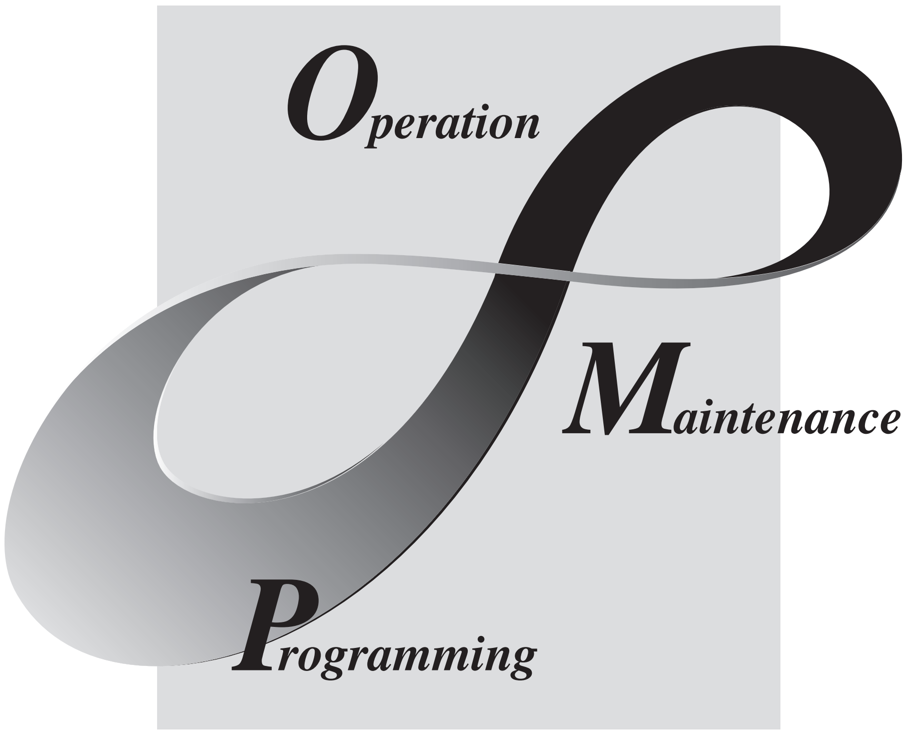
GX IEC Developer Version 7

Beginner's Manual

MITSUBISHI

MELSOFT



MELSOFT
Integrated FA Software

SW10D5C-MEDOC3-E

• SAFETY PRECAUTIONS •

(Always read these instructions before using this equipment.)

Before using this product, please read this manual and the relevant manuals introduced in this manual carefully and pay full attention to safety to handle the product correctly.
The instructions given in this manual are concerned with this product. For the safety instructions of the programmable controller system, please read the CPU module user's manual.
In this manual, the safety instructions are ranked as "DANGER" and "CAUTION".

◇ DANGER	Indicates that incorrect handling may cause hazardous conditions, resulting in death or severe injury.
⚠ CAUTION	Indicates that incorrect handling may cause hazardous conditions, resulting in medium or slight personal injury or physical damage.

Note that the ⚠CAUTION level may lead to a serious consequence according to the circumstances.
Always follow the instructions of both levels because they are important to personal safety.

Please save this manual to make it accessible when required and always forward it to the end user.

[Design Instructions]

◇ DANGER

- For data change, program change, and status control made to the PLC which is running from a Personal computer, configure the interlock circuit externally so that the system safety is ensured. The action to be taken for the system at the occurrence of communication errors caused by such as loose cable connection must be determined for online operation of PLC from Personal computers.

⚠ CAUTION

- Be sure to read the manual careful and exercise an appropriate amount of caution connecting to PLC CPU and performing online operations (PLC CPU program change during RUN, forced input/output operation, RUN-STOP or other operation condition changes, remote control operation) while the personal computer is operating.
  Regarding the PLC CPU program change during RUN (Online change), the program may be corrupted or have other problems depending on operation conditions. Exercise the appropriate amount of caution with regard to the Caution points in the Reference Manual.
- Please refer to the manual of each module for online module change and swap module during run, since there is restriction on the exchangeable module.

About this Manual

Beginner's Manual for
MELSOFT GX IEC Developer
Art. no.: 43596

Version			Changes / Additions / Corrections
A	03/1995	ME	First issue
B	05/1996	ME	Software update
C	07/1997	ME	Software update
D	01/1998	ME	Software update
E	08/2000	pdp-rs	Update to software version 2.40
F	06/2001	pdp-rs	Update to software version 4.00
G	05/2002	rs/pdp	Update to software version 5.00
H	09/2003	ow/pdp	Update to software version 6.00
I	09/2004	ow/pdp	Update to software version 6.10
J	09/2005	ow/pdp	Update to software version 7.00
K	11/2006	ow/pdp	Update to software version 7.01
L	09/2007	ow/pdp	Update to software version 7.02
M	10/2008	ow/pdp	Update to software version 7.03

Typographic Conventions

Use of notes

Notes containing important information are clearly identified as follows:

NOTE | Note text

Use of examples

Examples containing important information are clearly identified as follows:

Example ▽ Example text

△

Numbering in figures and illustrations

Reference numbers in figures and illustrations are shown with white numbers in a black circle and the corresponding explanations shown beneath the illustrations are identified with the same numbers, like this:
❶ ❷ ❸ ❹

Procedures

In some cases the setup, operation, maintenance and other instructions are explained with numbered procedures. The individual steps of these procedures are numbered in ascending order with black numbers in a white circle, and they must be performed in the exact order shown:

① Text

② Text

③ Text

Footnotes in tables

Footnote characters in tables are printed in superscript and the corresponding footnotes shown beneath the table are identified by the same characters, also in superscript.

If a table contains more than one footnote, they are all listed below the table and numbered in ascending order with black numbers in a white circle, like this:

① Text

② Text

③ Text

Character formatting and orientation aids

Menu names, menu commands, submenu commands, and dialog box options are printed in **boldface** type. Examples: The menu item **New** in the menu **Project** or the options **PLC interface** and **Computer Link** in the dialog box **Transfer-Setup**.

Please keep this manual in a place where it is always available for the users.

Contents

1 Introduction

1.1 This manual…

...is a compact guide to using GX IEC Developer, suitable both for beginners and experienced users upgrading from other systems. The manual includes explanations of the terms and structural concepts of IEC programming and an introduction to the new IEC 61131-3 standard. The "Getting Started" chapter provides a precise step-by-step description of how to use GX IEC Developer, including a sample project. This executable application is used to demonstrate the operation of the program with the help of the exercises provided in this manual.

1.2 The Reference Manual…

… contains detailed descriptions of all menus and menu options. Refer to it whenever you need more comprehensive information on the ins and outs of the system.

1.3 If you are not yet familiar with MS Windows …

… please at least read the Windows Fundamentals section in the Windows User's Guide, or work through the Windows Tutorial accessible through the Help menu of the Windows Program Manager. This will teach you what you need to know about using the basic elements of MS Windows, and the operating procedures that are identical in all Windows application programs.

1.4 If you are not yet familiar with the IEC 61131-3 standard…

… please do take the time to read the "Introduction to the IEC 61131-3 Standard" chapter. This section explains the most important new terms and concepts of this industrial standard. A glossary of all the terms is provided in the Appendix of the Reference Manual.

1.5 If you already have IEC 61131-3 experience and want to get to work right away…

… then you can go straight to the "Getting Started" section for immediate results. This chapter provides clear, step-by-step descriptions of all important GX IEC Developer operations, from creating a new project to downloading your finished program to the controller.

1.6 If you get stuck…

… do not despair, help is never far away! If you run up against seemingly insoluble problems, or if you have questions about GX IEC Developer or the connected programmable controller (PLC) configuration, please first refer to the manuals and documentation. Many answers and solutions can also be found directly in the GX IEC Developer context-sensitive online help system, which can always be accessed by pressing the F1 key. Make use of the **Search** command in the **Help** menu as well, as this will often locate the information you need. If you can't find answers to your questions in any of these places, contact your local MITSUBISHI ELECTRIC representative or call our European headquarters in Ratingen directly. The addresses and phone numbers are provided on the back covers of all our manuals.

2 Getting to Know GX IEC Developer

2.1 What's New in GX IEC Developer?

GX IEC Developer is a Windows program:

GX IEC Developer uses the graphical user interface of MS Windows for fast, intuitive operation. This means that instead of laboriously searching through a labyrinth of program structures, you can implement your controller applications quickly and efficiently.

GX IEC Developer increases your productivity:

The modular architecture of GX IEC Developer brings big advantages for complex programming projects. Frequently-needed program blocks and functions only need to be created once. Thanks to the building block system you can then insert them again and again wherever and whenever required. This significantly reduces your programming overheads, enabling you to make major changes to your programs with just a few simple operations.

GX IEC Developer is a multi-language system:

GX IEC Developer supports programming in different languages. Several graphical and text-based editors help you to write tailor-made programs quickly and easily, choosing the language that best suits the problem.

GX IEC Developer is your link to the IEC world:

GX IEC Developer supports the new IEC 61131-3 standard for PLC (programmable controller) programming. This standard lays down the specifications for standardized PLC control programs.

2.2 Introduction to the IEC 61131-3 Standard

IEC 61131-3 is the new international standard for PLC programs, defined by the International Electrotechnical Commission (IEC). It defines the programming languages and structuring elements used for writing PLC programs.

Structured Programming

The structured programming approach replaces the former unwieldy collection of individual instructions with a clear arrangement of the program into individual program modules. These modules are referred to as **Program Organisation Units** (POUs), which form the basis of this new approach to programming.

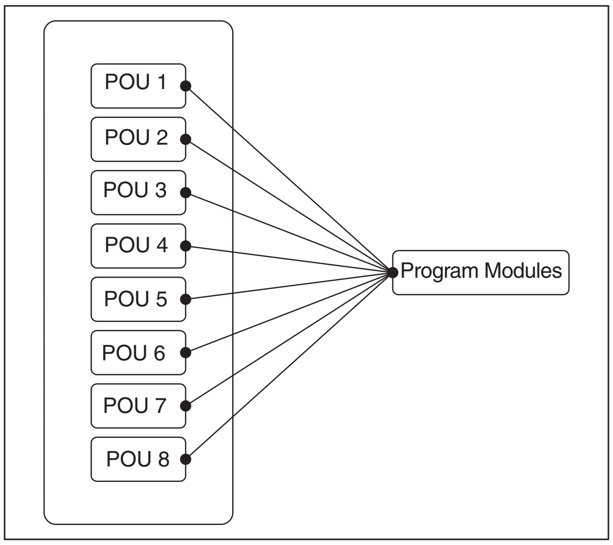


Fig. 2-1:
Program organisation units (POUs) are used to implement all programming tasks.

There are three different classes of POUs, classified on the basis of their functionality:

- **Programs**
- **Functions**
- **Function blocks**

POUs declared as functions and function blocks are effectively programming instructions in their own right, and they can be used as such in every module of your programs.

The final program is assembled from the POUs that you define as programs. This process is handled by the task management, in the Task Pool. Program POUs are put together in groups referred to as **Tasks.**

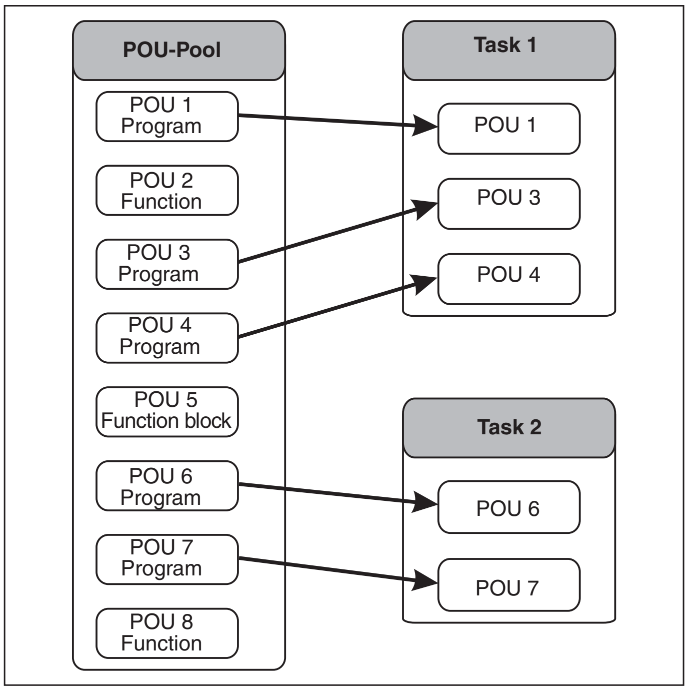


Fig. 2-2:
The program POUs are grouped together in tasks.

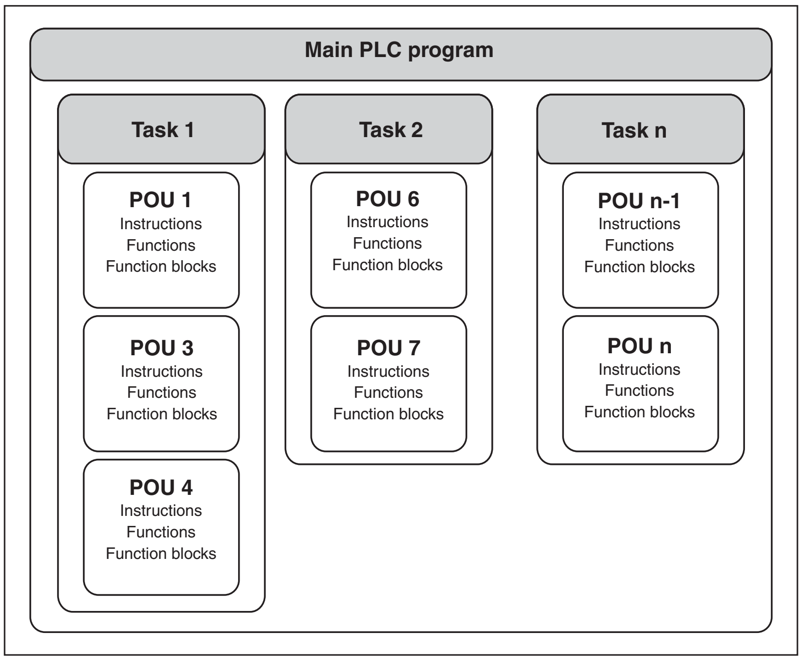

Main PLC program
Task 1
POU 1 Instructions Functions Function blocks
POU 3 Instructions Functions Function blocks
POU 4 Instructions Functions Function blocks
Task 2
POU 6 Instructions Functions Function blocks
POU 7 Instructions Functions Function blocks
Task n
POU n-1 Instructions Functions Function blocks
POU n Instructions Functions Function blocks

Fig. 2-3:
In turn, all the tasks are grouped together to form the actual PLC program.

The **Sequential Function Chart language (SFC)** is also an aid for writing structured PLC programs. It is particularly well suited for programming sequential operations.

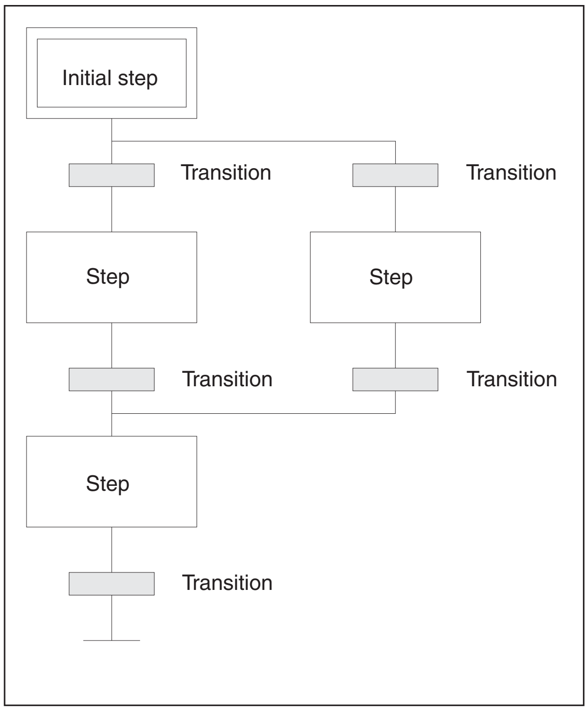


Fig. 2-4:
An SFC sequence consists of a series of steps and transitions (transition or continuos conditions).

Programming Languages

The actual PLC program code contained in the program organisation units (POUs) and the steps and transitions of an SFC sequence can be written in any of the available programming languages. The language used will depend on the nature and size of the programming task.

- **The Text Editors:**
  Instruction List (IL)
  Structured Text (ST)
- **The Graphical Editors:**
  Ladder Diagram (LD)
  Function Block Diagram (FBD)
  Sequential Function Chart (SFC)

Variables

Before you can actually start writing a PLC program you must first decide what variables you are going to need in the program module you are working on. Each POU has a list of **local variables**. These are the variables that can only be used within the POU they are defined and declared for. The **global variables**, which can be used by all the POUs in the program, are declared in a separate list.

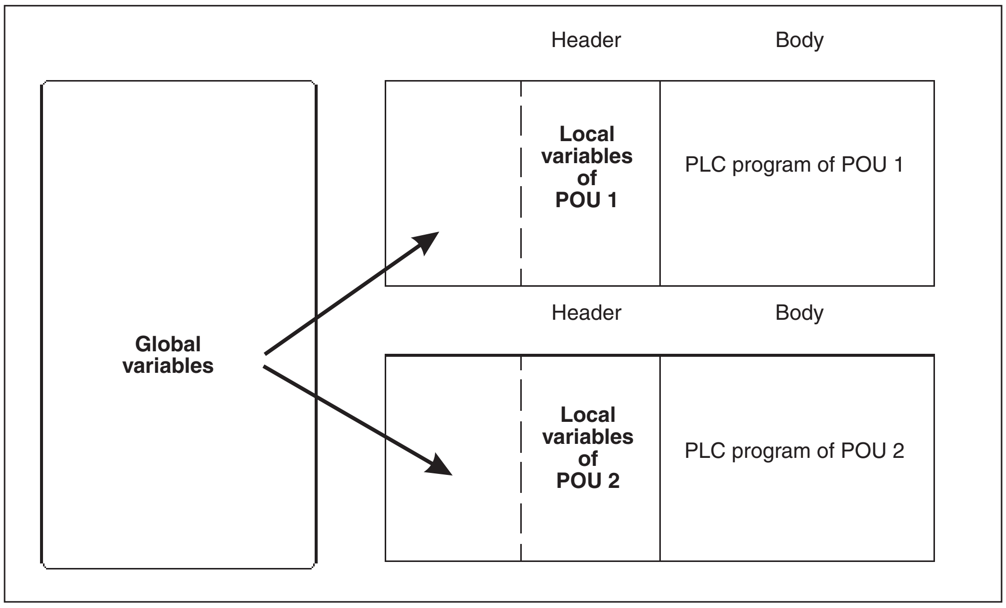


Fig. 2-5: *Global and local variables*

3 Basic Terms Used in IEC 61131-3

3.1 Projects

Every GX IEC Developer project consists of the following elements:

- The Library Pool:
  - the programming instructions contained in the standard library
  - the programming instructions contained in the manufacturer library
- The PLC parameters
- The tasks in the Task Pool
- The structured data types in the DUT Pool
- The global variables
- The program organisation units in the POU Pool

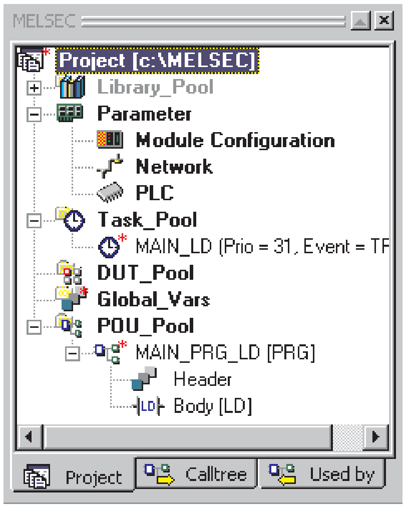


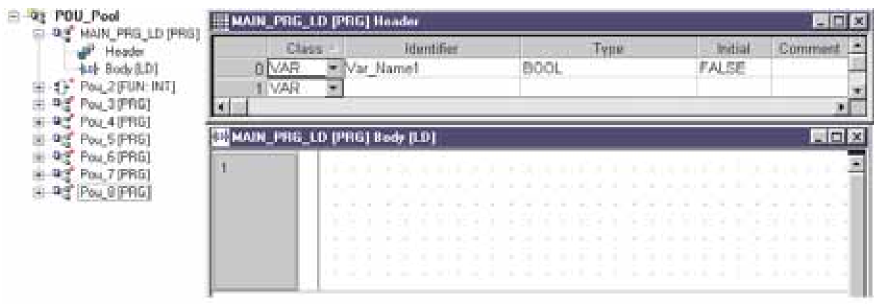
Fig. 3-1
The program element objects are displayed in the Project Navigator window.

3.2 Program Organisation Units (POUs)

Each program organisation unit consists of

- a **header** and
- a **body**.

The variables to be used in the POU are defined (declared) in the header.



	Class	Identifier	Type	Initial	Comment
0	VAR	Var_Name1	BOOL	FALSE	
1	VAR				

Fig. 3-2: *POU header (top) and POU body (bottom)*

The body contains the actual PLC program.

POUs are divided into three classes on the basis of their functionality:

- Programs [PRG],
- Functions [FUN] and
- Function blocks [FB]

3.3 Programs, Function Blocks and Functions

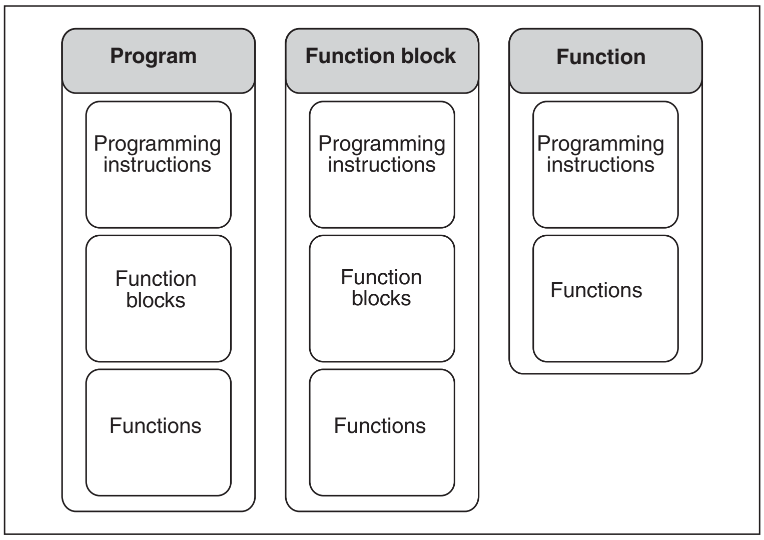


Fig. 3-3: *Programs, function blocks, and functions*

The **program POU** is the standard program organisation unit. Program POUs can contain programming instructions from libraries, functions and function blocks. The execution of the program POUs is controlled by tasks.

POUs declared as **functions** or **function blocks** are independent program elements. They function effectively as programming instructions that can be replaced whenever necessary, and they can also be used in other program modules, just like ordinary instructions.

NOTES

Function blocks can be called by program POUs and other existing function blocks, but not from functions. The function blocks themselves can contain programming instructions from the libraries, functions and other existing function blocks.

Function blocks pass one or more output variables as their result. All the values of the output variables and the internal values within the function block are stored for the following execution of the function block. These values are then used the next time the function block is invoked. This means that invoking the same function block twice with the same input parameters does not necessarily result in the same output values!

Functions can be called by program POUs, function blocks and other existing functions. Functions can contain programming instructions from the libraries and other existing functions.

