

PLC GX Works2 Basics

This training course (e-learning) is designed for those using the GX Works2 software for the first time to create sequence programs.

Introduction Purpose of the Course

This course provides basic knowledge of using the GX Works2 software for programming, debugging, and checking the operation of a programmable controller (PLC). The course is intended for those who create sequence programs for the MELSEC-Q series, MELSEC-L series, and MELSEC-F series controllers.

Introduction Course Structure

The contents of this course are as follows.
We recommend that you start from Chapter 1.

Chapter 1 - PLC System Control Method

The programming language and software used for programming are introduced here.

Chapter 2 - Program Design

You will learn how to design a program based on control items and hardware configuration.

Chapter 3 - Programming

You will learn how to program using the dedicated software GX Works2.

Chapter 4 - Debugging

You will learn how to write sequence programs to the CPU module and debug them.

Chapter 5 - Final Test

Passing grade: 60% or higher.

Introduction How to Use This e-Learning Tool

Go to the next page		Go to the next page.
Back to the previous page		Back to the previous page.
Move to the desired page		"Table of Contents" will be displayed, enabling you to navigate to the desired page.
Exit the learning		Exit the learning. Window such as "Contents" screen and the learning will be closed.

Introduction Cautions for Use

Safety precautions

When you learn by using actual products, please carefully read the safety precautions in the corresponding manuals.

Precautions in this course

- The displayed screens of the software version that you use may differ from those in this course.

Chapter 1 PLC System Control Method

This course is intended for persons who work with engineering software. It covers some of the fundamental concepts of managing MELSEC-Q, L, and F series systems.

GX Works 2 (GXW2) uses Internationally standardized programming languages including Sequential Function Chart (SFC) language, Instruction List (IL)*1, Ladder Logic, Function Block Diagram (FBD)*2 and Structured Text (ST).

Programs are developed using a personal computer running "engineering software," GX works2, and it is usually written to the programmable controller CPU via a USB, Ethernet*3 cable or Serial cable. The CPU module may be reprogrammed as many times is necessary to adapt to any required change in the desired control.

*1 Future plan for GX works2.

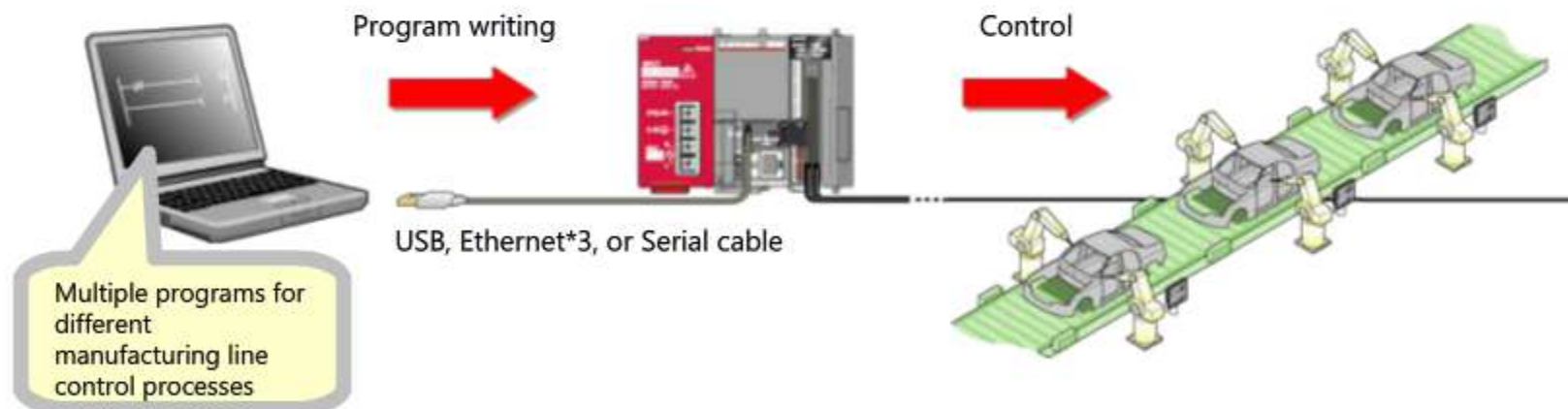
*2 Currently called Structured Ladder in GX works2, IEC compliance planned.

*3 Ethernet is a registered trademark of Xerox Corp.

Personal computer
(programming software)

PLC system

Automobile manufacturing line



In this course, ladder logic (one of the most popular PLC programming languages) is used in the example program. Although the example uses an L Series PLC, the contents of this course apply equally well to Q Series systems.

The basic control method is the same also for MELSEC-F series, but some of the operations and functions are different.

1.1

PLC System Construction Procedure

This e-learning course covers the software design steps (shown in green) necessary for implementing a programmable controller system.

Hardware design

(1) System design MELSEC-Q/MELSEC-L Basics Course



(2) Product selection MELSEC-Q/MELSEC-L Basics Course



(3) Advance preparation MELSEC-Q/MELSEC-L Basics Course



(4) Installation and wiring MELSEC-Q/MELSEC-L Basics Course



(5) Wiring check MELSEC-Q/MELSEC-L Basics Course

**Software design**

(6) Program design Chapter 2



(7) Programming Chapter 3



(8) Debugging Chapter 4



(9) Operation

**Scope of this
course**

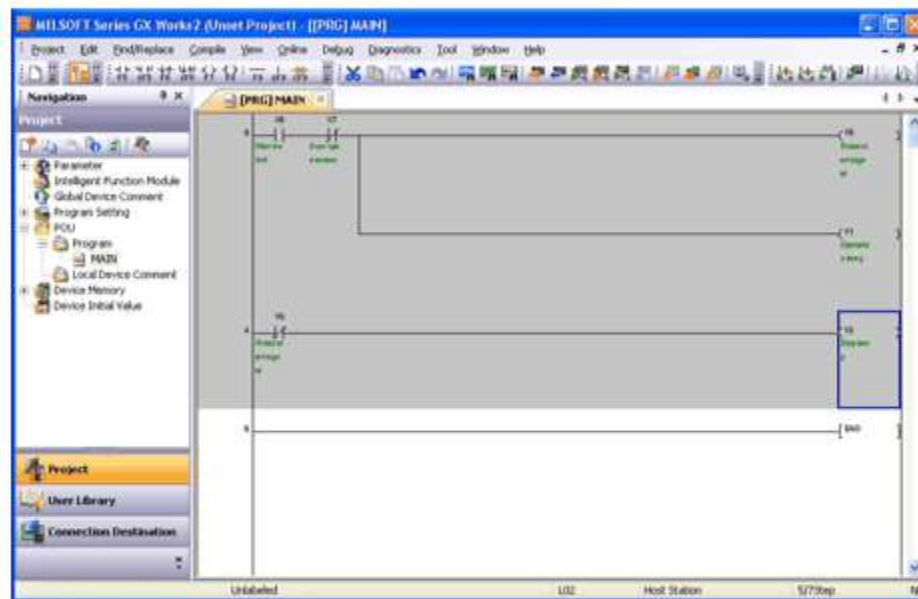
1.2

Requirements for Programming

In this course, the focus is on how to use the programmable controller engineering software GX Works2 to develop the example system program.

A few major functions of GX Works2 are listed below.

- Memory and file management
- Developing programmable controller programs
- Managing program documentation (comments, etc.)
- Reading writing data (especially programs) from/to the CPU module
- Verifying program operation
 - Software simulation of PLC hardware
 - Force I/O on or off
 - Monitor I/O and memory address status
- Perform maintenance and troubleshooting duties

