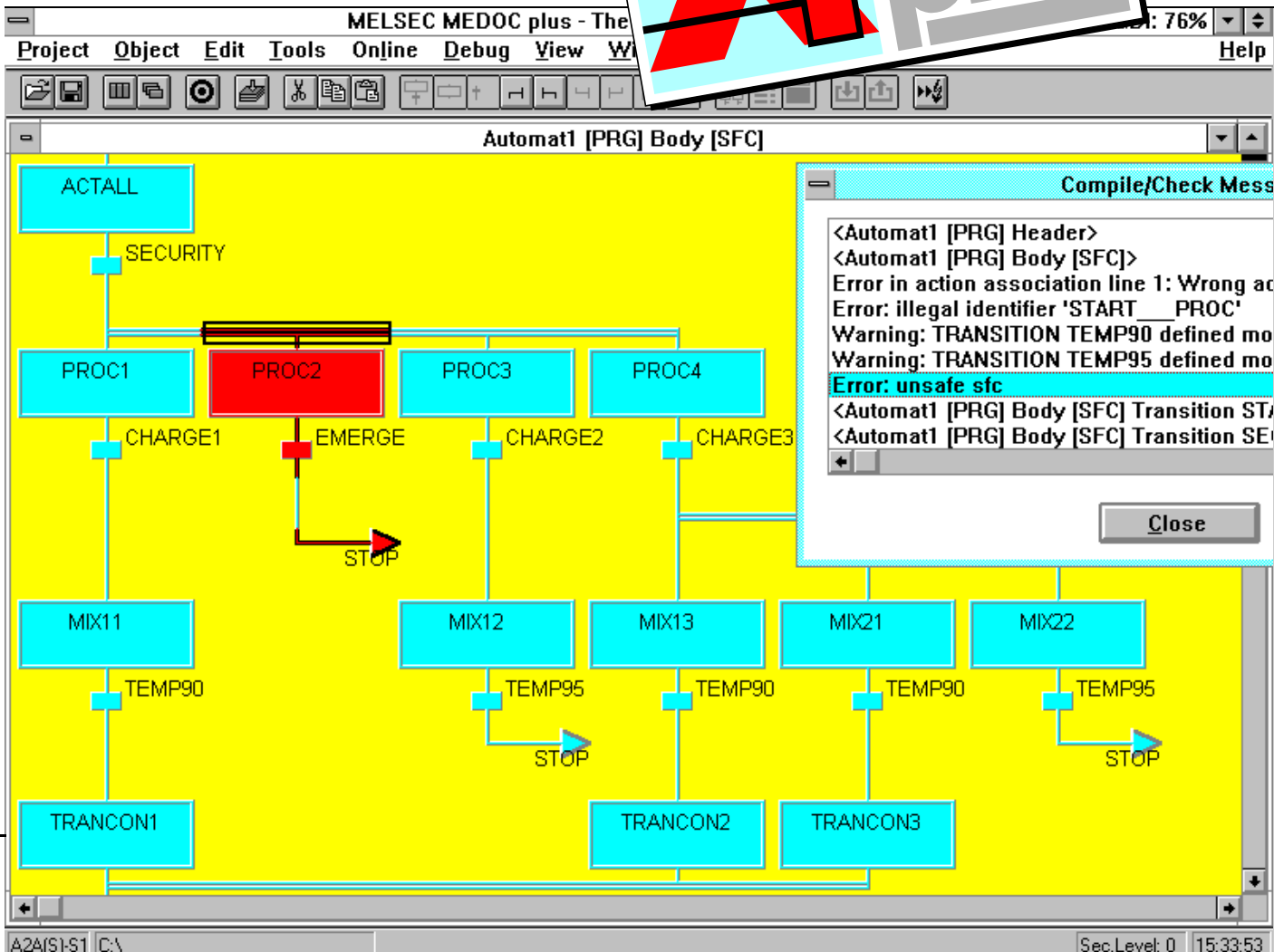
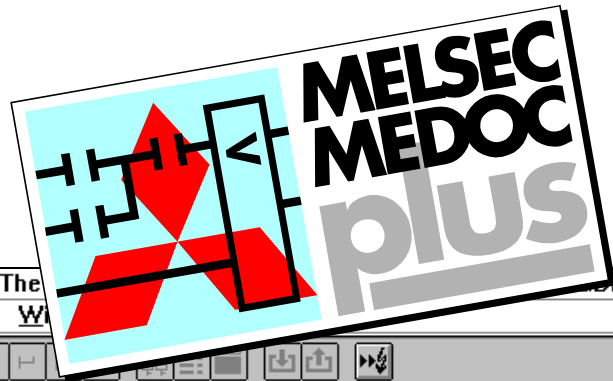


MELSEC MEDOC *plus*

IEC Programming and Documentation System

efesotomasyon.com

Beginner's Manual



**MELSEC MEDOC plus
IEC Programming and
Documentation System**

Beginner's Manual

About this Manual

The texts and illustrations in this manual are provided exclusively as a guide to the IEC programming and documentation system MELSEC MEDOC *plus*. Separate manuals are available for MITSUBISHI ELECTRIC's various series of MELSEC programmable logic controllers.

The MELSEC MEDOC *plus* software is supplied under a legal License Agreement and may only be used and copied subject to the terms of this License Agreement.

All rights reserved. Without the prior explicit consent in writing from MITSUBISHI ELECTRIC no part of this manual may be reproduced, stored in a retrieval system of any kind, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording or otherwise, or translated into another language.

MITSUBISHI ELECTRIC reserves the right to make changes to the technical specifications and information in this handbook at any time and without prior notice.

The IEC 1131.1 standard cited in this manual is available from the publishers Beuth Verlag in Berlin (Germany).

4th edition

Copyright January 1998

MITSUBISHI ELECTRIC EUROPE B.V.

Factory Automation

Gothaer Strasse 8

D - 40880 Ratingen

Germany

Phone: (+2102) 486-0

Fax: (+2102) 486-717

Printed in Germany

Article number: 43596-D (Beginner's Manual)

Chapter1:	Introduction	1 – 1
	This manual...	1 – 1
	The Reference Manual...	1 – 1
	If you're not yet familiar with MS Windows	1 – 1
	If you're not yet familiar with the IEC 1131.3 standard...	1 – 1
	If you already have IEC 1131.3 experience and want to get to work right away...	1 – 1
	If you get stuck...	1 – 2
	Typographic Conventions and other Tips	1 – 2
Chapter 2:	Getting to Know MELSEC MEDOC <i>plus</i>	2 – 1
	What's New in MELSEC MEDOC <i>plus</i> ?	2 – 1
	Introduction to the IEC 1131.3 Standard	2 – 2
Chapter 3:	Basic Terms Used in IEC 1131.3 Standard Programming . . .	3 – 1
	Projects	3 – 1
	Program Organisation Units (POUs)	3 – 2
	Programs, Function Blocks and Functions	3 – 3
	Parameters and Instancing	3 – 4
	Tasks	3 – 6
	Variables	3 – 7
	Programming Languages	3 – 10
	Networks	3 – 10
	The Text Editors	3 – 10
	The Graphical Editors	3 – 15
Chapter 4:	Installation	4 – 1
	Hardware Requirements	4 – 1
	Copyright	4 – 1
	Installing MELSEC MEDOC <i>plus</i>	4 – 1

Chapter 5: The User Interface 5 – 1

The Elements of the User Interface	5 – 1
The Menu Bar	5 – 2
The Toolbar	5 – 2
Windows	5 – 2
The Status Bar	5 – 2
The Project Navigator	5 – 2
Declaration Tables	5 – 3
The Editors	5 – 5

Chapter 6: Getting Started 6 – 1

S1 Creating New Projects	6 – 2
S2 Creating Tasks	6 – 4
S3 Declaring Global Variables	6 – 5
S4 Creating Program Organisation Units	6 – 7
S5 Programming POU Headers	6 – 8
S6 Programming POU Bodies	6 – 9
Programming Examples	6 – 10
Inputs and outputs in ladder diagram language (LD)	6 – 10
A Sum Function in FBD Language	6 – 12
I/O Signal Configuration Parameters	6 – 15
Timers in LD/FBD/IL	6 – 16
Sequential Function Chart Language	6 – 22
S7 Checking PLC Programs (syntax check)	6 – 34
S8 Configuring Tasks	6 – 35
S9 Compiling Projects	6 – 37
S10 Communications Port Setup	6 – 38
S11 Downloading Programs to the PLC	6 – 39
S12 Monitoring Programs	6 – 40
S13 Uploading Data from the PLC	6 – 42

Chapter 7: Sample Program: CarPark 7 – 1

Project Structure	7 – 1
The Task 'Main'	7 – 2
The Task 'Door_Operate'	7 – 2
Create the new 'CarPark' project	7 – 3
Create the tasks	7 – 3
Declare the global variables	7 – 3
Create the program organisation units	7 – 4
Project Navigator Window	7 – 4
Program the headers	7 – 5
Header of the 'Control' POU	7 – 5
Header of the 'Counter' POU	7 – 6
Header of the 'Door_Control' POU	7 – 6
Program the bodies	7 – 7
Body of the 'Control' POU	7 – 7
Body of the 'Counter' POU	7 – 9
Body of the 'Door_Control' POU	7 – 10
Configure the tasks	7 – 12
The 'Main' task	7 – 12
The 'Door_Operate' Task	7 – 13

Chapter 8: Importing 8 – 1**INDEX**

Introduction

This manual...

...is a compact guide to using MELSEC MEDOC *plus*, 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 1131.3 standard. The 'Getting Started' chapter provides a precise step-by-step description of how to use MELSEC MEDOC *plus*, 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.

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.

If you're 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.

If you're not yet familiar with the IEC 1131.3 standard...

... please do take the time to read the 'Introduction to the IEC 1131.3 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.

If you already have IEC 1131.3 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 MELSEC MEDOC *plus* operations, from creating a new project to downloading your finished program to the controller.

If you get stuck...

... don't despair, help is never far away! If you run up against seemingly insoluble problems, or if you have questions about MELSEC MEDOC *plus* or the connected programmable logic controller (PLC) configuration, please first refer to the manuals and documentation. Many answers and solutions can also be found directly in the MELSEC MEDOC *plus* context-sensitive online help system, which can always be accessed by pressing I. 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.

Typographic Conventions and other Tips

In the rest of this manual **MELSEC MEDOC *plus*** will normally be referred to with the abbreviation **MM+**.

Menu names, menu commands, submenu commands and dialog box options are printed in **boldface** type, e.g. the menu command **New** in the **Project** menu, or the **CPU Port** and **Computer Link (AJ71C24)** options in the **Transfer Setup** dialog box.



These two symbols are used to identify the separate instructions for mouse and keyboard users.



Examples: *This symbol identifies 'how-to' examples.*



Note: *This symbol draws your attention to notes, hints and valuable or useful information.*



WARNING: *Always pay particular attention to the warnings marked with this symbol. Failure to follow these instructions may lead to data loss, damage to your hardware or other serious problems.*

Getting to Know MELSEC MEDOC **plus**

What's New in MELSEC MEDOC **plus**?

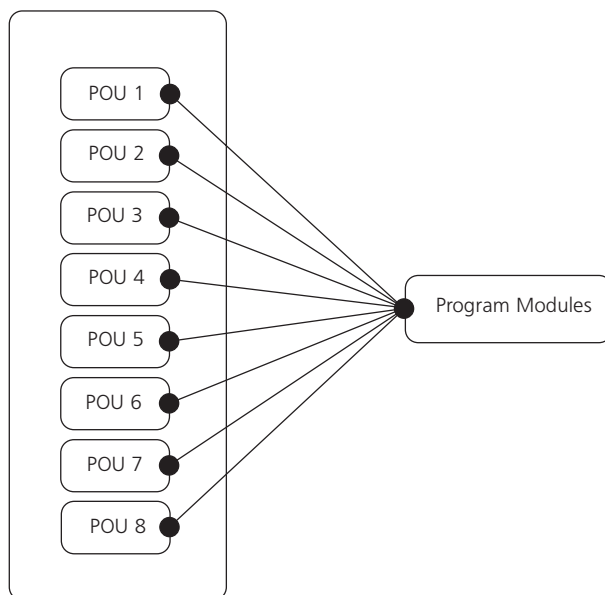
- *MELSEC MEDOC plus is a Windows program:* MELSEC MEDOC **plus** uses the graphical user interface of MS Windows for fast, intuitive operation. This means that instead of laboriously searching through a labyrinth of program structures, you can implement your controller applications quickly and efficiently.
- *MELSEC MEDOC plus increases your productivity:* The modular architecture of MELSEC MEDOC **plus** 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.
- *MELSEC MEDOC plus is a multi-language system:* MELSEC MEDOC **plus** supports programming in four different languages. Three graphical editors and one text-based editor help you to write tailor-made programs quickly and easily, choosing the language that best suits the problem.
- *MELSEC MEDOC plus is your link to the IEC world:* MELSEC MEDOC **plus** supports the new IEC 1131.3 standard for PLC (programmable logic controller) programming. This standard lays down the specifications for standardised PLC control programs.

Introduction to the IEC 1131.3 Standard

IEC 1131.3 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.



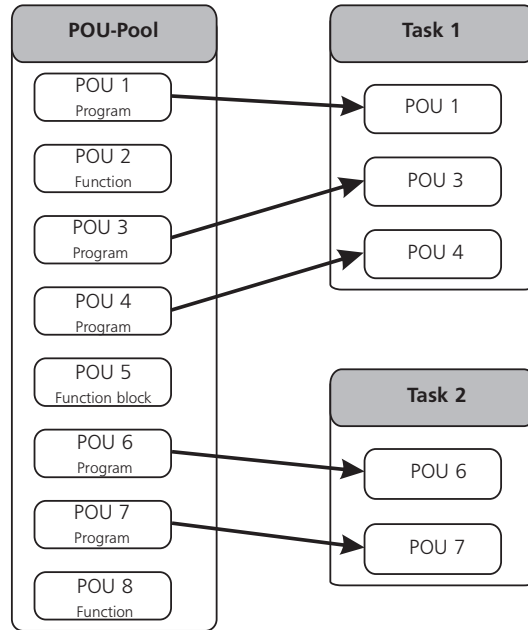
Program organisation units (POUs) are used to implement all programming tasks.

