SIEMENS

SIMATIC

Working with STEP 7 V5.1

Getting Started

This manual is part of the documentation package with the order number :
6ES7 810-4CA05-8BA0

Edition 08/2000
A5E00069681-03

Safety Guidelines

This manual contains notices which you should observe to ensure your own personal safety, as well as to protect the product and connected equipment. These notices are highlighted in the manual by a warning triangle and are marked as follows according to the level of danger:

Danger
indicates that death, severe personal injury or substantial property damage will result if proper precautions are not taken.

Warning
indicates that death, severe personal injury or substantial property damage can result if proper precautions are not taken.

Caution
indicates that minor personal injury or property damage can result if proper precautions are not taken.

Note
draws your attention to particularly important information on the product, handling the product, or to a particular part of the documentation.

Qualified Personnel

Only qualified personnel should be allowed to install and work on this equipment. Qualified persons are defined as persons who are authorized to commission, to ground, and to tag circuits, equipment, and systems in accordance with established safety practices and standards.

Correct Usage

Note the following:

Warning
This device and its components may only be used for the applications described in the catalog or the technical descriptions, and only in connection with devices or components from other manufacturers which have been approved or recommended by Siemens.

This product can only function correctly and safely if it is transported, stored, set up, and installed correctly, and operated and maintained as recommended.

Trademarks

SIMATIC®, SIMATIC HMI® and SIMATIC NET® are registered trademarks of SIEMENS AG.

Some of other designations used in these documents are also registered trademarks; the owner's rights may be violated if they are used by third parties for their own purposes.

Siemens AG
Bereich Automatisierungs- und Antriebstechnik
Geschaeftsgebiet Industrie-Automatisierungssysteme
Postfach 4848, D- 90327 Nuernberg

Disclaimer of Liability

We have che#ked the contents of this manual for agreement with the hardware and software described. Since deviations cannot be precluded entirely, we cannot guarantee full agreement. However, the data in this manual are reviewed regularly and any necessary corrections included in subsequent editions. Suggestions for improvement are welcomed.

Welcome to STEP 7...

...the SIMATIC standard software for creating programmable logic control programs in Ladder Logic, Function Block Diagram, or Statement List for SIMATIC S7-300/400 stations.

About This Getting Started Manual

In this manual, you will get to know the basics of SIMATIC STEP 7. We will show you the most important screen dialog boxes and the procedures to follow using practical exercises, which are structured so that you can start with almost any chapter.

Each section is split into two parts: a descriptive part, marked in gray, and a process-oriented part, marked in green. The instructions start with an arrow in the green margin and may be spread out over several pages, finishing in a full stop and a box containing related topics.

Previous experience of working with the mouse, window handling, pull-down menus, etc. would be useful, and you should preferably be familiar with the basic principles of programmable logic control.

The STEP 7 training courses provide you with in-depth knowledge above and beyond the contents of this Getting Started manual, teaching you how entire automation solutions can be created with STEP 7.

Requirements for Working with the Getting Started Manual

In order to carry out the practical exercises for STEP 7 in this Getting Started manual, you require the following:

- A Siemens programming device or a PC
- The STEP 7 software package and the authorization diskette
- A SIMATIC S7-300 or S7-400 programmable controller
  (for Chapter 7 "Downloading and Debugging the Program").

Additional Documentation on STEP 7

- STEP 7 Basic Information
- STEP 7 Reference Information

After you have installed STEP 7, you will find the electronic manuals in the Start menu under **Simatic > Documentation** or alternatively, you can order them from any Siemens sales center. All of the information in the manuals can be called up in STEP 7 from the online help.

Have fun and good luck!

SIEMENS AG

Contents

In Chapters 6 and 7, you configure the hardware and test your program.

In Chapters 8 to 11, you can extend your knowledge to include new functions.

1 Introduction to STEP 7

1.1 What You Will Learn

Using practical exercises, we will show you how easy it is to program in Ladder Logic, Statement List, or Function Block Diagram with STEP 7.

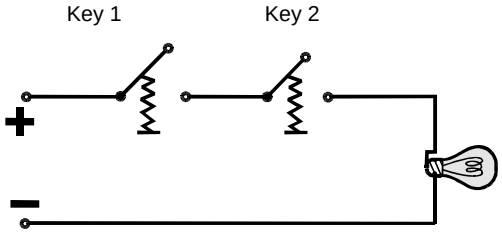
Detailed instructions in the individual chapters will show you step-by-step the many ways in which you can use STEP 7.

Creating a Program with Binary Logic

In Chapters 2 to 7, you will create a program with binary logic. Using the programmed logic operations, you will address the inputs and outputs of your CPU (if present).