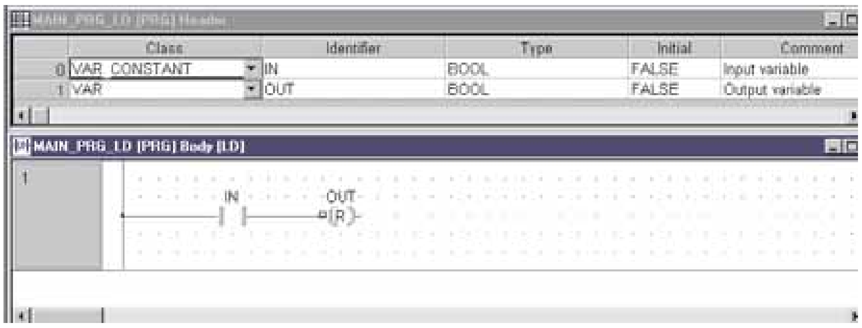
Functions always pass an output value, and they do not store any internal status information. Thus, you should always get the same output value every time you invoke a function with the same input parameters.

Item	Function Block	Function
Internal variable storage	Storage	No storage
Instancing	Required	Not required
Outputs	No output One output Multiple Outputs	One output
Repeated execution with same input values	Does not always deliver the same output value	Always delivers the same output value

Tab. 3-1: *Differences: Function Blocks and functions*

3.4 Parameters and Instancing

Functions and function blocks have formal parameters and actual parameters. **Formal parameters** are the variables used when a function or function block is created. The formal parameters of the programming instructions in the standard and manufacturer libraries are not visible to the user. **Actual parameters** are the variables that are passed to the function or function block instance (copy) when it is used in another POU. Actual parameters can be defined variables, hardware addresses or constants.



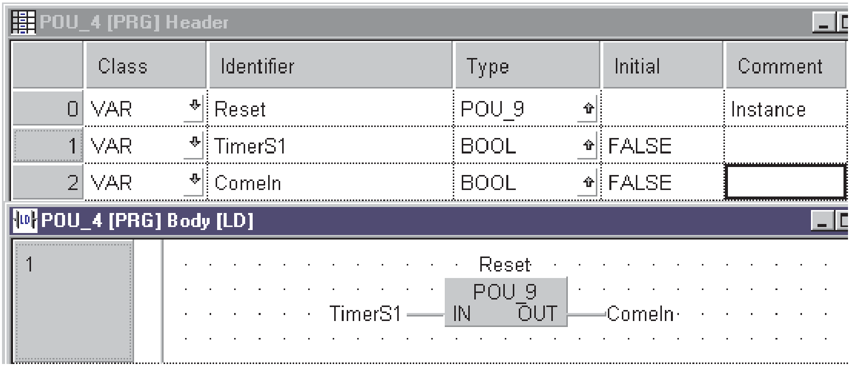
MAIN_PRG_LD [PRG] Header

	Class	Identifier	Type	Initial	Comment
0	VAR_CONSTANT	IN	BOOL	FALSE	Input variable
1	VAR	OUT	BOOL	FALSE	Output variable

MAIN_PRG_LD [PRG] Body [LD]

Fig. 3-4: *The program organisation unit POU_9 is a function block [FB]. The variables "IN" and "OUT" used in this program module are declared in the header. "IN" and "OUT" are the formal parameters.*

Function blocks can only be called as **instances**. The process of "instancing", or making a copy of the function block, is performed in the header of the POU in which the instance is to be used. In this header the function block is declared as a variable and the resulting instance is assigned a name. Note that you can declare multiple instances with different names from one and the same function block within the same POU. The instances are then called in the body of the POU and the actual parameters are passed to the formal parameters. Each instance can be used more than once. For details on activating instances of function blocks in the individual editors please refer to the chapter "Programming Languages".



POU_4 [PRG] Header

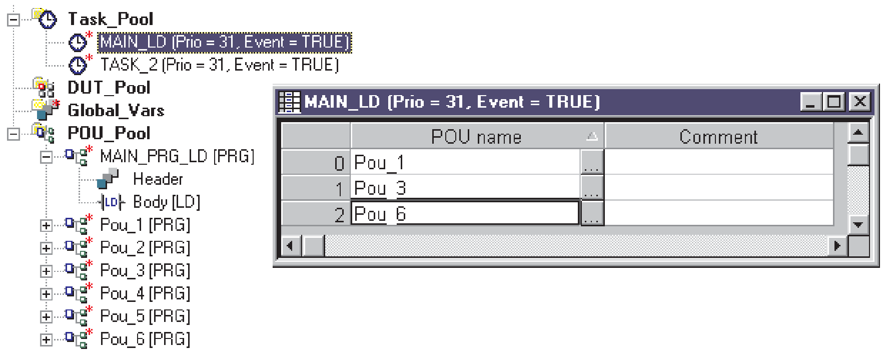
	Class	Identifier	Type	Initial	Comment
0	VAR	Reset	POU_9		Instance
1	VAR	TimerS1	BOOL	FALSE	
2	VAR	ComeIn	BOOL	FALSE	

POU_4 [PRG] Body [LD]

Fig. 3-5: *"Reset" ID an instance of function block POU_9. "IN" and "OUT" are the formal parameters; "TimerS1" and "ComeIn" are the actual parameters of the instance.*

3.5 Tasks

A **task** contains one or more program organisation units declared as programs [PRG]. The task controls the processing of these programs by the controller.



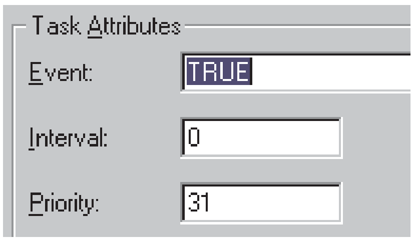
Task_Pool
MAIN_LD (Prio = 31, Event = TRUE)
TASK_2 (Prio = 31, Event = TRUE)
DUT_Pool
Global_Vars
POU_Pool
MAIN_PRG_LD [PRG]
Header
Body [LD]
Pou_1 [PRG]
Pou_2 [PRG]
Pou_3 [PRG]
Pou_4 [PRG]
Pou_5 [PRG]
Pou_6 [PRG]

MAIN_LD (Prio = 31, Event = TRUE)

	POU name	Comment
0	Pou_1	
1	Pou_3	
2	Pou_6	

Fig. 3-6: *This project consists of two tasks, MAIN_LD and TASK_2.*

If a project contains more than one task you can define execution conditions for the individual tasks:



Task Attributes
Event: TRUE
Interval: 0
Priority: 31

Fig. 3-7:
Event*: Execute, if the variable ID TRUE.*
Interval*: Execute at defined time intervals*
Priority*: Execute in a defined priority order*

3.6 Variables

Variables are similar to operands. They contain the values of inputs, outputs or the internal memory locations of the PLC system.

A distinction is made between two different variable types, on the basis of their "scope" within the program as a whole:

- Local variables
- Global variables

Local variables: When program elements are declared as Local Variables, GX IEC Developer automatically uses some of its System Variables as appropriate storage devices within a specific POU. These variables are exclusive to each POU and are not available to any other routine within a project.

Global variables: Global Variables can be regarded as “shared” variables and are the interface to physical PLC devices. They are made available to all POU’s and reference an actual physical PLC I/O or named internal devices within the PLC. External HMI and SCADA devices may interface with the user program using Global Variables.

Declaring Variables

Before you can begin with the actual programming, you should declare the variables you are going to use in the project as a whole (global variables) and in the individual POUs (local variables).

Each variable declaration has the following elements:

- Class
- Identifier,
- Absolute address (global variables only),
- Data type,
- Initial value (automatically),
- Comment (optional),
- Remark (global variables only).

IEC61131-3 Verses MELSEC Variables

GX IEC Developer supports program creation, using either symbolic declarations (tag names), or absolute Mitsubishi addresses (X0, M0 etc), assigned to the program elements.

The use of symbolic declarations complies with IEC 61131.3.

If symbolic declarations are used, then the tag names must be cross referenced to real PLC addresses.

Local Variable List

For a particular POU to access a Global Variable, it must be declared in its Local Variable List (LVL), in the POU Header.

The LVL can be made up of both Global Variables and Local Variables.

A Local Variable can be thought of as an intermediate result, i.e. if the program performs a five stage calculation, using three values and ending with one result, traditionally, the programmer would construct software, which produced several intermediate results, held in data registers before ending with the final register result.

It is likely that these intermediate results, serve no purpose other than for storage and only the final result is used elsewhere.

With GX IEC Developer, the intermediate results can be declared, as Local Variables and in this case, only the original three numbers and the result, declared as Global Variables.

The Global Variable List

The Global Variable List (GVL) provides the interface for all names, which relate to real PLC addresses, i.e. I/O data registers etc.

The GVL is available and can be read by all POU’s created in the project.

Class

The class keyword assigns the variable a specific property that defines how it is to be used in the project.

Class	Use in POUs:			Meaning
	PRG	FUN	FB	
VAR	X	X	X	Variable that is only used within the POU
VAR_CONSTANT	X	X	X	Local variable with unchangeable initial value used within the POU
VAR_INPUT	—	X	X	Variable passed from outside that cannot be altered within the POU
VAR_OUTPUT	—	—	X	Variable passed (output) by the POU
VAR_IN_OUT	—	—	X	Local variable passed from outside and passes (output) by the POU, can be altered within the POU
VAR_GLOBAL	X	—	X	Global variable declared in the Global Variable List
VAR_GLOBAL_CONSTANT	X	—	X	Global variable with unchangeable initial value declared in the Global Variable List

Tab. 3-2: *Available classes*

Identifiers and Absolute Addresses

Each variable is given a symbolic address, i.e. a name. This is referred to as the **identifier**; it consists of a string of alphanumeric characters and underline characters. The identifier must always begin with a letter or an underline character. Spaces and mathematical operator characters (e.g. +, -, *) are not permitted.

Examples of identifiers:	FAULT
	ZEROSIG
	LIM_SW_5

When global variables are declared they should also be assigned **absolute addresses** that reference the memory location of the variable in the CPU or a physical input or output. If you do not assign the absolute addresses manually, they are assigned automatically.

When local variables are declared in the header of the POU they are automatically assigned a suitable memory location in the CPU.

You can use either the IEC syntax (IEC-Addr.) or the MITSUBISHI syntax (MIT-Addr.) to assign the absolute addresses. Two address columns are available.

As soon as you have entered an address in one of these columns, the other address also appears. You can enter either of the two address formats in both columns. If, for instance, you enter a MITSUBISHI address in the IEC column, GX IEC Developer identifies it immediately, places it in the correct column and produces the matching IEC address in the other column.

IEC Address	MITSUBISHI Address	Meaning
%QX0	Y0	Output Y0
%IX31	X1F	Input X1F
%MW0.450	D450	Data register D450

Tab. 3-3: *Examples of absolute addresses*

Use upper case letters only and no spaces or mathematical operator characters (e.g. +, -, *) in addresses.

3.7 Data Types

GX IEC Developer supports the following data types.

3.7.1 Simple Types

The data type of a variable defines the number of bits it contains, how they are processed and the variable's value range. The following data types are available.

Data type		Value range	Size
BOOL	Boolean	0 (FALSE), 1 (TRUE)	1 bits
INT	Integer	-32.768 to 32.767	16 bits
DINT	Double integer	-2.147.483.648 to 2.147.483.647	32 bits
WORD	Bit string 16	0 to 65.535	16 bits
DWORD	Bit string 32	0 to 4.294.967.295	32 bits
REAL	Floating-point value	3.4 +/- 38 (7 digits)	32 bits
TIME	Time value	T#-24d-0h31m23s648.00ms to T#24d20h31m23s647.00ms	32 bits
STRING	Character string	max. 50 characters	

Tab. 3-4: *Available simple data types*

NOTE

Please note that not every data type can be processed by every PLC type!

Initial Value

The initial values are set automatically by the system and cannot be changed by the user.

Comment

You can add a comment up to 64 k characters long for each variable.

Remark

You can add additional user information.

3.7.2 Complex Data Types

Arrays

An array is a field or matrix of variables of a particular type.

For example, an **ARRAY [0..2] OF INT** is a one dimensional array of three integer elements (0,1,2). If the start address of the array is D0, then the array consists of D0, D1 and D2.

Identifier	**Address**	**Type**	**Length**
Motor_Volts	D0	ARRAY	[0...2] OF INT

In software, program elements can use e.g. Motor_Volts[1] and Motor_Volts[2] as declarations, which in this example mean that D1 and D2 are addressed.

Arrays can have up to three dimensions, for example: ARRAY [0...2, 0...4] has three elements in the first dimension and five in the second.

Arrays can provide a convenient way of ‘indexing’ tag names, i.e. one declaration in the Local or Global Variable Table can access many elements.

Data Unit Types (DUT)

User defined Data Unit Types (DUT), can be created. This can be useful for programs which contain common parts, for example; the control of six identical silos. Therefore a data unit type, called ‘Silo’ can be created, composing patterns of different elements, i.e. INT, BOOL etc.

When completing a global variable list, identifiers of type Silo can be used. This means that the predefined group called ‘Silo’ can be used with the elements defined as required for each silo, thus reducing design time and allowing re-use of the DUT.

3.8 Programming Languages

GX IEC Developer supports five programming languages: Two text languages, two graphical languages, and one structuring language.

- Text language:
  Instruction List language IL (IEC IL and MELSEC IL), **Structured Text ST**
- Graphical languages:
  Ladder Diagram language LD, Function Block Diagram language FBD
- Structuring language:
  Sequential Function Chart language SFC

WARNING:
You cannot change the programming language once you have selected it. Even though it is physically possible to switch to another language, you will lose the entire contents of the unit's body if you attempt to do so!

3.8.1 Networks

In all the editors - with the exception of the SFC editor and ST editor - your PLC program is divided into a number of program sections referred to as **networks**. Each network is assigned a name (the network label) which can be used as a destination for jump (goto) instructions.

NOTE Each network can contain no more than one contiguous circuit unit.

3.8.2 The Text Editors

The following text editors are supported:

- MELSEC Instruction List
- IEC Instruction List
- Structured Text

The structure of all Instruction List types is identical. Each Instruction List consists of a sequence of controller instructions. Each controller instruction begins on a new line and consists of a programming instruction and its parameters and variables. However, there are significant differences in the way the controller instructions are executed.

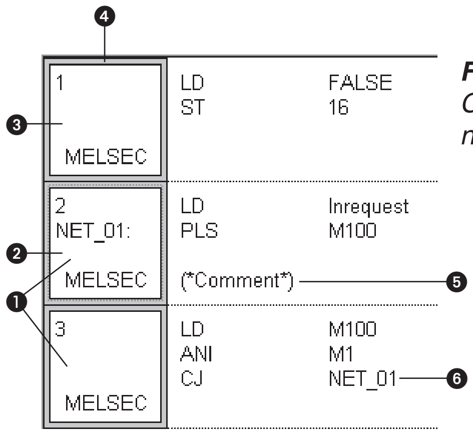
The MELSEC Instruction List Language (MELSEC IL)

MELSEC Instruction List programs are written following the rules of DIN 19239 and the programming rules familiar from the MELSEC MEDOC software. You can only use genuine MELSEC programming instructions (see Appendix of the Reference Manual). MELSEC Instruction List programs can only contain MELSEC networks. Access to IEC programming instructions is not possible.

The IEC Instruction List Language (IEC IL)

The IEC Instruction List language allows you to combine IEC networks and MELSEC networks in a single program.

The IEC networks are programmed according to the IEC 61131-3 rules, and you can use both IEC programming instructions and the adapted MELSEC instructions (see Appendix in Reference Manual).



```
LD      FALSE
ST      16

LD      Inrequest
PLS     M100

LD      M100
ANI     M1
CJ      NET_01
```

Fig. 3-8:
Combined Instruction List networks

Number	Description
❶	MELSEC network
❷	Network label To enter the network label "NET_01:" first double-click on the network bar.
❸	MELSEC network
❹	Network bar
❺	Comment text must be enclosed between (* and *) character pairs.
❻	The "CJ" instruction performs a jump to the specified destination network.

Tab. 3-5: *Key to figure above*

Structured Text (ST)

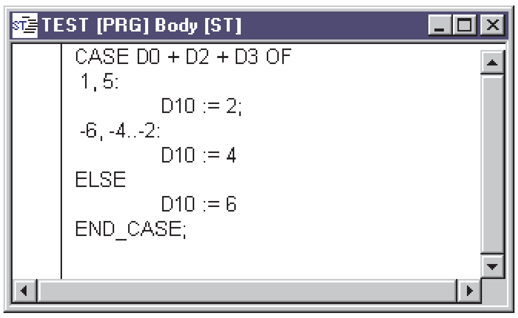
ST is a text-oriented editor (programming language), similar to PASCAL and supports mathematical functions and a simple creation of loops.

ST body does not contain a network list because it always consists of only one network.

ST is an editor from the IEC 61131 programming standard. The Structured Text editor is compatible to the IEC 61131-3.

All IEC 61131 (IEC 61131-3: PART3-1992) standard functions are supported.

All MELSEC instructions are supported.



```
CASE D0 + D2 + D3 OF
 1, 5:
            D10 := 2;
 -6, -4..-2:
            D10 := 4
ELSE
            D10 := 6
END_CASE;
```

Fig. 3-9.
Structured Text body

The Accumulator

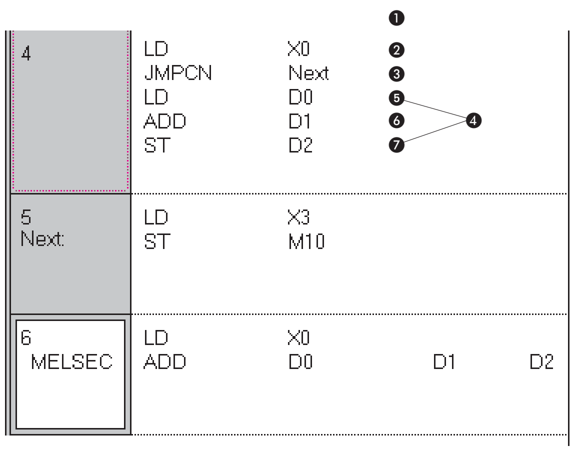
In the IEC editor the result of each operation is stored in an **accumulator** directly after execution. This accumulator always contains the operation result of the last instruction programmed.

NOTE

You do not have to program input conditions (execution conditions) for the operations in this editor. Execution is always based on the contents of the bit accumulator.

Example ▽

The following illustrates the difference between programming in the MELSEC and IEC editors. We want to program the addition D0(5)+D1(10) = D2(15) to be executed when input X0 is active.



```
4        LD       X0
         JMPCN    Next
         LD       D0
         ADD      D1
         ST       D2

5        LD       X3
Next:    ST       M10

6        LD       X0
 MELSEC  ADD      D0          D1          D2
```

Fig. 3-10:
Code for the addition ...

... in the IEC editor

... in the MELSEC editor

Number	Description
❶	The bit accumulator is undefined at the beginning of the network.
❷	The accumulator now contains a value of 0 or 1, depending on the state of input X0.
❸	The JMPCN instruction (JumpConditionalNot) will be executed if the value in the accumulator is 0. The instructions in section ❹ are skipped and the program branches to the "Next:" network. If the value in the accumulator is 1, JMPCN is ignored and the instructions in ❹ are executed. The accumulator then still contains the status of X0, i.e. 1 in this case.
❺	Writes the contents of data D0, i.e. 5, to the accumulator.
❻	Adds the value in D0 to the value in D1. After the addition the result (15) is stored in the accumulator.
❼	Stores the result of the addition to D2. The accumulator still contains the value 15.

Tab. 3-6: *Key to figure above*

△

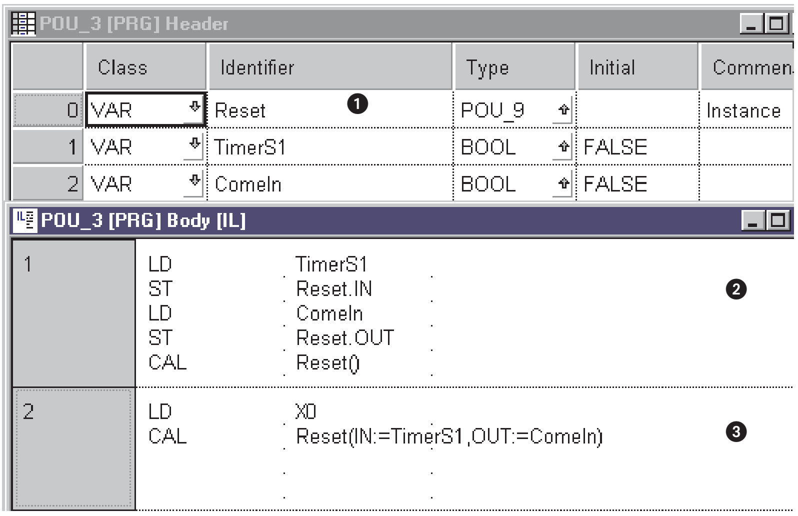
Calling Function Blocks

Function blocks can only be called as instances, using the following operators:

CAL	(Call)
CALC	(CallConditional))
CALCN	(CallConditionalNot)

CAL is always executed. CALC and CALCN first poll the status of the bit accumulator; they are executed only if its value is 1 (CALC) or 0 (CALCN).

The instance name is assigned in the header of the POU. The actual parameters must then be passed to the formal parameters in the code programmed in the body.



	Class	Identifier	Type	Initial	Commen
0	VAR	Reset	POU_9		Instance
1	VAR	TimerS1	BOOL	FALSE	
2	VAR	ComeIn	BOOL	FALSE	

```
1   LD     TimerS1
    ST     Reset.IN
    LD     ComeIn
    ST     Reset.OUT
    CAL    Reset()

2   LD     X0
    CAL    Reset(IN:=TimerS1,OUT:=ComeIn)
```

Fig. 3-11: *The formal parameters of function block POU_9 are "IN" and "OUT". Actual parameters "TimerS1" and "ComeIn" are passed to these formal parameters.*

Number	Description
❶	Declares the instance "Reset" of function block POU_9.
❷	There are two ways to pass actual parameters to formal parameters.
❸	There are two ways to pass actual parameters to formal parameters.

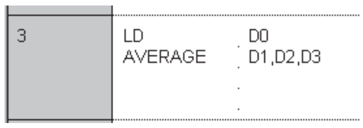
Tab. 3-7: *Key to figure above*

Calling Functions

When you call a function, you must pass the necessary actual parameters to its formal parameters.

A total of n - 1 actual parameters are assigned to every function, where n = total number of function parameters. This is because the first parameter must always be written to the bit accumulator with the LD instruction.

Example ▽



```
3    LD         D0
     AVERAGE    D1,D2,D3
```

Fig. 3-12:
Use of "Average", a function written by the user in IEC IL language. The function has 4 input parameters.

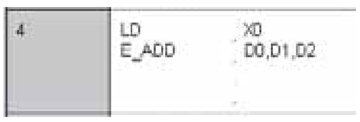
The "Average" function is programmed to perform the following operation:
(D0 + D1 + D2 + D3) : 4.

When the function has been executed the bit accumulator contains the resulting average value of the four input parameters.

△

LD must also be used to pass the first parameter for the EN/ENO functions
(e.g. E_ADD, E_MUL, E_XOR). Their first parameter is always the Boolean EN input
(EN = ENable).

Example ▽



```
4    LD         X0
     E_ADD      D0,D1,D2
```

Fig. 3-13:
This writes actual parameter X0 to the EN input. The 3 parameters for execution of the addition are programmed with the function itself.

The "E_ADD" function performs the following operation: D0 + D1 = D2.

Following execution of this function the bit accumulator will contain the status of the ENO output (ENO = ENable Out), which in term has the same status as the EN input.

△

3.8.3 The Graphical Editors

The Ladder Diagram Language (LD)

You can use all available programming instructions in the ladder diagram language (see Appendix in Reference Manual).

Ladder diagrams consist of contacts (break and make contacts), coils, function blocks and functions. These elements are linked with horizontal and vertical lines, referred to as interconnects. These interconnects always begin at the power bar on the left, which is sometimes also referred to as the rail.

NOTE Each network can contain no more than one contiguous circuit unit.

The functions and function blocks are displayed as shaded blocks in the editing window. In addition to their input and output parameters, some also have a Boolean input (EN = ENable) and a Boolean output (ENO = ENable Out).