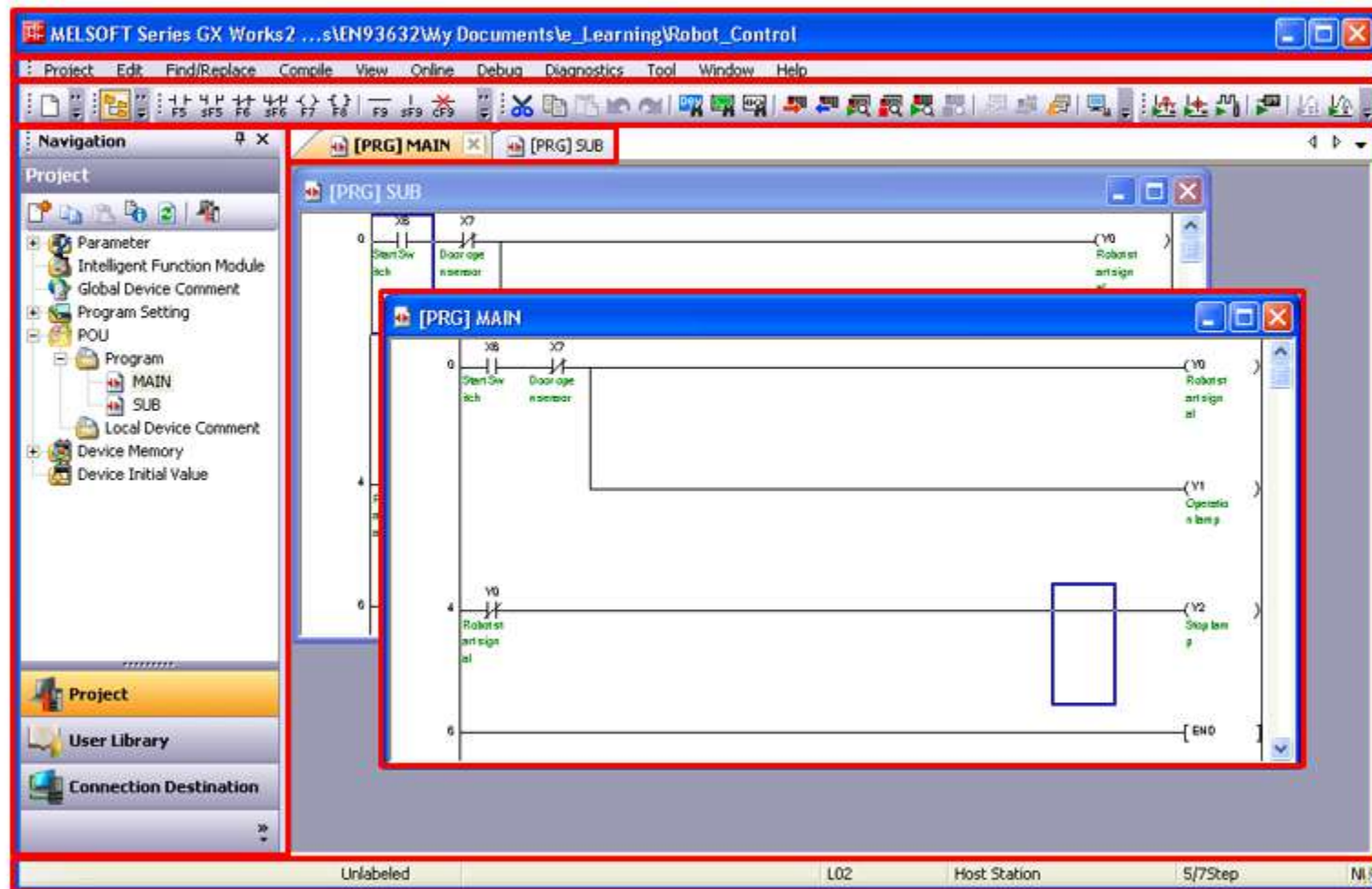


1.3

GX Works2 Screen Configuration

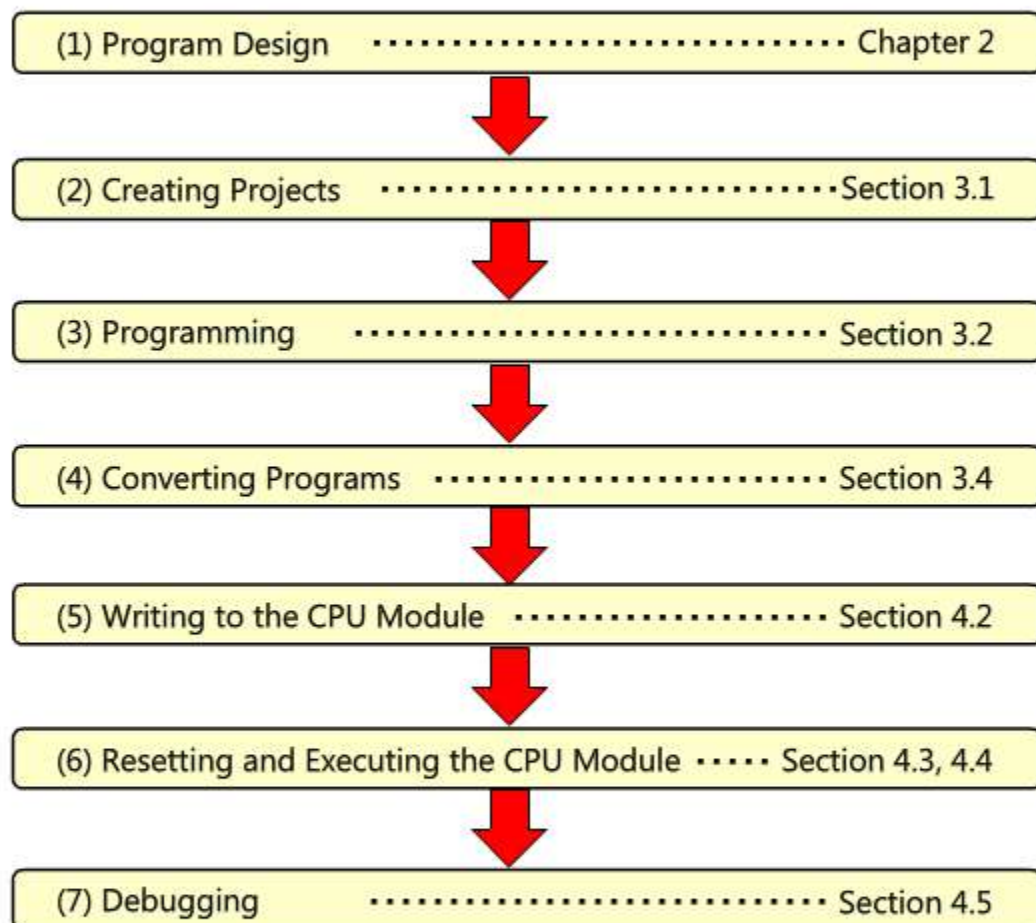
The GX Works2 screen configuration is shown below.

Place the mouse cursor in a red frame to display the respective function.



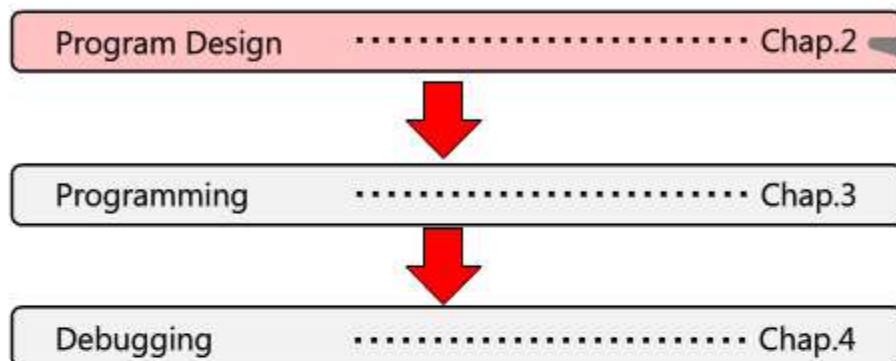
1.4**Sequence Program Creation Procedure**

Create a sequence program according to the following procedure.



Chapter 2 Program Design

In Chapter 2, you will learn how to design programs, including defining the contents of control and converting them into a program.



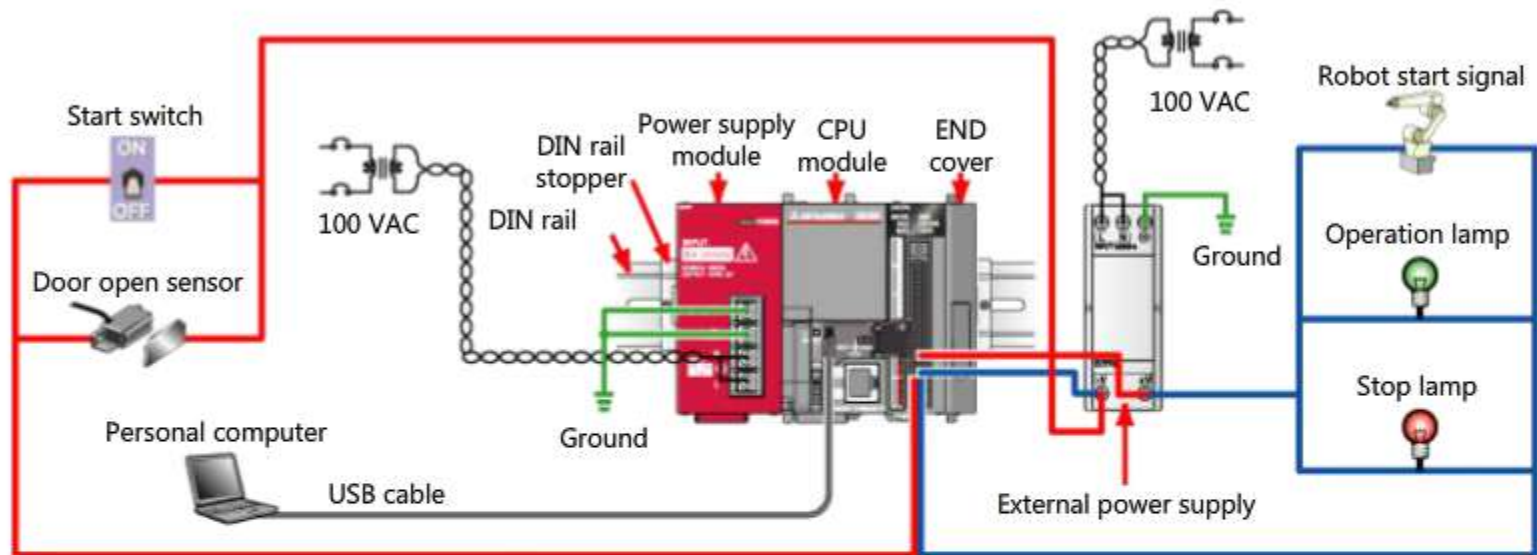
Learning steps in Chapter 2

- 2.1 Hardware Configuration of Example System Used for Learning
- 2.2 Defining Control Items
- 2.3 Creating a Correspondence Table of I/O Devices and Device Numbers
- 2.4 Designing a Program

2.1 Hardware Configuration of Example System Used for Learning

In this course, you will construct a PLC system (called “example system” hereafter), which starts the robot according to a procedure.

A diagram of the hardware configuration of the example system is shown below with a list of hardware components.