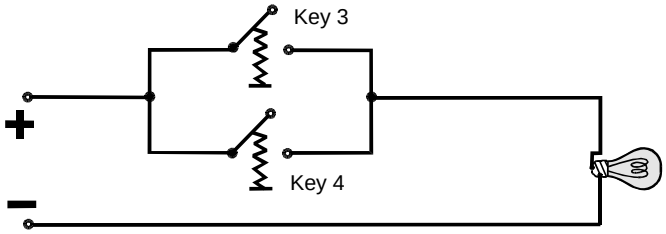
The programming examples in the Getting Started manual are based, among other things, on three fundamental binary logic operations.

The first binary logic operation, which you will program later on, is the AND function. The AND function can be best illustrated in a circuit diagram using two keys.



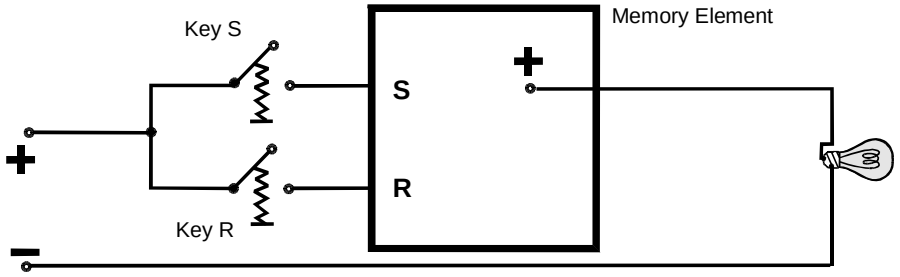
If both Key 1 **and** Key 2 are pressed, the bulb lights up.

The second binary logic operation is the OR function. The OR function can also be represented in a circuit diagram.



If **either** key 3 **or** key 4 is pressed, the bulb lights up.

The third binary logic operation is the memory element. The SR function reacts within a circuit diagram to certain voltage states and passes these on accordingly.