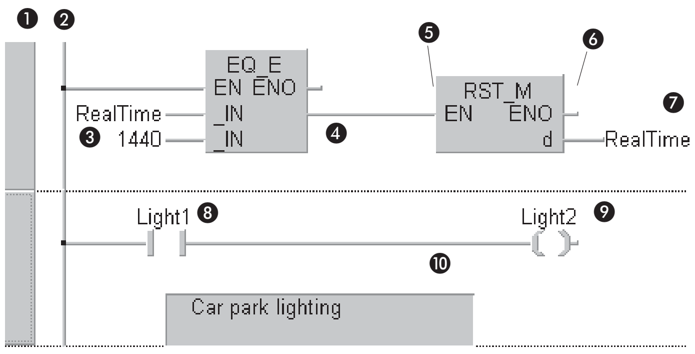


Fig. 3-14: *Graphical programming in the ladder diagram editor*

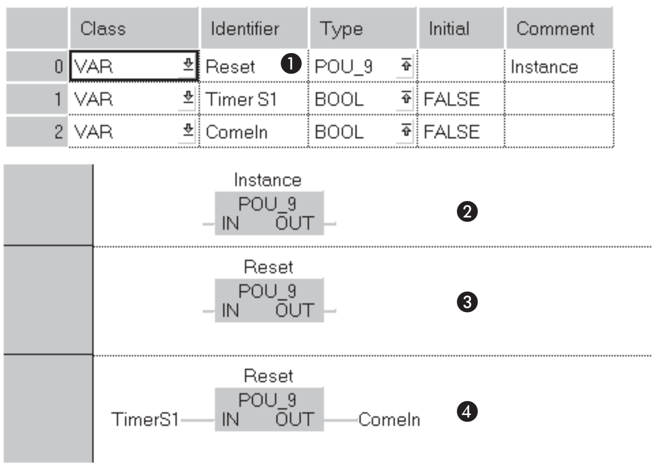
Number	Description
❶	Network bar
❷	Power bar
❸	Input variable
❹	Output variable
❺	EN input
❻	ENO output
❼	Output variable
❽	Contact
❾	Coil
❿	Comment

Tab. 3-8: *Key to figure above*

Calling Function Blocks

Function blocks can only be called as instances. The instance name must be declared in the header of the POU.

In the ladder diagram editor, the name of the function block is displayed inside the shaded block. The instance name declared in the header must be entered directly above the block. Then the actual parameters must be passed from outside to the formal parameters shown inside the block.



	Class	Identifier	Type	Initial	Comment
0	VAR	Reset ❶	POU_9		Instance
1	VAR	Timer S1	BOOL	FALSE	
2	VAR	ComeIn	BOOL	FALSE	

Fig. 3-15:
Calling function blocks

Number	Description
❶	Declaration of "Reset", an instance of function block POU_9.
❷	Activation of function block POU_9. The word "Instance" above the shaded block indicates that you must enter the function block's instance name here.
❸	The instance name "Reset" has been entered.
❹	Next, the actual parameters "TimerS1" and "ComeIn" are passed to the formal parameters "IN" and "OUT".

Tab. 3-9: *Key to figure above*

The Function Block Diagram Language (FBD)

In the function block diagram language you can also use all programming instructions (see Appendix in Reference Manual). They are displayed as shaded blocks which are connected with the horizontal and vertical interconnect lines. Power bars are not used in this language.

In addition to the normal input and output parameters some blocks also have a Boolean input (EN = ENable) and a Boolean output (ENO = ENable Out).

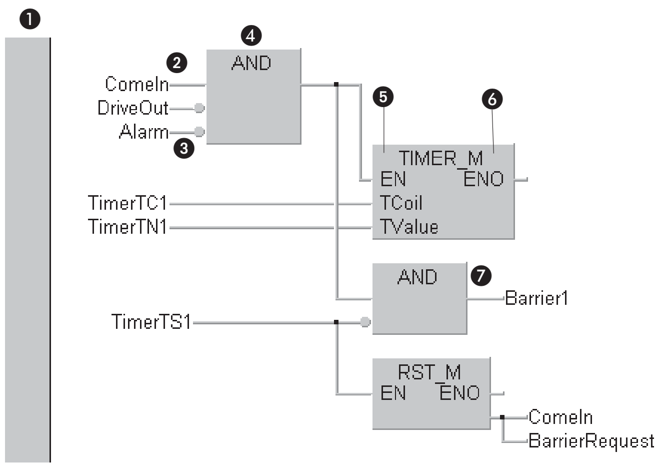


Fig. 3-16:
Graphical programming in the function block language editor

Number	Description
❶	Network bar
❷	Input variable (normal)
❸	Input variable (negation)
❹	Function
❺	EN input
❻	ENO output
❼	Output variable

Tab. 3-10: *Key to figure above*

Calling Function Blocks and Functions

In the function block language, function blocks and functions are called in exactly the same way as in the ladder diagram language.

The Sequential Function Chart Language (SFC)

SFC is a structuring language which allows clear representation of complex processes.

NOTE The program is the only available program organisation unit (POU) in this language.

The basic elements of the SFC language are steps and transitions.
From 0 to n **actions** can be assigned to each **step**. An action can be a Boolean variable (output or relay) or a PLC program. These programs can be written using any of the editors - including the Sequential Function Chart language itself. All actions are listed in the Action_Pool in the Project Navigator window.
Each **transition** is assigned a **transition condition**. Transition conditions can be written using any of the editors - except Sequential Function Chart itself. All transitions are also listed in the Project Navigator window. Transitions pass control to the next step in the program sequence when their condition evaluates as logical true.

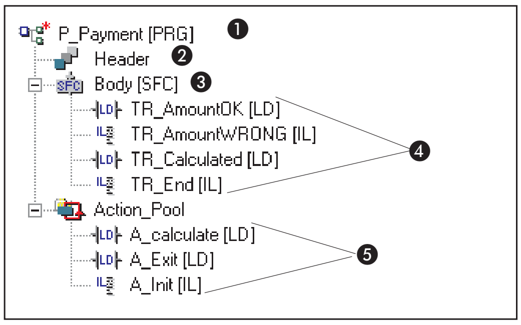


Fig. 3-17:
Sample SFC project

Number	Description
❶	The "P_Payment" program organisation unit, which is declared as a program [PRG].
❷	The header contains the POU's variables.
❸	The PLC program was written with the SFC editor.
❹	The individual transitions can be written with different editors.
❺	The Action Pool contains the individual actions, which can also be written in different editors.

Tab. 3-11: *Key to figure above*

Assignment of actions to steps and of transition conditions to transitions is performed with the following toolbar icons:

Fig. 3-18:
Activate action/transition condition

Fig. 3-19:
Deactivate action/transition condition

Sequencing Rules

A sequence always begins with an Initial Step, identified by a double outline. The initial step does not have to be at the physical beginning of the sequence, it can also be placed in other locations.

Steps are displayed as shaded blocks with names. Transitions are shown as small boxes placed directly on the vertical connecting lines between the steps.

Only one step can be active at any one time; this also applies in sequences with selective branching. A step is activated when the directly preceding step is deactivated and the transition condition (i.e. the continue condition) is satisfied. If the continue conditions of two or more transitions are fulfilled at the same time in a sequence with selective branching, execution priority is defined by the order of the sequences from left to right. This means that only the sequence that is furthest to the left will be executed. Even if their continue conditions are satisfied, the sequences further to the right will not be executed.

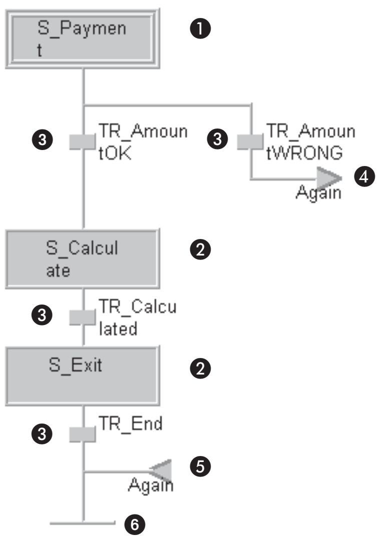


Fig. 3-20:
Graphical programming in the Sequential Function Chart editor

Number	Description
❶	Initial step
❷	Step
❸	Transition
❹	Jump exit point
❺	Jump entry point
❻	Final step

Tab. 3-12: *Key to figure above*

Sequences can also contain left and right "divergences" and "convergences" (i.e. alternative branches for different transition conditions). These branches are identified by a double horizontal interconnect lines.

Jumps are also allowed within sequences. These are effected with exit points (jump instructions) and entry points (labels).

Every step can be declared as a macro step, consisting in turn of a sequence. Macro steps are identified by two additional horizontal lines within the block. The only limitation on the nesting depth is the memory capacity of the controller.

NOTE

You will find more detailed information on the sequencing rules of the SFC language in the Reference Manual.
You can find a detailed example in Chapter 6 of this manual (Step 6).

4 Installation

4.1 Hardware Requirements

4.1.1 Recommended Hardware Configuration

- Pentium II 350 processor or above
- 64 MB RAM (Microsoft Windows® 2000)
  128 MB RAM (Microsoft Windows® XP)
  1024 MB RAM (Microsoft Windows® Vista)
- Serial interface (RS-232)
- USB port
- Hard disk with at least 200 MB free space
- CD/DVD-ROM drive
- 17" (43 cm) VGA monitor (1024 x 768 pixels)

4.1.2 Software Requirements

GX IEC Developer is a 32-bit product. The following operating systems are supported:

- Microsoft Windows® 2000 Professional (with ServicePack 2 or higher)
- Microsoft Windows® XP Professional (up to ServicePack 3)
- Microsoft Windows® XP Home Edition (up to ServicePack 3)
- Microsoft Windows® Vista (32-Bit) (up to ServicePack 1)

Versions of Microsoft Windows which are based on double-byte character sets (e. g. Japanese) are not supported.

4.2 Copyright

WARNING:
This software is protected by copyright. By opening the distribution disks package you automatically accept the terms and conditions of the License Agreement. You are only permitted to make one single copy of the original distribution CD-ROM for your own backup and archiving purposes.

4.3 Installing GX IEC Developer

During the installation procedure the setup program will create a directory on your hard disk to copy all the GX IEC Developer files into.

4.3.1 Installing GX IEC Developer on your hard disk

① Make sure that the correct Microsoft Windows version is properly installed on your computer. For information on using Microsoft Windows please refer to the Windows User's Guide.

② Start Microsoft Windows.

③ Insert the installation CD-ROM in the CD-ROM drive.
The GX IEC Developer installation program starts automatically
(if not, execute the file SETUP.EXE on the installation CD-ROM).

④ Follow the instructions that appear on the screen.

⑤ Enter the user name, company name, and serial number of the software.

⑥ Follow the instructions that appear on the screen.

⑦ When the installation procedure is finished the program will create a new program group in the Start menu containing the GX IEC Developer program icon.

For further details on the necessary Microsoft Windows procedures please refer to your Microsoft Windows documentation.

4.3.2 Starting GX IEC Developer

① In the Start menu click on the GX IEC Developer program icon. The icon is located in: Start > Programs > MELSOFT Application > GX IEC Developer. This starts GX IEC Developer and displays the start-up screen.

② Confirm with the Enter key.

4.3.3 Quitting GX IEC Developer

You can quit GX IEC Developer directly at any point in the program by pressing the key combination Alt F4.

Or:

Click on the **Quit** command in the **Project** menu.

5 The User Interface

5.1 The Elements of the User Interface

The Project Navigator window and the complete menu bar are both only displayed after opening an existing project or creating a new one (see Step1 in chapter 6 "Getting Started"). The illustration below shows a variety of different windows: The Project Navigator, PLC Parameter and the Header and Body windows of a POU. You can resize and arrange the windows on the screen to suit your individual preferences.

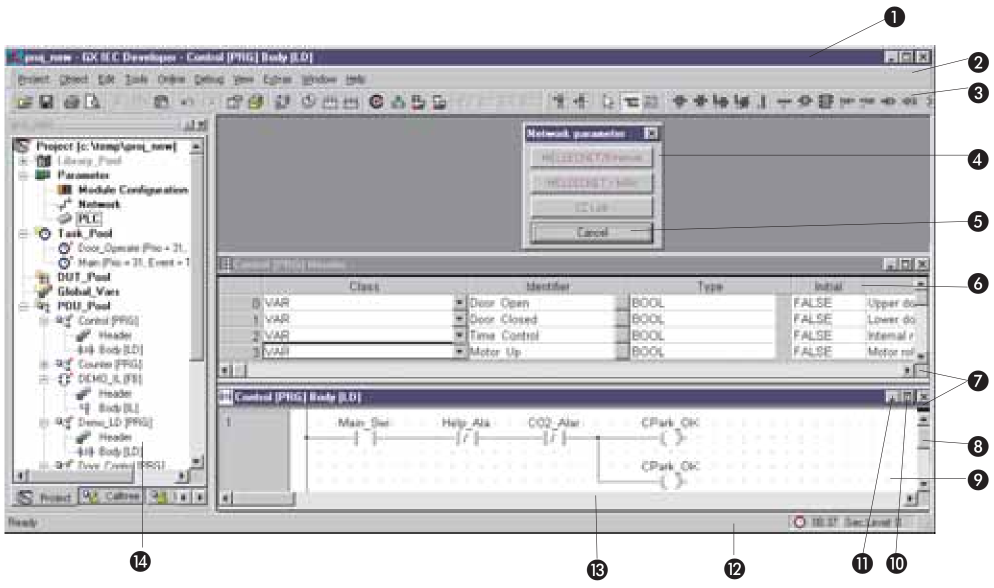


Fig. 5-1: *User interface*

Item	Description
❶	Application title bar
❷	Menu bar
❸	Toolbar
❹	Dialogue box
❺	Button
❻	Declaration table (header)
❼	Object window
❽	Vertical scrollbar
❾	Editor (body)
❿	"Maximise" button
⓫	"Minimise" button
⓬	Status bar
⓭	Horizontal scrollbar
⓮	Project Navigator window

Tab. 5-1: *Key to figure above*

5.1.1 The Menu Bar

The GX IEC Developer menu bar uses the standard Windows procedures. When you select one of the menu titles in the menu bar, a drop-down list of available commands is displayed. Commands with an arrow symbol open a submenu of additional commands. Selecting a command opens a dialogue or data entry box. The menu structure and the available options are context-sensitive, changing depending on what you are currently doing in the program. Options displayed in light grey are not currently available for selection.

NOTE A list of all menu commands with explanations is provided in the Appendix of the Reference Manual.

5.1.2 The Toolbar

The toolbar enables you to select the most important menu commands directly by clicking on the corresponding icons. The toolbar is context-sensitive, i.e. different tool icons are displayed depending on what you are currently doing in GX IEC Developer.

NOTE A complete list of all the available tools and icons is provided at the end of the Reference Manual.

5.1.3 Windows

GX IEC Developer allows you to edit multiple objects at the same time (e.g. body, header, task). A window is opened on the screen for each object. You can change the size and position of the windows on the screen as you wish. Objects often contain more information than can be displayed in the window; when this happens, horizontal and vertical scroll bars are included that can be used to "scroll" the contents of the windows up and down and from side to side.

5.1.4 The Status Bar

The status bar at the bottom of the screen is used to display information on the current status of your project. You can disable the status bar if you wish, and you can also configure the information to be displayed to suit your needs.

5.1.5 The Project Navigator

The Project Navigator is the "control centre" used for selecting and handling the objects used in GX IEC Developer. This is the starting place for all operations performed on GX IEC Developer objects. The Project Navigator window is not displayed until you open a project. Closing the Project Navigator window automatically closes the project currently on screen.

Using the Project Navigator

In the Project Navigator tree you can expand a branch by clicking on its **[+]** symbol and collapse a branch by clicking on its **[–]** symbol. Expanded and collapsed branches are identified by different symbols **[–]** or **[+]** in the tree. You can also expand or collapse branches by double-clicking on the appropriate branch icons. Double-clicking on the lowest level opens the window of the object on that level.

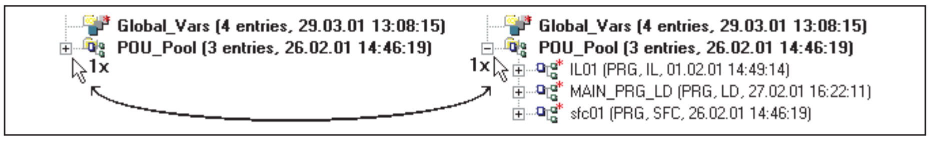

Global_Vars (4 entries, 29.03.01 13:08:15)
POU_Pool (3 entries, 26.02.01 14:46:19)
1x
Global_Vars (4 entries, 29.03.01 13:08:15)
POU_Pool (3 entries, 26.02.01 14:46:19)
1x
IL01 (PRG, IL, 01.02.01 14:49:14)
MAIN_PRG_LD (PRG, LD, 27.02.01 16:22:11)
sfc01 (PRG, SFC, 26.02.01 14:46:19)

Fig. 5-2: „Manoeuvring" with the Project Navigator

You can only perform the Cut, Copy, Paste and Delete operations on POU and Task objects. You can copy and delete multiple objects at the same time. To select individual multiple objects, hold down the CTRL key and click on the objects one after the other with the left mouse button. To select a consecutive group of multiple objects, first select the first object with a single click, then hold down the SHIFT key and click on the last object in the list you want to select.

NOTE The **Extended Information** command in the **View** menu enables or disables the display of additional information with the items in the Project Navigator window.

Project Navigator views

Three different views can be selected for the Project Navigator via tabs below the Project Navigator window:

Project

This view gives a total overview of the project. It contains all elements of the project.

Calltree

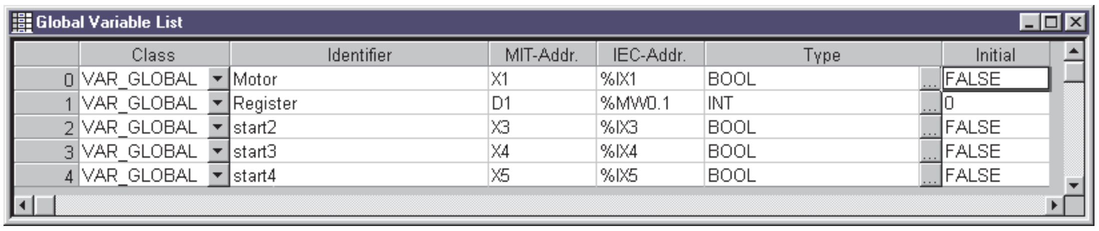
For this view, the corresponding root items are tasks or also POUs, if they are not related to a specific task. As subitems all used POUs are shown. Additionally, it can be defined per object, if used global variables should also be shown.

Used by

This view has exactly two root items. The first root item is the POU pool with its POUs as nodes. Subitems of the POUs and global variables are always POUs calling respectively using the corresponding POU or global variable.

5.2 Declaration Tables

The local variables of program organisation units (POU Header) and global variables are defined in declaration tables.



Global Variable List

	Class	Identifier	MIT-Addr.	IEC-Addr.	Type	Initial
0	VAR_GLOBAL	Motor	X1	%IX1	BOOL	FALSE
1	VAR_GLOBAL	Register	D1	%MW0.1	INT	0
2	VAR_GLOBAL	start2	X3	%IX3	BOOL	FALSE
3	VAR_GLOBAL	start3	X4	%IX4	BOOL	FALSE
4	VAR_GLOBAL	start4	X5	%IX5	BOOL	FALSE

Fig. 5-3: *Global Variable List*

Working with tables

You can access every cell in a table directly by clicking with the mouse. When the insertion mark (cursor) appears you are in editing mode and can make entries. You can move around in tables with the following keys and key combinations:

Key	Movement
↑	Line up
→	Cell right
↓	Line down
←	Cell left
Tab	Step through all cells from left to right
Shift Tab	Step through all cells from right to left
Shift Enter	Insert new line

You can also add new lines to a table with the **New Declaration** command from the **Edit** menu. You can insert a new line at the beginning, end of the table or before or after the current line.

Deleting Tables and Lines

Clicking on one of the shaded line number boxes at the left selects the corresponding line. Clicking on the empty uppermost box in the number box column selects the entire table. You can then delete the selected line or table by pressing the DEL key.

NOTE

The program performs these delete operations immediately, without prompting for confirmation. If you inadvertently delete something you can recover it by selecting the **Undo** command in the **Edit** menu. **Undo** only works if you select it directly after the delete operation, however!

Formatting Tables

You can adjust the width of the table columns to suit your individual needs. Move the mouse pointer to the dividing line between the shaded column title boxes, so that the pointer changes to a double-headed arrow. Then press and hold the left mouse button and drag the shaded dividing line until the column has the desired width.

5.3 The Editors

Your PLC programs are always divided into a number of logical program sections - referred to as "networks". These networks can be assigned names (labels) which can then be used as jump destinations within the PLC program. New networks are inserted with the **New Network** command in the **Edit** menu.

To open an editing window, simply double-click on a Body entry in the Project Navigator window.

Text Editors

- IEC Instruction List
- MELSEC Instruction List
- Structured Text

Graphical Editors

- Ladder Diagram
- Function Block Diagram
- Sequential Function Chart
  Please refer to the Reference Manual for full details on using the SFC language.

5.3.1 Using the text editors

All cursor movements and editing functions are similar to those of a standard word processor. The following additional conventions also apply in the text editors:

- To activate editing mode, click on the surface of a network with the mouse pointer.
- Each program line contains one controller instruction, with the following syntax:
  `Operator  TABSTOP Operand(s)`
- The operator and the operands must always be separated by tabstops.
- Pressing F2 when the cursor is in the first column displays a list of available programming instructions; pressing it in the second column displays a list of available operands (variables).
- You can also enter optional comments, which can be one or more lines long. Comments must be enclosed between (* and *) characters.
- You can move around in the program lines with the normal cursor keys.

5.3.2 Using the graphical editors

Working in the graphical editors is similar to using a drawing program. You can add elements to the networks in the editing windows by selecting symbols in the toolbar and with the commands in the Tools menu. The following elements are available:

- Contact (input, LD only)
- Coil (output, LD only)
- Programming instructions
- Input variable
- Output variable
- Return instruction
- Jump label
- Comment

Once you have positioned the elements, you then connect them with one another using interconnect lines.

NOTE In Chapter 6 (Step 6) you will learn how to use the different editors

6 Getting Started

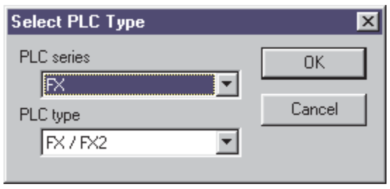
This chapter contains an introductory outline of all the steps required to create a new project with GX IEC Developer, with clear instructions on the procedures necessary in each step.

6.1 Step 1: Creating New Projects

How to create a new project

① Select **New** in the **Project** menu.

② This displays the **Select PLC Type** dialogue box. Select your PLC in the **PLC Type** field and confirm your selection with **OK**.

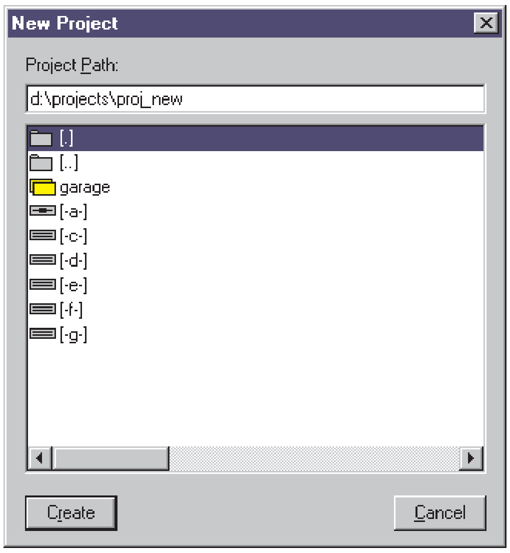


PLC series: FX
PLC type: FX / FX2

Fig. 6-1
CPU type selection

③ The **New Project** dialogue box is displayed. Select or enter the path under which you wish to store the new project.

