String Functions

Function	Name	Argument data type	Return value type	Operation	Example
LEN(string)	String length	STRING	INT, UINT, WORD	Gets the string length.	a:= LEN(b)
LEFT(<target string>, <number of characters>)	Get substring from left	Target string: STRING No. of characters: INT, UINT, WORD	STRING	Gets part of the string from the left.	a:= LEFT(b, c)
RIGHT(<target string>, <number of characters>)	Get substring from right	Target string: STRING No. of characters: INT, UINT, WORD	STRING	Gets part of the string from the right.	a:= RIGHT(b, c)
MID(<target string>, <number of characters>, <position>)	Get substring	Target string: STRING No. of characters: INT, UINT, WORD Position: INT, UINT, WORD	STRING	Gets part of the string.	a:= MID(b, c, d)
CONCAT(<string1>, <string2>, …) *Up to 31 strings can be specified.	Concatenate	Strings: STRING	STRING	Joins strings.	a:= CONCAT(b, c, ...)
INSERT(<target string>, <substring>, <position>)	Insert substring	Target string: STRING Insert substring: STRING Position: INT, UINT, WORD	STRING	Inserts a substring into the string.	a:= INSERT(b, c, d)
DEL(<target string>, <number of characters>, <position>)	Delete substring	Target string: STRING No. of characters: INT, UINT, WORD Position: INT, UINT, WORD	STRING	Deletes part of the string.	a:= DEL(b, c, d)
REPLACE(<target string>, <replacement string>, <number of characters>, <position>)	Replace string	Target string: STRING Replacement string: STRING No. of characters: INT, UINT, WORD Position: INT, UINT, WORD	STRING	Replaces a substring.	a:= REPLACE(b, c, d, e)
FIND(<target string>, <search string>)	Find string	Target string: STRING Search string: STRING	INT, UINT, WORD	Finds a substring.	a:= FIND(b, c)

OMRON Expansion Functions

Function	Name	Argument data type	Return value type	Operation	Example
WRITE_TEXT(<write string>, <filename>, <delimiter>, <parameter>)	Create text file	Write string: STRING Filename: STRING Delimiter: STRING Parameter: INT, UINT, WORD	—	Creates a text file.	WRITE_TEXT(a, b, c, d)
TXD_CPU(<send string>)	Send string (CPU serial port)	Send string: STRING	—	Sends a string (CPU's serial port).	TXD_CPU(a)
TXD_SCB(<send string>, <serial port number>)	Send string (SCB serial port)	Send string: STRING Serial port number: INT, UINT, WORD	—	Sends a string (SCB's serial port).	TXD_SCB(a, b)
TXD_SCU(<send string>, <SCU unit number>, <serial port number>, <logical port number>)	Send string (SCU serial port)	Send string: STRING SCU unit number: INT, UINT, WORD Serial port number: INT, UINT, WORD Logical port number: INT, UINT, WORD	—	Sends a string (SCU's serial port)	TXD_SCU(a, b, c, d)
RXD_CPU(<receive string>)	Receive string (CPU serial port)	Receive string: INT, UINT, WORD	STRING	Receives a string (CPU's serial port).	a:= RXD_CPU(b)
RXD_SCB(<receive string>, <serial port number>)	Receive string (SCB serial port)	Receive string: INT, UINT, WORD Serial port number: INT, UINT, WORD	STRING	Receives a string (SCB's serial port).	a:= RXD_SCB(b, c)
RXD_SCU (<receive string>, <SCU unit number>, <serial port number>, <logical port number>)	Receive string (SCU serial port)	Receive string: INT, UINT, WORD SCU unit number: INT, UINT, WORD Serial port number: INT, UINT, WORD Logical port number: INT, UINT, WORD	STRING	Receives a string (SCB's serial port).	a:= RXD_SCU(b, c, d, e)
DEG_TO_RAD(argument)	Convert degrees to radians	REAL, LREAL	REAL, LREAL	Converts degrees to radians.	a:= DEG_TO_RAD
RAD_TO_DEG(argument)	Convert radians to degrees	REAL, LREAL	REAL, LREAL	Converts radians to degrees.	a:= RAD_TO_DEG

OMRON Corporation
Industrial Automation Company
Control Devices Division H.Q.
PLC Division
Shiokoji Horikawa, Shimogyo-ku,
Kyoto, 600-8530 Japan
Tel: (81) 75-344-7084/Fax: (81) 75-344-7149

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

OMRON ELECTRONICS LLC
One Commerce Drive Schaumburg,
IL 60173-5302 U.S.A.
Tel: (1) 847-843-7900/Fax: (1) 847-843-7787

OMRON ASIA PACIFIC PTE. LTD.
No. 438A Alexandra Road # 05-05/08 (Lobby 2),
Alexandra Technopark, Singapore 119967
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:

In the interest of product improvement,
specifications are subject to change without notice.

OMRON Industrial Automation Global: **www.ia.omron.com**

Cat. No. R144-E1-03