Item	Component	Model	Description
PLC system	Power supply module	L61P	Supplies power to modules including the CPU module and I/O module.
	CPU module	L02CPU	Controls the PLC system.
	END cover	L6EC	Attached to the right side of the system block.
	USB cable	MR-J3USBCBL3M	Connects the personal computer, in which GX Works2 is installed, to the CPU module.
	Personal computer	—	Runs with GX Works2 installed.
External power supply	—	—	Supplies power to external I/O devices.
External I/O equipment	Switch	—	Set to ON to start control.
	Sensor	—	Detects whether the door is open or closed.
	Robot	—	Operates in accordance with control signals.
	Two lamps	—	Light according to the operation status.

2.2

Defining Control Items

The first step of designing a program is to identify the devices to be controlled and the I/O devices necessary for the desired control. In the example system, control of the starting and stopping operation of a robot is performed. The robot will be prevented from starting if the door to the safety fence is open, and stopped if the door is opened during operation. See the animation below for a better understanding of how the example system will operate.

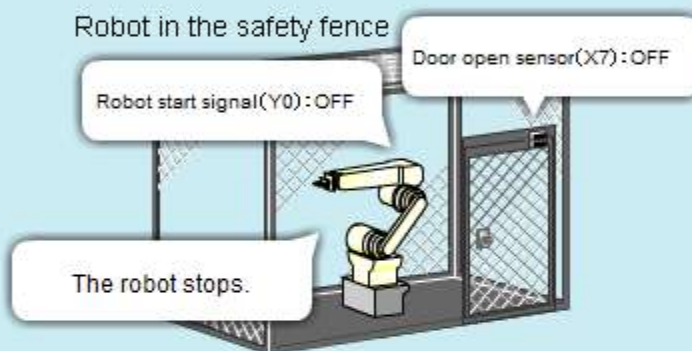
Example system operation

Click inside the red circle

Robot control panel



Robot in the safety fence



When you set the **start switch (X6)** to OFF, the **robot start signal (Y0)** turns off to stop robot operation. Simultaneously, the **operation lamp (Y1)** on the control panel turns off, and the **stop lamp (Y2)** turns on.

Replay



Previous

2.3 Creating a Correspondence Table of I/O Devices and Device Numbers

It is a good idea to create a table that includes all I/O devices and registers used in a PLC and their corresponding information for any program that is created. It reduces the chance mistakes will be made during the design and programming process and serves to increase programming efficiency. If a correspondence table already exists for the system, such as one created by the person who configured the hardware, make use of it.

The table below is a correspondence table for the example system used in this course

I/O device name	Device No.	I/O type	Device type	Description
Start switch	X6	Input	Bit	This switch starts or stops robot operation.
Door open sensor	X7	Input	Bit	This sensor checks whether the door of the safety fence of the robot is open. When the door opens, the sensor turns on. When the door closes, the sensor turns off.
Robot start signal	Y0	Output	Bit	When this signal turns on, the robot starts operation.
Operation lamp	Y1	Output	Bit	This lamp lights while the robot is operating.
Stop lamp	Y2	Output	Bit	This lamp lights while the robot is stopped.

* If word data is used, the initial value, setting range (upper and lower limits), data type (signed, real, etc.), and comment should be included in the table.

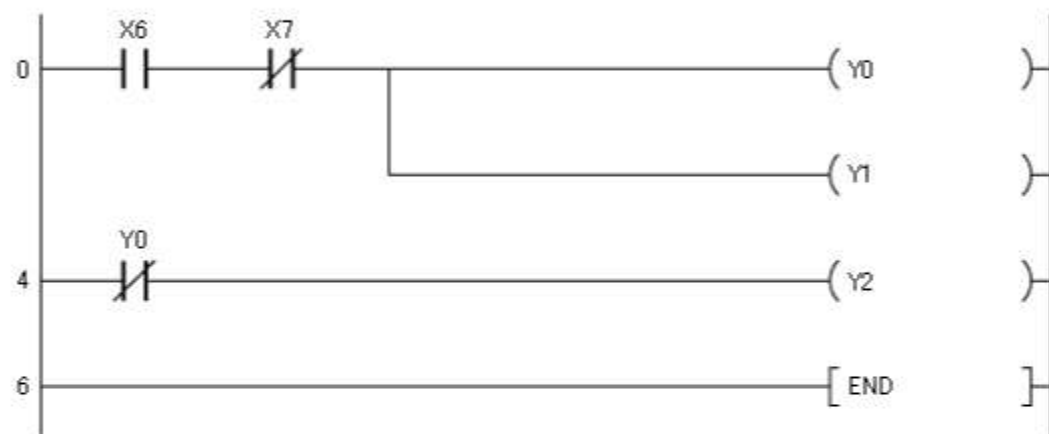
This information will be useful for designing and modifying programs.

2.4

Designing Programs

Design a program using the Ladder logic language based on the control items and the I/O correspondence table. The ladder program and I/O correspondence table designed for the example system is shown below.

Ladder program

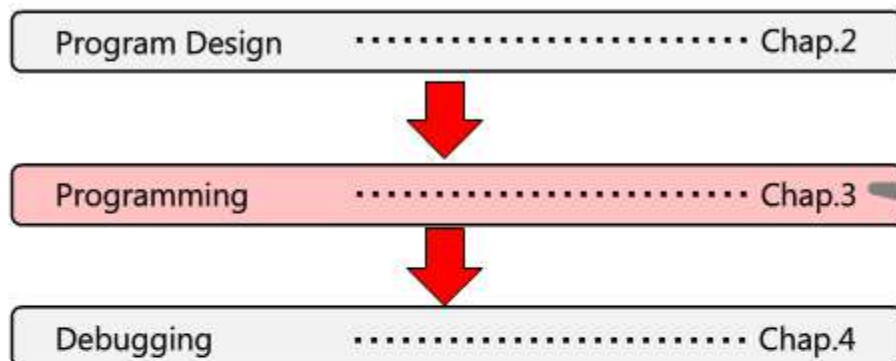


I/O correspondence table

I/O device name	Type	Device No.
Start switch	Input	X6
Door open sensor	Input	X7
Robot start signal	Output	Y0
Operation lamp	Output	Y1
Stop lamp	Output	Y2

Chapter 3 Programming

In Chapter 3, you will learn how to program the designed program using GX Works2.



Learning steps in Chapter 3

- 3.1 Creating Projects
- 3.2 Creating Programs
- 3.3 Making Programs Easy to Understand
- 3.4 Converting Programs into Executable Form
- 3.5 Saving Projects

3.1

Creating Projects

The first step to writing a program is to create a project.
The project is a collection of data GX Works2 uses to manage programs.
The following table lists the major components of a project.

Data type	Description
Program	The source code and compiled code for CPU sequence operations.
Comment	A type of documentation displayed inside the program. See Section 3.3 "Making Programs Easy to Understand" for details.
Parameter	Contains most of or all of the setting and configuration information for a system.
Transfer setup	The connection route information necessary for establishing communications between the system running GX Works2 and the CPU module.

Ladder program

GX Works2 allows you to select the following two project types.
The example program in this course uses the "simple project" type.

Project type	Description
Simple project	This project type is backwards compatible with GX Developer projects. Simple projects may be converted into Structured projects later, but not the other way around.
Structured project	These projects are capable of using an additional programming language called Structured Ladder. Additionally, programs can be separated into many small parts and the frequently used pieces of code can easily be modularized and reused using a user library. Labels can similarly be modularized for easy re-use. This can improve programming and debugging efficiency, especially for very large projects.

Labels

Labels are user created names that become aliases for device addresses. They can be used globally, locally, or system-wide when implemented in conjunction with MELSOFT Navigator. Simple projects may be created with or without the ability to use labels. For the example project, labels will not be used.

3.1

Creating Projects

To begin creating the example project, make the following settings.