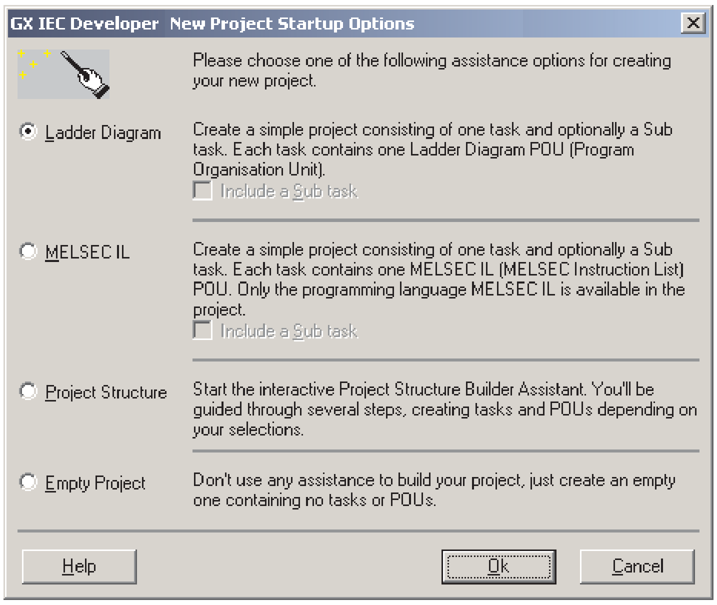
④ Enter a name for the new project at the end of the path (the project name is also the name of the subdirectory/folder in which it is stored). When you are satisfied, click on the **Create** button to create the project.



Project Path: d:\projects\proj_new

Fig. 6-2:
In this example a project called ***PROJ_NEW*** *is being created in the subdirectory* ***D:\PROJECTS.***
Please note that PROJ_NEW is not a single file, but rather a subdirectory created by GX IEC Developer to contain all the project files.

⑤ In the dialogue box **GX IEC Developer New Project Startup Options** click on the **Empty Project** option button and confirm with **OK**.


GX IEC Developer New Project Startup Options

Please choose one of the following assistance options for creating your new project.

[✓ Ladder Diagram] Create a simple project consisting of one task and optionally a Sub task. Each task contains one Ladder Diagram POU (Program Organisation Unit).
☐ Include a Sub task

MELSEC IL Create a simple project consisting of one task and optionally a Sub task. Each task contains one MELSEC IL (MELSEC Instruction List) POU. Only the programming language MELSEC IL is available in the project.
☐ Include a Sub task

Project Structure Start the interactive Project Structure Builder Assistant. You'll be guided through several steps, creating tasks and POUs depending on your selections.

Empty Project Don't use any assistance to build your project, just create an empty one containing no tasks or POUs.

Fig. 6-3: *GX IEC Developer then creates the empty new project as defined.*

As soon as you have created a new project the Project Navigator window is displayed on the screen automatically with all the standard entries for the project.

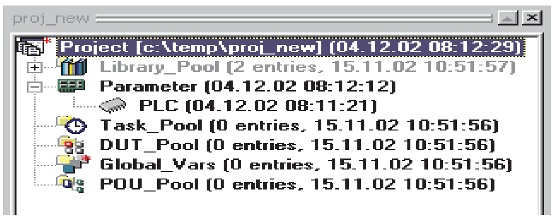

```
proj_new
Project [c:\temp\proj_new] (04.12.02 08:12:29)
  Library_Pool (2 entries, 15.11.02 10:51:57)
  Parameter (04.12.02 08:12:12)
      PLC (04.12.02 08:11:21)
  Task_Pool (0 entries, 15.11.02 10:51:56)
  DUT_Pool (0 entries, 15.11.02 10:51:56)
  Global_Vars (0 entries, 15.11.02 10:51:56)
  POU_Pool (0 entries, 15.11.02 10:51:56)
```

Fig. 6-4:
Project Navigator showing standard entries for the project and the optionally activated additional informations

The project entries are displayed in a hierarchical tree structure, which always contains the following standard components:

– Project Name
– Library Pool
– PLC Parameters
– Task Pool
– Data Unit Types Pool
– Global Variables
– Program Organisation Units

Additional information is optionally displayed in brackets behind each entry in the Project Navigator window. You can activate the display of these details by clicking on **Extended Information** in the **View** menu (a ✓ check mark is displayed next to the option when it is enabled).

The standard GX IEC Developer window background colour is light grey. You can change all the colours to suit your personal taste with **Colors** in the **View** menu.

6.2 Step 2: Creating Tasks

How to define a new task

① Select the Project Navigator window.

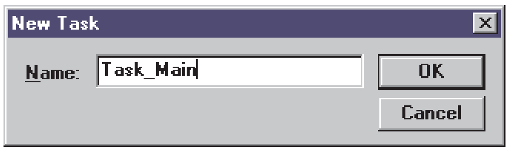
② Select **New** in the **Object** menu, then select the **Task** option.

Or

Click on the New Task icon in the toolbar:

NOTE This tool icon is only displayed in the toolbar when the Project Navigator window is displayed on the screen, i.e. when a project is open.

The **New Task** dialogue box is displayed.



Task_Main

Fig. 6-5:
Defining a new task

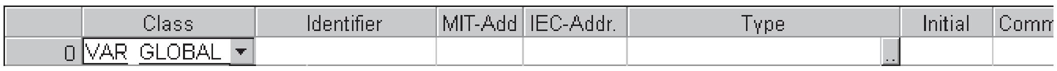
③ Enter a name (max. 32 characters) for the new task and confirm with **OK**. GX IEC Developer then creates the task and displays the name in the Task Pool in the Project Navigator window.

NOTE Assignment of the program organisation units (POUs) to the tasks and definition of the task attributes will be performed later on in Step 8.

6.3 Step 3: Declaring Global Variables

How to declare global variables

① Double-click on the **Global_Vars** branch in the Project Navigator. This opens the Global Variable List window on the right hand side of the screen, containing the declaration table for entry of the variables.



	Class	Identifier	MIT-Add	IEC-Addr.	Type	Initial	Comm
0	VAR_GLOBAL						

Fig. 6-6: *Declaration table of Global Variable List*

② Click in the first cell in the **Class** column with the mouse cursor, then click on the up arrow button and select VAR_GLOBAL or VAR_GLOBAL_CONSTANT.

③ Press Tab to move to the **Identifier** column, then enter the identifier for your first global variable.

④ Press Tab to move to the **MIT-Addr**. or **IEC-Addr**. column. Enter the absolute address of the global variable.

⑤ Press Tab to move to the **Type** column and click on the up arrow button with the mouse to open the Type Selection dialogue box.

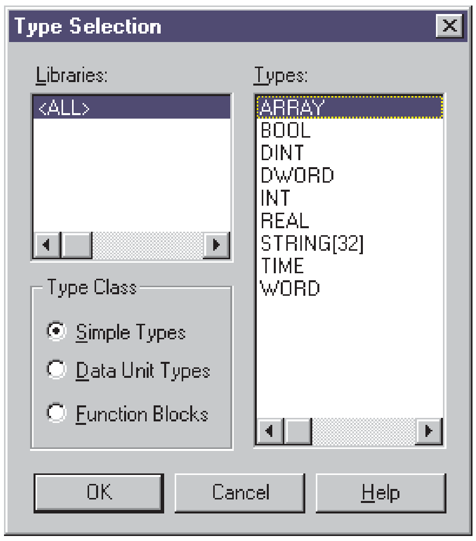


Fig. 6-7:
Data type selection

⑥ Select **Simple Types** in the Type Class field.

⑦ Select the appropriate data type from the list on the left.

The initial value in the **Initial** column is assigned automatically and cannot be changed by the user.

⑧ If you want to enter a comment text for the variable press Tab to move to the **Comment** column, then enter your text.

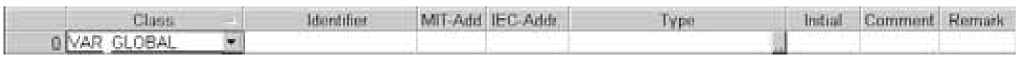
⑨ To enter another variable,

- If the editing cursor is active in the Comment column (white background with blinking cursor), you can create a new variable declaration line by pressing Tab.

Fig. 6-8:
Entering another variable

or

- Select any cell in the last line of the table and press Shift Enter.



	Class	Identifier	MIT-Addr	IEC-Addr	Type	Initial	Comment	Remark
0	VAR_GLOBAL							

Fig. 6-9: *Entering another variable*

or

- Select **New Declaration** in the **Edit** menu and then select the position in which it is to be inserted from the submenu.

or

- Copy an existing declaration line: First select the line by clicking on its number button at the left, then press Ctrl C to copy it to the clipboard. Then select the insertion position by clicking on the appropriate number button and press one of the icons to insert a new line above or below the selected line. Click on the number of the new line and press Ctrl V to overwrite the new line with the copied line from the clipboard.

⑩ Save your new entries with **Object - Save.**

NOTE The terms **identifier**, **address** and **data type** are defined and explained in Chapter 3.

6.4 Step 4: Creating Program Organisation Units

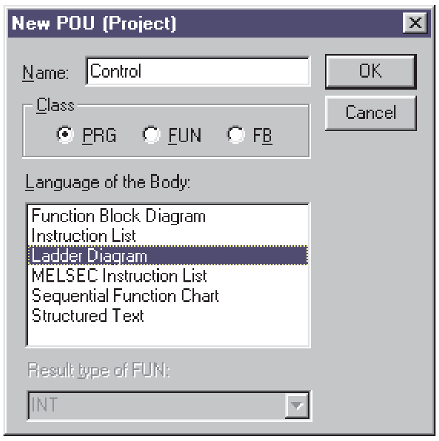
Program organisation units (POUs) always consist of two main parts, a header and a body.

How to create a new program organisation unit

① Click on the New POU icon in the toolbar:

NOTE This tool icon is only displayed in the toolbar when the Project Navigator window is displayed on the screen, i.e. when a project is open.

② Enter a name for the new POU and specify whether it is to be created as a program (PRG), a function (FUN) or a function block (FB). Then select the programming language/editor to be used for the creation of the PLC program in the POU's body. When you are satisfied that all your entries are correct select **OK** to add the new POU to the project.


New POU (Project)

Name: Control

Class: [✓ PRG] FUN FB

Language of the Body:
- Function Block Diagram
- Instruction List
- [✓ Ladder Diagram]
- MELSEC Instruction List
- Sequential Function Chart
- Structured Text

Result type of FUN: INT

Fig. 6-10:
A new POU called "Control" is being defined and declared as a "Program" (PRG) type.

The PLC program in the body of the POU is going to be written in ladder diagram language.

The new "Control" POU is then added to the project and appears in the POU Pool in the Project Navigator window.

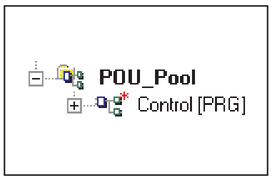


Fig. 6-11:
The **[+]** *symbol to the left of "Control" in the project tree indicates that this entry has subordinate entries that are currently collapsed. The asterisk in front of the term "Control" indicates that this POU has not yet been compiled.*

③ Double-click on "Control" to open the subordinate entries.

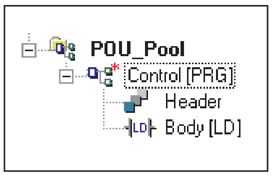


Fig. 6-12:
Every POU has two main components: a header and a body containing the actual program in the selected programming language.

6.5 Step 5: Programming POU Headers

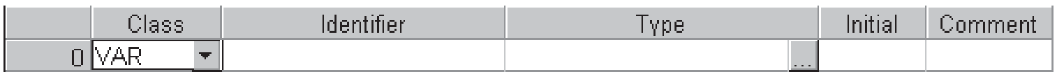
The POU header is used to declare and store the variables used by the program that the POU contains. In addition to global and local variables, these declarations can also include instances of function blocks.

How to program the POU header

① Check that the Header and Body entries are displayed under the POU entry in the POU Pool and expand them if necessary (see ③ in Step 4).

② Double-click on the "Header" entry in the Project Navigator window. This opens a window containing the declaration table for the header's local variables on the right hand side of the screen.

③ To declare the variables, proceed in exactly the same way as with the global variables in Step 3 above, entering the class, identifier and data type for each variable.



	Class	Identifier	Type	Initial	Comment
0	VAR		...		

Fig. 6-13: *Declaring variables for the program header*

④ Select **Save** in the **Object** menu to store your entries

NOTE

If you wish to enter global variables in the header you can copy them from the global variables declaration table (Ctrl C) and then insert them in this declaration table (Ctrl V).

The terms **class**, **identifier** and **data type** are defined and explained in Chapter 3.

6.6 Step 6: Programming POU Bodies

The body contains the actual code of the PLC program. The programming language used is shown in the information in brackets following the "Body" entry in the project tree.

How to program the POU body

① Check that the Header and Body entries are displayed under the POU entry in the POU Pool and expand them if necessary (see ③ in Step 4).

② Double-click on the "Body" entry in the Project Navigator. The editing window of the editor for the selected programming language is opened on the right hand side of the screen. It contains one network.

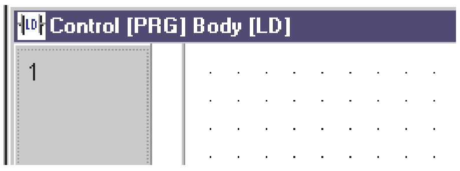


Fig. 6-14:
Programming a POU body in a network editor

If you wish, you can disable the background grid display by clicking on **Grid** in the **View** menu (a ✓ check mark is displayed in the menu when the function is enabled).
You can adjust the size of the background grid with the **Environment** option in the **View** menu. Please note that the value you enter for the grid spacing changes the setting for the entire screen setup, and not just for the selected programming editor.

③ Now you can start writing your PLC program.

You will find programming examples for the various programming languages and editors on the following pages.

④ Select **Save** in the **Object** menu to save the body of your POU.

NOTE

You can increase the size of the editing area with the mouse. Position the mouse pointer on the lower edge of the network bar box at the left of the editing window (the pointer changes to a double arrow when it is over the resize line). To resize the editing area, hold down the left mouse button, drag the dotted line to the desired position and then release the button.

6.7 Programming Examples

6.7.1 Inputs and outputs in ladder diagram language (LD)

Programming inputs and outputs in the LD editor

① In the Project Navigator window, double-click on a program body entry defined with the ladder diagram language (LD).

② Click on the “Contact” tool icon in the toolbar.

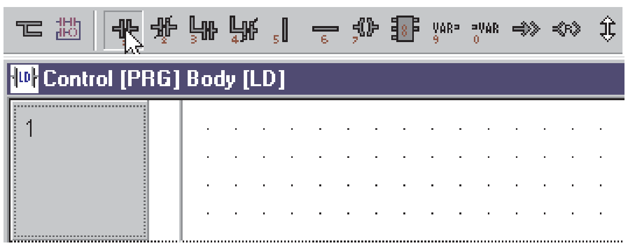

Control [PRG] Body [LD]

Fig. 6-15:
Selecting the "Contact" tool

③ Move the mouse pointer to the desired position and press the left mouse button to place the input contact.

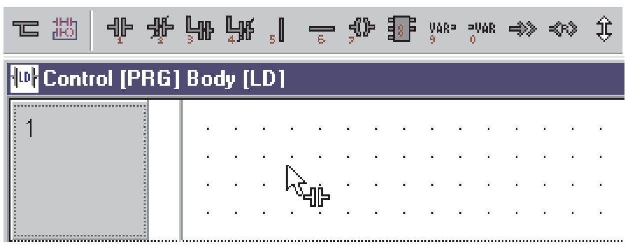

Control [PRG] Body [LD]

Fig. 6-16:
Placing the input contact

④ Click on the “Coil” tool icon in the toolbar.

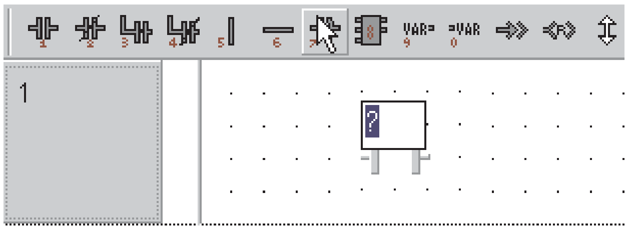


Fig. 6-17:
Selecting the "Coil" tool

⑤ Move the mouse pointer to the desired position and press the left mouse button to place the output coil.

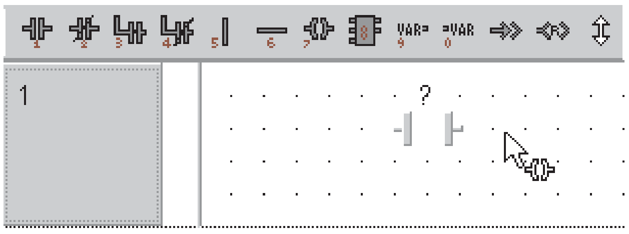


Fig. 6-18:
Placing the output coil

⑥ Click on the"Interconnect/Line" tool icon in the toolbar

or

Click with the right mouse button to open the context menu. Select the option “Line”.

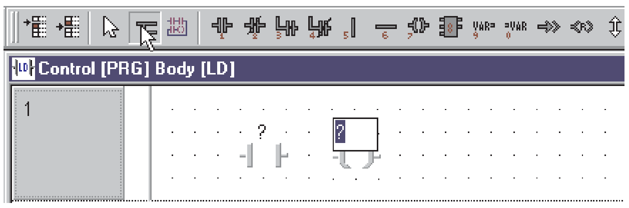


Fig. 6-19:
Selecting the "Line" tool

⑦ Position the pointer over the left network bar and click the left mouse button. Draw a line to the connection point of the output coil and left-click again.

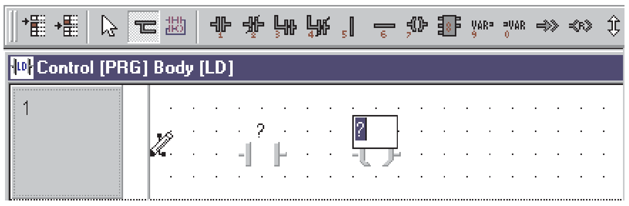


Fig. 6-20:
Drawing a connection line

The "?" character that appears above the input contact and output coil symbols serves as dummy, which you must replace with declared operand names or a direct address (X, Y).

⑧ Click on the "Select mode" tool icon in the toolbar. Using the mouse pointer, select the "?" dummy character over the contact and the coil and overwrite the dummy character with appropriate variable names. Alternatively, you can also press F2 to display the operand selection list and select a name from the list.

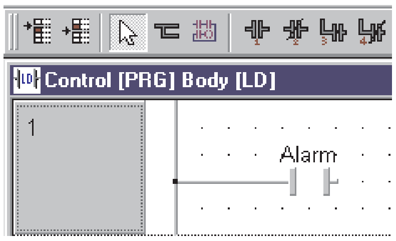


Fig. 6-21:
Selecting the "Selection" tool

NOTE For the described procedures the Auto Connect mode is not activated.

6.7.2 A Sum Function in FBD Language

Programming a sum function in the FBD editor

Steps ① through ⑥ below are exactly the same in the ladder diagram and function block diagram editors. Only the tools displayed in the toolbar are different in each case.

① In the Project Navigator window, double-click on a body entry defined with the function block diagram language (FBD).

② Click on the “Function Block” tool icon in the toolbar.

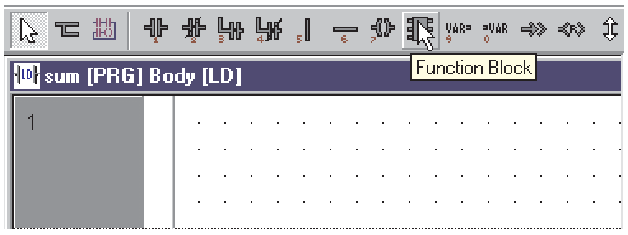


Fig. 6-22:
Selecting the "Function Block" tool

③ Double-click on the ADD instruction in the selection box displayed.

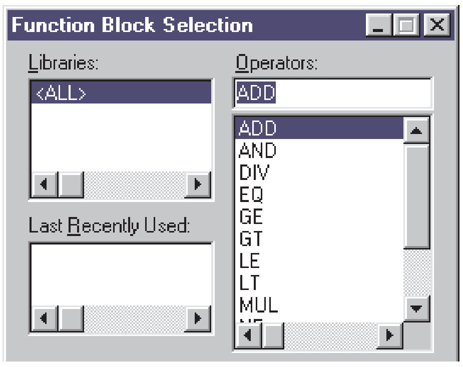


Fig. 6-23:
Selecting the "ADD" instruction

④ Position the mouse pointer and press the left mouse button to place the function block.

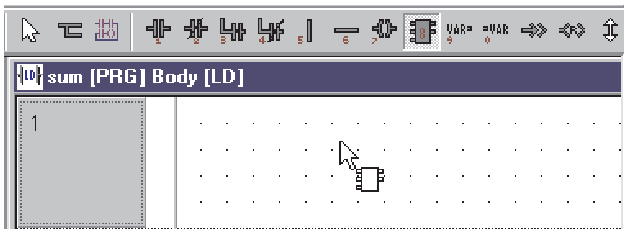


Fig. 6-24:
Placing the function block

⑤ Click on the dummy character “?” of the input variable. Overwrite the first dummy character with the number 12 and the second dummy character with the number 8.

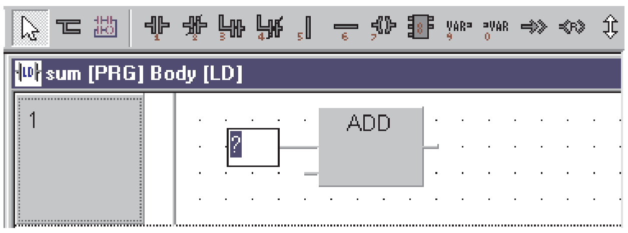
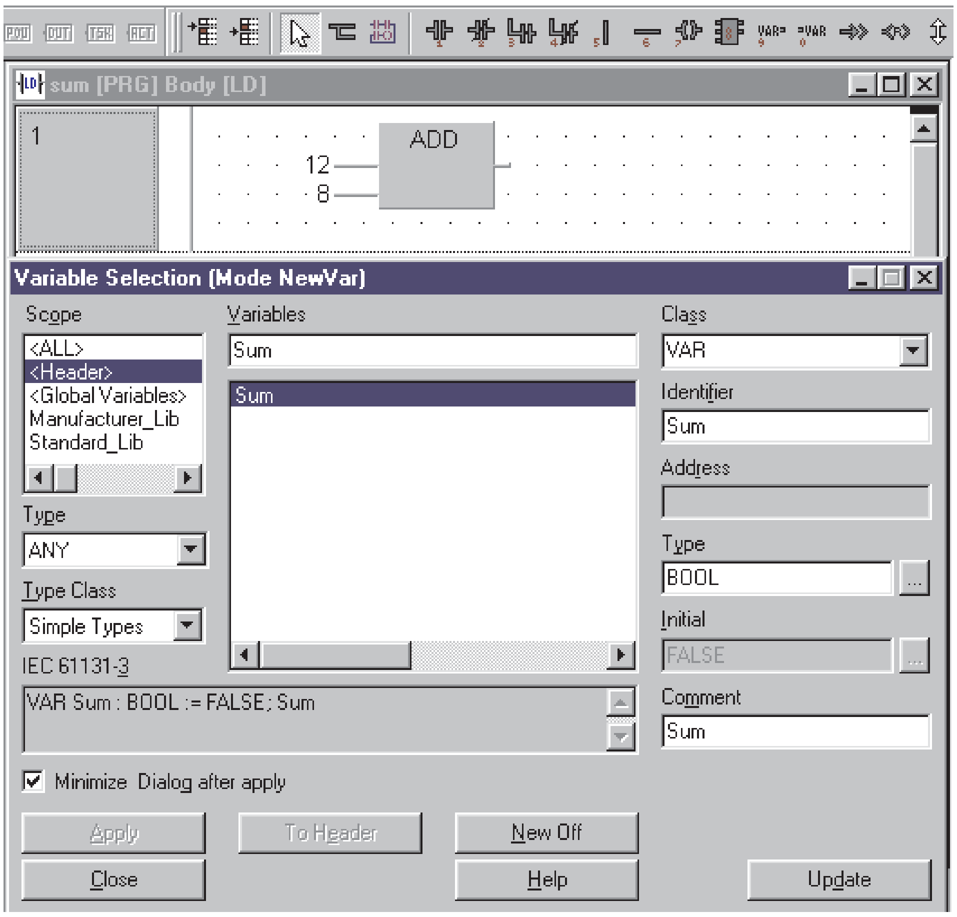


Fig. 6-25: *Overwriting the dummy character "?"*

⑥ Click on the dummy character “?” of the output variable and then press F2 to display the operand list. Select "Sum" and confirm with **OK**.