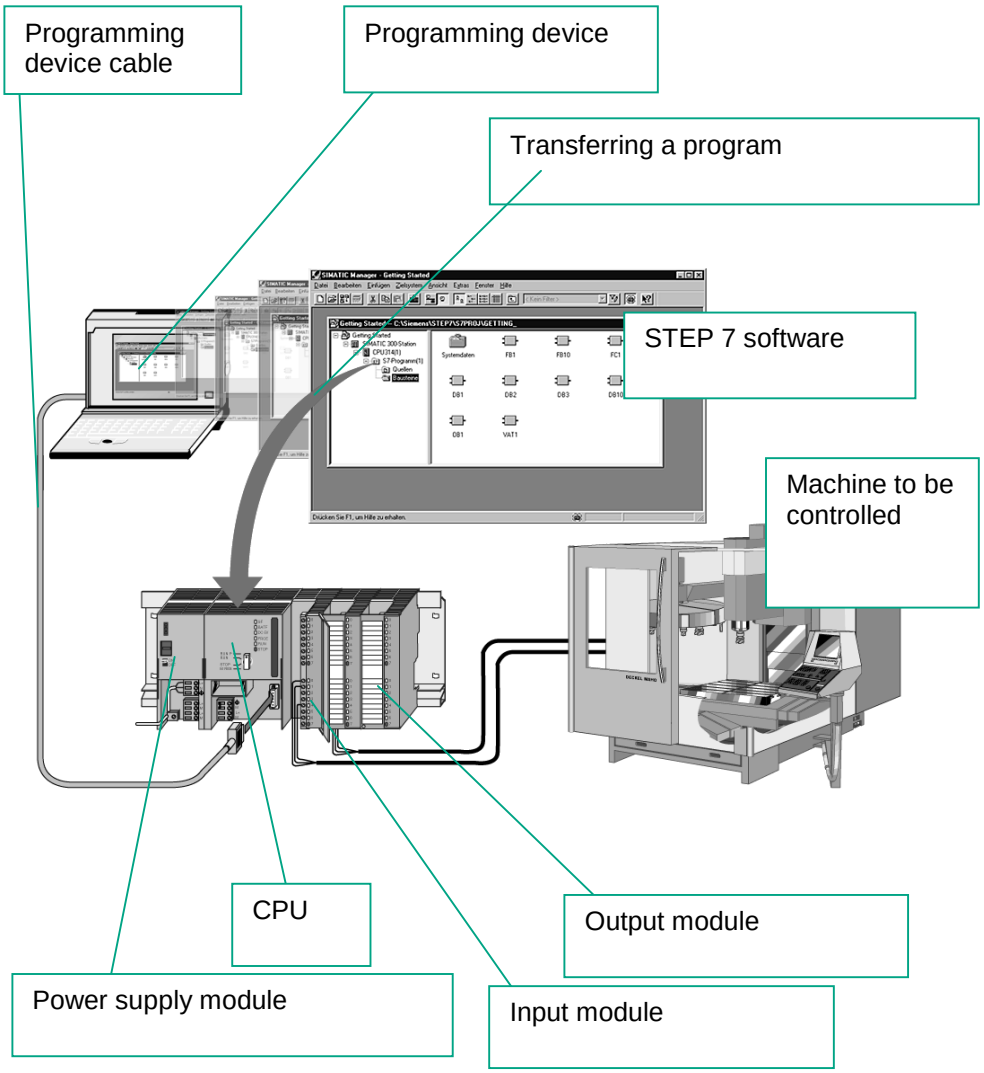


If key S is pressed, the bulb lights up and remains lit until key R is pressed.

1.2 Combining Hardware and Software

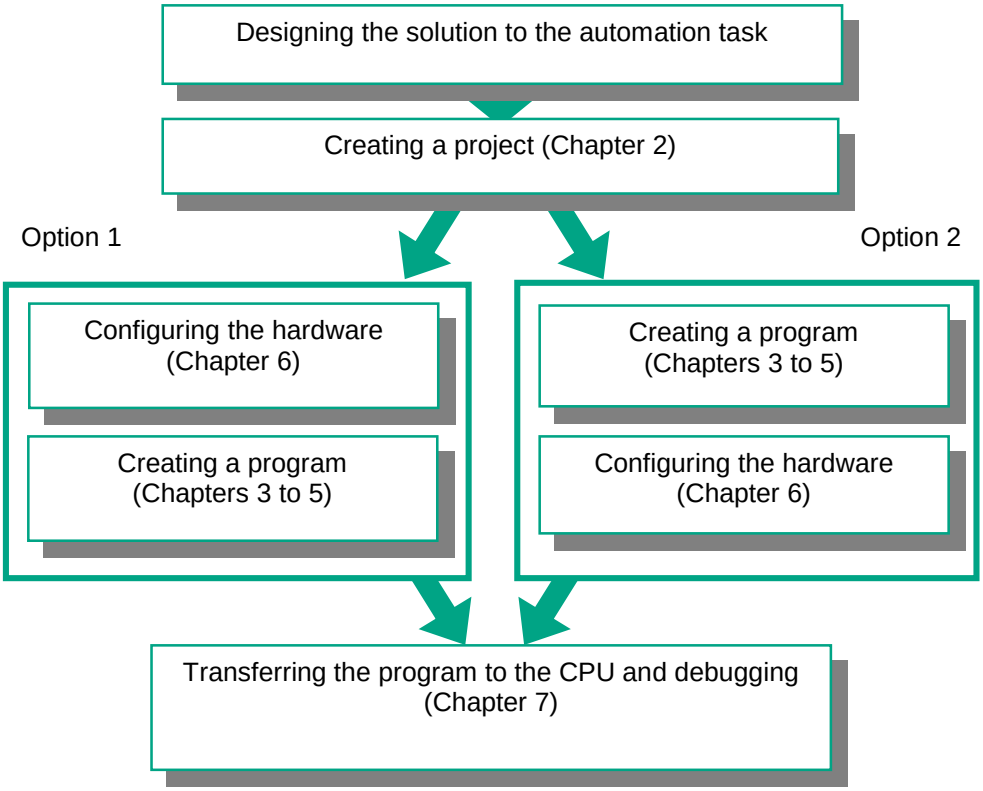
Using the STEP 7 software, you can create your S7 program within a project. The S7 programmable controller consists of a power supply unit, a CPU, and input and output modules (I/O modules).

The programmable logic controller (PLC) monitors and controls your machine with the S7 program. The I/O modules are addressed in the S7 program via the addresses.



1.3 Basic Procedure Using STEP 7

Before you create a project, you should know that STEP 7 projects can be created in different orders.



If you are creating comprehensive programs with many inputs and outputs, we recommend you configure the hardware first. The advantage of this is that STEP 7 displays the possible addresses in the Hardware Configuration Editor.

If you choose the second option, you have to determine each address yourself, depending on your selected components and you cannot call these addresses via STEP 7.

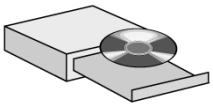
In the hardware configuration, not only can you define addresses, but you can also change the parameters and properties of modules. If you want to operate several CPUs, for example, you have to match up the MPI addresses of the CPUs.

Since we are only using a small number of inputs and outputs in the Getting Started manual, we will skip the hardware configuration for now and start with the programming.