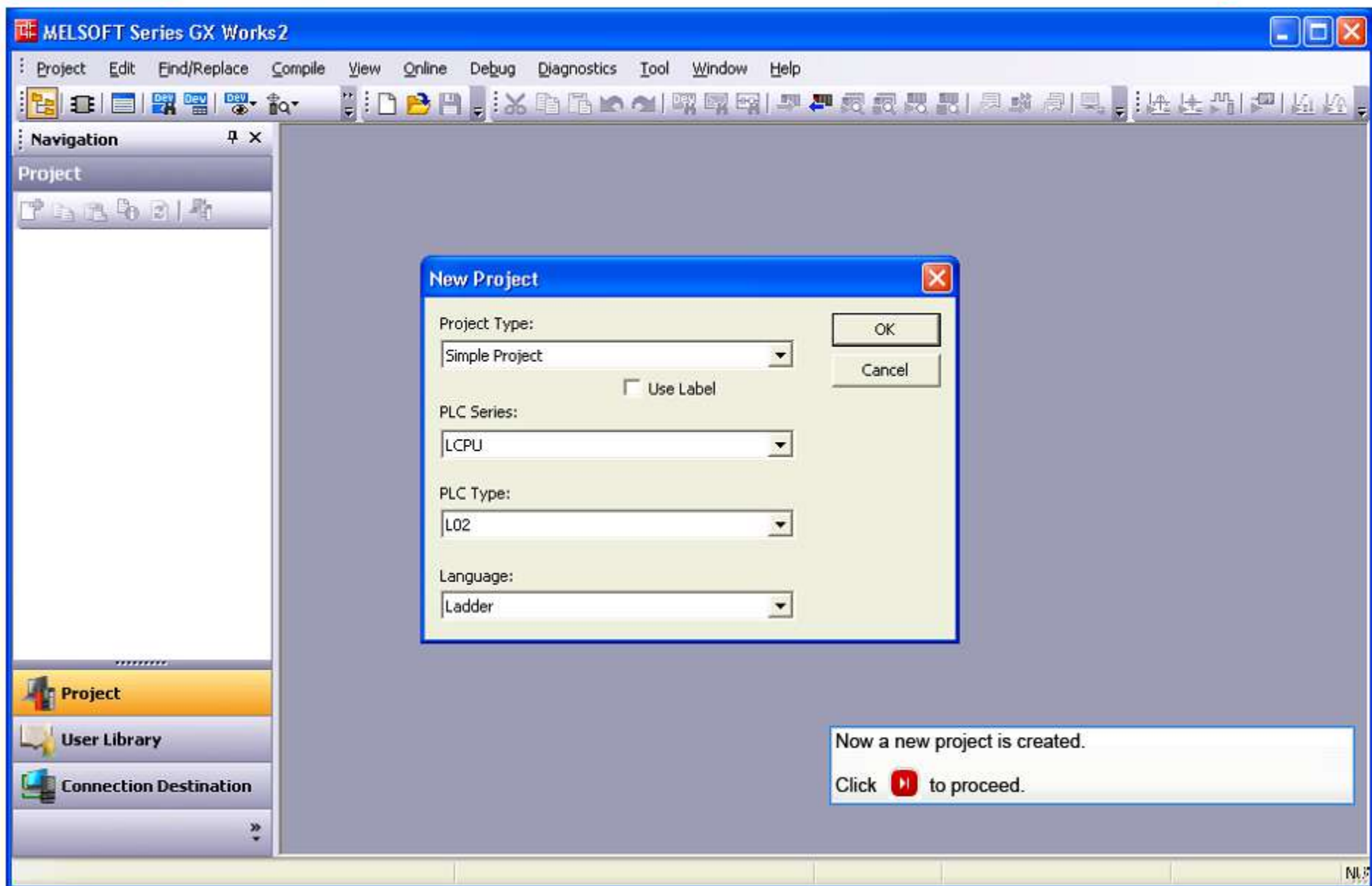
Before creating a project, the programmable controller series and model name, as well as the project type to be used, should be known.

Item	Description
Project type	The project type determines what features are available when writing programs. For this example, please choose "simple project."
Use label	If the ability to write programs using labels is required, check this item. The example program does not use labels. Therefore, leave this box unchecked.
PLC series	The PLC series determines the models available for selection in the PLC type drop-down list. For this example, please choose "LCPU."
PLC type	The PLC type determines how the compiler converts user programs into machine code. Choose the PLC model that will be programmed, in this case, "L02."
Programming language	The programming language determines the type of program of the first automatically created program (MAIN). Additional programs using different languages may be added later. For this example, please choose "Ladder."

Please review the next page, where the processes of creating a new project is simulated.

3.1

Creating Projects



3.2

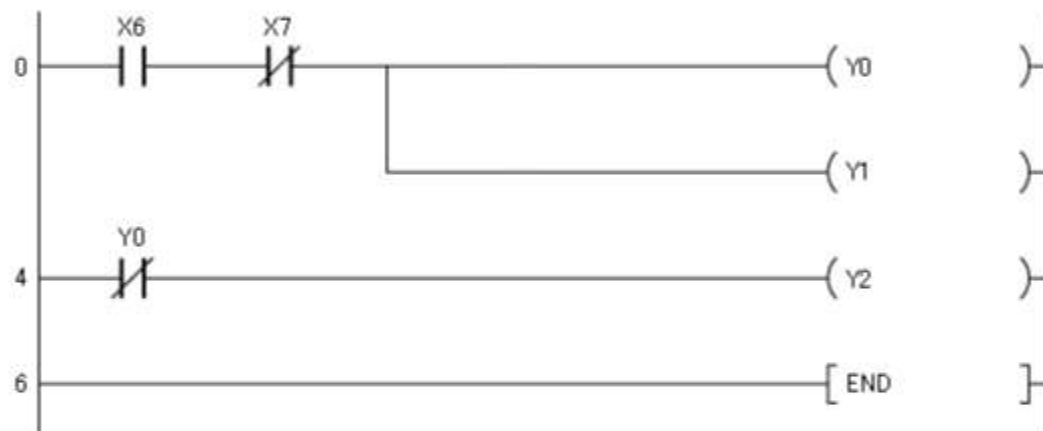
Creating Programs

After creating a project, let's create a program.

Create the following program and learn the basic operations (instruction input, change, delete, copy & paste, and ruled-line input/delete).

The program designed for the example system in Chapter 2 is shown below.

Program for example system



On the next page, try creating this program using the simulated window.

3.2

Creating Programs

MELSOFT Series GX Works2 (Unset Project) - [[PRG] MAIN]


Project Edit Find/Replace Compile View Online Debug Diagnostics Tool Window Help

Navigation

Project

- Parameter
- Intelligent Function Module
- Global Device Comment
- Program Setting
- POU
 - Program
 - MAIN
 - Local Device Comment
- Device Memory
- Device Initial Value

[PRG] MAIN

Now the Ladder circuit program is completed.
Click  to proceed.

Unlabeled L02 Host Station 0/15Step

3.3

Making Programs Easy to Understand

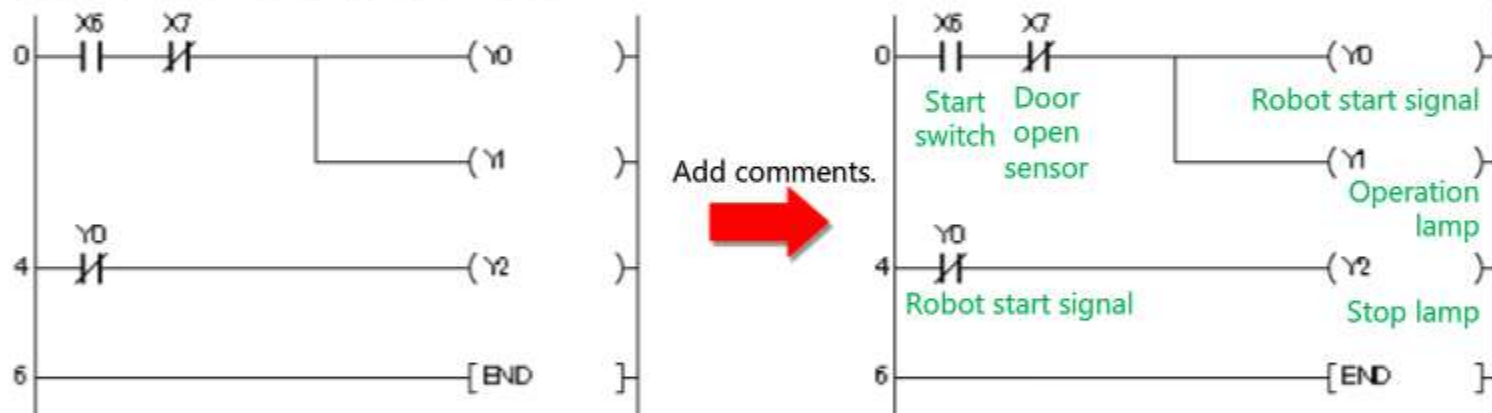
In its current state, the visual representation of the program only contains devices, instructions, lines, and step numbers. When looking at a complex program, it can be difficult to determine what the program is doing.

- Hard to find programming mistakes such as incorrect device numbers or instructions.
- Overall, difficult to perform operational analysis, debugging, and program expansion.
- If the original program developer can no longer maintain the program, the task of learning how the program operates for anyone else can be daunting and perhaps impossible.

Countermeasures

Include **documentation** in the program, to allow anyone to quickly understand how the program works.

As a matter of good practice, all programmers should add detailed comments their programs to allow themselves and others to better understand the program.



GX Works2 allows for three different types of comments to be used.

For more details, refer to the GX Works2 Simple Projects manual.

Comment type	Scope of comment
Device comment	Input up to 32 characters to be displayed under the selected device (I/O or other memory address.)
Statement	Input up to 64 characters per statement to be added at the top of the selected ladder block (above the step number). Each ladder block may have multiple statements.
Note	Input up to 32 characters to be displayed above the selected coil or application instruction.

The next page simulates the process of adding device comments to the example program.

3.3

Making Programs Easy to Understand

MELSOFT Series GX Works2 (Unset Project) - [[PRG] MAIN]

Project Edit Find/Replace Compile View Online Debug Diagnostics Tool Window Help

Navigation

Project

- Parameter
- Intelligent Function Module
- Global Device Comment
- Program Setting
- POU
 - Program
 - MAIN
- Local Device Comment
- Device Memory
- Device Initial Value


[PRG] MAIN

0 X6 Start switch X7 Door open sensor Y0 Robot start signal Y1 Operation lamp Y2 Stop lamp

4 Y0 Robot start signal

6 [END]

Unlabeled L02 Host Station 5/75Step

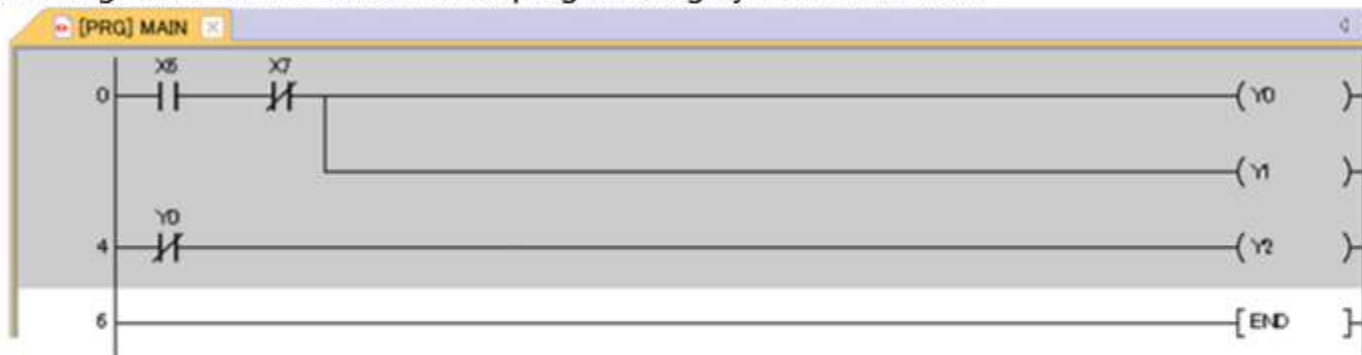
Device comment input is completed.
Click  to proceed.