NOTE The variables will only appear in the operand list if the header in which they are declared has been saved.



Variables: Sum
Class: VAR
Identifier: Sum
Type: ANY
Type Class: Simple Types
Type: BOOL
Initial: FALSE
Comment: Sum

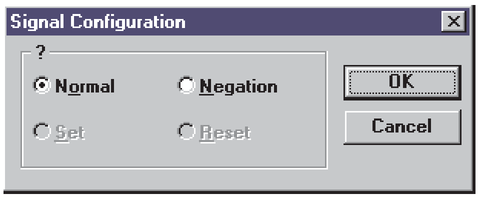
```
VAR Sum : BOOL := FALSE; Sum
```

Fig. 6-26: *Declared variables in the header must have been saved*

6.7.3 I/O Signal Configuration Parameters

Setting the signal configuration parameters of inputs and outputs

① Double-click on an input contact, an output coil or the connection point of a variable in a function block to display the Signal Configuration dialogue box. Select the appropriate options, then confirm with OK.



Signal Configuration

?

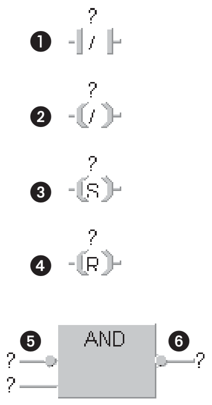
(● Normal) ○ Negation

○ Set ○ Reset

OK

Cancel

Fig. 6-27:
Signal configuration



Number	Description
❶	Negated input contact (LD only)
❷	Negated output coil (LD only)
❸	Set output coil (LD only)
❹	Reset output coil (LD only)
❺	Negated input variable (LD and FBD)
❻	Negated output variable (LD and FBD)

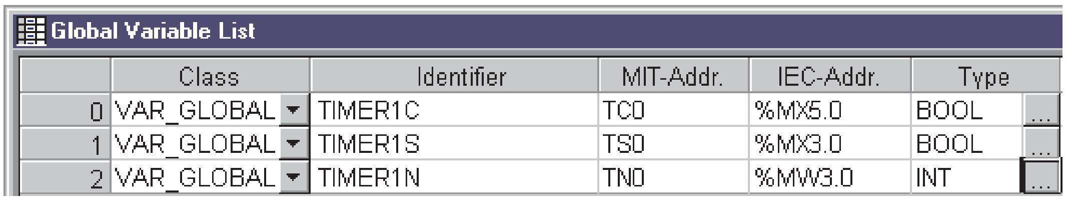
Tab. 6-1: *Key to figure above*

6.7.4 Timers in LD/FBD/IL

Description of the timer device

All timers must have the following four elements:

TValue:	Set point value
TN:	Actual value
TC:	Output coil
TS:	Input contact


Global Variable List

	Class	Identifier	MIT-Addr.	IEC-Addr.	Type
0	VAR_GLOBAL	TIMER1C	TC0	%MX5.0	BOOL
1	VAR_GLOBAL	TIMER1S	TS0	%MX3.0	BOOL
2	VAR_GLOBAL	TIMER1N	TN0	%MW3.0	INT

Fig. 6-28: *The elements TN, TC and TS must be declared in the global variables list.*

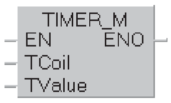


Fig. 6-29:
The element TValue is passed to the function directly.

The timer example

The following example shows how to program a timer and a function block instance (see Chapter 3) in ladder diagram, function block diagram and Instruction List languages.

Objective

When "Input1" is set the 100-ms timer "Timer1" must start to count and continue until it reaches a value of 100. We want "Output1" to be set when "Input2" is set, and we want "Output1" to be
reset again when the set point value of 100 is reached.

Algorithm:

"Input2" and the timer contact "Timer1S" (TS) are responsible for switching "Output1". This function will be performed by the user-programmed function block SET_RST."

"Input1" activates the timer, i.e. it controls the switching of the timer coil "Timer1C" (TC). The set point value TValue is 100. The timer function will be implemented by using the manufacturer function TIMER_M.

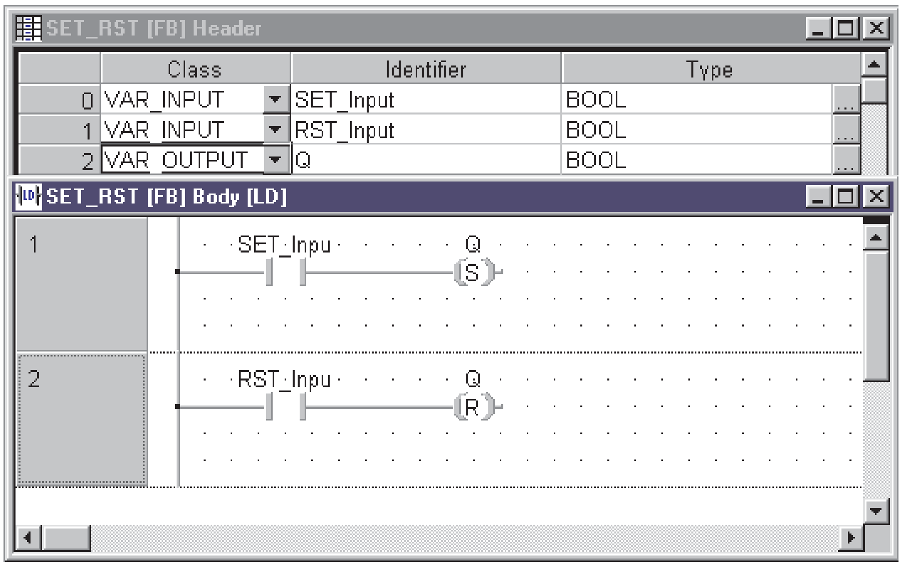
Creating the program

Step 1: Program the function block SET_RST

① Create a POU called **"SET_RST"** and define it as a function block to be programmed in ladder diagram language (see Step 4).

② Enter the following three variables in the header: SET, RST and Q.

③ Insert two network circuit blocks in the body.


SET_RST [FB] Header

	Class	Identifier	Type
0	VAR_INPUT	SET_Input	BOOL
1	VAR_INPUT	RST_Input	BOOL
2	VAR_OUTPUT	Q	BOOL

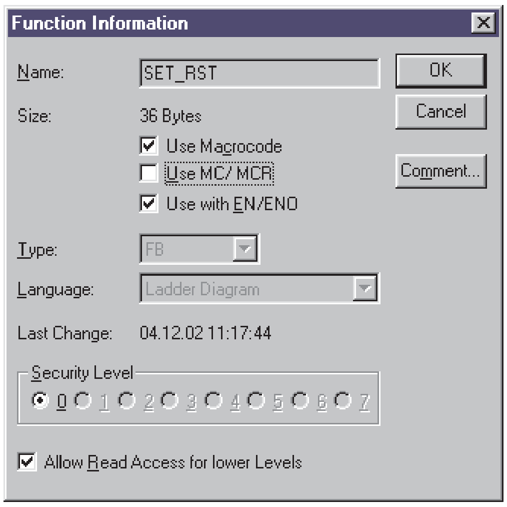
SET_RST [FB] Body [LD]

Fig. 6-30: SET and RST are input variables (VAR_INPUT). Q is an output variable (VAR_OUTPUT).

Q is set when SET is active, and Q is reset when RST is set.
You can configure the S (set) and R (reset) parameters by double-clicking on the coil symbol.

④ Save the header and the body (**Object - Save**).

⑤ Select the "SET_RST" POU in the Project Navigator window and press Alt Enter. The following dialogue box is then displayed:


Function Information

Name: SET_RST
Size: 36 Bytes
[✓] Use Macrocode
[ ] Use MC/ MCR
[✓] Use with EN/ENO
Type: FB
Language: Ladder Diagram
Last Change: 04.12.02 11:17:44
Security Level: [● 0] ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7
[✓] Allow Read Access for lower Levels

Fig. 6-31: Function information

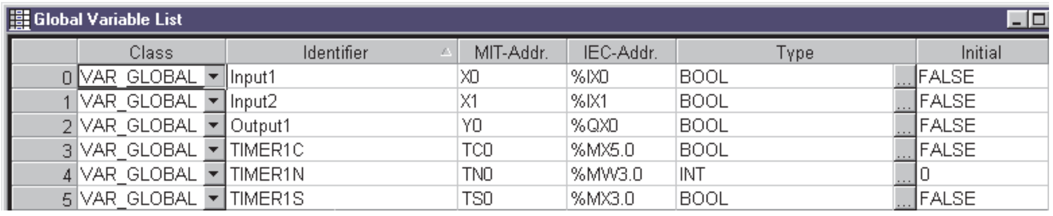
⑥ Activate the EN/ENO Contacts check box to assign an EN input and an ENO output to the function block and activate the Macrocode to create an optimized code.

Step 2: Define the global variables

The timer elements TC, TS and TN **must** first be declared in the Global Variable List. They can then be called in the header of the POU in which the timer is going to be used. In this example, the input and output variables are also declared globally.

① Open the global variable declaration table (see Step 3).

② Declare the following variables.



Global Variable List

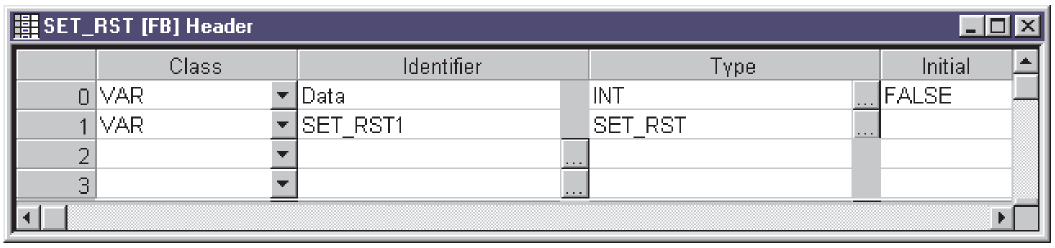
	Class	Identifier	MIT-Addr.	IEC-Addr.	Type	Initial
0	VAR_GLOBAL	Input1	X0	%IX0	BOOL	FALSE
1	VAR_GLOBAL	Input2	X1	%IX1	BOOL	FALSE
2	VAR_GLOBAL	Output1	Y0	%QX0	BOOL	FALSE
3	VAR_GLOBAL	TIMER1C	TC0	%MX5.0	BOOL	FALSE
4	VAR_GLOBAL	TIMER1N	TN0	%MW3.0	INT	0
5	VAR_GLOBAL	TIMER1S	TS0	%MX3.0	BOOL	FALSE

Fig. 6-32: *Declaring the global variables*

Step 3: Create the Timer POU (ladder diagram)

① Create a new POU as a program using the ladder diagram language (see Step 4).

② Open the header of the new POU and declare the following variables:



SET_RST [FB] Header

	Class	Identifier	Type	Initial
0	VAR	Data	INT	FALSE
1	VAR	SET_RST1	SET_RST	
2				
3				

Fig. 6-33: *Declaring the variables in the header*

You can speed up this process by copying the variables that you have already entered in the Global Variable List and inserting them here.

Use global variables = VAR_GLOBAL
Use as local variables = VAR
The variable SET_RST1 is an instance of the function block SET_RST (see Chapter 3 for details on instancing).

③ Save the header (**Object - Save**).

④ Open the body of the POU.
The timer we shall use is a function that is stored in the manufacturer library.

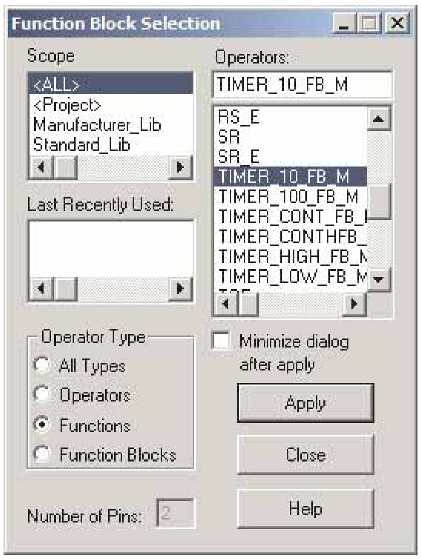


Fig. 6-34:
The timer function is called ***TIMER_M***

You can find instructions on how to insert function blocks in the editing window in the section on ”Sum Function in FBD”.

⑤ Create the following PLC program:

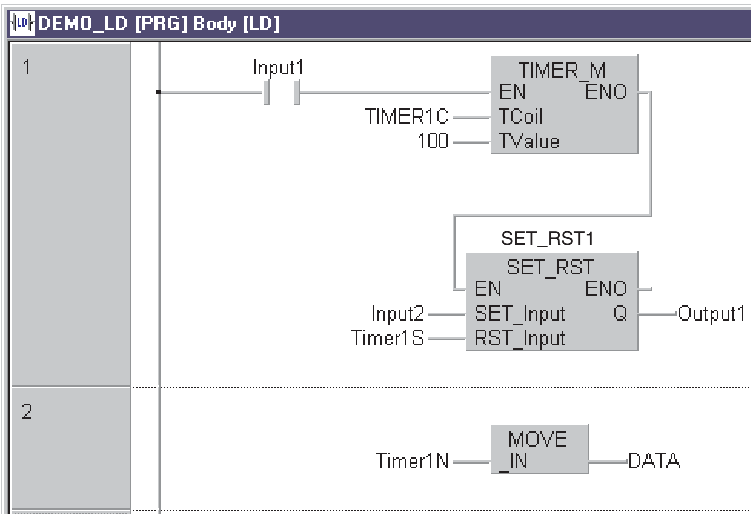


Fig. 6-35:
Ladder Diagram program example

The function blocks used in the program can be found in the following libraries:

Manufacturer Library:	TIMER_M
Function blocks:	SET_RST
IEC Standard Library:	MOVE

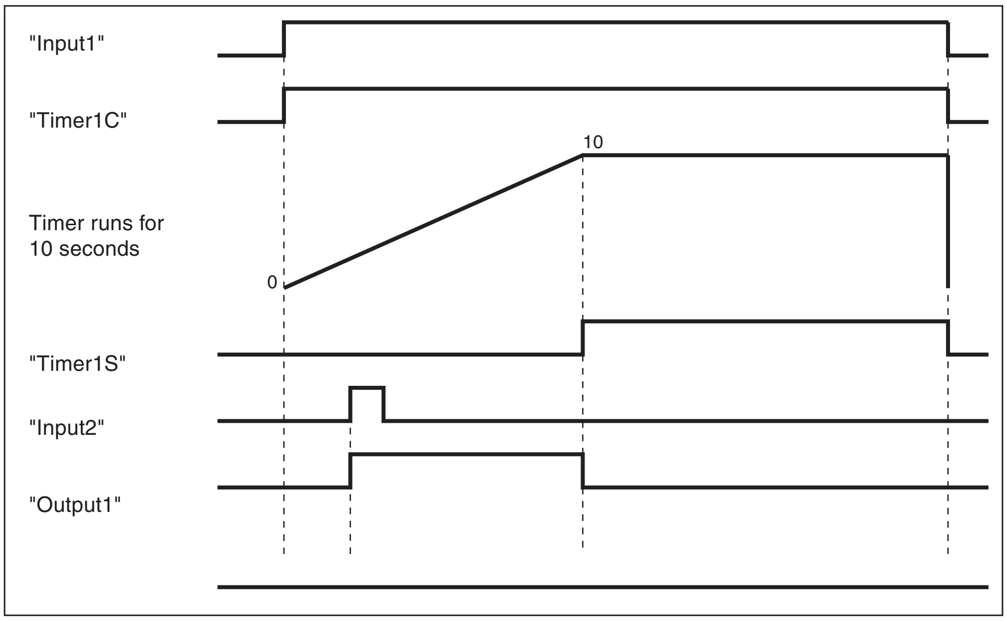


Fig. 6-36: *Timer sequence*

Timer sequence

The timer sequence begins when INPUT1 is set and the timer starts to run. If INPUT2 is set, OUTPUT1 is switched on.

When the 10-second period has elapsed, timer contact TIMER1S is set and OUTPUT1 is switched off again.
If Input2 still remains set or is set, Output1 will be set again.

The lower program block containing the MOVE instruction is only necessary to make it possible to follow the 10-second sequence in monitoring mode.

Programming the timer in function block diagram language

The following illustration shows how to realize the same program using function block diagram language:

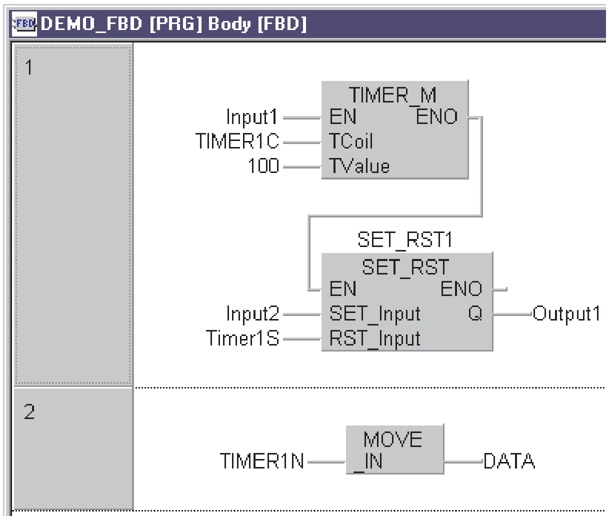


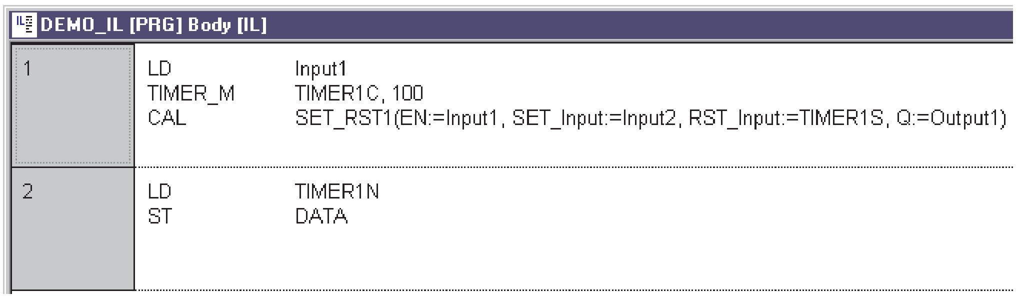
Fig. 6-37:
Function Block Diagram program example

Function block diagram language

Timer:	
Coil:	TIMER2C
Contact:	TIMER2S
Actual value:	TIMER2N

Programming the timer in Instruction List language

The following illustration shows how to realize the same program using Instruction List language:



DEMO_IL [PRG] Body [IL]

```
1    LD        Input1
     TIMER_M   TIMER1C, 100
     CAL       SET_RST1(EN:=Input1, SET_Input:=Input2, RST_Input:=TIMER1S, Q:=Output1)

2    LD        TIMER1N
     ST        DATA
```

Fig. 6-38: Instruction List program example

Instruction list language

Timer:	
Coil:	TIMER1C
Contact:	TIMER1S
Actual value:	TIMER1N

NOTE Please refer to Chapter 3 for detailed instructions on how to call functions and pass the actual parameters to the formal parameters of function block instances in the Instruction List language.

6.7.5 Sequential Function Chart Language

You can find a basic introduction to the SFC programming language in Chapter 3 of this manual. More detailed information is provided in the Reference Manual. The following example is a step-by-step illustration of the procedures required to create an SFC program using the GX IEC Developer tools.

The "Process" sample program

The program is called "Process" and solves the problem using a variety of selective branching constructs and the Jump instruction.

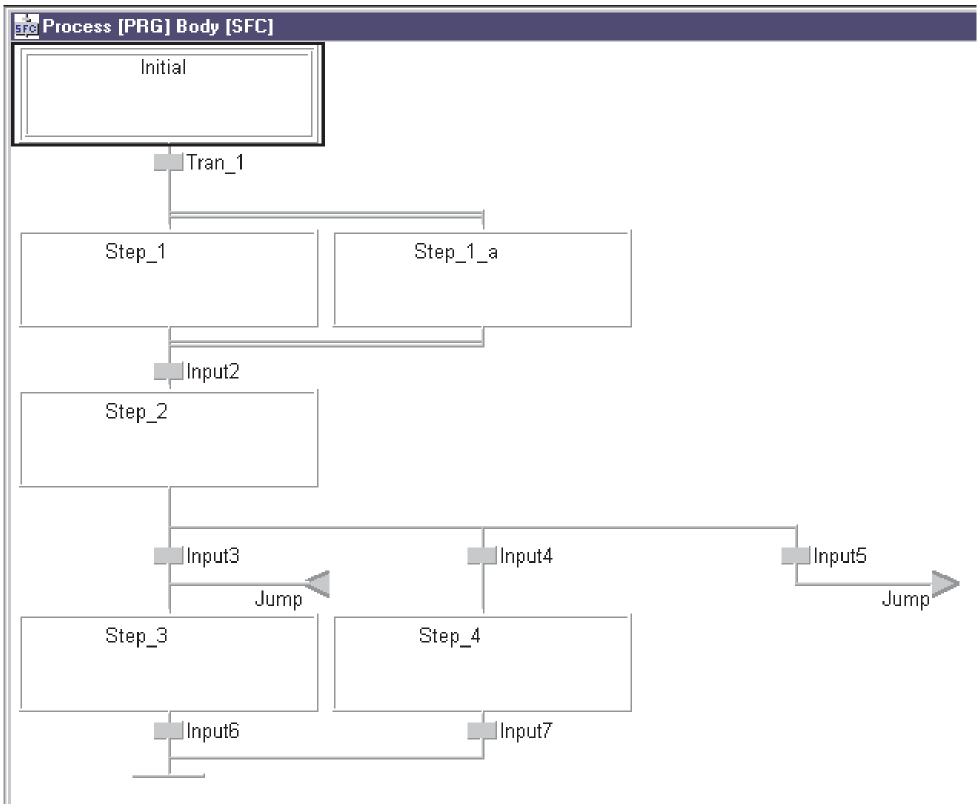


Fig. 6-39: *The "Process" program*

Program execution

- When the PLC is switched to RUN mode "Output1" is set.
- The transition "TRAN_1" performs a TRUE/FALSE poll of "Input1". If "Input1" is set "Step_1" and "Step_1_a" are both activated. "Output2" blinks and "Output3" is switched on continuously.
- The transition "Input2" polls "Input2". If the latter is set "Step_2" is executed and "Output4" is set.
- In the subsequent branches to "Input3", "Input4" and "Input5" a variety of program sequences then execute in parallel.

  If "Input3" is set, this activates "Step_3". "Input5" activates the jump exit point "Jump" which leads to the jump entry point "Jump" and also activates "Step_3". "Step_3" sets "Output5".

  If "Input4" is set, this triggers "Step_4", which then switches "Output6".
- "Input6" and "Input7" lead to the end of the program.

Creating the program

Perform steps 1 through 8 in the order described.

Step 1: Create the POU

① Create a new POU called "Process". Select PRG (program) as the class and Sequential Function Chart as the programming language (see Step 4).

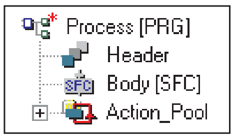
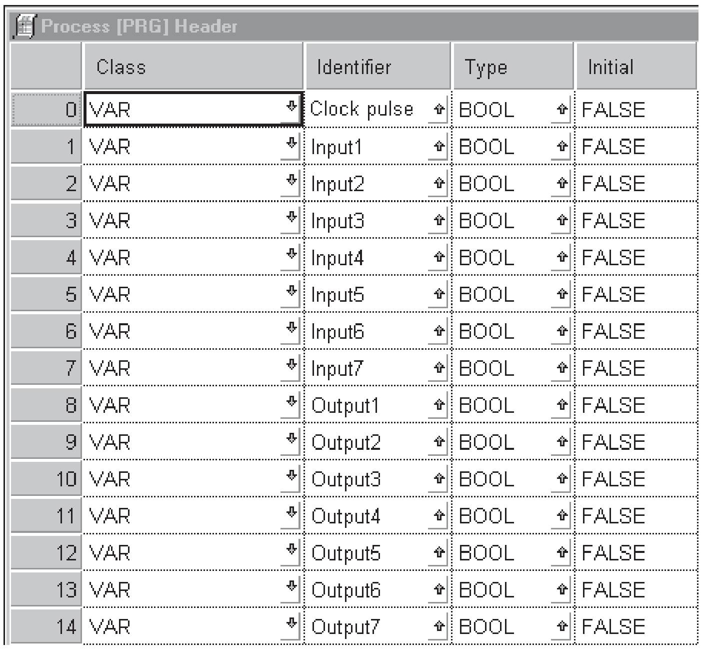


Fig. 6-40:
The new "Process" POU is displayed in the Project Navigator.