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

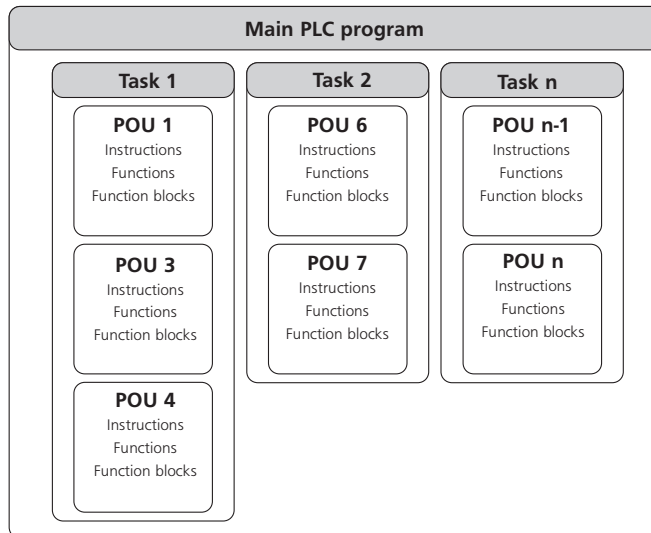
- **Programs**
- **Functions**
- **Function blocks**

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 POU's that you define as programs. This process is handled by the task management, in the Task Pool. Program POU's are put together in groups referred to as **Tasks**.

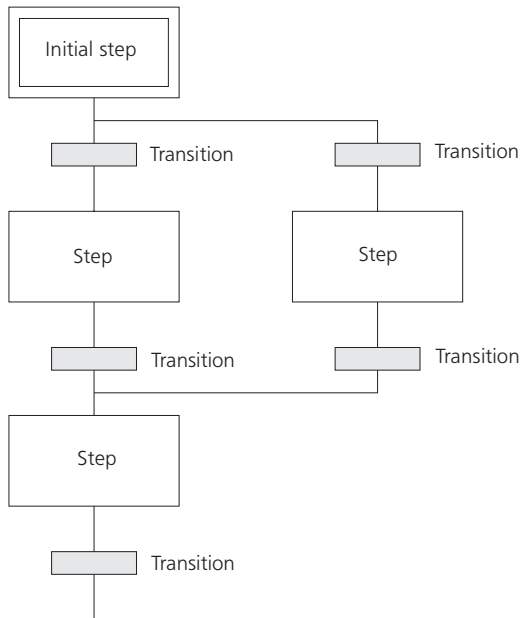


The program POU's are grouped together in tasks.



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.



An SFC sequence consists of a series of steps and transitions (transition or continue 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 Editor:

Instruction List (IL)

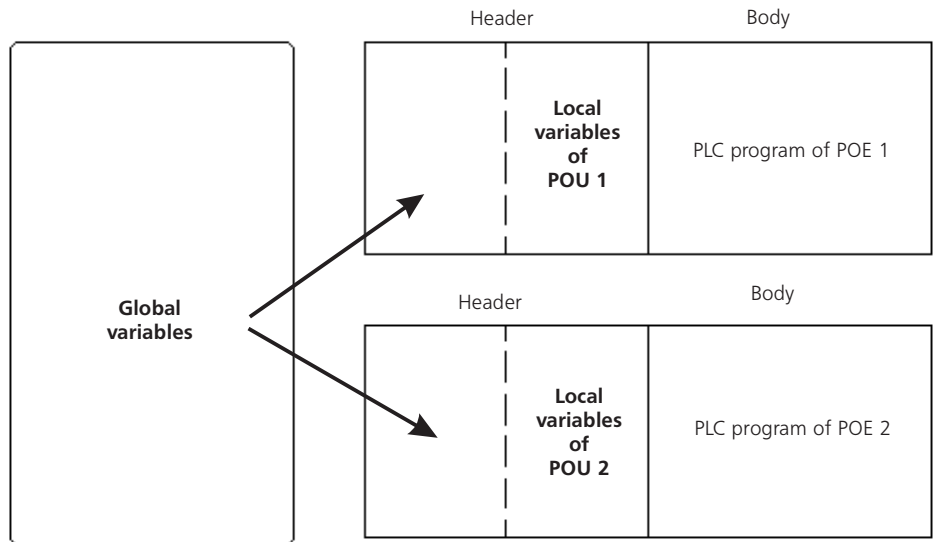
■ The Graphical Editors:

Ladder Diagram (LD)

Function Block Diagram (FBD)

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**, which is where the variables that can only be used within that POU are defined and declared. The **global variables**, which can be used by all the POUs in the program, are declared in a separate list.

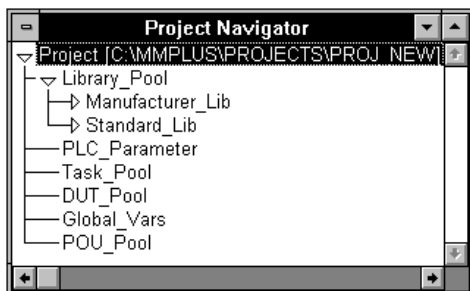


Basic Terms Used in IEC 1131.3 Standard Programming

Projects

Every MM+ 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



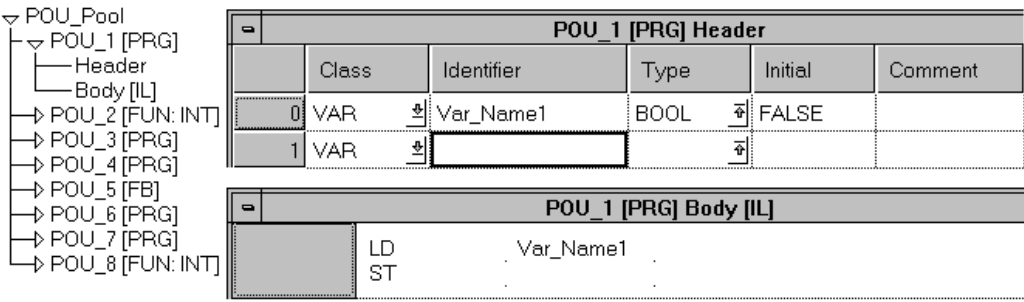
The program element objects are displayed in the Project Navigator window.

Program Organisation Units (POUs)

Each program organisation unit consists of

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

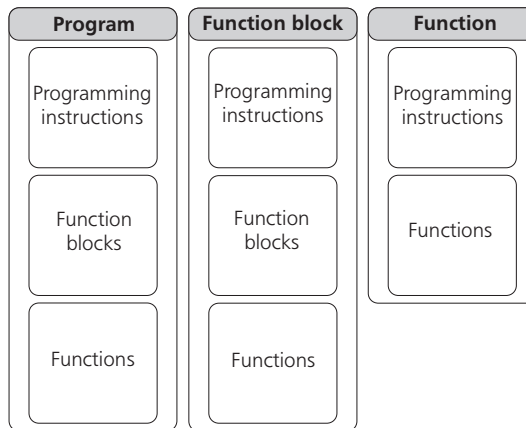
The variables to be used in the POU are defined (declared) in the header. All the variables you are going to use must always be declared before you start programming. The body contains the actual PLC program.



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

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

Programs, Function Blocks and Functions



The **program POU** is the primary 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.



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 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.

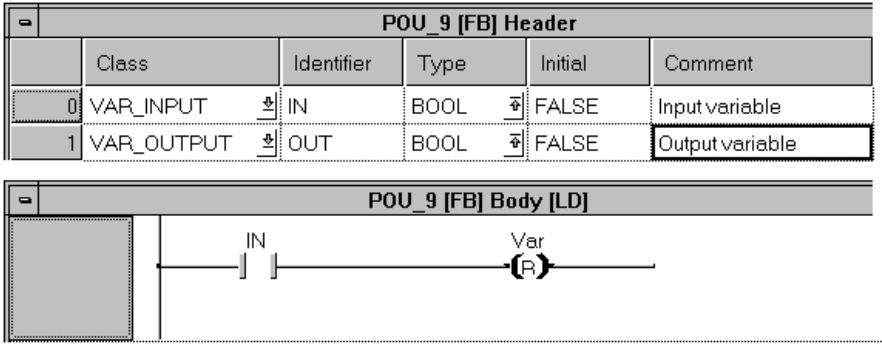
Functions can only pass one output value, and they do not store any internal status information. Thus, you should always get the same output value every time you invoke a function with the same input parameters.

Differences: Function Blocks – Functions

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

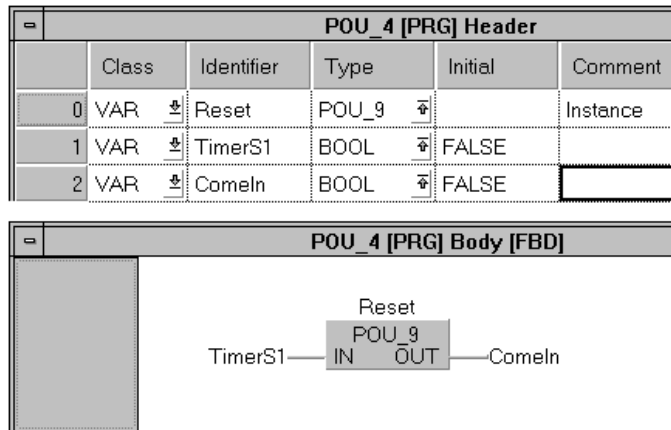
Parameters and Instancing

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.



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 'instantiating', 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'.



'Reset' is an instance of function block POU_9.

'IN' and 'OUT' are the formal parameters; 'TimerS1' and 'Comeln' are the actual parameters of the instance.

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

- TASK_1 (Prio = 31, Event = TRUE)
- TASK_2 (Prio = 31, Event = TRUE)

DUT_Pool

Global_Vars

POU_Pool

- POU_1 [PRG]
- POU_2 [FUN: INT]
- POU_3 [PRG]
- POU_4 [PRG]
- POU_5 [FB]
- POU_6 [PRG]
- POU_7 [PRG]
- POU_8 [FUN: INT]
- POU_9 [FB]

Task TASK_1 (Prio = 31, Event = TRUE)

	POU-Name	Comment
0	POU_1	
1	POU_3	
2	POU_4	

This project consists of two tasks, TASK_1 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

Event: Execute on the rising edge of a variable

Interval: Execute at defined time intervals

Priority: Execute in a defined priority order

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:

- Global variables, and
- Local variables.

Global variables are created for the entire project. They have global scope; this means that they are accessible from all POU's, thus making it possible to exchange data between the individual modules that make up the entire PLC program project.

Local variables' scope only applies for a single specific POU. They cannot be used or referenced in other POU's.

Declaring Variables

Before you can begin with the actual programming, you must declare all the variables you are going to use in the project as a whole (global variables) and in the individual POU's (local variables).

Each variable declaration has the following elements:

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

POU_9 [FB] Header					
	Class	Identifier	Type	Initial	Comment
0	VAR_INPUT	IN	BOOL	FALSE	Input variable
1	VAR_OUTPUT	OUT	BOOL	FALSE	Output variable

The declaration table for local variables in the header of the POU_9 function block

Class

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

Class	Use in POU's:			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_EXTERNAL	X	-	X	Global variable used in the POU header
VAR_EXTERNAL_CONSTANT	X	-	X	Global variable with unchangeable initial value used in the POU header
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

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 must also be assigned **absolute addresses** that reference the memory location of the variable in the CPU or a physical input or output.

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, MM+ identifies it immediately, places it in the correct column and produces the matching IEC address in the other column.

Examples of absolute addresses:

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

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

Data 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 bit
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 (Q series only)	Character string	max. 50 characters	

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 characters long for each variable.

Autoextern option

Global variables activated in the Autoextern column are automatically added to all existing and new POU's (programs and function blocks only).

Programming Languages

MELSEC MEDOC *plus* supports four programming languages: one text language, two graphical languages and a structuring language.

- Text language:
Instruction List language IL (IEC IL and MELSEC IL)
- 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!*

Networks

In all the editors – with the exception of the SFC 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.*

The Text Editors

Two different instruction list types are supported:

- MELSEC Instruction List
- IEC Instruction List

The structure of both these 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.

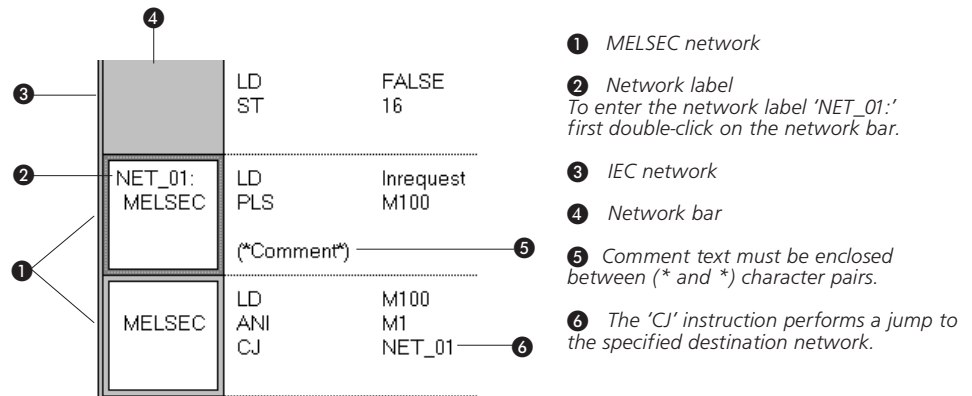
The MELSEC Instruction List Language (MELSEC IL)

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 1131.3 rules, and you can use both IEC programming instructions and the adapted MELSEC instructions (see Appendix in Reference Manual).