3.4

Converting Programs into Executable Form

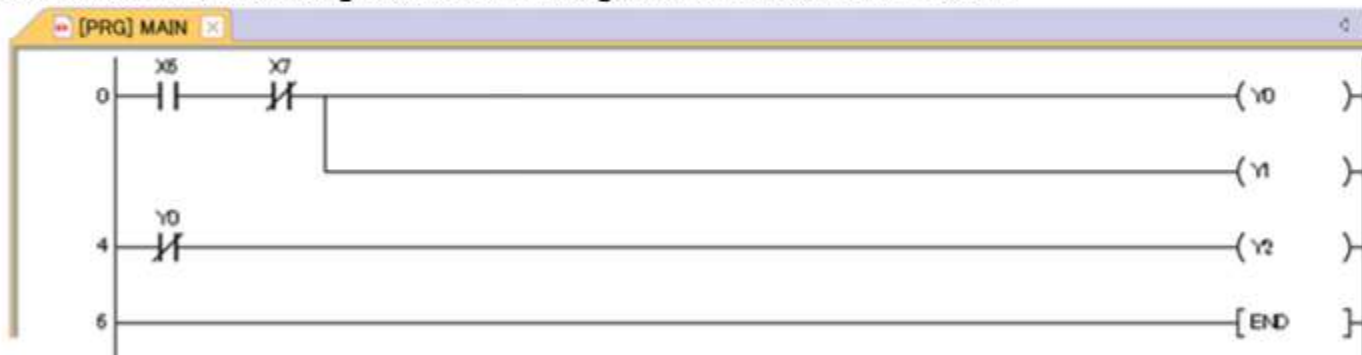
After completing the program, you need to convert it into a form that can be executed in the CPU module. Unconverted programs cannot be executed or saved.

The background color of unconverted programs is gray as shown below.



Convert

After conversion, the background color changes to white as shown below.



On the next page, try converting the program using the simulated window.

3.4

Converting Programs into Executable Form

MELSOFT Series GX Works2 (Unset Project) - [[PRG] MAIN]


Project Edit Find/Replace Compile View Online Debug Diagnostics Tool Window Help

Navigation

Project

- Parameter
- Intelligent Function Module
- Global Device Comment
- Program Setting
- POU
 - Program
 - MAIN
- Local Device Comment
- Device Memory
- Device Initial Value

When the program is converted, the background color changes from gray to white.

The program is converted.
Click  to proceed.

Unlabeled L02 Host Station 5/75Step

3.5

Saving Projects

After program conversion is finished, save the project including the programs. If GX Works2 is terminated without saving the project, the associated programs will be discarded, so you should save your project regularly.

When saving a new project, specify the following types of project information. (This is not required for overwrite-saving.) You should include information which makes it easy for others to understand the contents of control of the program, the system name, etc.

Item	Required	Description
Save destination path	✓	Specify the folder to which a workspace is to be allocated.
Workspace/project list		If one or more workspaces already exist in the folder specified at "Save destination path," the existing workspaces are listed.
Workspace name	✓	Specify a workspace name with up to 128 characters.
Project name	✓	Specify a project name with up to 128 characters.
Title		Specify a project title with up to 128 characters. This parameter is useful when you want to assign a long name that does not fit in "Project name."

The **workspace** is a folder to manage multiple projects.

An example of using a workspace is shown below. (Projects are managed for each vehicle type in the automobile manufacturing line.)

Workspace name	Project name	Title
Automobile manufacturing line	Type-A manufacturing line	Normal operation program for controlling the type-A manufacturing line
	Type-B manufacturing line	Normal operation program for controlling the type-B manufacturing line
	Type-C manufacturing line	Normal operation program for controlling the type-C manufacturing line

Notes:

- If a project containing an unconverted program is saved, only the unconverted program is discarded. Before saving a project, perform program conversion as you learned in Section 3.4.
- Specify the save destination path, workspace name, and project name so that the total number of characters does not exceed 150.

On the next page, try saving the project using the simulated window.

3.5

Saving Projects

MELSOFT Series GX Works2 C:\SequenceProgram\Learning\Robot_Control - [[PRG] MAIN]

Project Edit Find/Replace Compile View Online Debug Diagnostics Tool Window Help

Navigation

Project

- Parameter
- Intelligent Function Module
- Global Device Comment
- Program Setting
- POU
 - Program
 - MAIN
 - Local Device Comment
- Device Memory
- Device Initial Value

Unlabeled

L02

Host Station

6/75Step

NL

Diagram illustrating a ladder logic program for a robot control system. The program is titled "[PRG] MAIN".

The diagram shows three rungs:

- Rung 0:** A series connection of a normally open contact labeled "Start Switch" (X6) and a normally closed contact labeled "Door open sensor" (X7). This rung is connected to two outputs: "Robot start signal" (Y0) and "Operation lamp" (Y1).
- Rung 4:** A normally open contact labeled "Robot start signal" (Y0) is connected to a normally open output labeled "Stop lamp" (Y2).
- Rung 6:** The rung is terminated with an "END" block.

A message box in the bottom right corner states: "The project is saved. Click [Play icon] to proceed."

Chapter 4 Debugging

In Chapter 4, you will learn how to write sequence programs to the CPU module and debug them.

Program Design Chap.2



Programming Chap.3



Debugging Chap.4

Learning steps in Chapter 4

4.1 Debugging

4.1.1 Debugging a Program without Using the CPU Module

4.1.2 Changing the Status of an I/O Device

4.1.3 Monitoring the Device Status

4.2 Writing Programs to the CPU Module

4.3 Enabling Written Programs

4.4 Running Programs

4.5 Debugging Programs

4.6 Checking PLC System Operation

4.7 Operating the PLC System

4.1

What is Debugging?

Once a program or program segment has been written, it is necessary to test the code to ensure that it operates as expected.

Software defects (when code that is written does not perform as intended) are called "**bugs**", and the process of finding the cause of the unintended behavior and correcting it is known as "**debugging**."

Testing and debugging are essential steps in creating programs.

Particularly in programmable controllers because if bugs are present they could cause the system to stop, equipment to be damaged, or other accidents.

The following table lists a few of the functions in GX Works2 that can help the debugging process.

Function name	Description
Simulator	This function is used to simulate program execution even without a CPU module. This function can be used for debugging in an environment in which a CPU module is not available.
Monitor	This function enables monitoring the execution status and the status of each device during execution of the CPU module. Multiple monitor functions are available depending on the application, such as monitoring on the ladder, monitoring only registered devices, and monitoring all devices in a batch.
Change current value	This function can forcibly change the device status (bit: ON ↔ OFF, word: current value) during execution of the CPU module. This function is useful for changing the current value of a word device or the status of an internal relay.
Forced input output registration/cancellation	This function can forcibly change the status (ON ↔ OFF) of a registered I/O device during execution of the CPU module. For debugging or operation verification with a CPU module alone, this function can be used as a substitute for a switch.

These functions are explained in more detail with regards to the debugging process throughout the rest of this chapter.

Notes on debugging

Do not perform debugging tasks while the programmable controller is connected to physical I/O devices.

Bugs in the program, forced I/O devices, or word value changes could result in damage to external equipment or worse.

If a disconnected PLC system is unavailable, use the simulator function.

4.1.1

Debugging a Program without Using the CPU Module

If a CPU module is not available for debugging, use the **simulator function**.

A program can run on a virtual CPU module provided by the software without using an actual CPU module.



 On

 Off

Item	State	Description
Switch	RUN	Runs the virtual CPU module.
	STOP	Stops the virtual CPU module.
	RESET	Resets the virtual CPU module. (Enabled only in the STOP state)
LED	MODE	Indicates the MODE status of the virtual CPU.
	RUN	Indicates the run status of the virtual CPU. •On: RUN state •Off: STOP state
	ERR	Indicates the error status of the virtual CPU module. If an error is present, the LED will turn on or blink.
	USER	Indicates whether a user error has occurred in the virtual CPU. Turns on or blinks when an error occurs.

Notes on using the simulator function

- Debugging using the simulator function does not guarantee that the sequence program will operate correctly after debugging.
- The simulator function executes input/output of data with I/O modules using simulation memory. The function does not support some instructions, functions, and device memory. Therefore, the results of operation with the simulator function may differ from those with the actual CPU module.

On the next page, try using the simulator function with the simulated window.