1.4 Installing STEP 7

Regardless of whether you want to start with programming or configuring hardware, you first have to install STEP 7. If you are using a SIMATIC programming device, STEP 7 is already installed.

When installing the STEP 7 software on a programming device or PC without a previously installed version of STEP 7, note the software and hardware requirements. You can find these in the Readme.wri on the STEP 7 CD under **<Drive>:\STEP 7 \Disk1**.



If you need to install STEP 7 first, insert the STEP 7 CD in the CD-ROM drive now. The installation program starts automatically. Follow the instructions on the screen.

If the installation does not start automatically, you can also find the installation program on the CD-ROM under **<Drive>:\STEP 7 \Disk1\setup.exe**.

Once the installation is complete and you have restarted the computer, the "SIMATIC Manager" icon will appear on your Windows desktop.

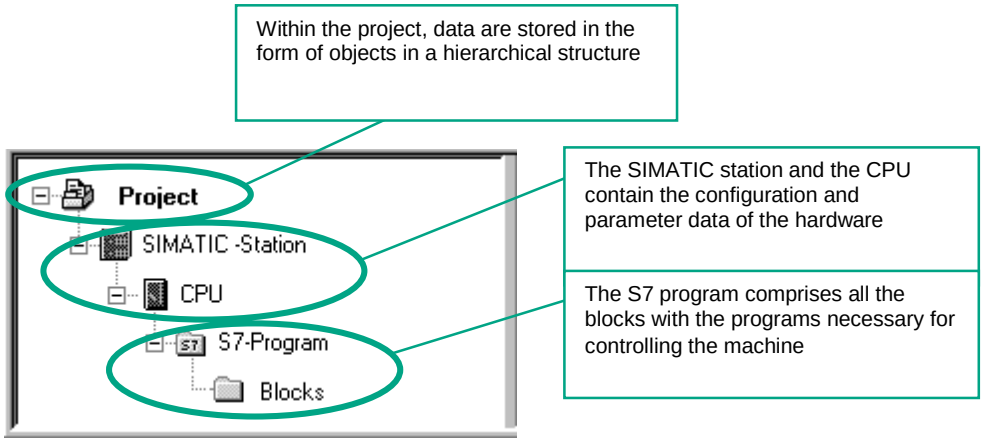
If you double-click the "SIMATIC Manager" icon following installation, the STEP 7 Wizard will be started automatically.

You can find additional notes on installation in the Readme.wri file on the STEP 7 CD under **<Drive>:\STEP 7 \Disk1\Readme.wri**.

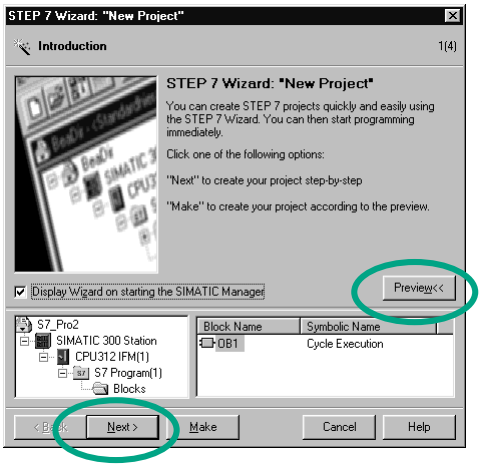
2 The SIMATIC Manager

2.1 Starting the SIMATIC Manager and Creating a Project

The SIMATIC Manager is the central window which becomes active when STEP 7 is started. The default setting starts the STEP 7 Wizard, which supports you when creating a STEP 7 project. The project structure is used to store and arrange all the data and programs in order.

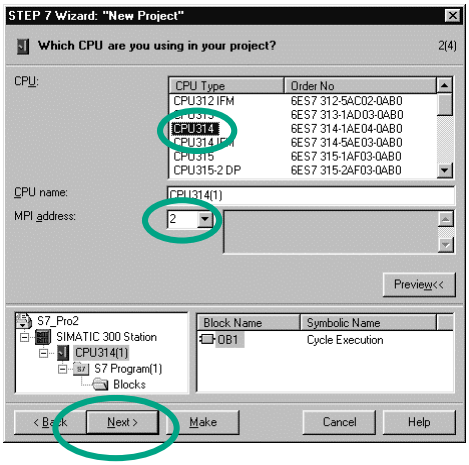


Double-click the **SIMATIC Manager** icon. The STEP 7 Wizard is activated.



In the **preview**, you can toggle the view of the project structure being created on and off.

To move to the next dialog box, click **Next**.