In addition to the header and the body, each POU written in SFC language also has an action pool entry in which the actions assigned to the POU are stored.

Step 2: Declare the variables in the header

① Open the header and enter the local variables to be used in the POU (see Step 3). Global variables do not need to be declared here, as they can be used directly as global variables.



Process [PRG] Header

	Class	Identifier	Type	Initial
0	VAR	Clock pulse	BOOL	FALSE
1	VAR	Input1	BOOL	FALSE
2	VAR	Input2	BOOL	FALSE
3	VAR	Input3	BOOL	FALSE
4	VAR	Input4	BOOL	FALSE
5	VAR	Input5	BOOL	FALSE
6	VAR	Input6	BOOL	FALSE
7	VAR	Input7	BOOL	FALSE
8	VAR	Output1	BOOL	FALSE
9	VAR	Output2	BOOL	FALSE
10	VAR	Output3	BOOL	FALSE
11	VAR	Output4	BOOL	FALSE
12	VAR	Output5	BOOL	FALSE
13	VAR	Output6	BOOL	FALSE
14	VAR	Output7	BOOL	FALSE

Fig. 6-41:
In this example only local variables are used.

Step 3: Open the body

① Double-click on the "Body" entry in the Project Navigator window.

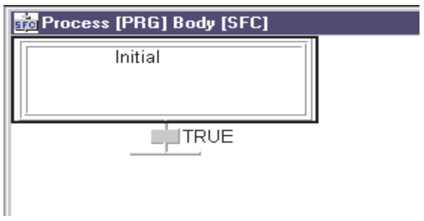


Fig. 6-42:
When you open the SFC editor the following elements are displayed:
- The Initial Step (double outline)
- The transition TRUE
- The Final Step

NOTE Steps to which no actions are yet associated are shown filled in white. The fill colour changes to grey when actions are associated. Your current position in the sequence is indicated by a "block cursor", displayed as a black rectangle around the elements that can be moved around at will in the editing window with the mouse or cursor keys. The tool icons activated in the toolbar change depending on the current position of this cursor; you cannot use all the tools at all positions in a program.

Step 4: Create the sequence

① Select the "TRUE" transition with the block cursor.

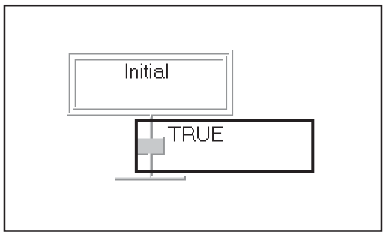


Fig. 6-43:
Selecting the "TRUE" transition

② Click on the tool icon

Inserts a new step and transition pair.

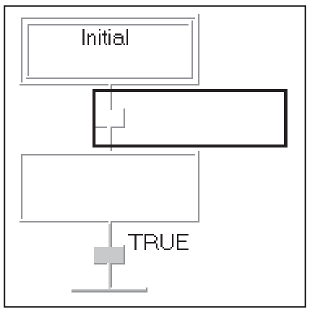


Fig. 6-44:
Inserted new step and transition pair

③ Select the step you have just inserted.

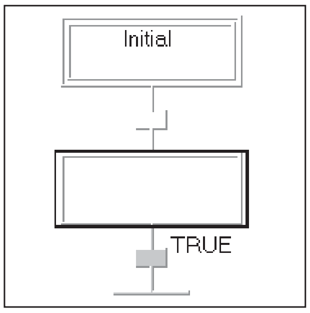


Fig. 6-45:
Selected new step

④ Click on the tool icon

Inserts a right divergence and a new step.

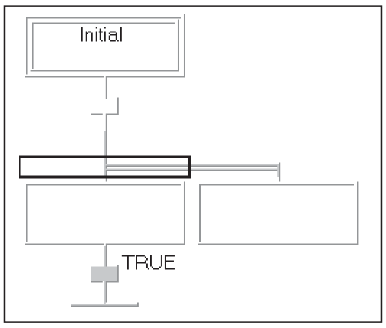


Fig. 6-46:
Inserted new right divergence and step

⑤ Select the "TRUE" transition.

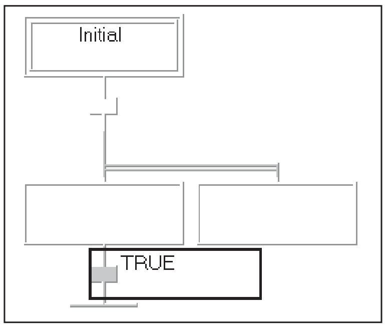


Fig. 6-47:
Selected "TRUE" transition

⑥ Click on the tool icon

Inserts a right convergence.

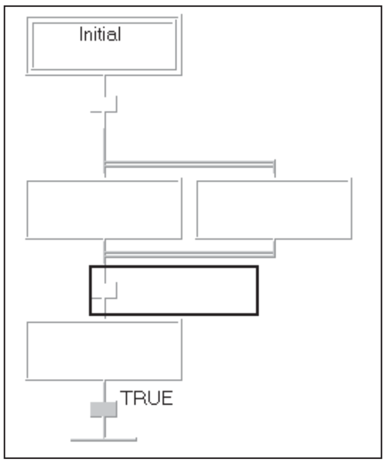


Fig. 6-48:
Inserted new right convergence

⑦ Select the "TRUE" transition.

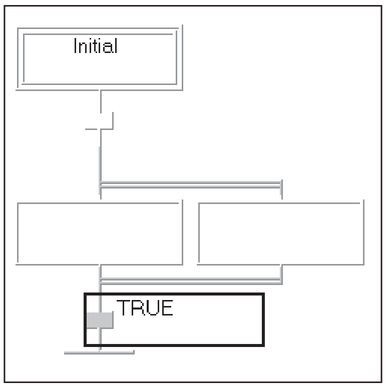


Fig. 6-49:
Selected "TRUE" transition

⑧ Click on the tool icon

Inserts a new step and transition pair.

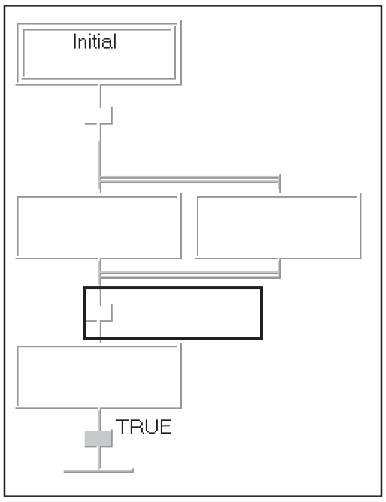


Fig. 6-50:
Inserted new step and transition pair

⑨ Select the "TRUE" transition.

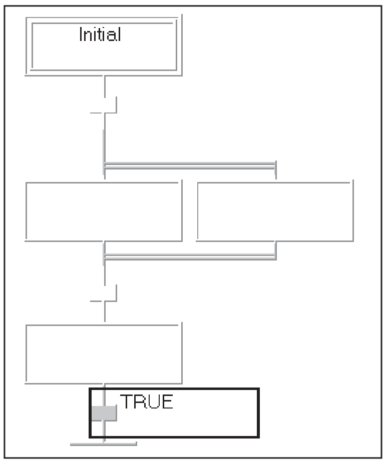


Fig. 6-51:
Selected "TRUE" transition

⑩ Click twice on the tool icon ⊢┐

Inserts two right divergences with transitions.

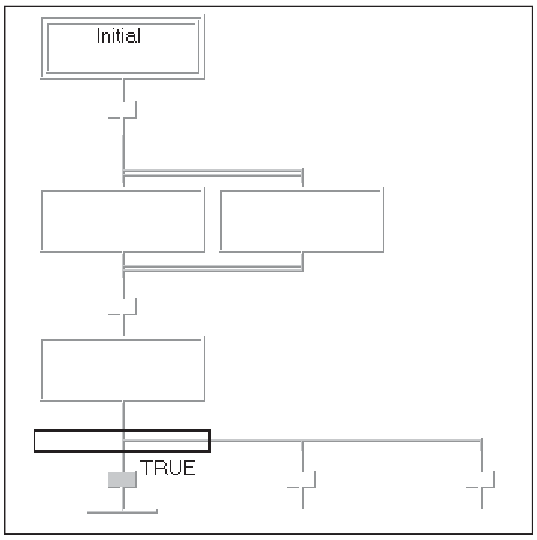


Fig. 6-52:
Two inserted new right divergences

⑪ Select the "TRUE" transition.

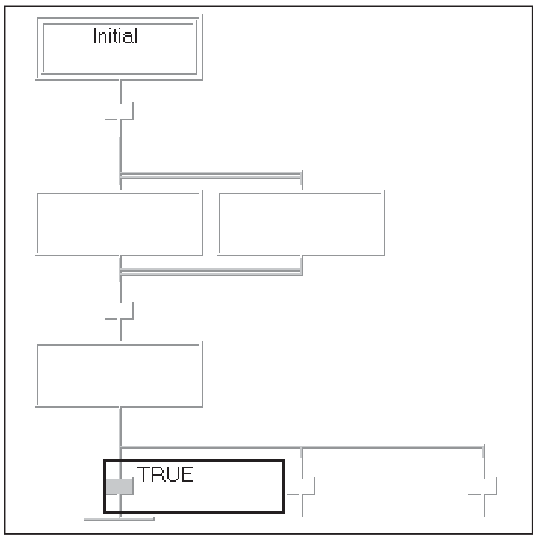


Fig. 6-53:
Selected "TRUE" transition

⑫ Click on the tool icon

Inserts a new step and transition pair.

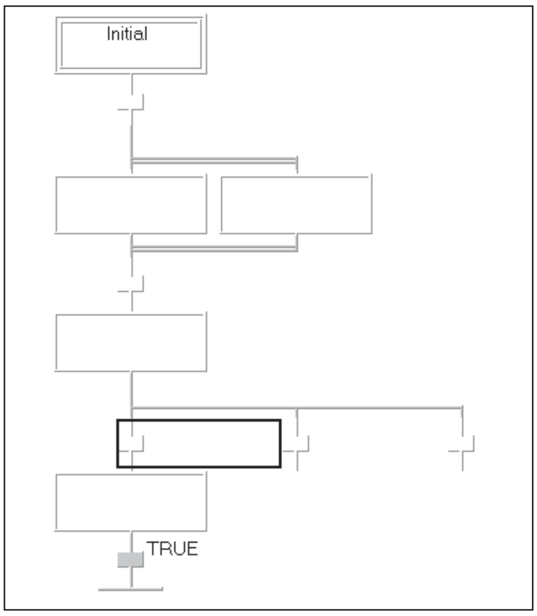


Fig. 6-54:
Inserted new step and transition pair

⑬ Click in the empty space next to the step you have just inserted.

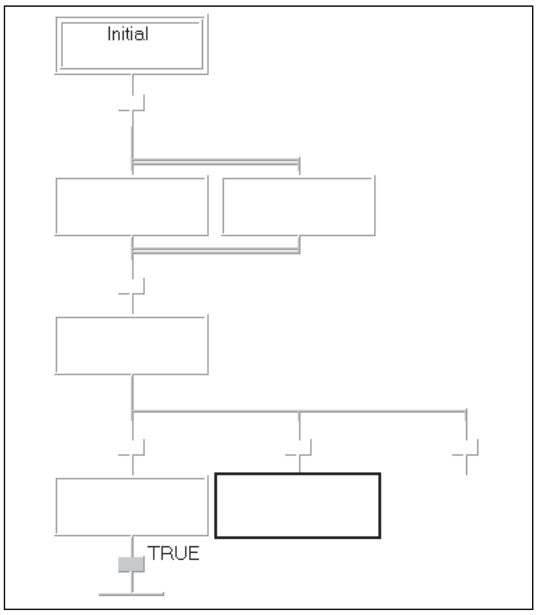


Fig. 6-55:
Selection rectangle for location of new step and transition pair

⑭ Click on the tool icon

Inserts a new step and transition pair.

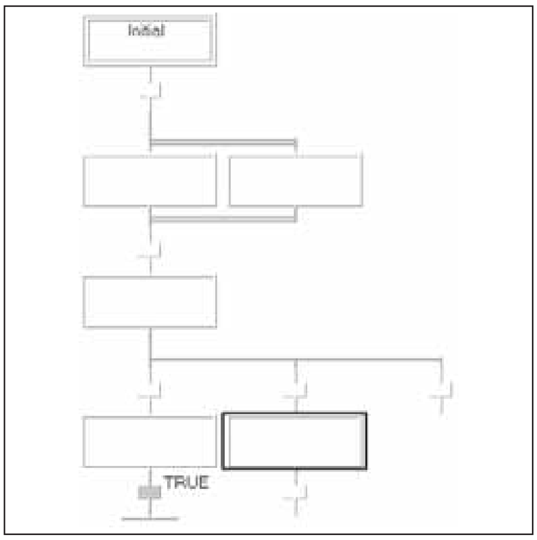


Fig. 6-56:
Inserted new step and transition pair

⑮ Click on the final step.

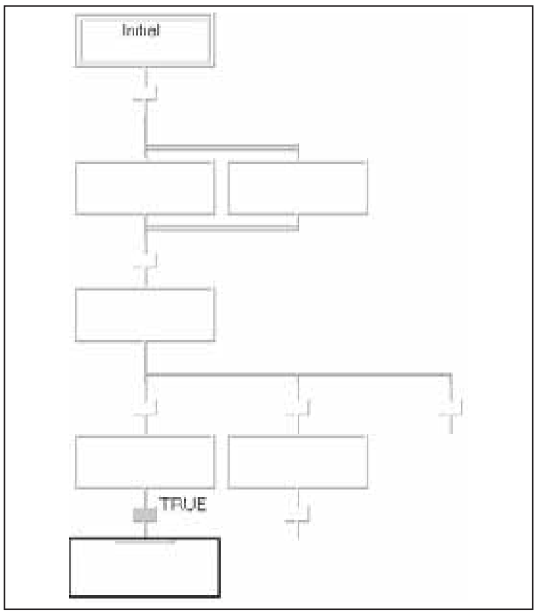


Fig. 6-57:
Selected final step

⑯ Click on the tool icon

⊢

Inserts a right convergence.

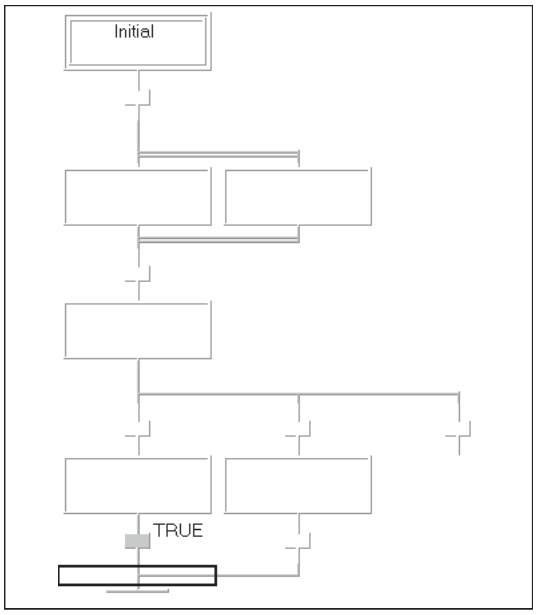


Fig. 6-58:
Inserted new right convergence

⑰ Click on the left hand step in the bottom row.

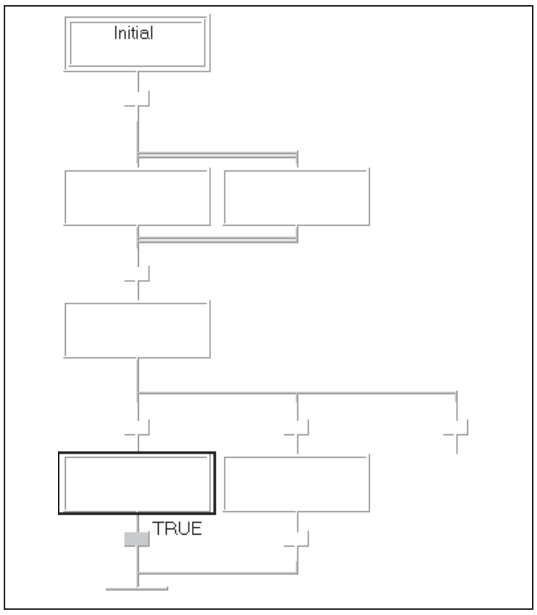


Fig. 6-59:
Selected step in the bottom row

⑱ Click on the tool icon

Inserts a jump entry point.

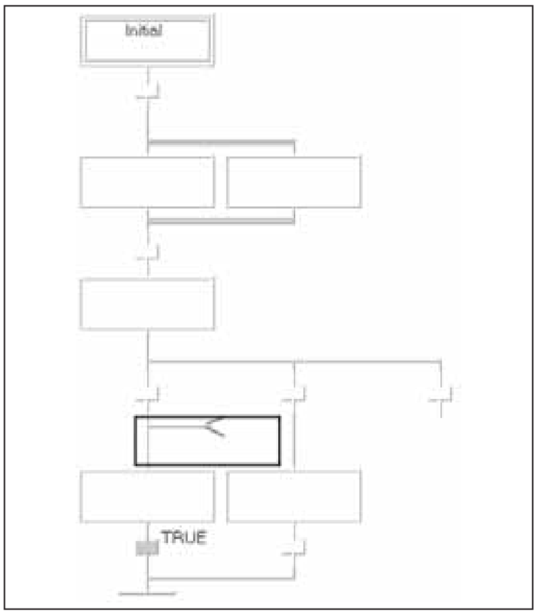


Fig. 6-60:
Inserted jump entry point

⑲ Click on the empty space below the free transition.

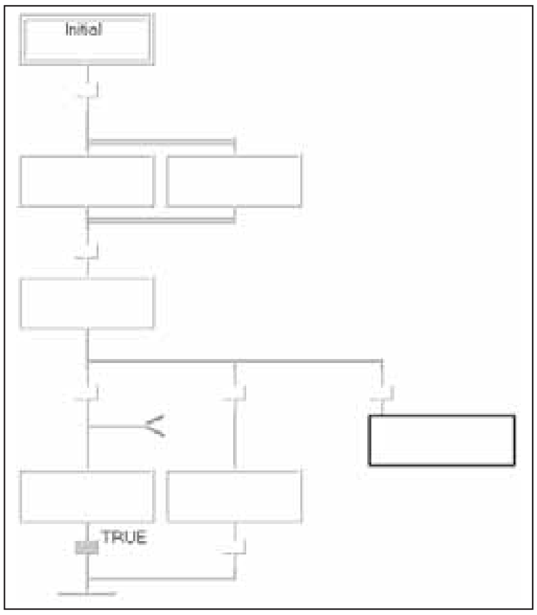


Fig. 6-61:
Selected position for new item

⑳ Click on the tool icon

Inserts a jump exit point.

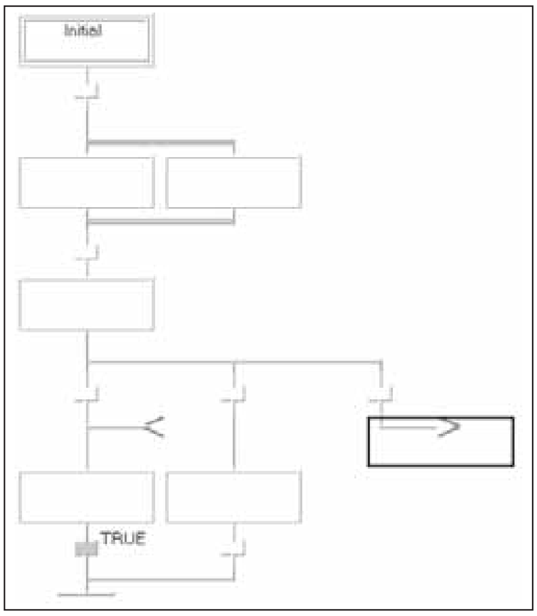


Fig. 6-62:
Inserted jump exit point

Step 5: Assign names to the steps and the jump exit/entry labels.

① Click on the element you want to assign a name to. Activates editing mode.

② Enter the name (Example: "Step_1" through "Step_4" and "Jump").

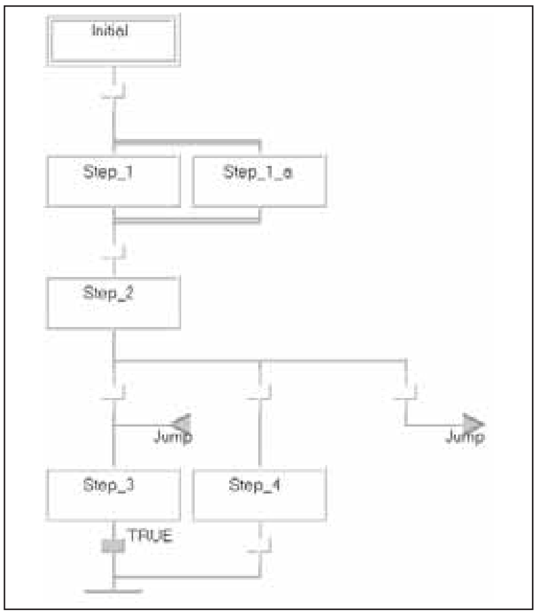


Fig. 6-63:
Assigned names to the steps and jump exit/entry labels

Step 6: Assign transition conditions to the transitions

NOTE You can use transition programs, TRUE/FALSE and Boolean variables (referenced by direct address or name) as transition conditions.

Assigning and creating a transition program

① Click on the transition to which you wish to assign a program.
Activates editing mode.

② Enter the program name (Example: "TRAN_1").

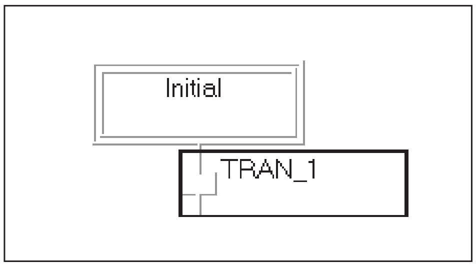


Fig. 6-64:
Entered program name

③ Click on the tool icon

The New Transition dialogue box is displayed, with the program name you just entered.

④ Select the programming language (Example: ladder diagram).

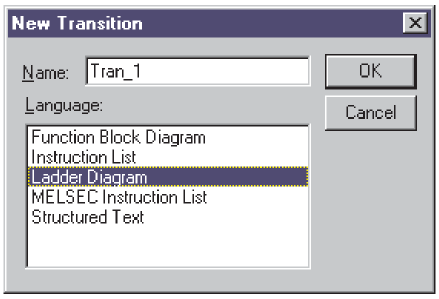


Fig. 6-65:
Programming language selection for transition

⑤ Click on **OK**.
The body of the transition program is displayed automatically.

⑥ Write the transition program.

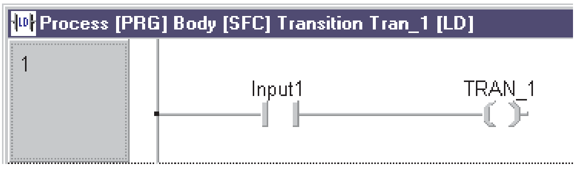


Fig. 6-66:
Transition program

Assigning an existing variable to a transition

① Double-click on the transition to which you wish to assign a variable.
Activates editing mode.

② Enter the name of an existing variable (Example: "Input2" to "Input7").

NOTE This overwrites the "TRUE" transition condition.