4.1.1

Debugging a Program without Using the CPU Module

MELSOFT Series GX Works2 C:\SequenceProgram\Learning\Robot_Control - [[PRG] MAIN]

Project Edit Find/Replace Compile View Online Debug Diagnostics Tool Window Help

Navigation [PRG] MAIN

Project


- Parameter
- Intelligent Function Module
- Global Device Comment
- Program Setting
- POU
 - Program
 - MAIN
 - Local Device Comment
- Device Memory
- Device Initial Value

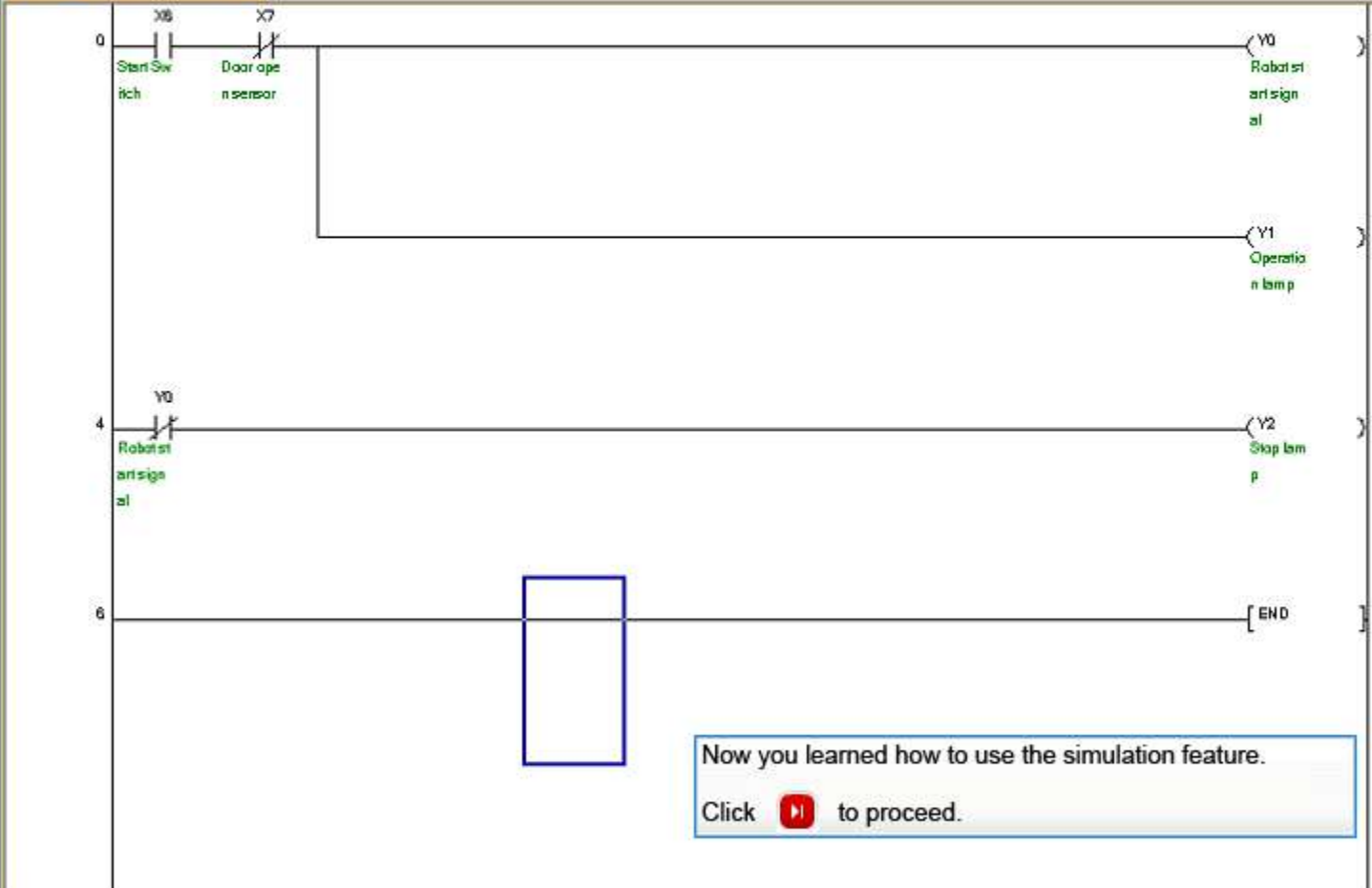
Unlabeled

L02

Host Station

6/75Step

Now you learned how to use the simulation feature.
Click  to proceed.



4.1.2

Changing the Status of an I/O Device

When debugging a sequence program with a CPU module to which no I/O device is connected or using the simulator function, use the **Forced Input Output Registration/Cancellation** function to change the ON/OFF state of an I/O device. The status of the registered I/O devices can be forcibly changed to ON or OFF with the software.

(MELSEC-Q and MELSEC-L series): From "Forced Input Output Registration/Cancellation" screen

(MELSEC-F series): From "Modify Value" screen

No.	Device	ON/OFF
1	X6	ON
2	X7	OFF
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		

Forced Input Output Registration/Cancellation screen (MELSEC-Q and MELSEC-L series)

Device/Label	Data Type	Setting Value
X6	Bit	ON

Modify Value screen (MELSEC-F series)

To change the states of other devices

To change the current device of a word device or the ON/OFF state of an internal relay, use the **current value change function**.

For details, refer to the manual.

4.1.3

Monitoring the Device Status

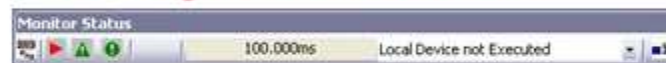
When simulation is started, monitoring automatically begins. To enter monitoring mode when connected to an actual programmable controller CPU, simply click Online, Monitor, and then Start Monitoring. Or use the keyboard short-cut, F3.
















During monitor mode the values and status of all devices used in the program can be seen overlaying the program code. This allows the user to see values changing including the effects of using the "forced input output registration/cancellation" function.

Additionally, the **Monitor Status** bar appears and includes basic information to determine the CPU or virtual CPU status. Refer to the table below to for an understanding of the information provided by the **Monitor Status** bar.

When connected to the CPU module

When using the simulator function



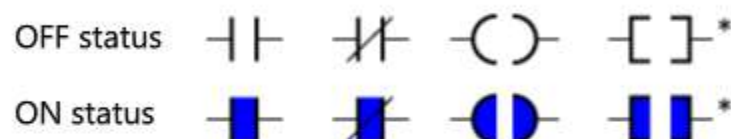
State	Icon/indication	Description
Connection status	 When connected to CPU module	Displays the status of the connection with the CPU module or simulator function.
	 When using the simulator function	
RUN/STOP status	 RUN	Displays the run status of the CPU (RUN or STOP).
	 STOP	
ERR. status	 ERR. off	Displays the error status of the CPU module.
	 ERR. on	
	 ↔  ERR blinking	
USER status	 USER off	Displays the user error status of the CPU module.
	 USER on	
	 ↔  USER blinking	
Scan time	 0.000ms	Displays the maximum scan time of the CPU module being monitored.
Unsupported instruction presence/absence status	 Unsupported instruction exists.	Displays whether an unsupported instruction exists when the simulator function is executed. Clicking the icon opens the Unsupported Instruction/Device window.
	 Unsupported instruction does not exist.	

4.1.3 Monitoring the Device Status

During monitoring mode, the current status of all devices in the program become visible.

Bit device status display (ON/OFF)

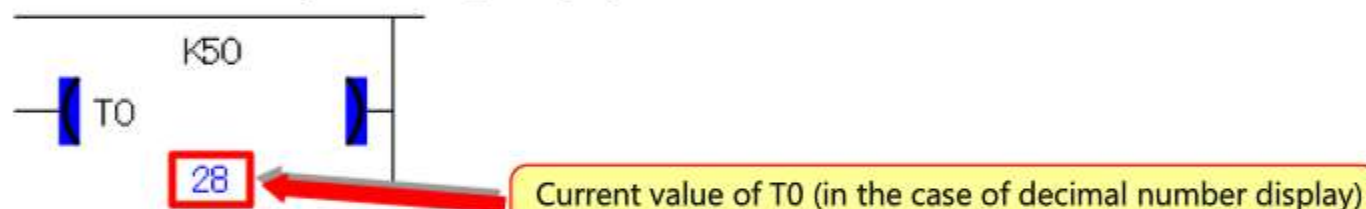
The ON/OFF status is displayed during monitoring as shown below.



* This kind of display applies to only SET, RST, PLS, PLF, SFT, SFTP, MC, and contact type comparison instructions.
Note that for the RST instruction, only the ON/OFF status is displayed.

Current value display of word device (decimal/hexadecimal number display)

The current value during monitoring is displayed as shown below.



Monitoring only specific devices

When monitoring a very large or complex program, it may be beneficial to only monitor certain devices of interest. To accomplish this, GX Works2 includes watch windows that allow the user to easily add the devices they are interested in, see their current status, and modify their values during monitoring. for details, refer to the GX Works2 Operation Manual (Common).