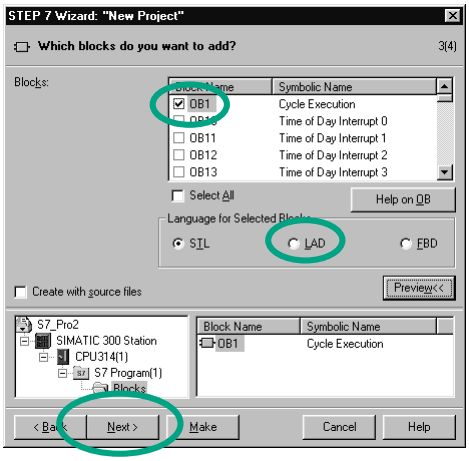
For the "Getting Started" sample project, select CPU 314. The example has been created in such a way that you can actually select the CPU you have been supplied with at any time.

The default setting for the MPI address is 2.

Click **Next** to confirm the settings and move to the next dialog box.

Every CPU has certain properties; for example, regarding its memory configuration or address areas. This is why you have to select the CPU before you start programming.

The MPI address (multipoint interface) is required in order for your CPU to communicate with your programming device or PC.



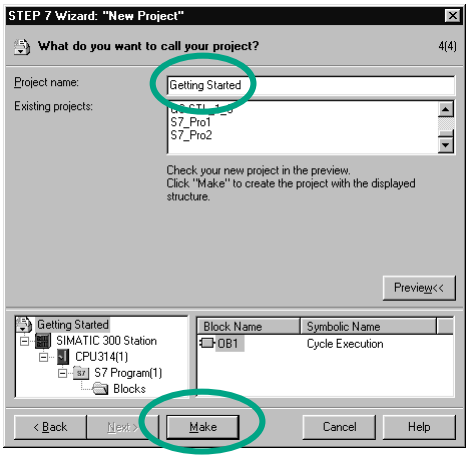
Select the organization block **OB1** (if this is not already selected).

Select one of the programming languages: Ladder Logic **(LAD)**, Statement List **(STL)**, or Function Block Diagram **(FBD)**.

Confirm your settings with **Next**.

OB1 represents the highest programming level and organizes the other blocks in the S7 program.

You can change the programming language again at a later date.



Double-click to select the suggested name in the "Project name" field and overwrite it with "Getting Started."

Click **Make** to generate your new project according to the preview.

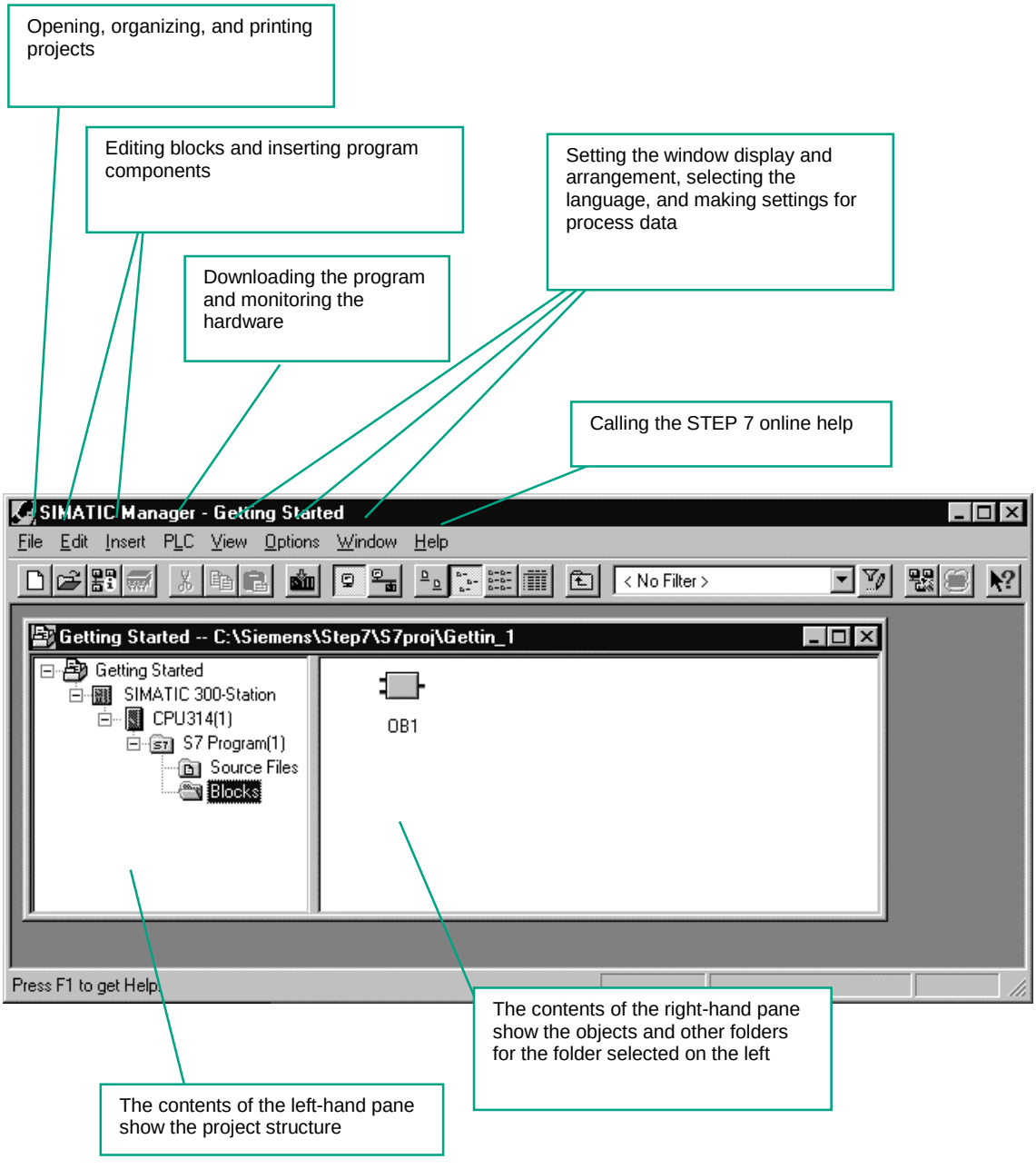
When you click the **Make** button, the SIMATIC Manager will open with the window for the "Getting Started" project you have created. On the following pages, we will show you what the created files and folders are for and how you can work effectively with them.