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.

POU_3 [PRG] Header					
	Class	Identifier	Type	Initial	Comments
0	VAR	Reset ❶	POU_9		❶ Declares the instance 'Reset' of function block POU_9.
1	VAR	TimerS1	BOOL	FALSE	
2	VAR	Comeln	BOOL	FALSE	

POU_3 [PRG] Body [IL]					
	LD	TimerS1			
	ST	Reset.IN			
	LD	Comeln			
	ST	Reset.OUT			
	CAL	Reset()			❷
	LD	X0			
	CALC	Reset(IN:=TimerS1,OUT:=Comeln)			❸

The formal parameters of function block POU_9 are 'IN' and 'OUT'. Actual parameters 'TimerS1' and 'Comeln' are passed to these formal parameters.

There are two ways (❷ and ❸) to pass actual parameters to formal parameters.

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:

LD	AVERAGE	D0	D1, D2, D3	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:

LD	E_ADD	X0	D0, D1, D2	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.

The Graphical Editors

The Ladder Diagram Language (LD)

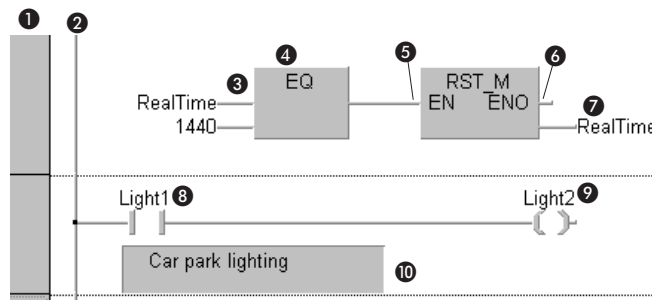
You can use all available programming instructions in the ladder diagram language (see Appendix in Reference Manual).

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).



Graphical programming in the ladder diagram editor

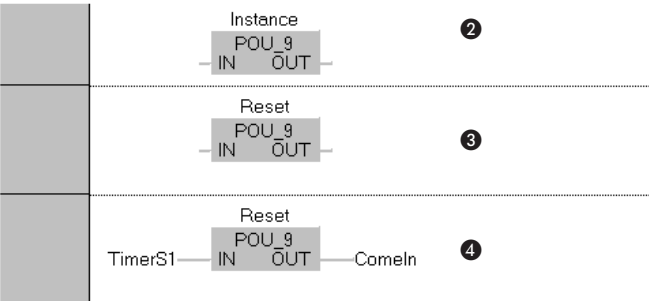
- ① Network bar
- ② Power bar
- ③ Input variable
- ④ Function
- ⑤ EN input
- ⑥ ENO output
- ⑦ Output variable
- ⑧ Contact
- ⑨ Coil
- ⑩ Comment

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 ①	POU_9		Instance
1	VAR	TimerS1	BOOL	FALSE	
2	VAR	Comeln	BOOL	FALSE	



① Declaration of 'Reset', an instance of function block POU_9.

② 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.

③ The instance name 'Reset' has been entered.

④ Next, the actual parameters 'TimerS1' and 'Comeln' are passed to the formal parameters 'IN' and 'OUT'.

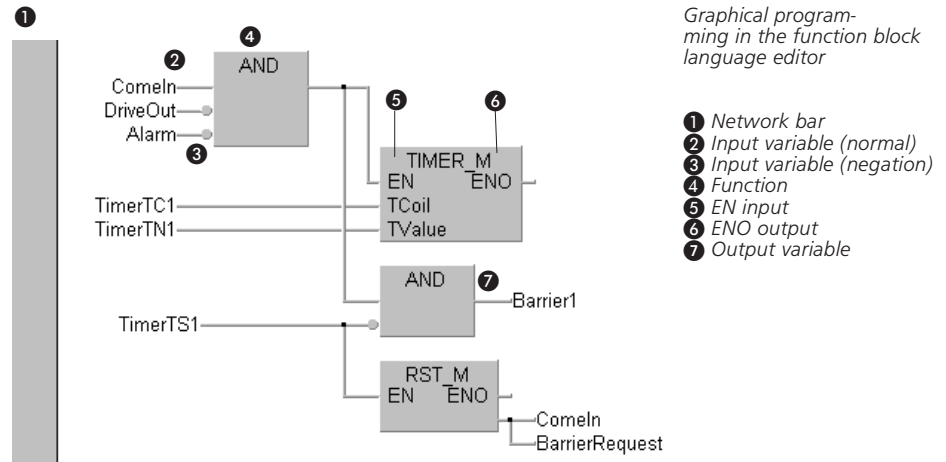
Calling Functions

When you call a function you must pass the necessary actual parameters (outside the shaded block) to the formal parameters (shown inside the block) (cf. ④ in the illustration 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).



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.

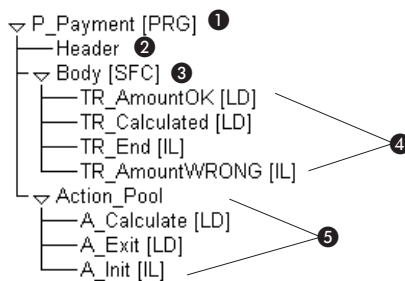


Note: *The program is the only available program organisation unit (POU) in this language.*

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.



① The 'P_Payment' program organisation unit, which is declared as a program [PRG].

② The header contains the POU's variables.

③ The PLC program was written with the SFC editor.

④ The individual transitions can be written with different editors.

⑤ The Action Pool contains the individual actions, which can also be written in different editors.

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



Activate action/transition condition



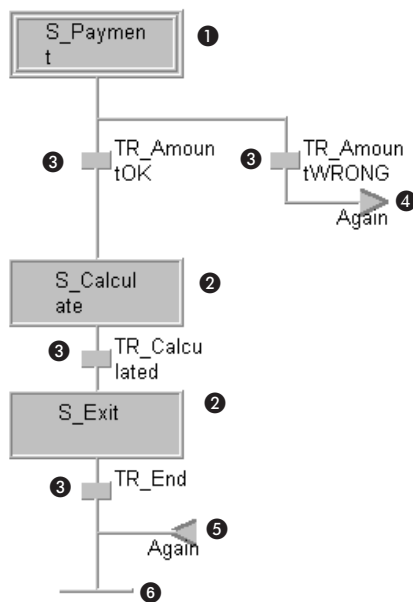
Deactivate action/transition condition

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.



Graphical programming in the sequential function chart editor

- ① Initial step
- ② Step
- ③ Transition
- ④ Jump exit point
- ⑤ Jump entry point
- ⑥ Final step

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 S6).

Installation

Hardware Requirements with MS Windows 3.1

Minimum Hardware Configuration

- 80386DX processor or above
- 4 MB RAM
- Hard disk with at least 20 MB free
- 3,5" high density floppy disk drive
- VGA-compatible graphics adapter
- Mouse
- 1 RS232 serial port for communication with the PLC system
- 1 printer port
- Printer
- 14"/35 cm diag. VGA monitor

Recommended Enhancements

- Pentium
- 8 MB RAM
- 1024 x 768 x 256 (width x height x colour depth)
- Small Fonts
- 17"/43 cm diag. VGA monitor

Software Requirements

- MS-DOS operating system version 3.3 or above
- Microsoft Windows version 3.1*/Microsoft Windows 95 (16 bit)

Copyright



IMPORTANT NOTICE: *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 disks for your own backup and archiving purposes.*

Installing MELSEC MEDOC plus

During this procedure the installation program will create a directory on your hard disk, into which all the MM+ files will be copied.

Installing MM+ on your hard disk

- ① Before you begin, make backup copies of the MM+ distribution disks. Write-protect the disks to make sure they cannot get erased accidentally. Install the program from the backup copies and store the original disks in a safe place.
- ② Make sure that Microsoft Windows 3.1* is properly installed on your computer. For information on using Windows please refer to the Windows User's Guide.
- ③ Start MS Windows.
- ④ Insert Installation Disk 1 in the floppy disk drive.
- ⑤ Open the Windows File Manager.
- ⑥ Select and open the floppy disk drive in which you have inserted the installation disk.
- ⑦ Double-click on the file INSTALL.EXE.
This starts the MM+ installation program.
- ⑧ Follow the instructions that appear on the screen.
- ⑨ When the installation procedure is finished the program will create a new program group containing the MM+ program icon.

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

Starting MELSEC MEDOC plus

- ① Double-click on the MM+ program icon. This starts MELSEC MEDOC *plus* and displays the start-up screen.
- ② Confirm with the \ominus key.

Quitting MELSEC MEDOC plus

You can quit MM+ directly at any point in the program by pressing the key combination $\alpha\mathcal{O}$.

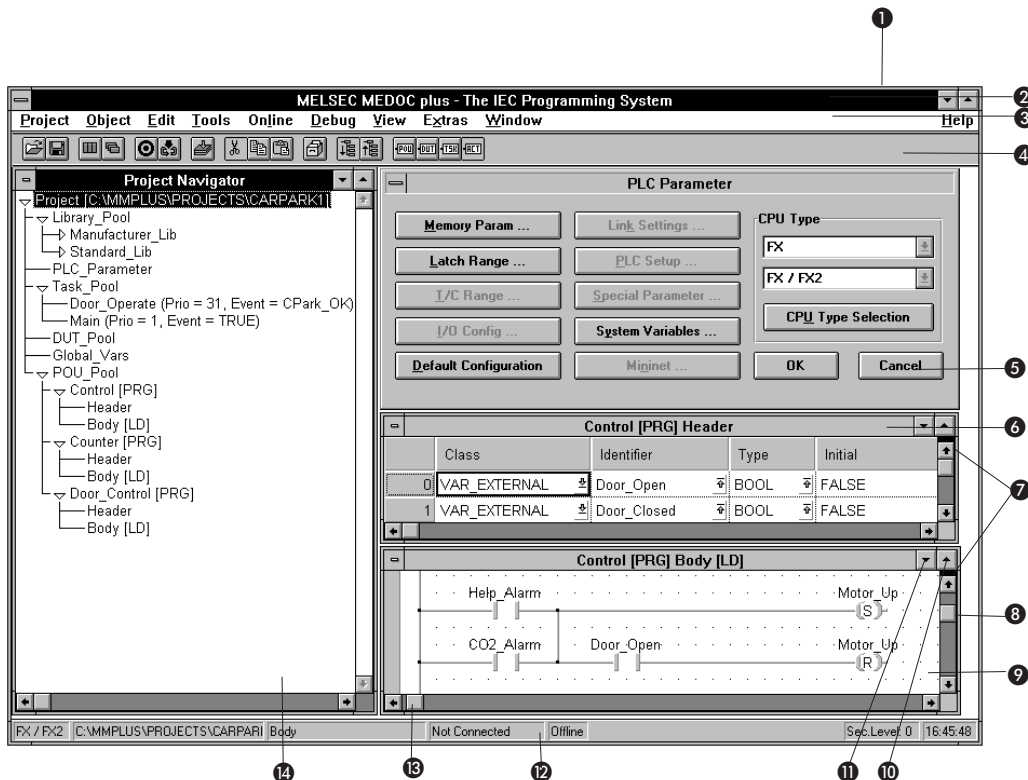
Or:

Double-click on the **Quit** command in the **Project** menu.

The User Interface

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 S1 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.



- | | | |
|-------------------------|------------------------------|----------------------------|
| ① Application title bar | ⑥ Declaration table (header) | ⑪ 'Minimise' button |
| ② Menu bar | ⑦ Object window | ⑫ Status bar |
| ③ Toolbar | ⑧ Vertical scrollbar | ⑬ Horizontal scrollbar |
| ④ Dialog box | ⑨ Editor (body) | ⑭ Project Navigator window |
| ⑤ Button | ⑩ 'Maximise' button | |

The Menu Bar

The MM+ 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 dialog 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.*

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 MM+.



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

Windows

MM+ 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.

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.

The Project Navigator

The Project Navigator is the 'control centre' used for selecting and handling the objects used in MM+. This is the starting place for all operations performed on MM+ 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

You can expand or collapse branches of the Project Navigator tree by double-clicking on the appropriate levels. Expanded and collapsed branches are identified by different arrow symbols in the tree. Double-clicking on the lowest level opens the window of the object on that level.



'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.

Declaration Tables

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

Global Variable List							
	Class	Autoextern	Identifier	MIT-Addr.	IEC-Addr.	Type	Initial
0	VAR_GLOBAL		Door_Open	X0	%IX0	BOOL	FALSE
1	VAR_GLOBAL		Door_Closed	X1	%IX1	BOOL	FALSE
2	VAR_GLOBAL		Motor_Up	Y0	%QX0	BOOL	FALSE
3	VAR_GLOBAL		Motor_Down	Y1	%QX1	BOOL	FALSE
4	VAR_GLOBAL		Enter_Up	X2	%IX2	BOOL	FALSE

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
w	Line up
x	Cell right
y	Line down
z	Cell left
	Step through all cells from left to right
j	Step through all cells from right to left
jø	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.

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

Graphical Editors

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

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.