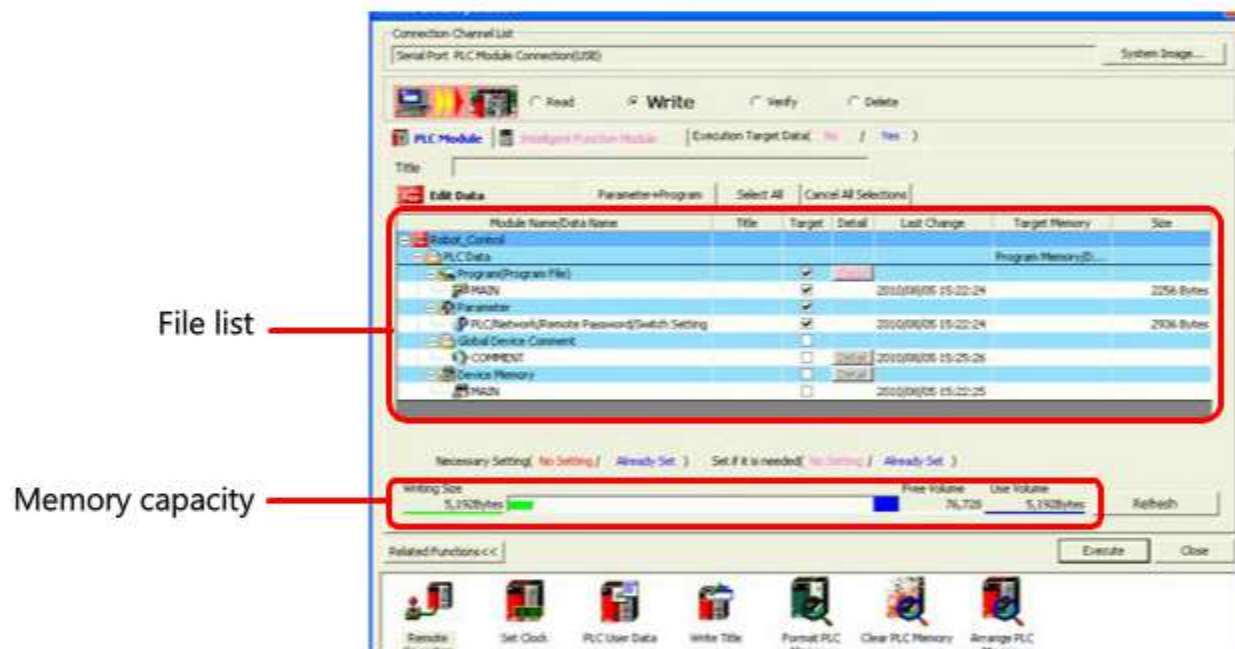
Watch 1					
Device/Label	Current Value	Data Type	Class	Device	Comment
X7	-	Bit		X7	Door open sensor
Y0	-	Bit		Y0	Robot start signal
Y1	-	Bit		Y1	Operation lamp
Y0	-	Bit		Y0	Robot start signal
Y2	-	Bit		Y2	Stop lamp
Y0	-	Bit		Y0	Robot start signal

4.2

Writing Programs to the CPU Module

Before performing any debugging using an actual CPU module, place the CPU in **STOP mode**, ensure a connection to the CPU has been established, and write the programs and parameters to program memory.

As seen in the screenshot below, the main functions of the **Write to PLC** window allow the user to select the desired files to be written, choose their location, and confirm the memory capacity of the CPU. The three buttons above the file list allow the user to quickly select the desired files to be written. The most common, which is used in the following simulation, is "**Parameter+Program**."



On the next page, try writing to the CPU module using the simulated window.

4.2

Writing Programs to the CPU Module

MELSOFT Series GX Works2 C:\SequenceProgram\Learning\Robot_Control - [[PRG] MAIN]

Project Edit Find/Replace Compile View Online Debug Diagnostics Tool Window Help

Navigation

Project

- Parameter
- Intelligent Function Module
- Global Device Comment
- Program Setting
- POU
 - Program
 - MAIN
 - Local Device Comment
- Device Memory
- Device Initial Value

[[PRG] MAIN

0 X6 Start Switch X7 Door open sensor

Y0 Robot start signal


Y1 Operation lamp

4 Y0 Robot start signal

Y2 Stop lamp

6 END

Unlabeled L02 Host Station 6/75Step

The program is now written to the PLC module.
Click  to proceed.

4.3

Enabling Written Programs

(MELSEC-F series): The following operation is not necessary.

(MELSEC-Q and MELSEC-L series): The following operation is necessary.

After writing a program to the CPU module, **reset** the CPU module.

Written programs are not enabled unless the CPU module is reset.

* This operation is not required if the simulator function is used for debugging.

Reset the CPU module as follows:

- (1) Press and hold the RESET/STOP/RUN switch on the front panel of the CPU module to the RESET position (for 1 second or more).

[Resetting in progress]

L02CPU	
MODE ■	ERR.
RUN	I/OERR.
BAT.	USER

MODE : On in green
RUN : Off
ERR. : Blinking

Press down for 1 second or more.



- (2) Release the switch after the lit MODE LED and blinking ERR. LED both turn off.

[Resetting complete]

L02CPU	
MODE ■	ERR.
RUN	I/OERR.
BAT.	USER

MODE : On in green
RUN : Off
ERR. : Off

RESET/STOP/
RUN switch

- (3) The switch returns to the STOP position to complete resetting.

4.4 Running Programs

MELSEC-Q and MELSEC-L series

After resetting is completed, run the program.
Put the CPU module in the **RUN status** as follows to run the program.

* This operation is not required if the simulator function is used for debugging.

(1) Turn the RESET/STOP/RUN switch on the front panel of the CPU module to the RUN position.

LED display in STOP status

L02CPU	MODE	ERR.	MODE : On in green
	RUN	I/OERR.	RUN : Off
	BAT.	USER	



(2) If the RUN LED lights in green, the program is running normally.

LED display in RUN status

L02CPU	MODE	ERR.	MODE : On in green
	RUN	I/OERR.	RUN : On in green
	BAT.	USER	

MELSEC-F series

After writing a program into the main unit, switch the main unit to the RUN status as follows to run the program. (Reset operation is not necessary.)

(1) Turn the RUN/STOP switch on the front panel of the main unit to the RUN position.



LED display in STOP status

(2) If the RUN LED lights, the program is running normally.



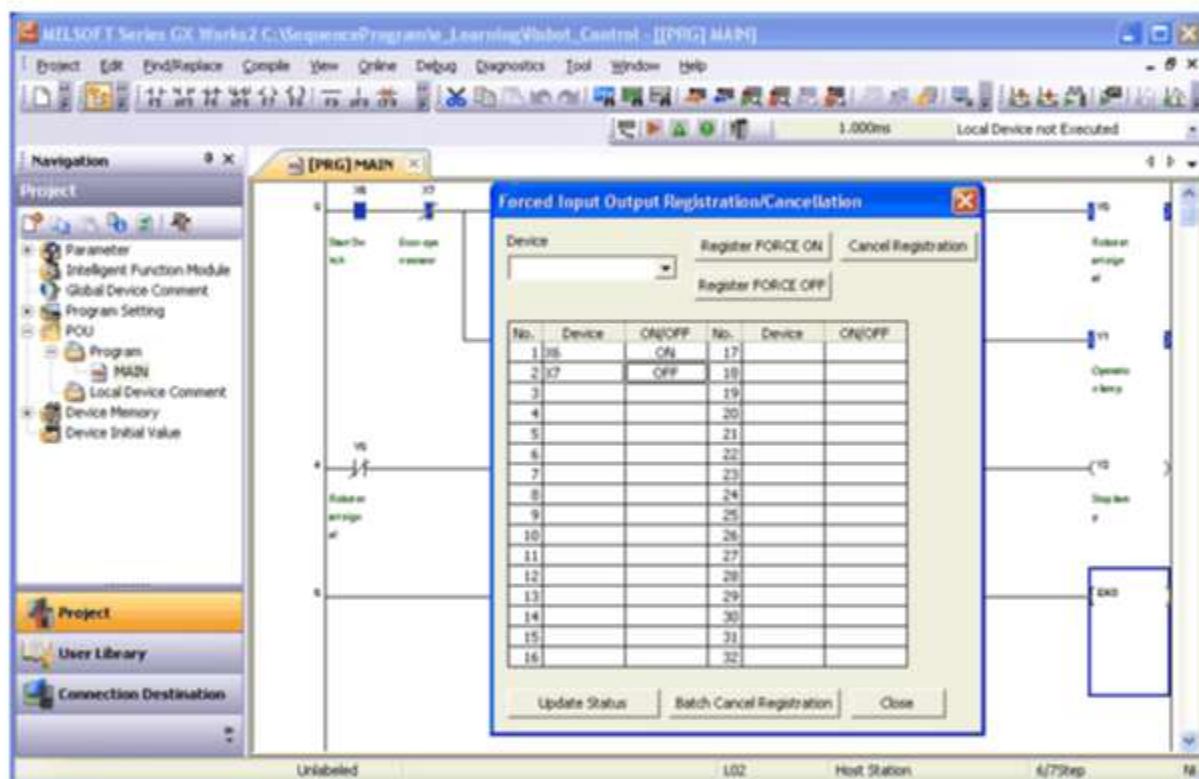
LED display in RUN status

4.5

Debugging Programs

After running the CPU module, use the forced input output registration/cancellation function to change the status of each device and monitor the result (output) on the ladder.

(Screen example of MELSEC-Q and MELSEC-L series)



On the next page, try program debugging using the simulated window.

4.5

Debugging Programs

MELSOFT Series GX Works2 C:\SequenceProgram\Learning\Robot_Control - [[PRG] MAIN]

Project Edit Find/Replace Compile View Online Debug Diagnostics Tool Window Help

1.000ms Local Device not Executed

Navigation

Project

- Parameter
- Intelligent Function Module
- Global Device Comment
- Program Setting
- POU
 - Program
 - MAIN
- Local Device Comment
- Device Memory
- Device Initial Value

Project

User Library

Connection Destination

[PRG] MAIN


0 X6 X7 Y0 Y1

Start Switch Door open sensor Robot start signal Operation lamp

4 Y0 Y2

Robot start signal Stop lamp

6 END

Debugging of the program is completed.
Click  to proceed.

Unlabeled L02 Host Station 6/75Step

4.6

Checking PLC System Operation

After program debugging is completed, write the program to the actual PLC system to finally check the operation. Operate the actual I/O equipment to confirm that it works as designed. Even when operating the I/O equipment, the status of each device can be checked using the monitor function of GX Works2.