The STEP 7 Wizard is activated each time the program is started. You can deactivate this default setting in the first dialog box for the Wizard. However, if you create projects without the STEP 7 Wizard, you must create each directory within the project yourself.

You can find more information under **Help > Contents** in the topic "Setting Up and Editing the Project."

2.2 The Project Structure in the SIMATIC Manager and How to Call the Online Help

As soon as the STEP 7 Wizard is closed, the SIMATIC Manager appears with the open project window "Getting Started." From here, you can start all the STEP 7 functions and windows.

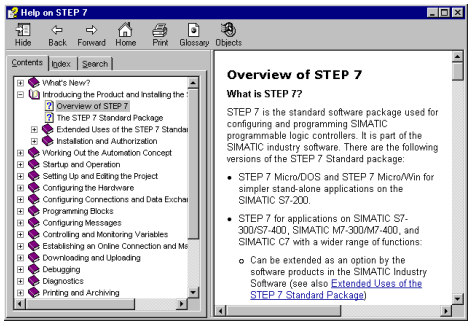


Calling the Help on STEP 7

F1

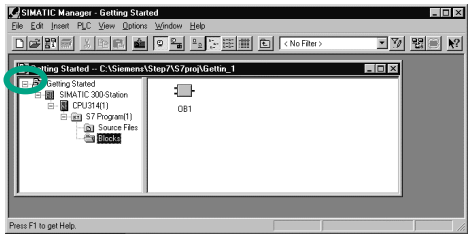
Option 1:

Place the cursor on any menu command and press the **F1** key. The context-sensitive help for the selected menu command will appear.



Option 2:
Use the menu to open the STEP 7 online help.
The contents page with various help topics appears in the left-hand pane and the selected topic is displayed in the right-hand pane.
Navigate to the topic you want by clicking the **+** sign in the **Contents** list. At the same time, the contents of the selected topic are displayed in the right-hand pane.
Using **Index** and **Find**, you can enter search strings and look for the specific topics you require.
Option 3:
Click the question mark button in the toolbar to turn your mouse into a help cursor. The next time you click on a specific object, the online help is activated.

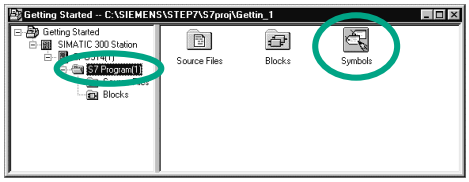
Navigating in the Project Structure



The project you have just created is displayed with the selected S7 station and CPU.

Click the **+** or **–** sign to open or close a folder.

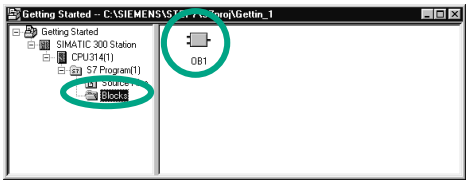
You can start other functions later on by clicking the symbols displayed in the right-hand pane.



Click the **S7 Program (1)** folder. This contains all the necessary program components.