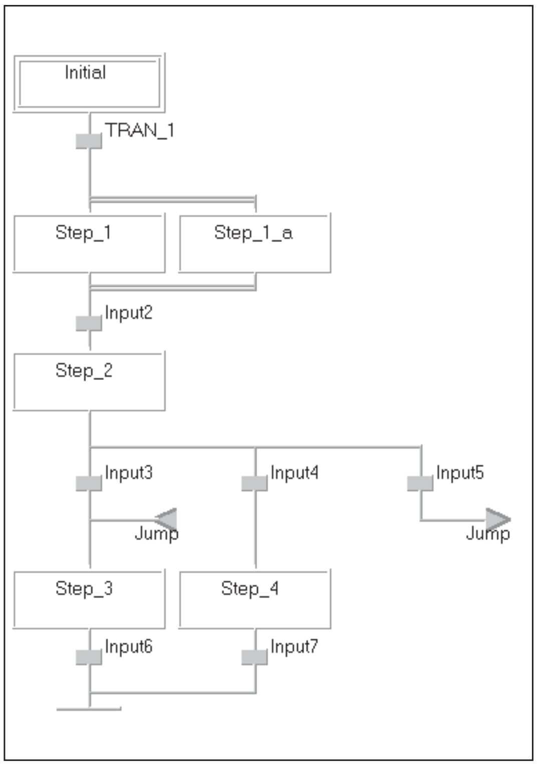


Fig. 6-67:
Existing variables assigned to transitions

Step 7: Create the actions

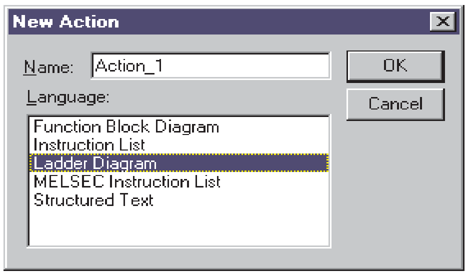
① Select the "Process" POU in the Project Navigator.

② Click on the tool icon

ACT

Displays the New Action dialogue box.

③ Enter a name for the action (Example: "Action_1") and select the programming language (Example: ladder diagram).



New Action

Name: Action_1

Language:

- Function Block Diagram
- Instruction List
- Ladder Diagram
- MELSEC Instruction List
- Structured Text

Fig. 6-68:
Programming language selection for action

The new action is displayed in the Action Pool in the Project Navigator window.

④ Double-click on "Action_1" to open the program editor.

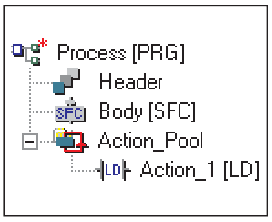


Fig. 6-69:
New action in the Action Pool in the Project Navigator

⑤ Enter the program.

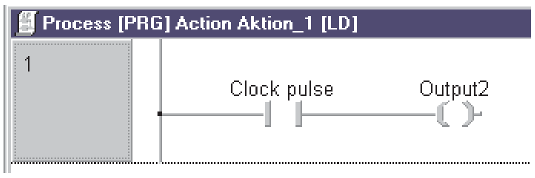


Fig. 6-70:
Action program

NOTE

Transition and action programs are written in exactly the same way as any other POU. You can write these programs using Instruction List, ladder diagram or function block diagram language. The Sequential Function Chart language itself is not supported for these programs, however.

Step 8: Assign actions or Boolean variables to the steps

① Select the step to which you wish to assign an action or variable (Example: "Step_1").

② Click on the tool icon

This displays the Action Association dialogue box, which is still empty.

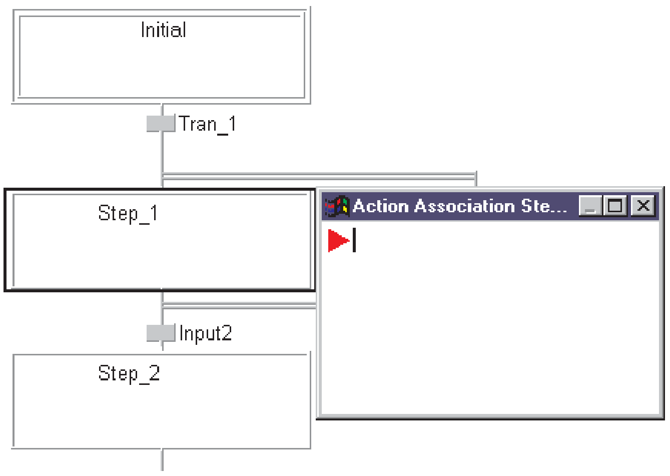
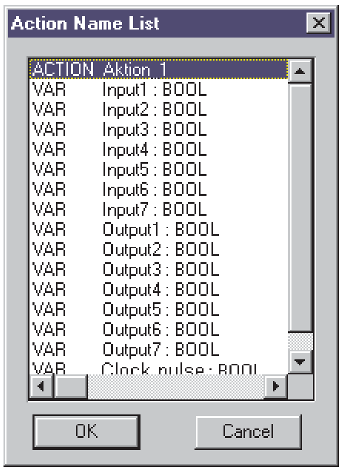


Fig. 6-71:
Opened Action Association dialogue box

③ Press the F2 key.
This displays the Action Name List box showing the actions and Boolean variables that are currently available.

④ Select the appropriate action/variable (Example:
"Initial" = "Output1"
"Step_1" = "Action_1"
"Step_1_a" = "Output3"
"Step_2" = "Output4"
"Step_3" = "Output5"
"Step_4" = "Output6"


Action Name List

```
ACTION  Aktion_1
VAR     Input1 : BOOL
VAR     Input2 : BOOL
VAR     Input3 : BOOL
VAR     Input4 : BOOL
VAR     Input5 : BOOL
VAR     Input6 : BOOL
VAR     Input7 : BOOL
VAR     Output1 : BOOL
VAR     Output2 : BOOL
VAR     Output3 : BOOL
VAR     Output4 : BOOL
VAR     Output5 : BOOL
VAR     Output6 : BOOL
VAR     Output7 : BOOL
VAR     Clock_pulse : BOOL
```

Fig. 6-72:
Action Name List

⑤ Close the Action Name List box by double-clicking on the control menu button. The step will now be displayed with a grey fill colour.

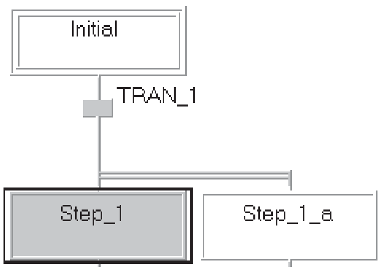


Fig. 6-73:
Fill colour indicates assigned actions or Boolean variables to the steps

6.8 Step 7: Checking PLC Programs (syntax check)

How to check your program for syntax errors

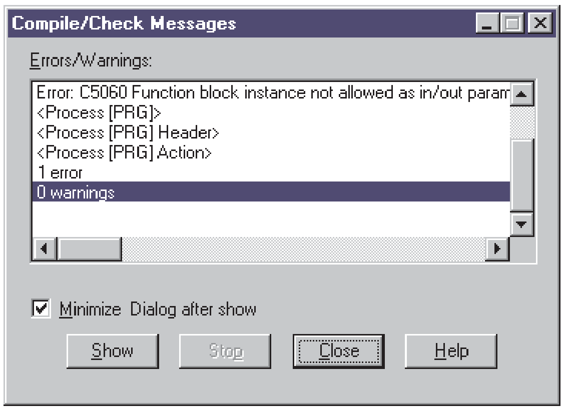
① Select the object to be checked in the Project Navigator window.

② Select **Check** in the **Object** menu

or

Click on the Syntax Check tool icon in the toolbar:

If the syntax check program finds any errors they are displayed and explained in the Compile/Check Messages box.



```
Error: C5060 Function block instance not allowed as in/out param
<Process [PRG]>
<Process [PRG] Header>
<Process [PRG] Action>
1 error
0 warnings
```

Fig. 6-74:
Errors are displayed and explained in the Compile/Check Messages box

③ You can display the source of any errors found automatically: Double-click on the corresponding error message in the Errors/Warnings list, or select the message and click on the **Show** button. This calls the object containing the error, with the source of the error highlighted in red.

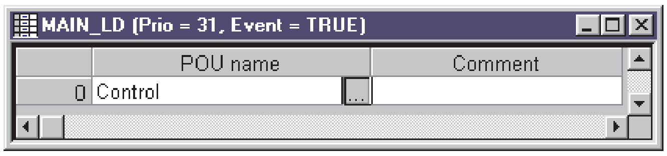
NOTE You can perform syntax checks both on individual objects and the entire project as a whole. You can also perform separate checks on the header and body of a single POU. Simply select the object to be checked in the Project Navigator (to check the entire project select the Project entry at the top of the tree).

6.9 Step 8: Configuring Tasks

In this section it is assumed that you have already created the tasks for your project (see Step2). Their entries are displayed in the Task Pool in the Project Navigator tree. Before you can use them in the program you must first specify the POUs you want to use in the tasks and configure the task attributes.

How to assign PRG type POUs to tasks

① Double-click on the task's entry in the Task Pool. A table is displayed on the right of the screen in which you can then assemble the task by specifying the POUs it is to contain.



MAIN_LD (Prio = 31, Event = TRUE)

	POU name	Comment
0	Control	

Fig. 6-75:
The task configuration table

② Click on the pop-up list icon in the first table line and select the POU you want to add to the task in the dialogue box displayed. Confirm your selection with **OK**. Only POUs defined as programs (PRG) are included in the dialogue box list. The name of the selected POU will then appear in the POU Name column in the first line of the table.

NOTE Only POUs that have not yet been included in a previously stored task are included in the dialogue box list.

Using the Tab key, move the cursor to the next cell of the table and enter a comment for the POU entry in the Comment column (optional).

Repeat step 2 for each additional POU you wish to use in the task. To insert a new table line for the next POU entry, select **New Declaration** in the **Edit** menu, then select the position at which the line is to be inserted in the submenu.

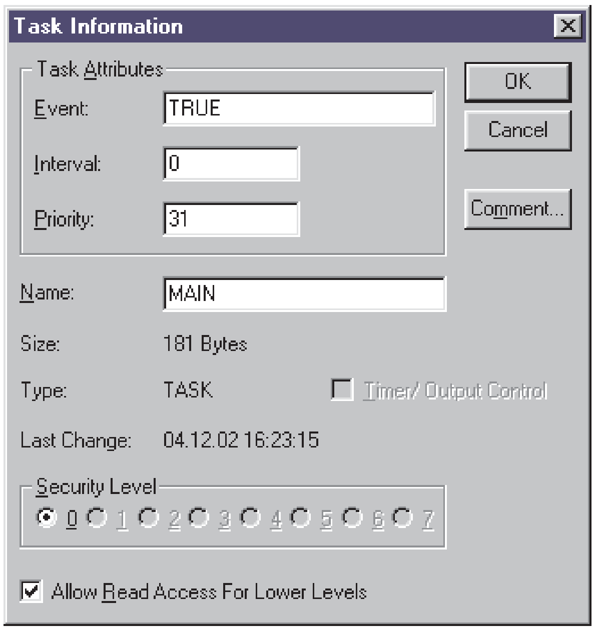
When the cursor is on a comment cell that is currently in edit mode (white background) you can insert a new line at the end of the table automatically by pressing Tab.

How to configure the task execution attributes

① Select the task to be configured in the Project Navigator window or in the task configuration table (select the grey number button in the first table column).

② Press Alt Enter to open the **Task Information** dialogue box.

The parameters in this dialogue box set the execution conditions and the security level for the current task. Tasks can be either event-triggered or interval-triggered. Full details on the various execution options can be found in the Reference Manual.


Task Information

Task Attributes

Event: TRUE

Interval: 0

Priority: 31

Name: MAIN

Size: 181 Bytes

Type: TASK

☐ Timer/ Output Control

Last Change: 04.12.02 16:23:15

Security Level

(● 0) ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7

☑ Allow Read Access For Lower Levels

Fig. 6-76:
Event = TRUE
(execute always)

Interval = 0
(because event-triggered)

Priority = 0
(maximum priority)

Priority = 31
(lowest priority)

The dialogue box also shows the current size of the task and the date and time of the last editing change.

NOTE

Please refer to the Reference Manual for details on configuration of the read/write access parameters.

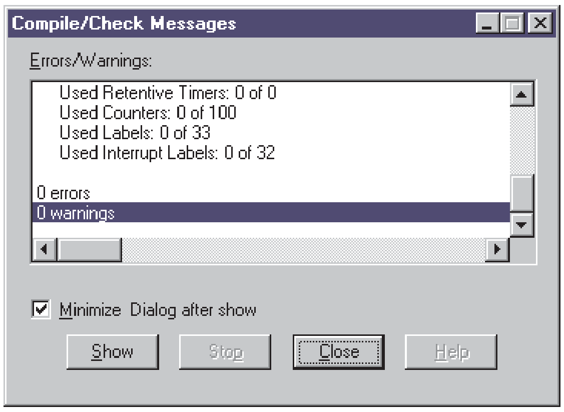
6.10 Step 9: Compiling Projects

When you compile a project the system translates the program code into executable form to prepare it for downloading to the controller CPU.

How to compile a project

① Select **Rebuild all** in the **Project** menu.

The progress of the compilation process and any errors found are documented in a status window.



```
    Used Retentive Timers: 0 of 0
    Used Counters: 0 of 100
    Used Labels: 0 of 33
    Used Interrupt Labels: 0 of 32

0 errors
0 warnings
```

Fig. 6-77:
Errors are displayed and explained in the Compile/Check Messages box

WARNING:
Compilation does not download the program code to the CPU, this must be done separately!
Always perform a syntax check on the entire project before attempting to compile it (Step 7).

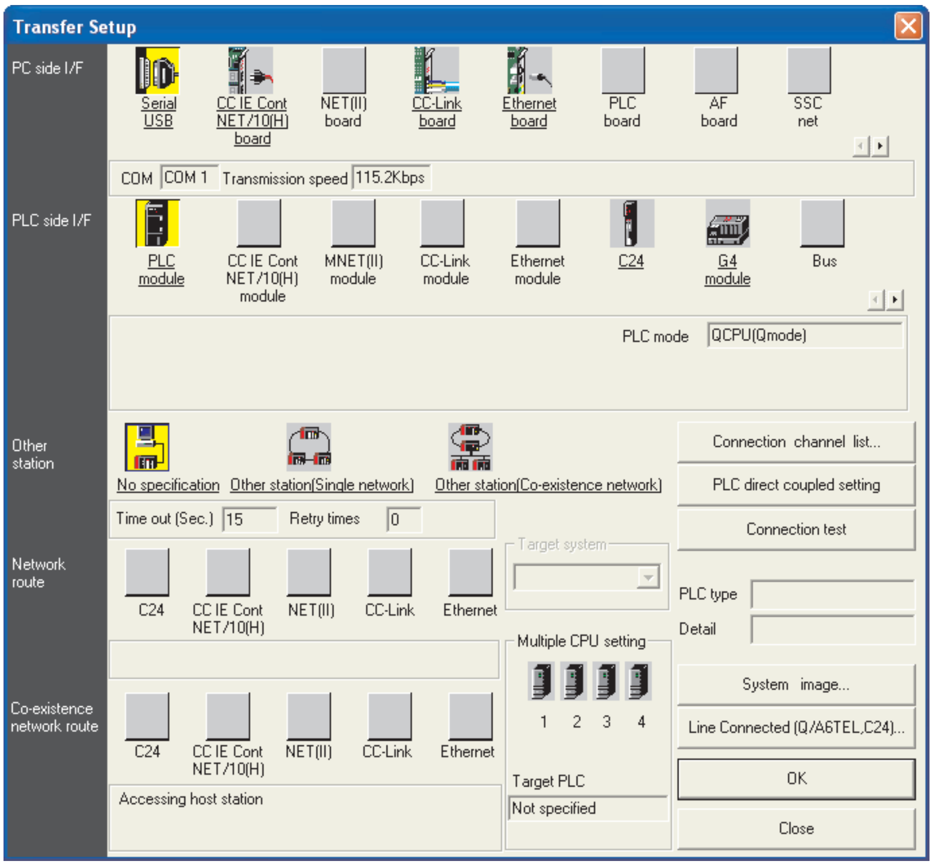
6.11 Step 10: Communications Port Setup

Before you can download a project to the PLC you must first configure the communications port you are going to use for this purpose.

NOTE Before you begin, make sure that you know precisely which physical interface on your personal computer is going to be used for transferring the data to the PLC system.

How to select and configure the communications port

① In the **Online** menu select **Transfer Setup**, then select **Ports**.
The **Connection Setup** dialogue box appears.



COM COM 1 Transmission speed 115.2Kbps

PLC mode QCPU(Qmode)

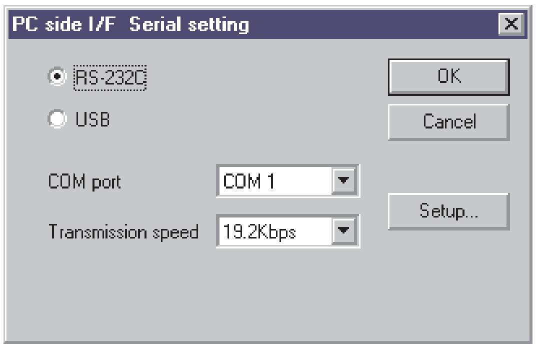
Time out (Sec.) 15 Retry times 0

Accessing host station

Not specified

Fig. 6-78: Interface selection in the Connection Setup

② Under **PLC side I/F** click on the button **Seriell/USB**.



(●) RS-232C
( ) USB

COM port COM 1

Transmission speed 19.2Kbps

Fig. 6-79: PC side I/F Serial setting

③ Select either USB or RS-232C. The COM-ports COM1–COM10 can be selected.

④ Confirm the entries in the dialogue boxes with **OK**.

6.12 Step 11: Downloading Programs (to PLC)

When your program project is complete and has been checked for errors and successfully compiled you can download it to the controller system for execution.

Connecting the PLC system

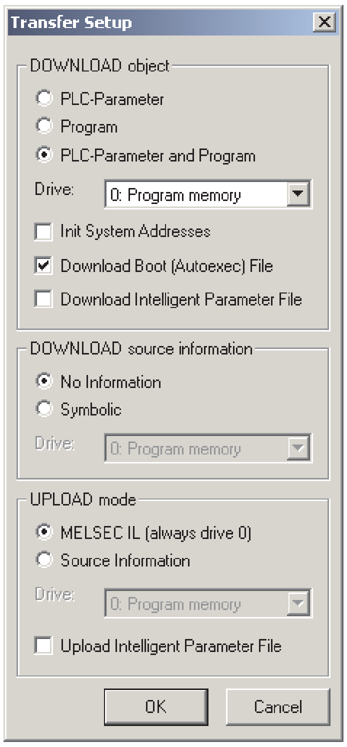
① Connect the PLC to your personal computer.

② Make sure that you plug the connection cable into the same port on the computer that you defined in the settings described in Step 10 above.

NOTE Please refer to the Reference Manual for details on the various different options available for connecting the PC and PLC systems.

How to download a program to the PLC

① Select **Transfer Setup** in the **Online** menu, then select **Project**.
The **Download Setup** dialogue box is displayed on the screen.


Transfer Setup

DOWNLOAD object
- ( ) PLC-Parameter
- ( ) Program
- (●) PLC-Parameter and Program

Drive: 0: Program memory

- [ ] Init System Addresses
- [✓] Download Boot (Autoexec) File
- [ ] Download Intelligent Parameter File

DOWNLOAD source information
- (●) No Information
- ( ) Symbolic

Drive: 0: Program memory

UPLOAD mode
- (●) MELSEC IL (always drive 0)
- ( ) Source Information

Drive: 0: Program memory

- [ ] Upload Intelligent Parameter File

OK Cancel

Fig. 6-80:
*The **Transfer Setup** dialogue box options are used to specify which data are downloaded to the PLC.*

② Click on **PLC Parameter and Program**, then confirm with OK.

WARNING:
You must always download the PLC parameter when you transfer a program to the PLC for the first time! Q/QnA series PLCs must be formatted first.

③ Select **Transfer** in the **Project** menu, then select **Download to PLC** to start the download. The transfer process is documented in a list box; if no error messages are displayed the transfer has been completed successfully.

6.13 Step 12: Monitoring Programs

In monitoring mode, GX IEC Developer can display the current status and changes of the variables/devices used by your program.

NOTE You can only monitor error-free programs that have been compiled and downloaded to the PLC system for execution.

① Select **Monitoring Mode** in the **Online** menu. A check mark ✓ in front of the option in the menu indicates that the mode is currently active, and the entries in the Project Navigator window switch to light grey.

② Open the body of the POU that you wish to monitor.

③ Select **Start Monitoring** in the **Online** menu.

NOTE The PLC must be in RUN mode for monitoring to be possible.

The following examples illustrate how the status changes of the variables are displayed in monitoring mode for the various programming languages supported by GX IEC Developer.

NOTE More detailed information on the display options and other monitoring mode features can be found in the Reference Manual (Chapter 8).

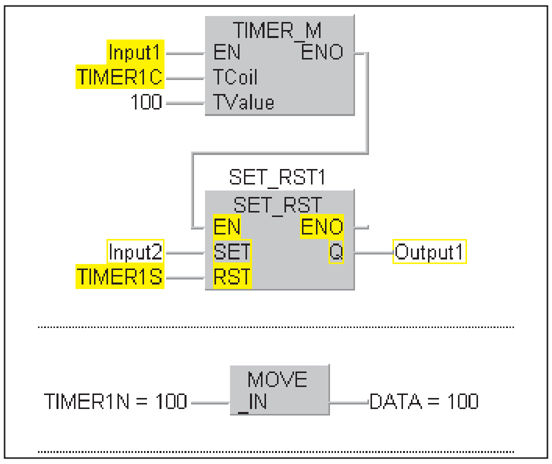


Fig. 6-81:
Function block diagram
Filled rectangle: Binary ON
Rectangle: Binary OFF
DATA = 40: 40 seconds have elapsed

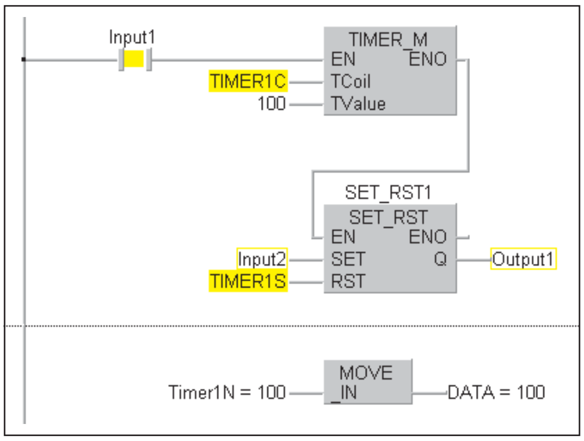
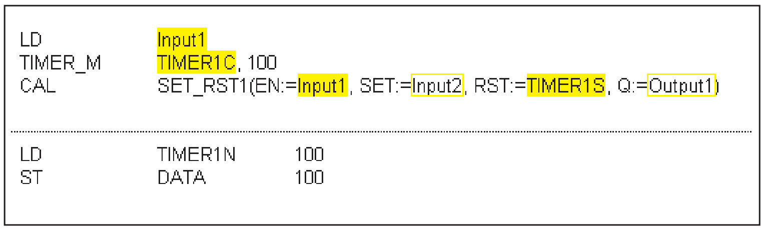


Fig. 6-82:
Ladder diagram
Filled field between the input contacts: ON
Filled rectangle: Binary ON
Rectangle: Binary OFF
DATA =100: 100 seconds have elapsed



```
LD          Input1
TIMER_M     TIMER1C, 100
CAL         SET_RST1(EN:=Input1, SET:=Input2, RST:=TIMER1S, Q:=Output1)

LD          TIMER1N       100
ST          DATA          100
```

Fig. 6-83:
Instruction list
ON/OFF status is indicated by filled /not filled rectangles.

6.14 Step 13: Uploading Data from the CPU

How to upload data from the PLC's CPU to GX IEC Developer

① Select **Transfer** in the **Project** menu, then select the **Upload from PLC** option.

② This displays the PLC Parameter dialogue box. Select the appropriate **CPU Type** and confirm with **OK** (see Step 1).

③ In the next dialogue box GX IEC Developer asks you to specify the path and name for the uploaded project data, which will be stored as a new project.
If you want to create a new project for the upload follow the instructions in Step 1.
If a project is already open you can abort the procedure by clicking on the **Cancel** button.

④ Click on Setup in the **Transfer Setup** dialogue box.