Example system operation

Click inside the red circle

Robot control panel



Robot in the safety fence

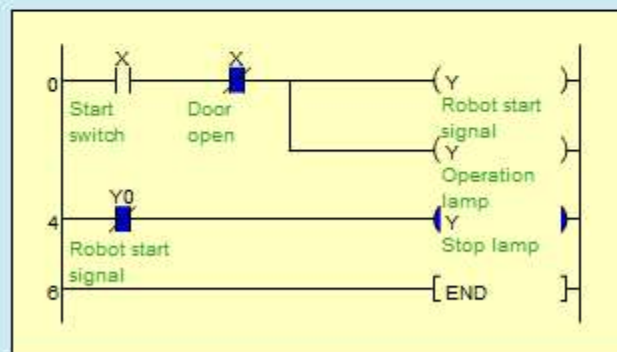


When you set the **start switch (X6)** to OFF, the **robot start signal (Y0)** turns off to stop robot operation. Simultaneously, the **operation lamp (Y1)** on the control panel turns off, and the **stop lamp (Y2)** turns on.

Replay



Previous



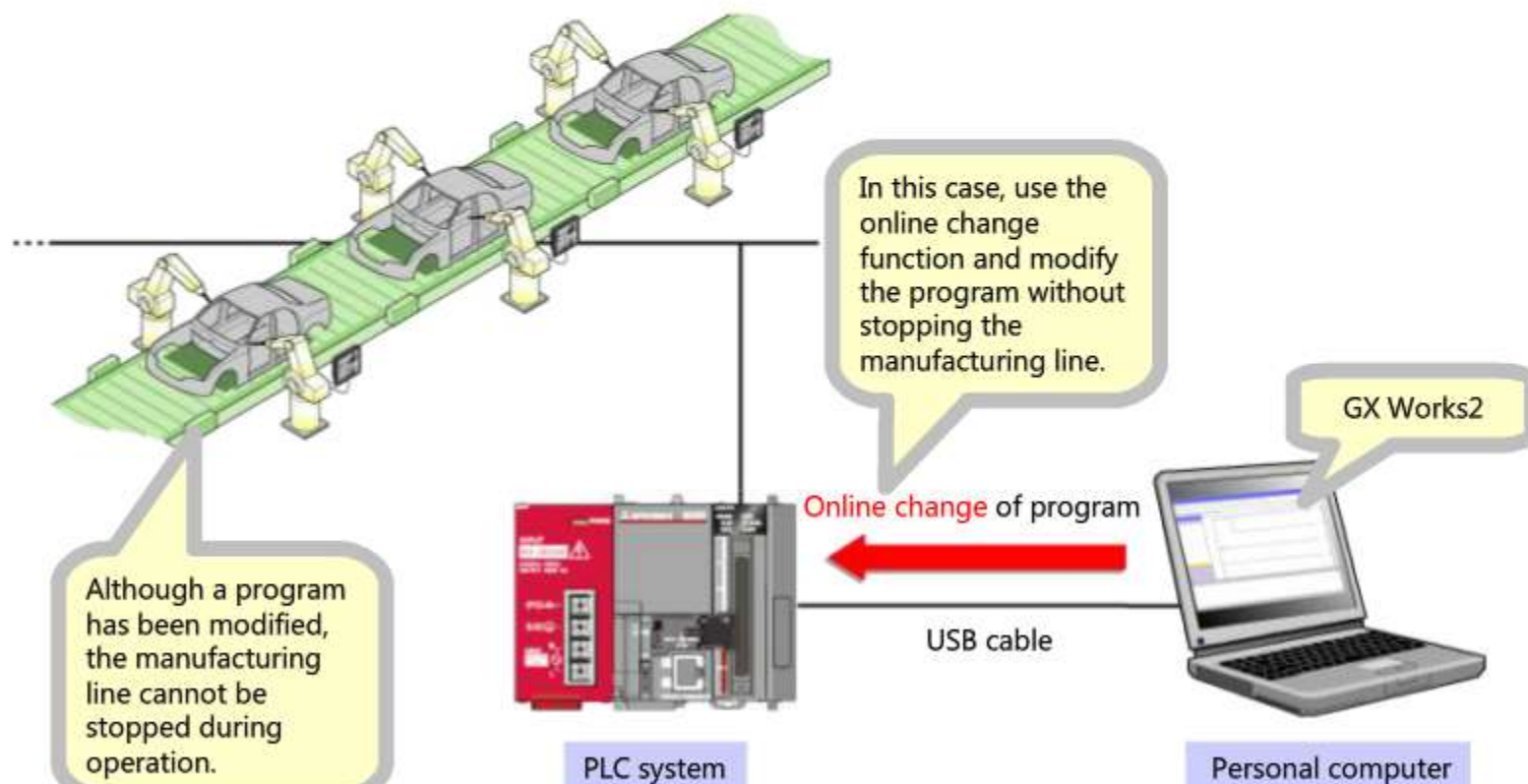
4.7

Operating the PLC System

After operation verification is completed, run the PLC system to start operation.

If the program needs to be modified in the running system

Program modification such as bug correction or system expansion may be required after system operation is started. Normally, the system (CPU module) needs to be stopped to write a modified program, but this is not always possible. To solve this problem, GX Works provides an online change function, which is used to write programs without stopping the running CPU module.

Example: 24-hour running automobile manufacturing line

On the next page, try the online change function using the simulated window.

4.7

Operating the PLC System

MELSOFT Series GX Works2 C:\SequenceProgram\Learning\Robot_Control - [[PRG] MAIN]

Project Edit Find/Replace Compile View Online Debug Diagnostics Tool Window Help

1.000ms Local Device not Executed

Navigation

Project

- Parameter
- Intelligent Function Module
- Global Device Comment
- Program Setting
- POU
 - Program
 - MAIN
- Local Device Comment
- Device Memory
- Device Initial Value

Project

User Library

Connection Destination

[PRG] MAIN


0 X0 X0 Y0 Y1

Start Switch Robot start signal

4 Y0 Y2

Robot start signal Stop lamp

6 [END]

Online change of the modified program has completed.
Click  to proceed.

Unlabeled L02 Host Station 2/75Step

4.8**Conclusion**

This completes the basic explanation of programmable controller software design.

In this course you have learned:

- The required elements for programming a PLC system
- Some basic guidelines for program design including the use of comments
- How to use GX works2 to perform the basic tasks of PC programming
- A few techniques used for debugging PLC programs

Test**Final Test**

Now that you have completed all of the lessons of the **PLC GX Works2 Basics** Course, you are ready to take the final test. If you are unclear on any of the topics covered, please take this opportunity to review those topics.

There are a total of 5 questions (15 items) in this Final Test.

You can take the final test as many times as you like.

How to score the test

After selecting the answer, make sure to click the **Answer** button. Your answer will be lost if you proceed without clicking the Answer button. (Regarded as unanswered question.)

Score results

The number of correct answers, the number of questions, the percentage of correct answers, and the pass/fail result will appear on the score page.

Correct Answers : **2**

Total Questions : **9**

Percentage : **22%**

To pass the test, you have to answer **60%** of the questions correct.

Proceed

Review

Retry

- Click the **Proceed** button to exit the test.
- Click the **Review** button to review the test. (Correct answer check)
- Click the **Retry** button to retake the test again.

Test**Final Test 1**

The program you were in charge of was taken over by another person, who found it difficult to understand the control items for the program. What is the proper countermeasure to prevent this problem?

- ☐ Using the comment function of GX Works2, give the program an appropriate title and explanation.
- ☐ Orally explain the control items to the new person.
- ☐ Avoid taking over a complex, large program.
- ☐ Transfer the correspondence table for I/O devices and device numbers together with the program.

[Answer](#)[Back](#)

Test**Final Test 2**

Complete the correct programming procedure.

Step 1 Program design

Step 2 (Q1)

Step 3 (Q2)

Step 4 Converting programs

Step 5 Saving projects

Step 6 (Q3)

Step 7 (Q4)

Step 8 Running the CPU module (RUN)

Step 9 (Q5)

Step 10 Checking PLC System Operation

Answer

Back

Test**Final Test 3**

Fill in the blanks to complete the explanation of what needs to be done after a program is completed.

Once a program has been written, it must be tested to ensure that it operates as expected.

A () (when code is written that does not perform as intended) is called

a () and the process of finding the cause and correcting it is called

().

This process is an essential step in creating programs.

[Answer](#)[Back](#)

Test**Final Test 4**

Select the appropriate application of each GX Works function.

Function	Application
Simulation	--Select--
Forced input output registration/cancellation	--Select--
Change current value	--Select--
Ladder monitor	--Select--
Watch	--Select--

[Answer](#)[Back](#)

Test**Final Test 5**

Select the correct description of the online change function.

- ☐ The function automatically stops the CPU, writes a program to the CPU, and then automatically runs the CPU.
- ☐ The function compares the program in the running CPU module with the program opened by GX Works2.
- ☐ The function can write a program to the CPU module after stopping the running CPU module safely.
- ☐ The function can write a program to the running CPU module without stopping it.

[Answer](#)[Back](#)

Test**Test Score**

You have completed the Final Test. Your results are as follows.
To end the Final Test, proceed to the next page.

Correct answers : **0**

Total questions : **5**

Percentage : **0%**

[Proceed](#)[Review](#)[Retry](#)

You failed the test.

You have completed the **PLC GX Works2 Basics** Course.

Thank you for taking this course.

We hope you enjoyed the lessons and the information you acquired in this course will be useful in the future.

You can review the course as many times as you want.

Review

Close