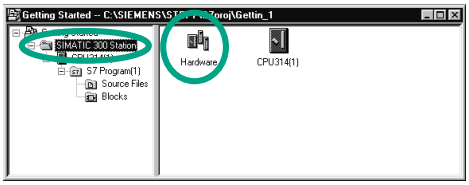
You will use the Symbols component in Chapter 3 to give the addresses symbolic names.

The Source Files component is used to store source file programs. These are not dealt with in the Getting Started manual.



Click the **Blocks** folder. This contains the **OB1** you have already created and, later on, all the other blocks.

From here, you will start programming in Ladder Logic, Statement List, or Function Block Diagram in Chapters 4 and 5.



Click the **SIMATIC 300 Station** folder. All the hardware-related project data are stored here.

You will use the Hardware component in Chapter 6 to specify the parameters of your programmable controller.

If you require further SIMATIC software for your automation task; for example, the optional packages PLCSIM (hardware simulation program) or S7 Graph (graphic programming language), these are also integrated in STEP 7. Using the SIMATIC Manager, for example, you can directly open the relevant objects such as an S7 Graph function block.

You can find more information under **Help > Contents** in the topics "Working Out the Automation Concept" and "Basics of Designing the Program Structure."

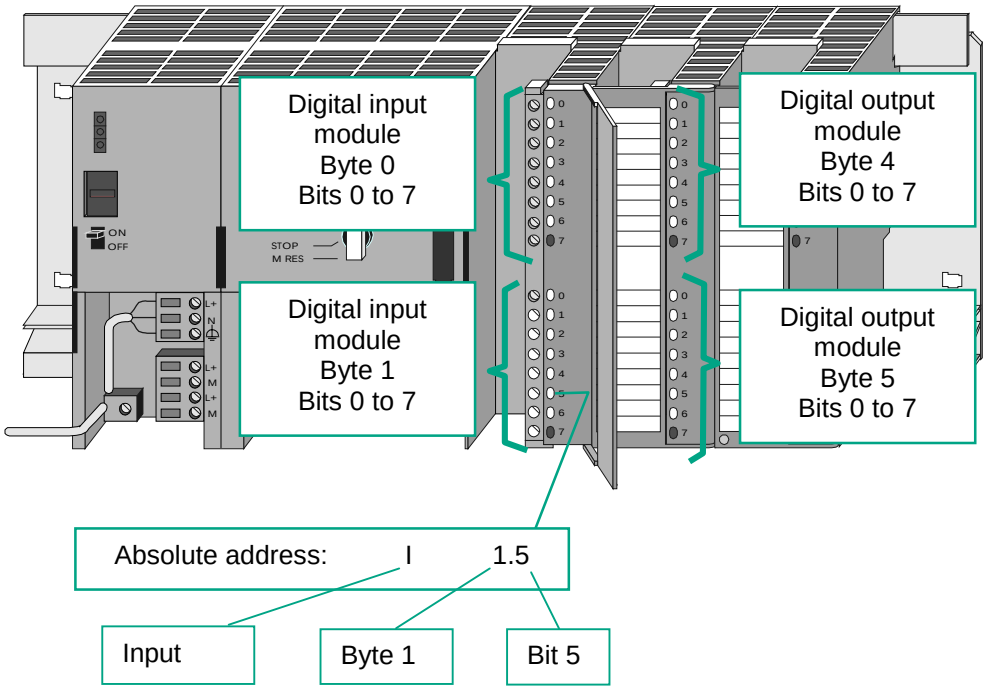
You can find more information on optional packages in the SIMATIC catalog ST 70, "Components for Completely Integrated Automation."

3 Programming with Symbols

3.1 Absolute Addresses

Every input and output has an absolute address predefined by the hardware configuration. This address is specified directly; that is, absolutely.

The absolute address can be replaced by any symbolic name you choose.



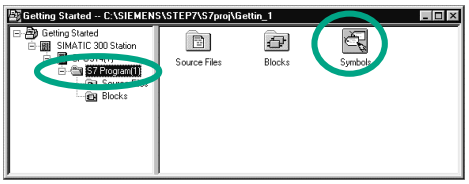
You should only use absolute programming if you do not have to address many inputs and outputs in your S7 program.

3.2 Symbolic Programming

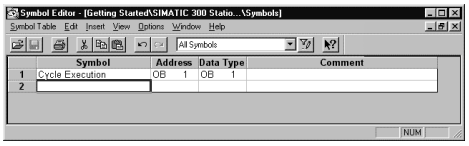
In the symbol table, you assign a symbolic name and the data type to all the absolute addresses which you will address later on in your program; for example, for input I 0.1 the symbolic name Key 1. These names apply to all parts of the program and are known as global variables.

Using symbolic programming, you can considerably improve the legibility of the S7 program you have created.