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 S6) you will learn how to use the different editors.*

Getting Started

efesotomasyon.com

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

Steps	Page
S1 Creating New Projects	6 – 2
S2 Creating Tasks	6 – 4
S3 Declaring Global Variables	6 – 5
S4 Creating Program Organisation Units	6 – 7
S5 Programming POU Headers	6 – 8
S6 Programming POU Bodies	6 – 9
<i>Programming examples:</i>	
● Inputs and outputs in LD	
● Sum function in FBD	
● I/O signal configuration parameters	
● Timers in LD/FBD/IL	
● Sequential function chart language	
S7 Checking PLC Programs (syntax check)	6 – 34
S8 Configuring Tasks	6 – 35
S9 Compiling Projects	6 – 37
S10 Communications Port Setup	6 – 38
S11 Downloading Programs to the PLC	6 – 39
S12 Monitoring Programs	6 – 40
S13 Uploading Data from the PLC	6 – 42

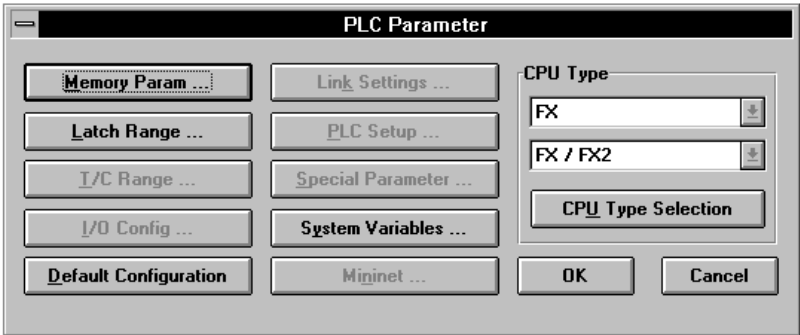
S1

Creating New Projects

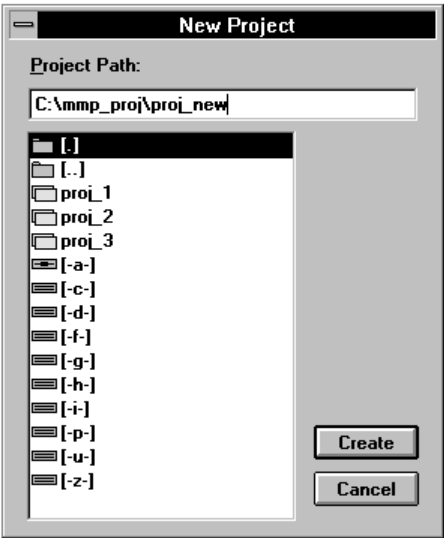
S1

How to create a new project

- ① Select **New** in the **Project** menu.
- ② This displays the **PLC Parameter** dialog box. Select your PLC in the **PLC Type** field and confirm your selection with **OK**.



- ③ The **New Project** dialog 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.



In this example a project called **PROJ_NEW** is being created in the subdirectory **C:\mmp_proj**.

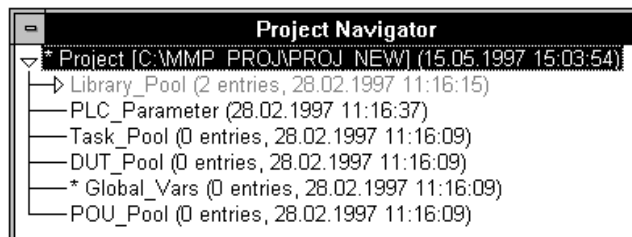
Please note that **PROJ_NEW** is not a single file, but rather a subdirectory created by MM+ to contain all the project files.

- ⑤ Click on the **Standard** option button in the **Project Wizard** dialog box.



MM+ then creates the 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.



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 displayed in brackets after each entry in the Project Navigator window. You can disable 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 MM+ window background colour is light grey. You can change all the colours to suit your personal taste with **Colors** in the **View** menu.

S2 ——— Creating Tasks ——— S2

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:



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



Defining a new task

- ③ Enter a name (max. 16 characters) for the new task and confirm with **OK**. MM+ 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 S8.

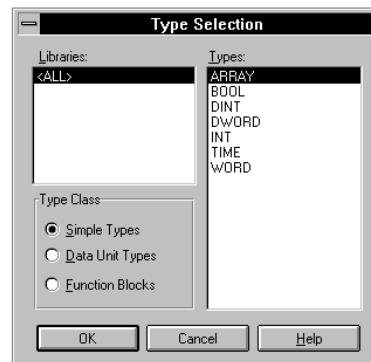
S3 — Declaring Global Variables — S3

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.

Global Variable List								
	Class	Autoextern	Identifier	MIT-Addr.	IEC-Addr.	Type	Initial	Comment
0	VAR_GLOBAL							

- 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 | to move to the **MIT-Addr.** or **IEC-Addr.** column. Enter the absolute address of the global variable.
- ⑤ Press | to move to the **Type** column and click on the up arrow button with the mouse to open the Type Selection dialog box.



- 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 | 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).

Comment

Or

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

	Class	Autoextern	Identifier	MIT-Addr.	IEC-Addr.	Type	Initial	Comment
0	VAR_GLOBAL							

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 bC to copy it to the clipboard. Then select the insertion position by clicking on the appropriate number button and press bV to insert.

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



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

S4 ————— Creating Program Organisation Units ————— S4

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

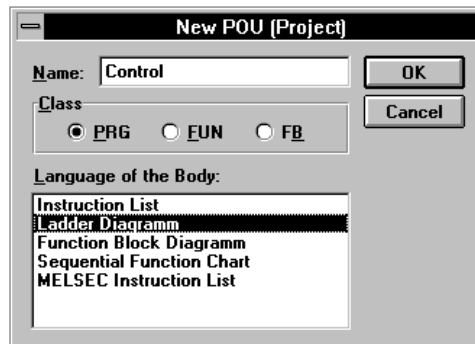
How to create a new program organisation unit

- 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.



A new POU called 'Control' is being defined and declared as a 'Program' (PRG) type.

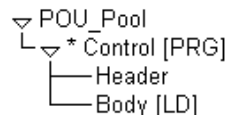
The PLC program in the body of the POU is going to be written in ladder diagram language.

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



The right arrow 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.



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

S5

Programming POU Headers

S5

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 S4).
- 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 S3 above, entering the class, identifier and data type for each variable.

Control [PRG] Header					
	Class	Identifier	Type	Initial	Comment
0	VAR				

- 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 (bC) and then insert them in this declaration table (bV).

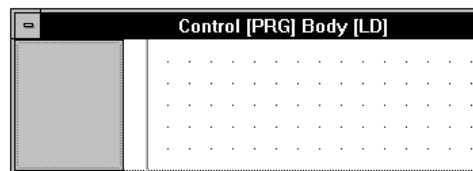
The terms **class**, **identifier** and **data type** are defined and explained in Chapter 3.

S6 — Programming POU Bodies — S6

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 S4).
- 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.



*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.

Programming Examples

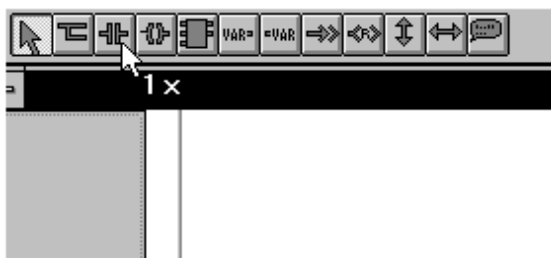
Inputs and outputs in ladder diagram language (LD)

Programming inputs and outputs in the LD editor

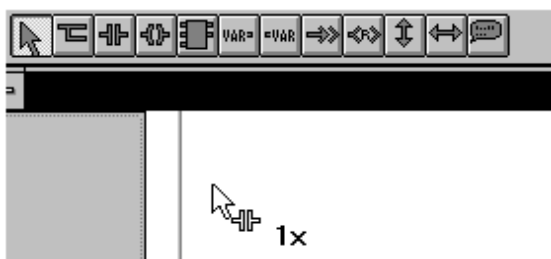


- ① In the Project Navigator window, double-click on a program body entry defined with the ladder diagram language (LD).

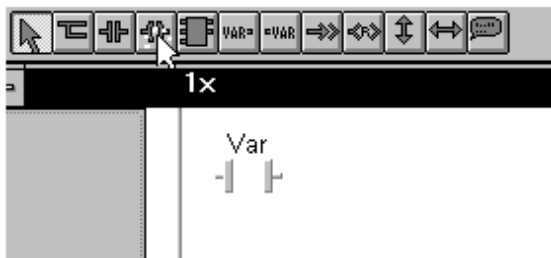
- ② Click on the "Contact" tool icon in the toolbar.



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



- ④ Click on the "Coil" tool icon in the toolbar.



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



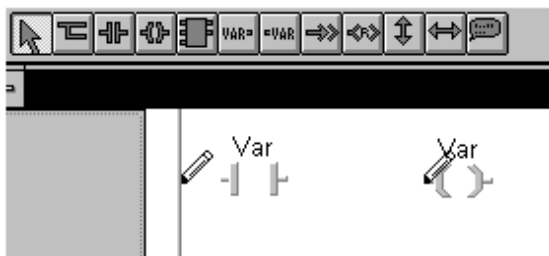
- ⑥ Click on the "Interconnect/Line" tool icon in the toolbar

or

Click with the right mouse button.

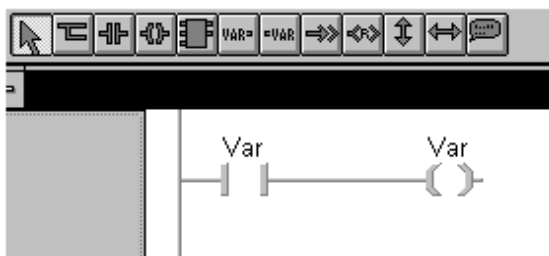


- ⑦ Position the pointer over the left network bar, hold down the left mouse button and draw a line to the connection point of the output coil.



The 'VAR' designations that appear above the input contact and output coil symbols are dummy names, 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 'VAR' dummy designators over the contact and the coil and overwrite the dummy name 'VAR' with appropriate variable names. Alternatively, you can press m to display the operand selection list and select a name from the list.



A Sum Function in FBD Language

Programming a sum function in the FBD editor

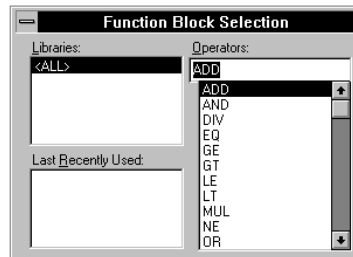


Steps ① through ⑫ below are exactly the same in the ladder diagram and function block diagram editors. Only the tools displayed in the toolbar are different in each case.

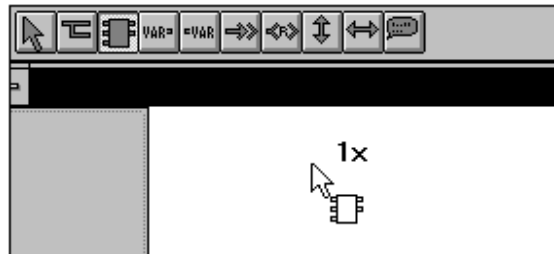
- ① In the Project Navigator window, double-click on a body entry defined with the function block diagram language (FBD).
- ② Click on the “Function Block” tool icon in the toolbar.



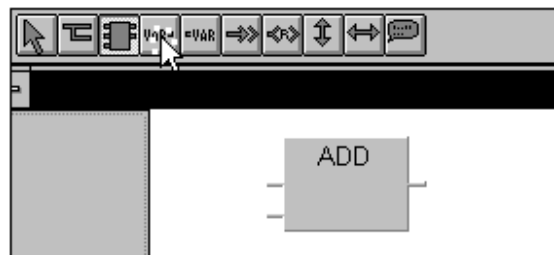
- ③ Double-click on the ADD instruction in the selection box displayed.



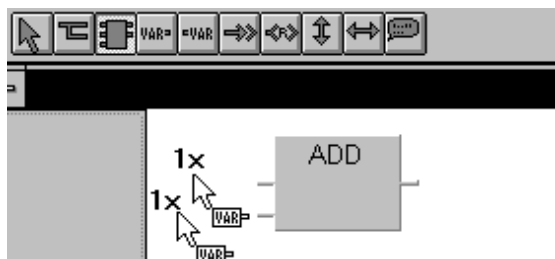
- ④ Position the mouse pointer and press the left mouse button to place the function block.



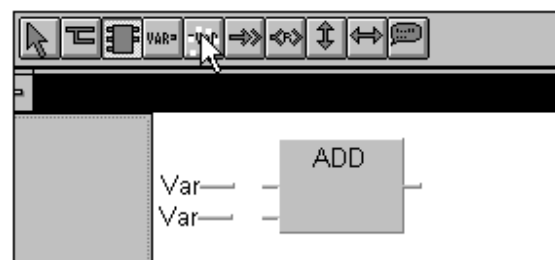
- ⑤ Click on the “Input Variable” tool icon in the toolbar.



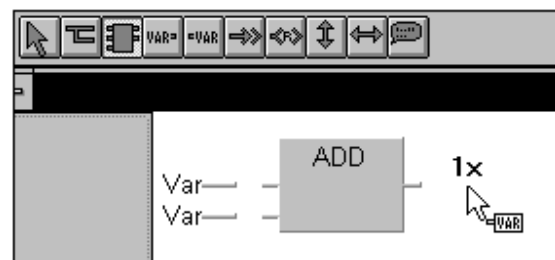
- ⑥ Position the mouse pointer next to the block's first input and press the left mouse button. Repeat step ⑤, and then enter the next input variable next to the block's second input.



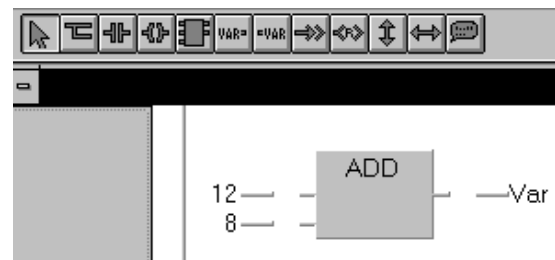
- ⑦ Click on the "Output Variable" tool icon in the toolbar.



- ⑧ Position the mouse pointer next to the output of the block and press the left mouse button.