⑤ This displays the **Transfer Setup (CPU port)** dialogue box. Select the correct port for your system configuration (see Step 10).

⑥ Confirm your entries in both dialogue boxes with **OK**.
This starts the upload procedure. Progress and any errors are documented in a list box.

7 Sample Program: CarPark

NOTE This sample program is only intended as an illustration of programming and program structure techniques in GX IEC Developer. In its present version it cannot be used as a basis for producing your own executable programs. The sample version has been written for a MELSEC FX series controller.

Description

The roll-up door of a car park building can be opened from inside and outside with a key-operated switch. Safety functions included in the program ensure that the door opens automatically in the event of an alarm, and that it does not remain open for too long when no cars are entering or leaving the car park. The program also keeps track of the number of cars in the building.

7.1 Project Structure

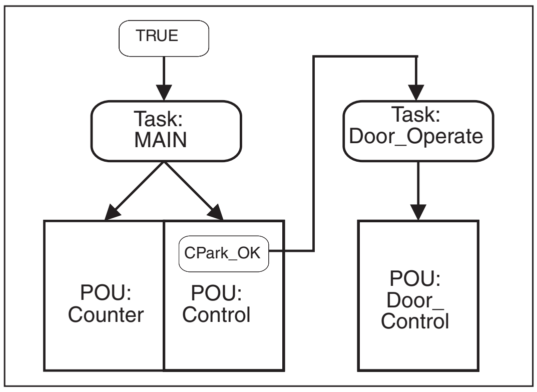


Fig. 7-1:
Project structure

7.1.1 The Task "Main"

... always runs in the background, with maximum priority. This task contains the POUs "Control" and "Counter", which perform the following functions:

POU "Control"

– Car park status check
– Close car park door if no car drives in or out within a 60-second period
– Open car park door when an alarm is triggered

POU "Counter"

– Count the cars

7.1.2 The Task "Door_Operate"

... is event-triggered. It is activated when the OK signal for the car park door (variable: CPark_OK) is set. This task contains the POU "Door_Control", which handles the following functions:

POU "Door_Control"

– Open car park door when the key switches inside and outside the car park are operated.
– Open car park door when an alarm is triggered

NOTES

GX IEC Developer allows you to apply an engineering design approach to project planning and programming. This is illustrated in the "CarPark" project. Steps S1 through S11 are fully documented.

In the sample program all the variables are already known and declared at the outset. Of course, this ideal situation is not always possible in actual projects; one often has to make corrections and add and delete variables in the course of the programming work. This flexible approach is fully supported in GX IEC Developer; the system allows you to edit, add and delete variable declarations at any time, both during programming and afterwards.

7.2 Create the new "CarPark" project (Step 1 in Chapter 6)

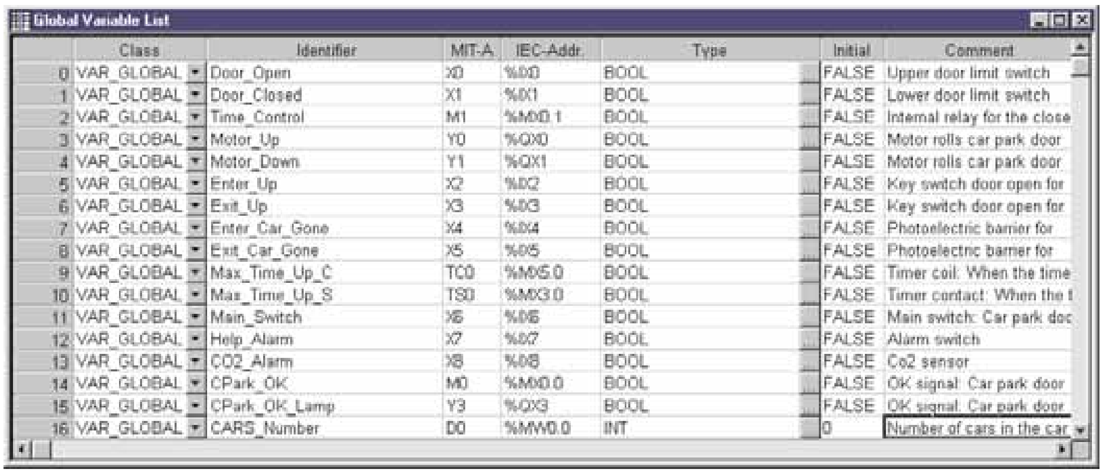
The first step is to create a new project. Refer to the instructions in S1 and enter "CarPark" as the project name in step ④.

7.3 Create the tasks (Step 2 in Chapter 6)

Create the "Main" and "Door_Operate" tasks.

7.4 Declare the global variables (Step 3 in Chapter 6)

Declare the global variables shown in the table below. The entries in the Comment column are optional.



Global Variable List

	Class	Identifier	MIT-A	IEC-Addr.	Type	Initial	Comment
0	VAR_GLOBAL	Door_Open	X0	%IX0	BOOL	FALSE	Upper door limit switch
1	VAR_GLOBAL	Door_Closed	X1	%IX1	BOOL	FALSE	Lower door limit switch
2	VAR_GLOBAL	Time_Control	M1	%MX0.1	BOOL	FALSE	Internal relay for the close
3	VAR_GLOBAL	Motor_Up	Y0	%QX0	BOOL	FALSE	Motor rolls car park door
4	VAR_GLOBAL	Motor_Down	Y1	%QX1	BOOL	FALSE	Motor rolls car park door
5	VAR_GLOBAL	Enter_Up	X2	%IX2	BOOL	FALSE	Key switch door open for
6	VAR_GLOBAL	Exit_Up	X3	%IX3	BOOL	FALSE	Key switch door open for
7	VAR_GLOBAL	Enter_Car_Gone	X4	%IX4	BOOL	FALSE	Photoelectric barrier for
8	VAR_GLOBAL	Exit_Car_Gone	X5	%IX5	BOOL	FALSE	Photoelectric barrier for
9	VAR_GLOBAL	Max_Time_Up_C	TC0	%MX5.0	BOOL	FALSE	Timer coil: When the time
10	VAR_GLOBAL	Max_Time_Up_S	TS0	%MX3.0	BOOL	FALSE	Timer contact: When the t
11	VAR_GLOBAL	Main_Switch	X6	%IX6	BOOL	FALSE	Main switch: Car park doc
12	VAR_GLOBAL	Help_Alarm	X7	%IX7	BOOL	FALSE	Alarm switch
13	VAR_GLOBAL	CO2_Alarm	X8	%IX8	BOOL	FALSE	Co2 sensor
14	VAR_GLOBAL	CPark_OK	M0	%MX0.0	BOOL	FALSE	OK signal: Car park door
15	VAR_GLOBAL	CPark_OK_Lamp	Y3	%QX3	BOOL	FALSE	OK signal: Car park door
16	VAR_GLOBAL	CARS_Number	D0	%MW0.0	INT	0	Number of cars in the car

Fig. 7-2: *Global Variable List*

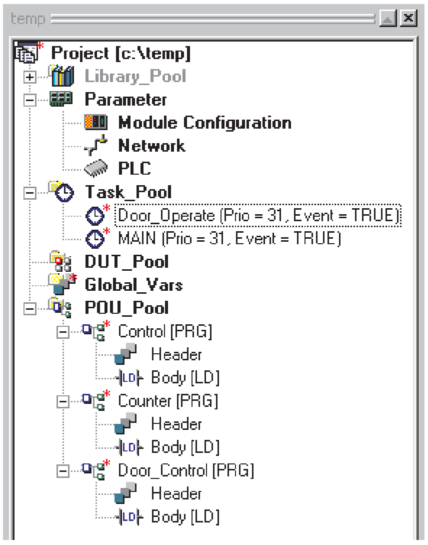
7.5 Create the program organisation units (Step 4 in Chapter 6)

Create the three program organisation units: "Control", "Counter" and "Door_Control". Define all three POUs as programs (PRG) and specify ladder diagram (LD) as the programming language.

Each POU consists of a header and a body. The header contains the declarations of the variables used by the POU, the body contains the actual PLC program code.

7.5.1 Project Navigator Window

All the tasks and POUs you create are automatically displayed in the Project Navigator window.



```
Project [c:\temp]
  Library_Pool
  Parameter
    Module Configuration
    Network
    PLC
  Task_Pool
    Door_Operate (Prio = 31, Event = TRUE)
    MAIN (Prio = 31, Event = TRUE)
  DUT_Pool
  Global_Vars
  POU_Pool
    Control [PRG]
      Header
      Body [LD]
    Counter [PRG]
      Header
      Body [LD]
    Door_Control [PRG]
      Header
      Body [LD]
```

Fig. 7-3
Tasks:
- *"Door_Operate"*
- *"Main"*

POUs:
- *"Control"*
- *"Counter"*
- *"Door_Control"*

7.6 Program the bodies (Step 6 in Chapter 6)

7.6.1 Body of the "Control" POU

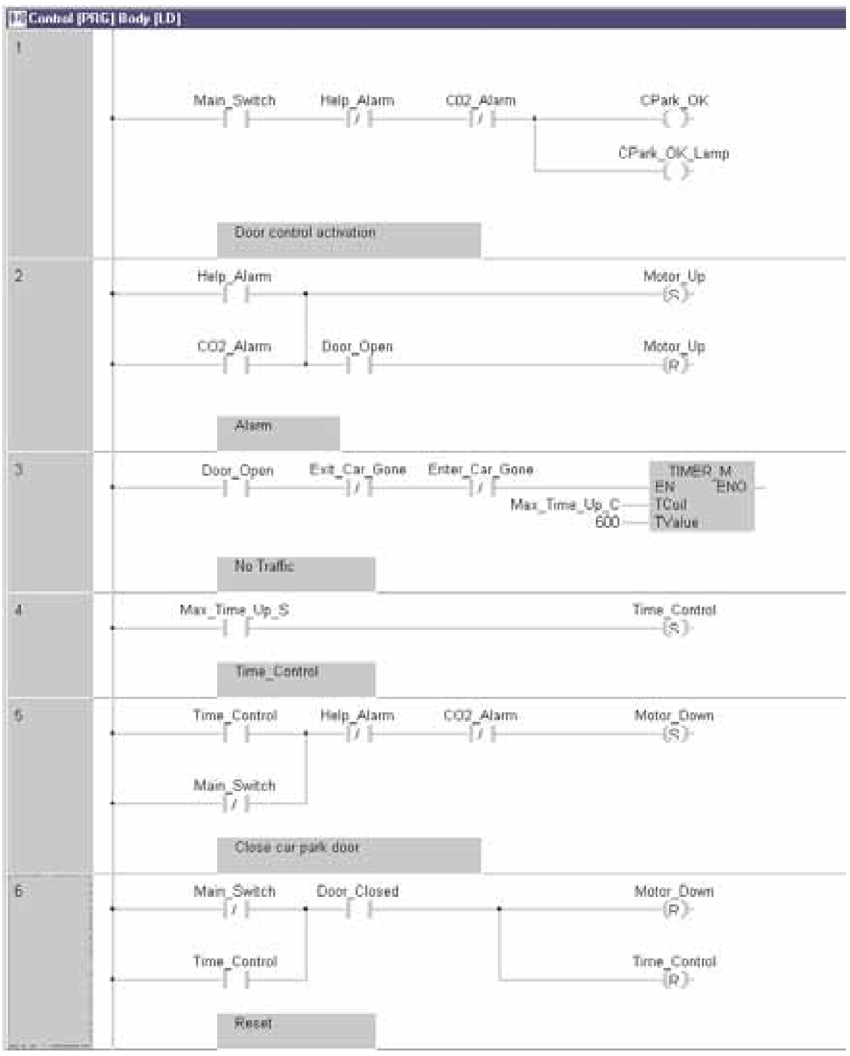


Fig. 7-4:
Door control activation

Help call and CO2-alarm

Timer activation

Time control

Close car park door

Reset

Door control activation

When the main switch is on and no help call or CO2 alarm is registered the OK signal (variable: "CPark_OK") for the "Door_Control" program organisation unit is set and the CPark_OK_Lamp is switched on.

Help call and CO2 alarm

As soon as an alarm is registered the motor rolls the car park door up. The motor is reset when the door activates the upper limit switch ("Door_Open").

Timer activation

When the door is open and the photoelectric barriers at the entrance ("Enter_Car_Gone") and the exit ("Exit_Car_Gone") do not register any vehicles the timer "Max_Time_Up_C" starts to count for 60 seconds.

Time control

The "Time_Control" relay is set as soon as the 60-second period has elapsed.

Close car park door

When no traffic is registered or the main switch is turned off and no alarm is registered the motor rolls the car park door down into the closed position.

Reset

When no traffic is registered or the main switch is turned off and the door reaches the lower limit switch ("Door_Closed") both the motor and the "Time_Control" relay are reset.

7.6.2 Body of the "Counter" POU

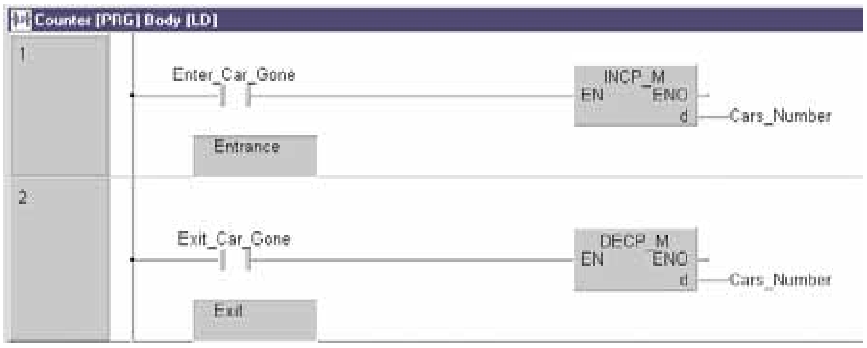


Fig. 7-5:
Entrance

Exit

Entrance

The program counts the cars driving into the car park by incrementing the total number stored in the "Cars_Number" data register every time a car enters.

Exit

Every time a car drives out of the building the program decrements the number stored in the data register. The result is that the number always corresponds to the exact number of cars in the building.

7.6.3 Body of the "Door_Control" POU

Conditions for activation of the door control routine

The Door_Control POU can only be executed when the "CPark_OK" variable in the Control POU is set. "CPark_OK" is only set if

- The main switch is on, *and*
- No help call alarm is registered, and
- No CO2 alarm is registered.

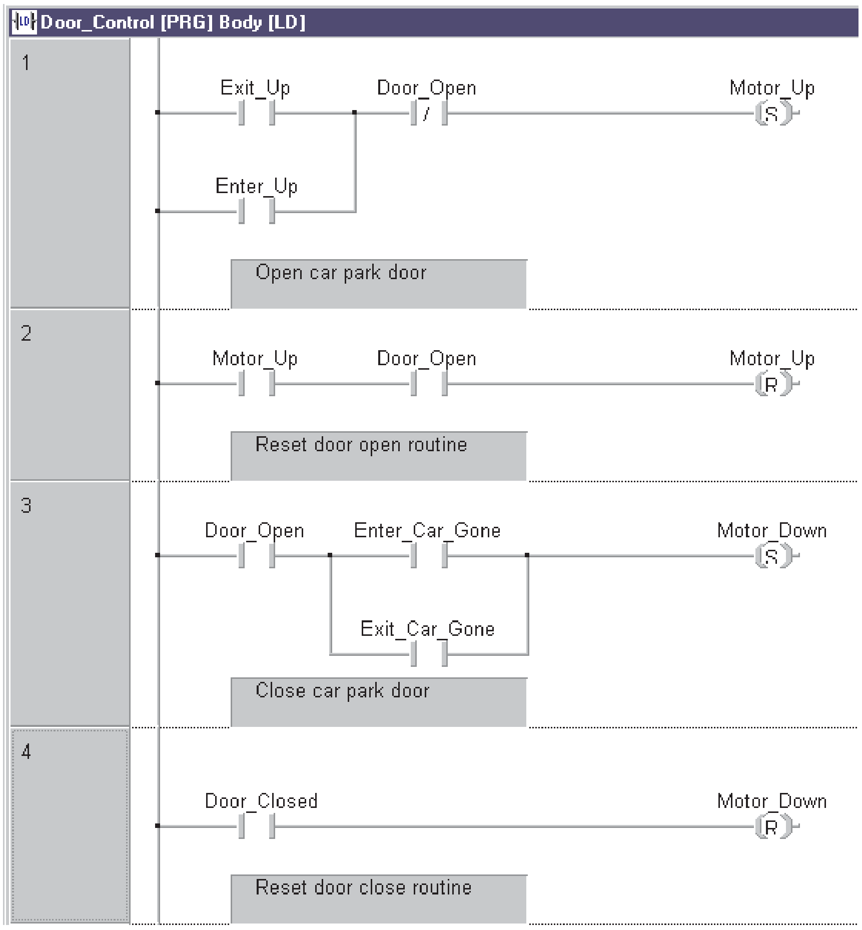


Fig. 7-6

Open car park door

Reset

Close car park door

Reset

Open car park door

When the car park door is closed the key switch inside ("Exit_Up") or outside ("Enter_Up") the building must be operated to open the door.

Reset

The motor is reset when the car park door reaches the upper limit switch ("Door_Open").

Close car park door

When a car passes through the photoelectric barrier after driving in ("Enter_Car_Gone") or out ("Exit_Car_Gone") of the building the motor starts to close the car park door.

Reset

When the door reaches the lower limit switch ("Door_Closed") the motor is reset.

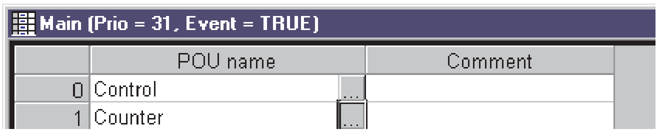
7.7 Configure the tasks (Step 8 in Chapter 6)

We have already created the two tasks needed by the program, "Main" and "Door_Operate" (see Step 2).

The next step is to assign the POUs to the tasks, which are still "empty". Double-click on the task name in the Project Navigator, then select the pop-up arrow icon in the cell in the POU Name column, and select the POU from the list displayed.

After assigning the POUs you must then configure the task attributes. Select the task in the Project Navigator window or open the task configuration table by double-clicking on its name, then press Alt Enter or select **Information** in the **Object** menu to open the Task Information dialogue box.

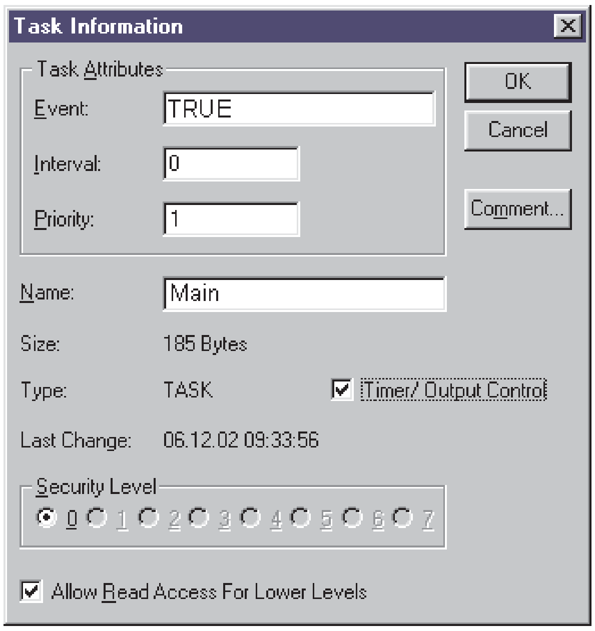
7.7.1 The "Main" task



Main (Prio = 31, Event = TRUE)

	POU name	Comment
0	Control	
1	Counter	

Fig. 7-7:
Assign the POUs "Control" and "Counter" to the "Main" task.



Task Information

Task Attributes
Event: TRUE
Interval: 0
Priority: 1

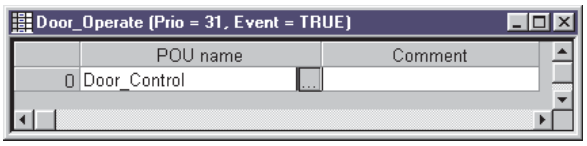
Name: Main
Size: 185 Bytes
Type: TASK [✓ Timer/ Output Control]
Last Change: 06.12.02 09:33:56

Security Level: [✓ 0] 1 2 3 4 5 6 7

[✓ Allow Read Access For Lower Levels]

Fig. 7-8:
Attributes of the "Main" task
Event: TRUE
... i.e. the task's two POUs "Control" and "Counter" both run continuously.
Priority: 1

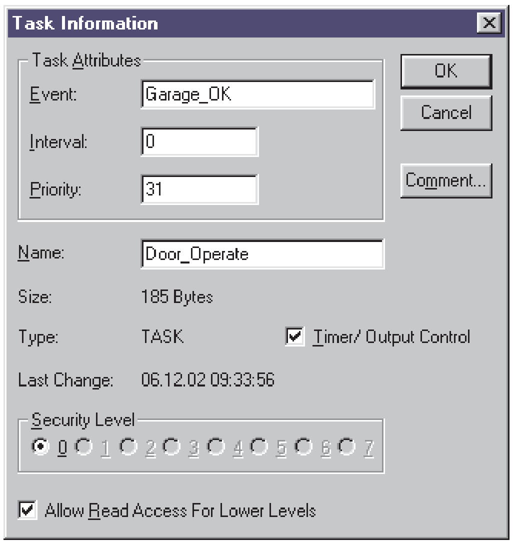
7.7.2 The "Door_Operate" Task



Door_Operate (Prio = 31, Event = TRUE)

	POU name	Comment
0	Door_Control	

Fig. 7-9:
The "Door_Operate" task contains the POU "Door_Control".



Task Information

Task Attributes
Event: Garage_OK
Interval: 0
Priority: 31

Name: Door_Operate
Size: 185 Bytes
Type: TASK [✓ Timer/ Output Control]
Last Change: 06.12.02 09:33:56

Security Level
[✓ 0] 1 2 3 4 5 6 7

[✓ Allow Read Access For Lower Levels]

Fig. 7-10:
Attributes of the "Door_Operate" task
Event: "CPark_OK"
... i.e. the associated POU Door_Control is only activated when the "CPark_OK" signal is set.
Priority: 31

NOTE Entry of the project data is now complete.

① Compile the project (Step 9),

② configure the ports of your personal computer (Step 10) and

③ download the program to the controller CPU (Step 11).

④ Monitoring mode for following the status of the program variables is explained in Step 12.

8 Importing

There are two different ways to import projects created with the older MELSEC MEDOC programming package for use in GX IEC Developer:

- Import by loading a MELSEC MEDOC print file
- Import by uploading directly from the CPU

Loading a print file to GX IEC Developer

Procedures in MELSEC MEDOC

① Select a file name as the printer port. The extension `TMP` is added automatically by the program.

② Make sure that only Instruction List and Name List are selected in the program listing options. The Header must be switched off!

③ Start the print procedure.

Procedures in GX IEC Developer

④ Open the body of an existing MELSEC Instruction List program or create a new POU and specify MELSEC Instruction List as the language. **Important:** Make very sure that the POU is declared as a program (PRG).

⑤ In a network click on the left field reading MELSEC.

⑥ Open the POU body, then select **Import MEDOC Network** in the **Tools** menu.

⑦ This opens a file selection box. Select the drive and directory, and then select the print file (`TMP`) that you want to load and confirm your choice with **OK**. This opens another dialogue box.

⑧ Confirm the settings with **OK**
(MEDOC Program = Instruction List only,
MEDOC Symbolic Names = Name list only).

⑨ If necessary, edit the system variables.

NOTES

The structure of MELSEC MEDOC programs can only be remained, if you select MELSEC mode in the Wizard before!

For further details refer to the Reference Manual

Index

GX IEC Developer Version 7

Beginner's Manual

MODEL	SW10D5C-MEDOC3-B-E
MODEL CODE	13JJ00
SH(NA)-080588ENG-C(0811)MEE	

MITSUBISHI ELECTRIC CORPORATION

HEAD OFFICE : TOKYO BUILDING, 2-7-3 MARUNOUCHI, CHIYODA-KU, TOKYO 100-8310, JAPAN
NAGOYA WORKS : 1-14 , YADA-MINAMI 5-CHOME , HIGASHI-KU, NAGOYA , JAPAN