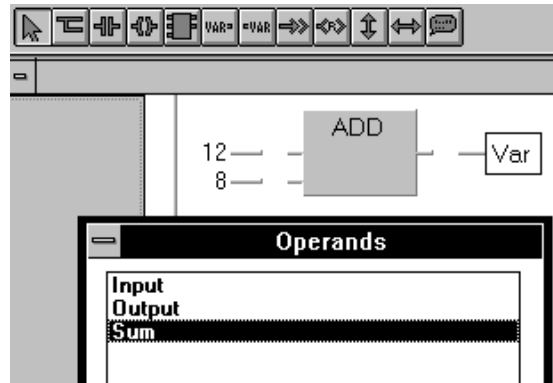


- ⑨ Click on the 'Var' dummy designators of the input variables and overwrite the first with a value of 12 and the second with 8.

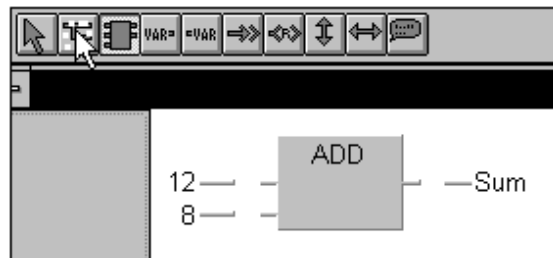


- ⑩ Click on the 'Var' dummy designator of the output variable and then press **m** to display the operand list. Select 'Sum' and confirm with **OK**.

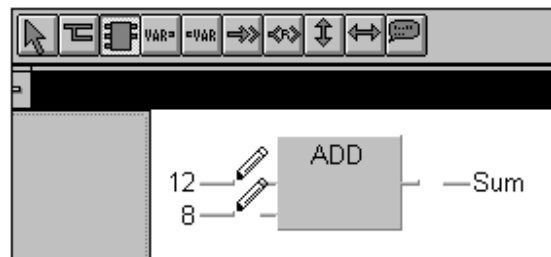
Note: The variables will only appear in the operand list if the header in which they are declared has been saved.



- ⑪ Click on the "Interconnect/Line" tool icon in the toolbar. (In the illustration the "Var" dummy designators of the input variables have been overwritten with two constants, and the variable designator "Sum" has been entered for the output variable.)



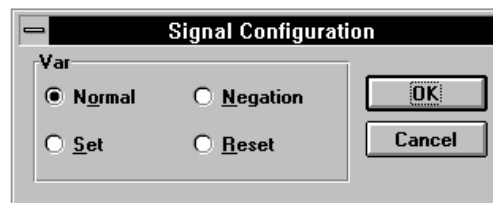
- ⑫ Draw an interconnect line between the first input variable and the output variable. Then draw a line to connect the second input variable to the function block.



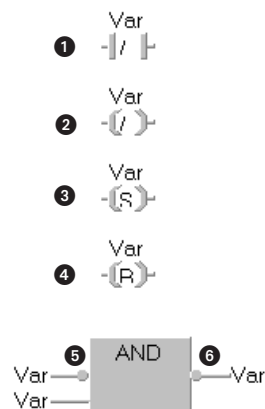
I/O Signal Configuration Parameters

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 dialog box. Select the appropriate options, then confirm with OK.



- ① Negated input contact(LD only)
- ② Negated output coil(LD only)
- ③ Set output coil (LD only)
- ④ Reset output coil(LD only)
- ⑤ Negated input variable(LD and FBD)
- ⑥ Negated output variable(LD and FBD)



Timers in LD/FBD/IL

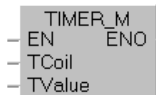
Description of the timer device

All timers must have the following four elements:

- TValue: Setpoint value
- TN: Actual value
- TC: Output coil
- TS: Input contact

Global Variable List						
	Class	Autoextern	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

The elements TN, TC and TS must be declared in the global variables list.



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 setpoint value of 100 is reached.

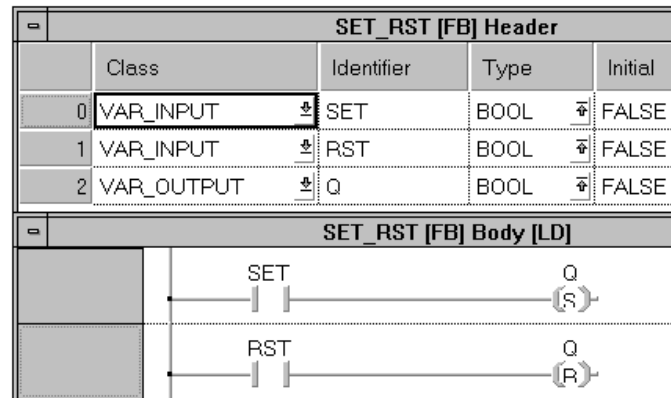
Algorithm: 'Input2' and the timer contact 'Timer1S' (TS) are responsible for switching 'Output1'. This function will be performed by the user-programmed function block SET_RST.

'Input1' activates the timer, i.e. it controls the switching of the timer coil 'Timer1C' (TC). The setpoint 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 S4, p. 6–7).
- ② Enter the following three variables in the header: SET, RST and Q.
- ③ Insert two network circuit blocks in the body.



- ④ Save the header and the body (**Object - Save**).
- ⑤ Select the 'SET_RST' POU in the Project Navigator window and press . The following dialog box is then displayed:

Function Information	
Name:	SET_RST OK
Size:	20 Bytes Cancel
<input type="checkbox"/> Use Macrocode <input type="checkbox"/> Use MC/ MCR <input checked="" type="checkbox"/> Use with EN/ENO Comment	
Type:	FB ↓
Language:	Ladder Diagram ↓
Last Change:	19.01.1996 14:16:33
Security Level <input checked="" type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 <input type="radio"/> 6 <input type="radio"/> 7	
<input checked="" type="checkbox"/> Allow Read Access for lower Levels	

- ⑥ Activate the EN/ENO Contacts check box to assign an EN input and an ENO output to the function block.

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 S3).
- ② Declare the following variables.

Global Variable List							
	Class	Autoextern	Identifier	MIT-Addr.	IEC-Addr.	Type	Initial
0	VAR_GLOBAL		TIMER1C	TC0	%MX5.0	BOOL	FALSE
1	VAR_GLOBAL		TIMER1S	TS0	%MX3.0	BOOL	FALSE
2	VAR_GLOBAL		TIMER1N	TN0	%MW3.0	INT	0
3	VAR_GLOBAL		Eingang1	X0	%IX0	BOOL	FALSE
4	VAR_GLOBAL		Eingang2	X1	%IX1	BOOL	FALSE
5	VAR_GLOBAL		Ausgang1	Y0	%QX0	BOOL	FALSE

Step 3: Create the Timer POU (ladder diagram)

- ① Create a new POU as a program using the ladder diagram language (see S4).
- ② Open the header of the new POU and declare the following variables:

DEMO_LD [PRG] Header					
	Class	Identifier	Type	Initial	
0	VAR_EXTERNAL	TIMER1C	BOOL	FALSE	
1	VAR_EXTERNAL	TIMER1S	BOOL	FALSE	
2	VAR_EXTERNAL	TIMER1N	INT	0	
3	VAR_EXTERNAL	Input1	BOOL	FALSE	
4	VAR_EXTERNAL	Input2	BOOL	FALSE	
5	VAR_EXTERNAL	Output1	BOOL	FALSE	
6	VAR	DATA	INT	0	
7	VAR	SET_RST1	SET_RST		

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_EXTERNAL

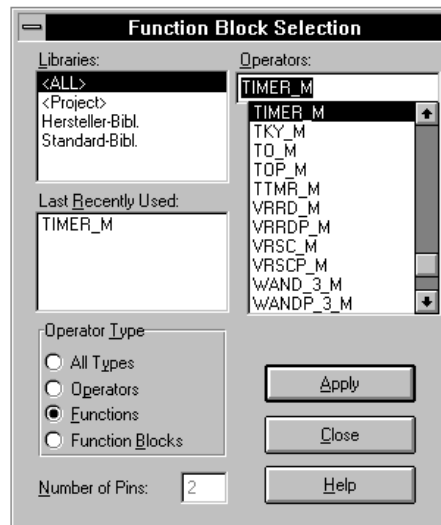
Use as local variables = VAR

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.

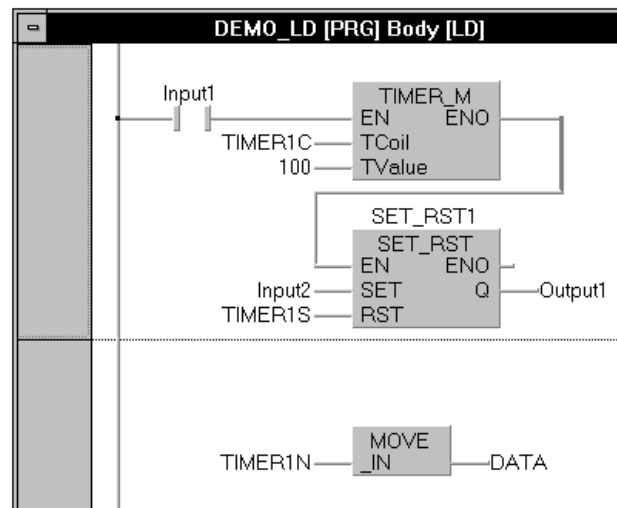
The timer we shall use is a function that is stored in the manufacturer library.



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" (p. 6–16).

- ⑤ Create the following PLC program:

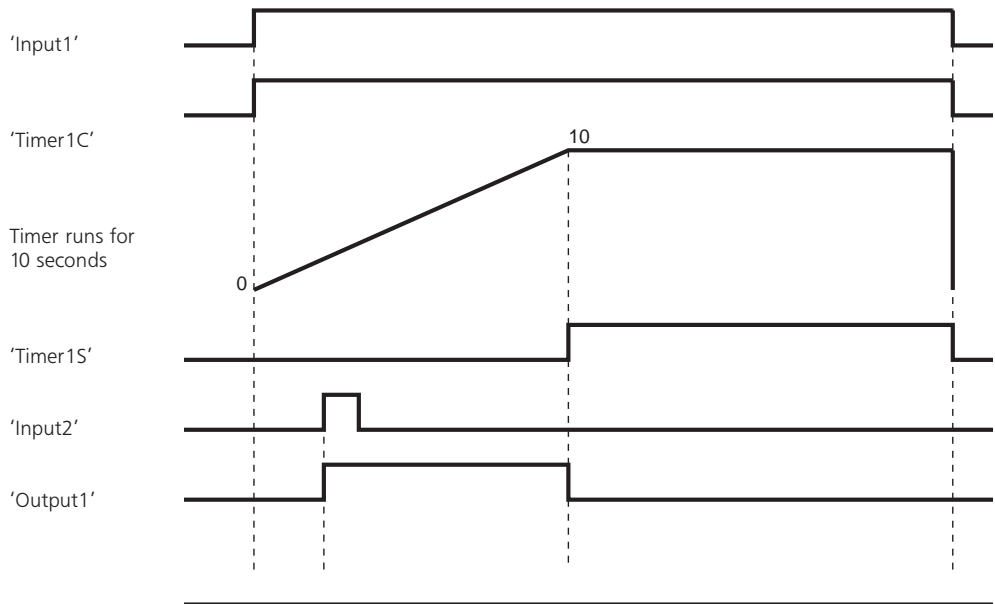


The function blocks used in the program can be found in the following libraries:

Manufacturer Library:
TIMER_M

Function blocks:
SET_RST

IEC Standard Library:
MOVE



Timer sequence

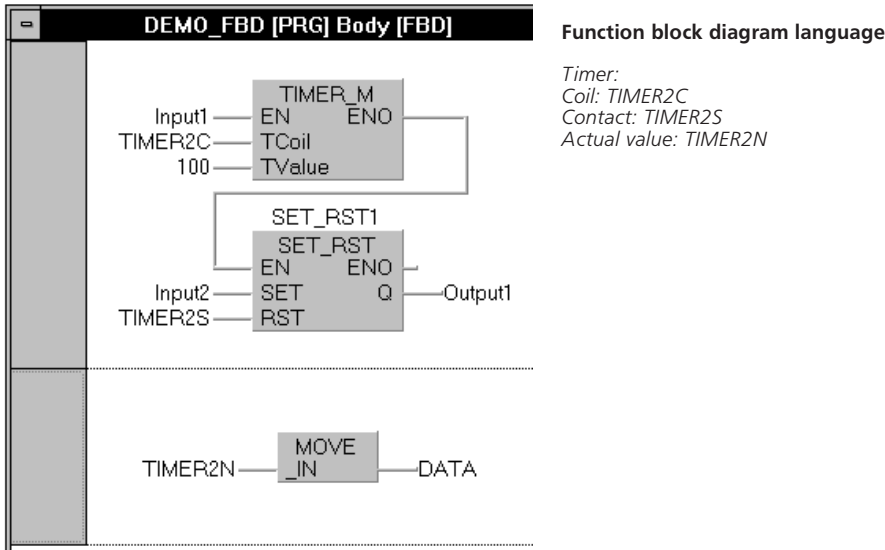
The timer sequence begins when INPUT1 is set and the timer starts to run. If INPUT2 is set, OUTPUT1 is switched on.

When the 10-second period has elapsed, timer contact TIMER1S is set and OUTPUT1 is switched off again.

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:



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]			Instruction list language
LD	Input1		
TIMER_M	TIMER3C, 100		
CAL	SET_RST1(EN:=Input1, SET:=Input2, RST:=TIMER3S, Q:=Output1)		Timer: Coil: TIMER3C Contact: TIMER3S Actual value: TIMER3N
LD	TIMER3N		
MOVE			
ST	DATA		



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.

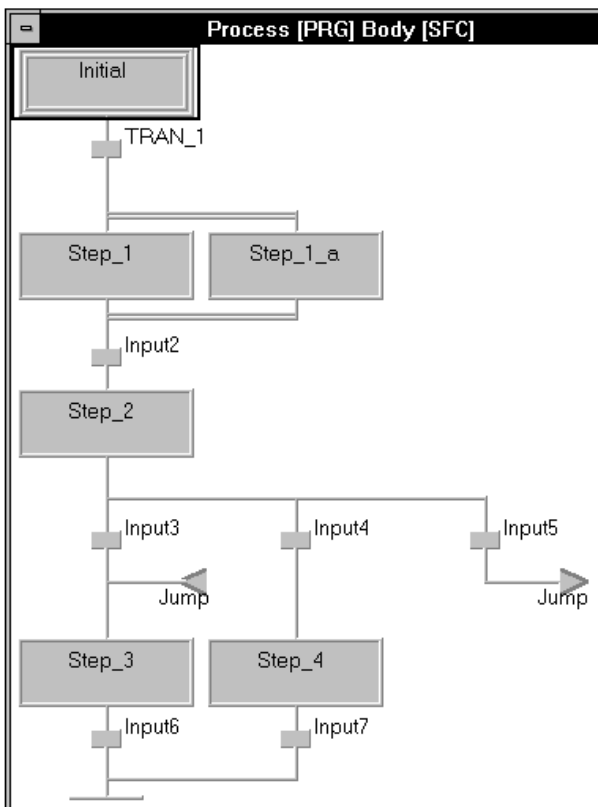
Sequential Function Chart Language

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 MM+ 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.



The 'Process' program

Program execution

- ① When the PLC is switched to RUN mode 'Output1' is set.
- ② 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.
- ③ The transition 'Input2' polls 'Input2'. If the latter is set 'Step_2' is executed and 'Output4' is set.
- ④ 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'.
- ⑤ '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 S4).

▾ Process [PRG]
 ├── Header
 ├── Body [SFC]
 └── Action_Pool

*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

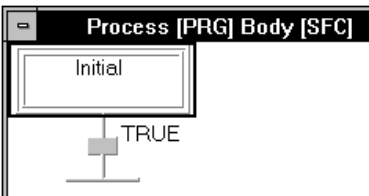
① Open the header and enter the variables to be used in the POU (see S3).

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

In this example only global variables are used. Global variables are referenced by using the VAR_EXTERNAL class.

Step 3: Open the body

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



When you open the SFC editor the following elements are displayed:

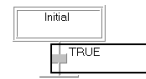
- The Initial Step (double outline)
- The transition TRUE
- The Final Step



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

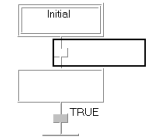
- ① Select the 'TRUE' transition with the block cursor.



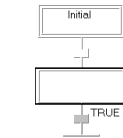
- ② Click on the tool icon



Inserts a new step/transition pair.



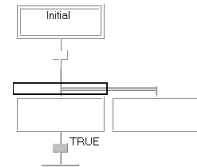
- ③ Select the step you have just inserted.



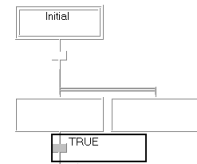
- ④ Click on the tool icon



Inserts a right divergence and a new step.



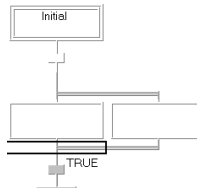
- ⑤ Select the 'TRUE' transition.



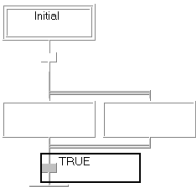
- ⑥ Click on the tool icon



Inserts a right convergence.



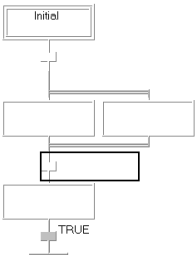
⑦ Select the 'TRUE' transition.



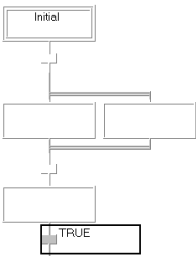
⑧ Click on the tool icon



Inserts a new step/transition pair.



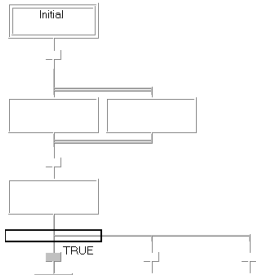
⑨ Select the 'TRUE' transition.



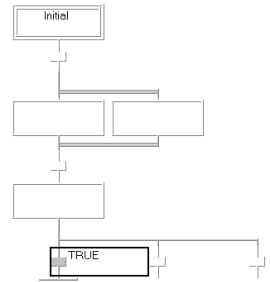
⑩ Click twice on the tool icon



Inserts two right divergences with transitions.



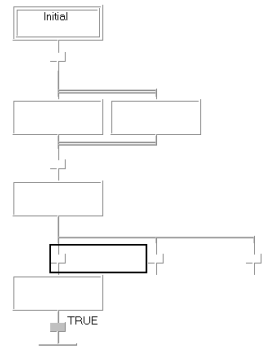
- ⑪ Select the 'TRUE' transition.



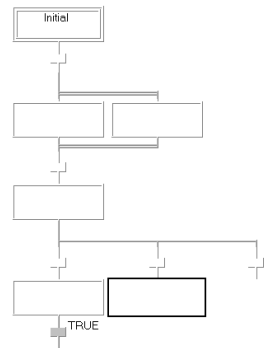
- ⑫ Click on the tool icon



Inserts a new step/transition pair.



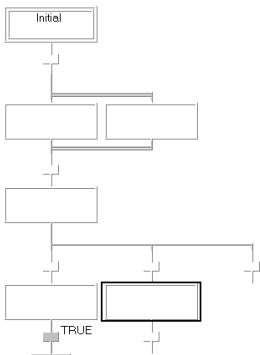
- ⑬ Click in the empty space next to the step you have just inserted.



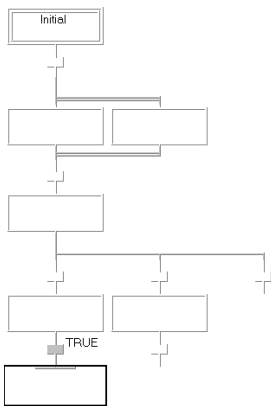
⑭ Click on the tool icon



Inserts a new step/transition pair.



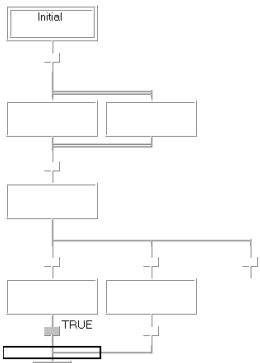
⑮ Click on the final step.



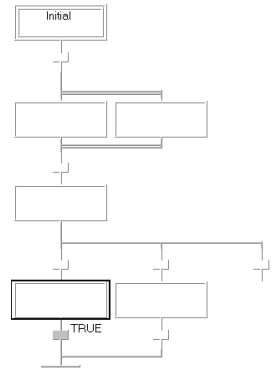
⑯ Click on the tool icon



Inserts a right convergence.



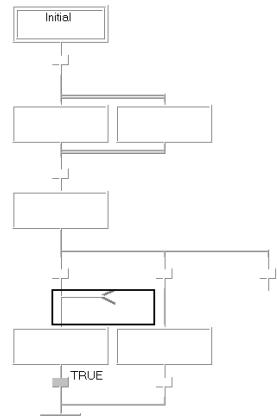
- ⑰ Click on the left hand step in the bottom row.



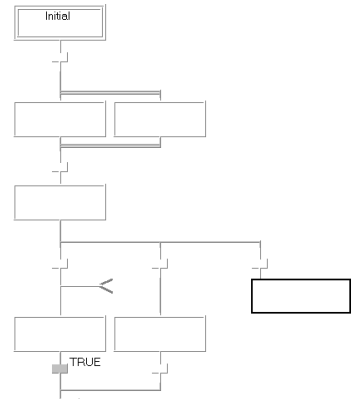
- ⑱ Click on the tool icon



Inserts a jump entry point.



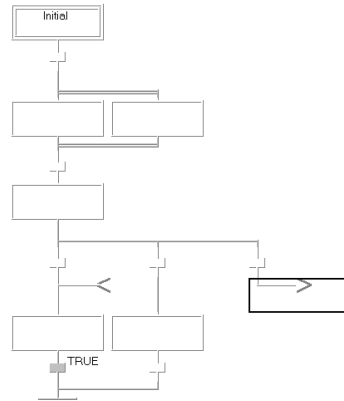
- ⑲ Click on the empty space below the free transition.



- ⑩ Click on the tool icon

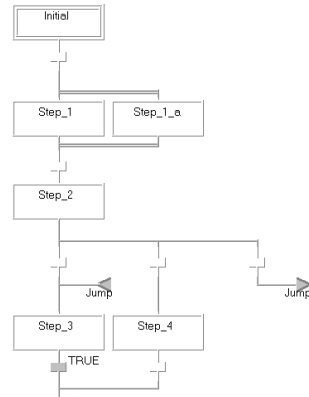


Inserts a jump exit point.



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

- ① Double-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').



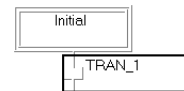


Step 6: Assign transition conditions to the transitions

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

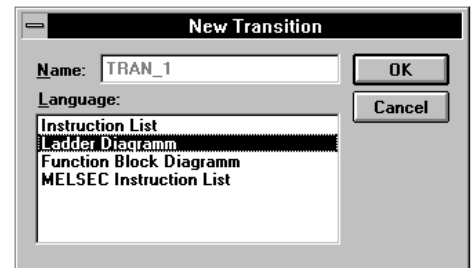


- ② Enter the program name (Example: 'TRAN_1').

- ③ Click on the tool icon



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



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

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

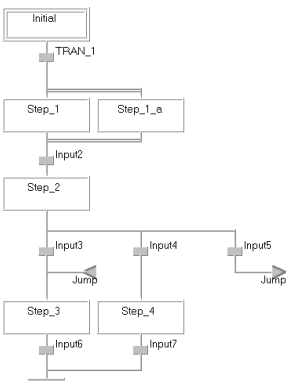
- ⑥ Write the transition program.




Assigning an existing variable to a transition

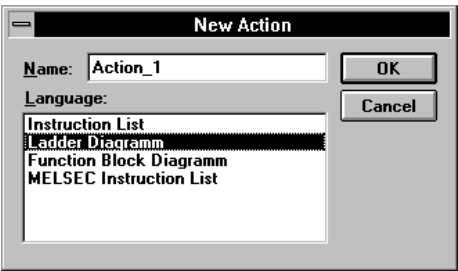
- ① Double-click on the transition to which you wish to assign a variable.
Activates editing mode.
- ② Enter the name of an existing variable (Example: 'Input2' to 'Input7').

Note: *This overwrites the 'TRUE' transition condition.*



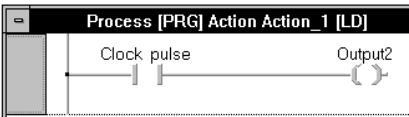
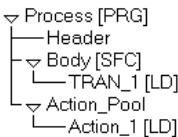
Step 7: Create the actions

- ① Select the 'Process' POU in the Project Navigator.
- ② Click on the tool icon 
Displays the New Action dialog box.
- ③ Enter a name for the action (Example: 'Action_1') and select the programming language (Example: ladder diagram).



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.
- ⑤ Enter the 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.*

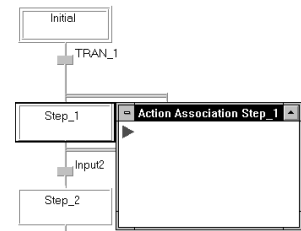
Step 8: Assign actions or Boolean variables to the steps

- 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** dialog box, which is still empty.



- ③ Press the m 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'

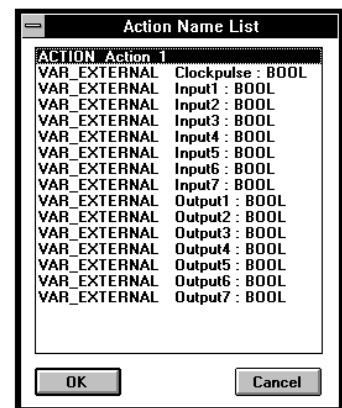
'Step_1' = 'Action_1'

'Step_1_a' = 'Output3'

'Step_2' = 'Output4'

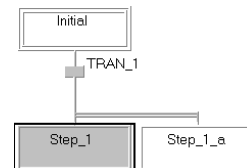
'Step_3' = 'Output5'

'Step_4' = 'Output6'



- 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.



S7 ——— Checking PLC Programs (syntax check) ——— S7

How to check your program for syntax errors

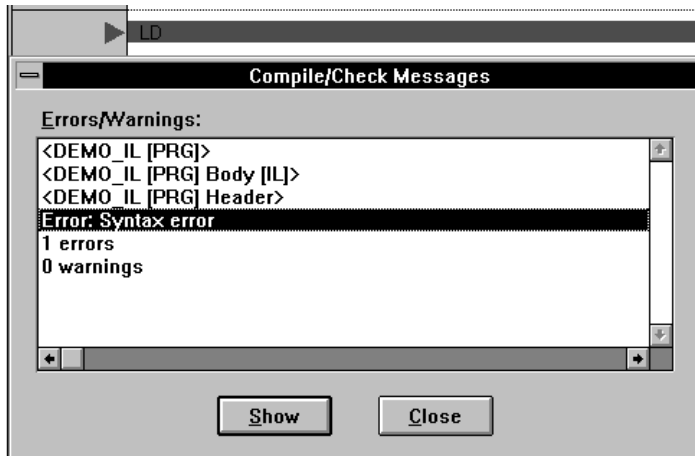
- ① Select the object to be checked in the Project Navigator window.
- ② Select **Check** in the **Object** menu

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.



- 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).

S8 ————— Configuring Tasks ————— S8

In this section it is assumed that you have already created the tasks for your project (see S2). 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.

Task Task_Main [Prio = 31, Event = TRUE]		
	POU-Name	Comment
0	Control	

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 dialog box displayed. Confirm your selection with **OK**. Only POUs defined as programs (PRG) are included in the dialog 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 dialog 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 |.

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 **OE** to open the **Task Information** dialog box.

The parameters in this dialog 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:

1

OK

Cancel

Comment

Name:

Main

Size:

96 Bytes

Type:

TASK

☐ Timer/ Output Control

Last Change:

11.02.1998 16:42:09

Security Level

☒ 0

☐ 1

☐ 2

☐ 3

☐ 4

☐ 5

☐ 6

☐ 7

☒ Allow Read Access for lower Levels

Event = TRUE
(execute always)

Interval = 0
(because event-triggered)

Priority = 0
(maximum priority)

Priority = 31
(lowest priority)

The dialog 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.

S9 ————— Compiling Projects ————— S9

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 **Compile Project** in the **Project** menu.

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



IMPORTANT: *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 (S7).

S10 — Communications Port Setup — S10

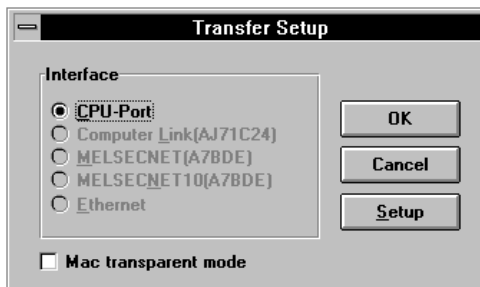
Before you can download a project to the PLC you must first configure the communications port you are going to use for this purpose.



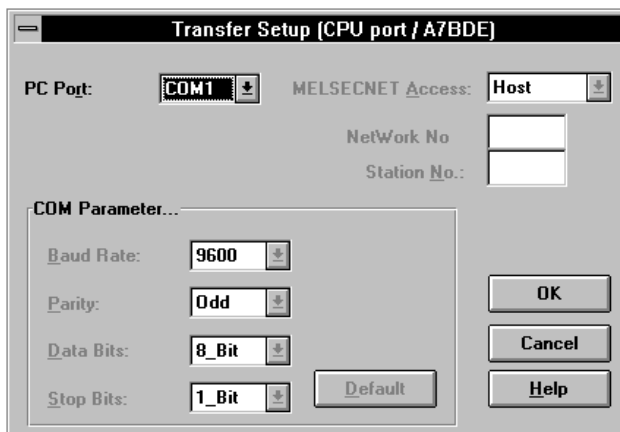
Note: Before you begin, make sure that you know precisely which physical interface on your personal computer is going to be used for transferring the data to the PLC system.

How to select and configure the communications port

- ① Select **Transfer Setup** in the **Online** menu, then select **Ports** to display the **Transfer Setup** dialog box.



- ② Click on the **Setup** button.



- ③ Select the desired serial port. **COM1** to **COM6** are available.
- ④ Confirm the entries in both dialog boxes with **OK**.

S11 ——— Downloading Programs (to PLC) ——— S11

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

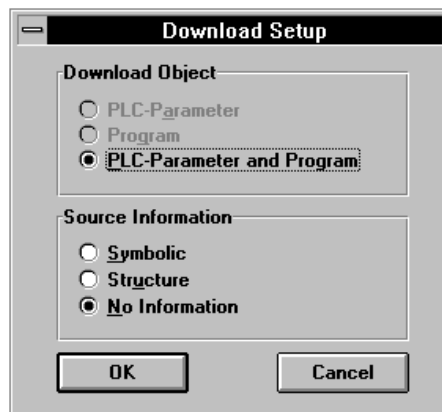
- ① Connect the PLC to your personal computer.
- ② Make sure that you plug the connection cable into the same port on the computer that you defined in the settings described in S10 above.



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** dialog box is displayed on the screen.



The **Download Setup** dialog box options are used to specify which data are downloaded to the PLC.

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



IMPORTANT: You must always download the PLC parameter when you transfer a program to the PLC for the first time!

- ③ Select **Transfer** in the **Project** menu, then select **MEDOC 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.

S12

Monitoring Programs

S12

In monitoring mode, MM+ 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.

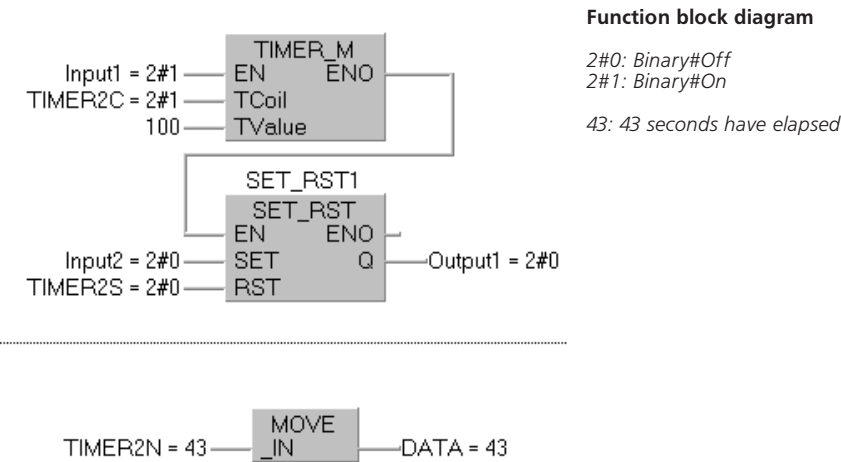


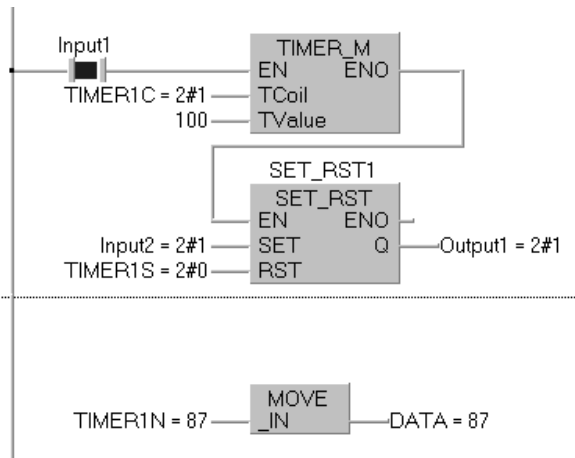
Important: *The PLC must be in RUN mode for monitoring to be possible.*

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



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





Ladder diagram

Filled field between the input contacts:
On

2#0: Binary#Off
2#1: Binary#On

87: 87 seconds have elapsed

LD	Input1	
TIMER_M	TIMER3C, 100	
CAL	SET_RST1(EN:=Input1, SET:=Input2, RST:=TIMER3S, Q:=Output1)	
LD	TIMER3N	38
MOVE		
ST	DATA	38

Instruction list

On/Off status is
indicated by
different colours.

S13 ——— Uploading Data from the CPU ——— S13

How to upload data from the PLC's CPU to MM+

- ① Select **Transfer** in the **Project** menu, then select the **PLC to MEDOC (MELSEC)** option.
- ② This displays the **PLC Parameter** dialog box. Select the appropriate **CPU Type** and confirm with **OK** (see S1).
- ③ In the next dialog box MM+ 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 S1.
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** dialog box.
- ⑤ This displays the **Transfer Setup (CPU port)** dialog box. Select the correct port for your system configuration (see S10).
- ⑥ Confirm your entries in both dialog boxes with **OK**.

This starts the upload procedure. Progress and any errors are documented in a list box.

Sample Program: CarPark

The arrowed references to the left of some of the headings below indicate where the instructions for the necessary procedures can be found in the 'Getting Started' chapter.

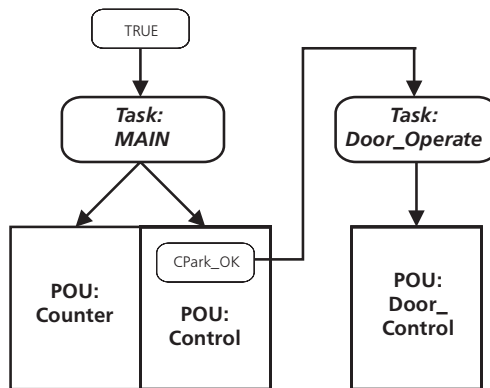


Note: This sample program is only intended as an illustration of programming and program structure techniques in MM+. In its present version it cannot be used as a basis for producing your own executable programs. The sample version has been written for a MELSEC FX series controller.

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.

Project Structure



The Task 'Main'

... always runs in the background, with maximum priority. This task contains the POU '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

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.
- Close the car park door after the car passes through the photoelectric barrier.



Note: MM+ 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 MM+; the system allows you to edit, add and delete variable declarations at any time, both during programming and afterwards.

➔ S1 Create the new 'CarPark' project

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.

➔ S2 Create the tasks

Create the 'Main' and 'Door_Operate' tasks.

➔ S3 Declare the global variables

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

Global Variable List									
	Class	Autoextern	Identifier	MIT-Addr.	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		Motor_Up	Y0	%QX0	BOOL	FALSE	Motor rolls car park door up	
3	VAR_GLOBAL		Motor_Down	Y1	%QX1	BOOL	FALSE	Motor rolls car park door down	
4	VAR_GLOBAL		Enter_Up	X2	%IX2	BOOL	FALSE	Key switch door open for entry	
5	VAR_GLOBAL		Exit_Up	X3	%IX3	BOOL	FALSE	Key switch door open for exit	
6	VAR_GLOBAL		Enter_Car_Gone	X4	%IX4	BOOL	FALSE	Photoelectric barrier for car entrance	
7	VAR_GLOBAL		Exit_Car_Gone	X5	%IX5	BOOL	FALSE	Photoelectric barrier for car exit	
8	VAR_GLOBAL		Max_Time_Up_C	TC0	%MX5.0	BOOL	FALSE	Timer coil: When the time has elapsed	
9	VAR_GLOBAL		Max_Time_Up_S	TS0	%MX3.0	BOOL	FALSE	Timer contact: When the time has elapsed	
10	VAR_GLOBAL		Main_Switch	X6	%IX6	BOOL	FALSE	Main switch: Car park door is in operation	
11	VAR_GLOBAL		Help_Alarm	X7	%IX7	BOOL	FALSE	Alarm switch	
12	VAR_GLOBAL		CO2_Alarm	X10	%IX8	BOOL	FALSE	CO2 sensor	
13	VAR_GLOBAL		CPark_OK	M0	%MX0.0	BOOL	FALSE	OK signal: Car park door can be used	
14	VAR_GLOBAL		CPark_OK_Lamp	Y3	%QX3	BOOL	FALSE	OK signal: Car park door can be used	
15	VAR_GLOBAL		Time_Control	M1	%MX0.1	BOOL	FALSE	Internal relay for the close process	
16	VAR_GLOBAL		Cars_Number	D0	%MW0.0	INT	0	Number of cars in the car park	

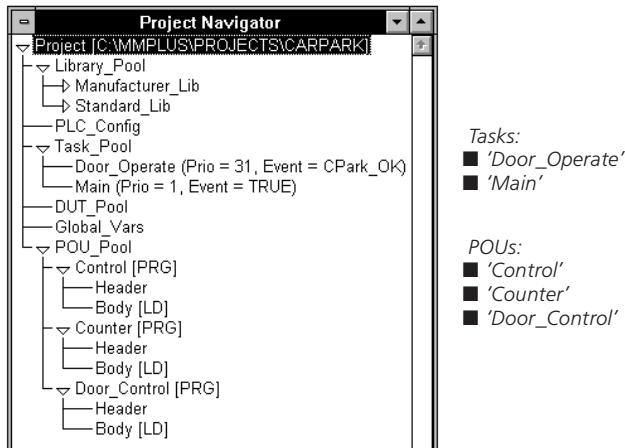
➔ S4 Create the program organisation units

Create the three program organisation units: 'Control', 'Counter' and 'Door_Control'. Define all three POU 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.





Project Navigator Window

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



➔ S5

Program the headers

All the variables used in the 'CarPark' sample program are declared as global variables. The next step is to declare the variables used in each POU in the POU headers. The quickest way to do this is to copy the variables in the global variable list ( ) and then paste them into the declaration tables of the headers of the individual POUs ( )



Note: The keyboard commands for selecting single and multiple lines in tables are listed in the appendix of the Reference Manual ("Table Keys").

Header of the 'Control' POU

Control [PRG] Header					
	Class	Identifier	Type	Initial	Comment
0	VAR_EXTEND	Door_Open	BOOL	FALSE	Upper door limit switch
1	VAR_EXTEND	Door_Closed	BOOL	FALSE	Lower door limit switch
2	VAR_EXTEND	Time_Control	BOOL	FALSE	Internal relay for the close process
3	VAR_EXTEND	Motor_Up	BOOL	FALSE	Motor rolls car park door up
4	VAR_EXTEND	Motor_Down	BOOL	FALSE	Motor rolls car park door down
5	VAR_EXTEND	Enter_Up	BOOL	FALSE	Key switch door open for entry
6	VAR_EXTEND	Exit_Up	BOOL	FALSE	Key switch door open for exit
7	VAR_EXTEND	Enter_Car_Gone	BOOL	FALSE	Photoelectric barrier for car entrance
8	VAR_EXTEND	Exit_Car_Gone	BOOL	FALSE	Photoelectric barrier for car exit
- 9	VAR_EXTEND	Max_Time_Up_C	BOOL	FALSE	Timer coil: When the time has elapsed the car park door is closed
- 10	VAR_EXTEND	Max_Time_Up_S	BOOL	FALSE	Timer contact: When the time has elapsed the car park door is closed
11	VAR_EXTEND	Main_Switch	BOOL	FALSE	Main switch: Car park door is in operation
12	VAR_EXTEND	Help_Alarm	BOOL	FALSE	Alarm switch
13	VAR_EXTEND	CO2_Alarm	BOOL	FALSE	CO2 sensor
14	VAR_EXTEND	CPark_OK	BOOL	FALSE	OK signal: Car park door can be used
15	VAR_EXTEND	CPark_OK_Lamp	BOOL	FALSE	OK signal: Car park door can be used

Header of the 'Counter' POU

Zaehler [PRG] Header					
	Schlüsselwort	Bezeichner	Typ	Vorgabe	Kommentar
0	VAR_EXTEND	Rein_Auto_Weg	BOOL	FALSE	Lichtschränke beim Hineinfahren
1	VAR_EXTEND	Raus_Auto_Weg	BOOL	FALSE	Lichtschränke beim Hinausfahren
2	VAR_EXTEND	Anzahl_Autos	INT	0	Anzahl der Autos, die sich in der Garage befinden

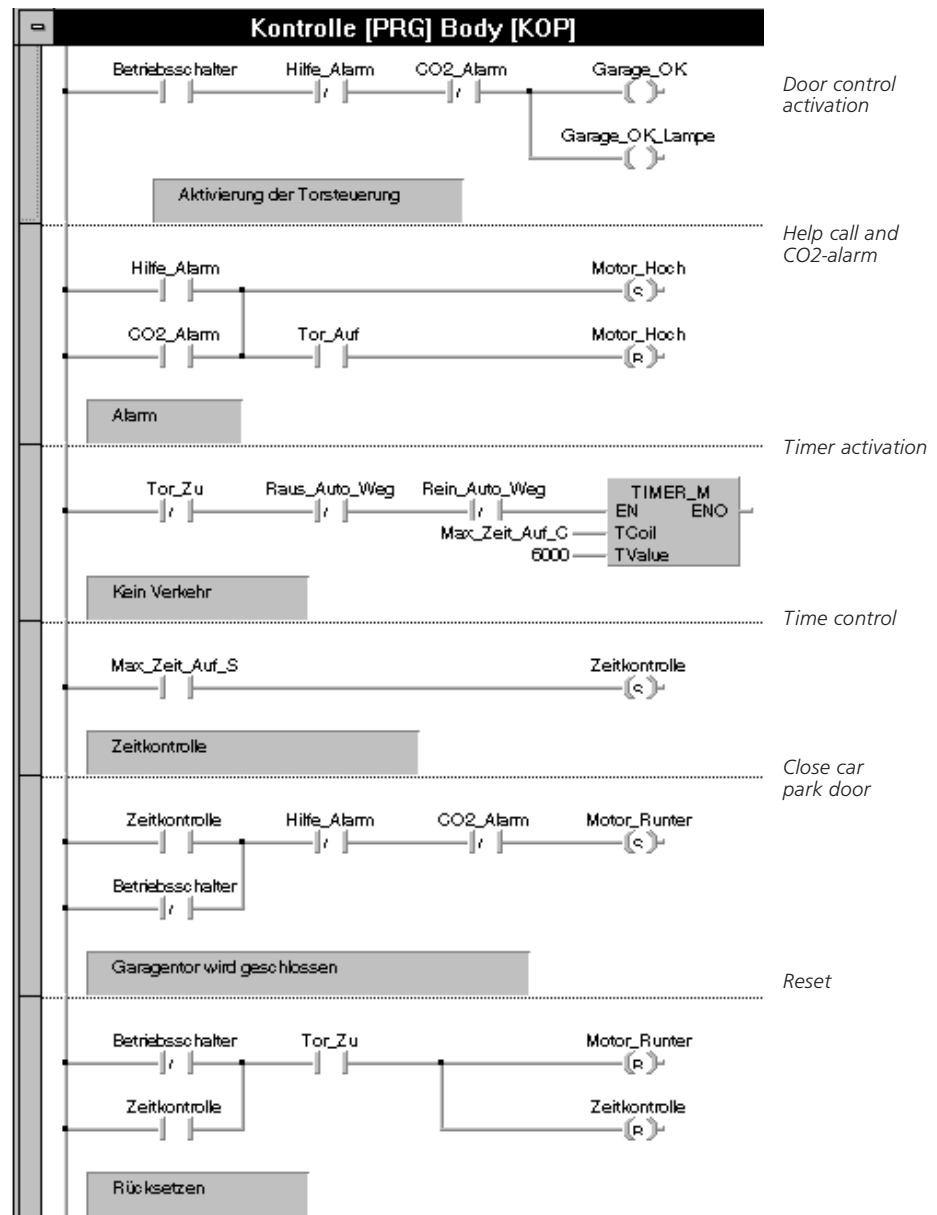
Header of the 'Door_Control' POU

Torsteuerung [PRG] Header					
	Schlüsselwort	Bezeichner	Typ	Vorgabe	Kommentar
0	VAR_EXTEND	Tor_Auf	BOOL	FALSE	Endschalter Tor oben
1	VAR_EXTEND	Tor_Zu	BOOL	FALSE	Endschalter Tor unten
2	VAR_EXTEND	Motor_Hoch	BOOL	FALSE	Motor zieht das Garagentor auf
3	VAR_EXTEND	Motor_Runter	BOOL	FALSE	Motor läßt das Garagentor runter
4	VAR_EXTEND	Rein_Auf	BOOL	FALSE	Schlüsselschalter beim Hineinfahren
5	VAR_EXTEND	Raus_Auf	BOOL	FALSE	Schlüsselschalter beim Hinausfahren
6	VAR_EXTEND	Rein_Auto_Weg	BOOL	FALSE	Lichtschränke beim Hineinfahren
7	VAR_EXTEND	Raus_Auto_Weg	BOOL	FALSE	Lichtschränke beim Hinausfahren
8	VAR_EXTEND	Max_Zeit_Auf_C	BOOL	FALSE	Timer-Spule: Nach Ablauf der Zeit wird das Garagentor geschlossen
9	VAR_EXTEND	Max_Zeit_Auf_S	BOOL	FALSE	Timer-Kontakt: Nach Ablauf der Zeit wird das Garagentor geschlossen

➔ S6

Program the bodies

Body of the 'Control' POU



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.

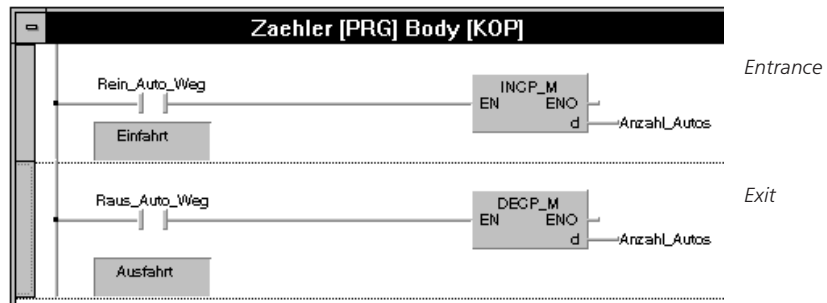
Close car park door

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.

Body of the 'Counter' POU



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

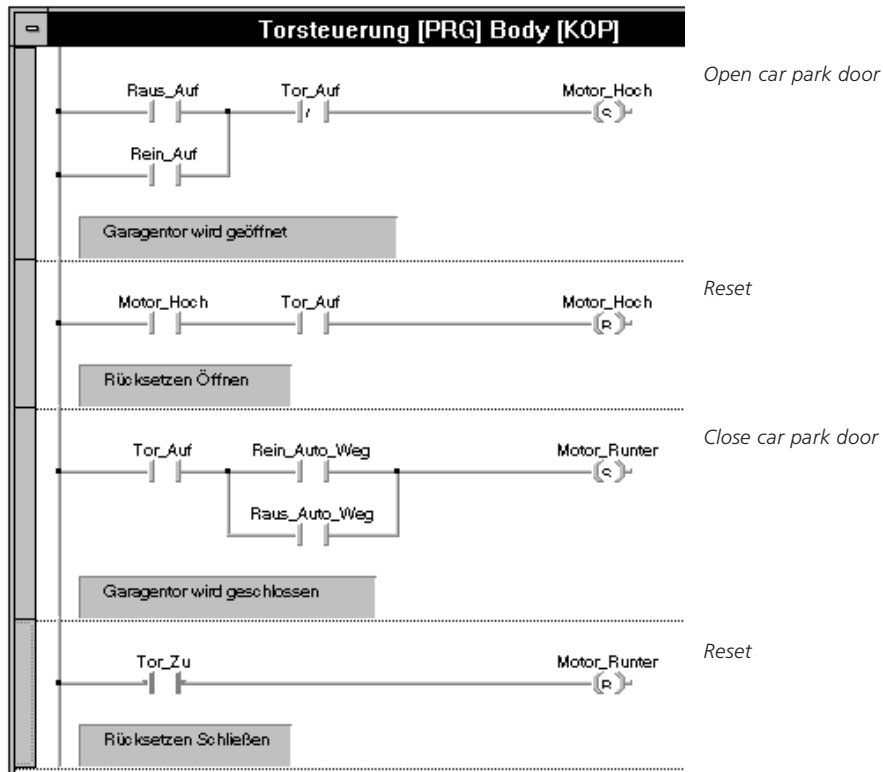
Every time a car drives out of the building the program decrements the number stored in the data register. The result is that the number always corresponds to the exact number of cars in the building.

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.



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.

➔ s8 **Configure the tasks**

We have already created the two tasks needed by the program, 'Main' and 'Door_Operate' (see S2).

The next step is to assign the POU's 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 POU's 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 dialog box.

The 'Main' task

Task Main (Prio = 1, Event = TRUE)		
	POU-Name	Comment
0	Control	
1	Counter	

Assign the POU's 'Control' and 'Counter' to the 'Main' task.

Task Information

Task Attributes

Event:

TRUE

Interval:

0

Priority:

1

OK

Cancel

Comment

Name:

Main

Size:

96 Bytes

Type:

TASK

☐ Timer/ Output Control

Last Change:

11.02.1998 16:42:09

Security Level

☒ 0

☐ 1

☐ 2

☐ 3

☐ 4

☐ 5

☐ 6

☐ 7

☒ Allow Read Access for lower Levels

Attributes of the 'Main' task

Event: TRUE

... i.e. the task's two POU's 'Control' and 'Counter' both run continuously.

Priority: 1

The 'Door_Operate' Task

Task Door_Operate (Prio = 31, Event = CPark_OK)		
	POU-Name	Comment
0	Door_Control	

The 'Door_Operate' task contains the POU 'Door_Control'.

Task Information

Task Attributes

Event:

CPark_OK

Interval:

0

Priority:

31

OK

Cancel

Comment

Name:

Door_Operate

Size:

97 Bytes

Type:

TASK

☒ Timer/ Output Control

Last Change:

11.02.1998 16:42:31

Security Level

☒ 0

☐ 1

☐ 2

☐ 3

☐ 4

☐ 5

☐ 6

☐ 7

☒ Allow Read Access for lower Levels

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.

- S9
- S10
- S11
- S12

Compile the project (S9),
configure the ports of your personal computer (S10) and
download the program to the controller CPU (S11).
Monitoring mode for following the status of the program variables is explained in S12.

Importing

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

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

Loading a print file to MM+

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 MM+

- 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 Open the POU body, then select **Import MEDOC Network** in the **Tools** menu.
- ⑥ 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 dialog box.
- ⑦ Confirm the settings with **OK**
(MEDOC Program = Instruction list only,
MEDOC Symbolic Names = Name list only).

The progress of the import procedure is documented in a status window.

A

Absolute address, 3-7
 Action, 3-18, 6-32
 Actual parameter, 3-4

B

Bit accumulator, 3-12
 Body, 3-2, 6-9

C

Checking, 6-34
 Class, 3-7, 3-8
 Colors, 6-3
 Comment, 3-7, 3-9
 Communications port, 6-38
 Compiling, 6-37

D

Data type, 3-7, 3-9
 Declaration table, 5-3
 Derived data type, 3-9
 Downloading, 6-39
 DUT Pool, 3-1

E

Editor, 5-5
 Elementary data type, 3-9
 Extended information, 6-3

F

Final step, 6-24
 Formal parameter, 3-4
 Function, 2-2, 3-2, 3-3, 3-14, 3-16, 3-17
 Function block, 2-2, 3-2, 3-3, 3-13, 3-16, 3-17
 Function Block Diagram, 2-4, 3-10, 3-17, 5-5, 6-12

G

Global variable, 2-5, 3-1, 3-7, 6-5
 Graphical editor, 2-4, 3-15, 5-5

H

Header, 3-2, 6-8
 Help, 1-2

I

Identifier, 3-7, 3-8
 IEC address, 3-8
 IEC Instruction List, 3-10, 5-5
 Importing, 8-1, 8-2
 Initial step, 3-19, 6-24
 Initial value, 3-7, 3-9
 Installing, 4-1
 Instance, 3-4
 Instancing, 3-4
 Instruction List, 2-4, 3-10

J

Jump entry point, 3-19, 6-29
 Jump exit point, 3-19, 6-30

L

Ladder Diagram, 2-4, 3-10, 3-15, 5-5, 6-10
 Local variable, 2-5, 3-7

M

Macro step, 3-20
 Manufacturer library, 3-1
 MELSEC Instruction List, 3-10, 5-5
 Menu bar, 5-2
 MITSUBISHI address, 3-8
 Monitoring, 6-40

N

Navigator, 5-2
 Negated contact/coil/variable, 6-15

P

Parameter, 3-4
 PLC configuration, 3-1
 POU Pool, 3-1
 Program, 2-2, 3-2, 3-3
 Program Organisation Unit, 2-2, 3-2, 6-7
 Programming language, 3-10
 Project, 3-1, 6-2

S

Sequential Function Chart, 2-4, 3-10, 3-18, 5-5, 6-22
 Signal configuration, 6-15
 Standard library, 3-1
 Status bar, 5-2
 Step, 3-18
 Structured programming, 2-2
 Syntax check, 6-34

T

Task, 2-3, 3-6, 6-4, 6-35
 Task Pool, 3-1
 Text editor, 2-4, 3-10, 5-5
 Timer, 6-16
 Toolbar, 5-2
 Transition, 3-18
 Transition condition, 3-18, 6-31

U

Uploading, 6-42
 User interface, 5-1

V

Variable, 2-5, 3-7

W

Windows, 5-2
 Wizard, 6-3