Working with the Symbol Editor



Navigate in the project window "Getting Started" until you reach **S7 Program (1)** and double-click to open the **Symbols** component.



Your symbol table currently only consists of the predefined organization block OB1.

	Symbol	Address	Data Type
1	Cycle Execution	OB 1	OB 1
2			

Click **Cycle Execution** and overwrite it with "Main Program" for our example.

	Symbol	Address	Data Type
1	Main Program	OB 1	OB 1
2	Green Light	Q 4.0	BOOL

Enter "Green Light" and "Q 4.0" in row 2. The data type is added automatically.

Comment

Click in the comment column of row 1 or 2 to enter a comment on the symbol. You complete your entries in a row by pressing **Enter**, which then adds a new row.

	Symbol	Address	Data Type
1	Main Program	OB 1	OB 1
2	Green Light	Q 4.0	BOOL
3	Red Light	Q 4.1	BOOL

Enter "Red Light" and "Q 4.1" in row 3 and press Enter to complete the entry.

In this way, you can assign symbolic names to all the absolute addresses of the inputs and outputs which your program requires.

Save the entries or changes you have made in the symbol table and close the window.

Because there are lots of names for the entire "Getting Started" project, you can copy the symbol table to your "Getting Started" project in Section 4.1.

Symbol Editor - [Getting Started\SIMATIC 300 Statio...\Symbols]

Symbol Table Edit Insert View Options Window Help

All Symbols

	Symbol	Address	Data Type	Comment
1	Automatic_Mode	Q 4.2	BOOL	Retentive output
2	Automatic_On	I 0.5	BOOL	For the memory function (switch on)
3	DE_Actual_Speed	MW 4	INT	Actual speed for diesel engine
4	DE_Failure	I 1.6	BOOL	Diesel engine failure
5	DE_Fan_On	Q 5.6	BOOL	Command for switching on diesel engine fan
6	DE_Follow_On	T 2	TIMER	Follow-on time for diesel engine fan
7	DE_On	Q 5.4	BOOL	Command for switching on diesel engine
8	DE_Preset_Speed_Reache	Q 5.5	BOOL	Display "Diesel engine preset speed reached"
9	Diesel	DB 2	FB 1	Data for diesel engine
10	Engine	FB 1	FB 1	Engine control
11	Engine_Data	DB 10	FB 10	Instance data block for FB10
12	Engines	FB 10	FB 10	Example of multiple instances
13	Fan	FC 1	FC 1	Fan control
14	Green_Light	Q 4.0	BOOL	Result of AND query
15	Key_1	I 0.1	BOOL	For the AND query
16	Key_2	I 0.2	BOOL	For the AND query
17	Key_3	I 0.3	BOOL	For the OR query
18	Key_4	I 0.4	BOOL	For the OR query
19	Main_Program	OB 1	OB 1	This block contains the user program
20	Manual_On	I 0.6	BOOL	For the memory function (switch off)
21	PE_Actual_Speed	MW 2	INT	Actual speed for petrol engine
22	PE_Failure	I 1.2	BOOL	Petrol engine failure
23	PE_Fan_On	Q 5.2	BOOL	Command for switching on petrol engine fan
24	PE_Follow_On	T 1	TIMER	Follow-on time for petrol engine fan
25	PE_On	Q 5.0	BOOL	Command for switching on petrol engine
26	PE_Preset_Speed_Reache	Q 5.1	BOOL	Display "Petrol engine preset speed reached"
27	Petrol	DB 1	FB 1	Data for petrol engine
28	Red_Light	Q 4.1	BOOL	Result of OR query
29	S_Data	DB 3	DB 3	Shared data block
30	Switch_Off_DE	I 1.5	BOOL	Switch off diesel engine
31	Switch_Off_PE	I 1.1	BOOL	Switch off petrol engine
32	Switch_On_DE	I 1.4	BOOL	Switch on diesel engine
33	Switch_On_PE	I 1.0	BOOL	Switch on petrol engine
34				

Press F1 for help. NUM

Here you can see the symbol table for the S7 program in the "Getting Started" example for Statement List.

Generally speaking, only one symbol table is created per S7 program, regardless of which programming language you have selected.

All printable characters (for example, special characters, spaces) are permitted in the symbol table.

The data type which was previously added automatically to the symbol table determines the type of the signal to be processed for the CPU. STEP 7 uses, among others, the following data types:

BOOL BYTE WORD DWORD	Data of this type are bit combinations. 1 bit (type BOOL) to 32 bits (DWORD).
CHAR	Data of this type occupy exactly one character of the ASCII character set.
INT DINT REAL	They are available for the processing of numerical values (for example, to calculate arithmetic expressions).
S5TIME TIME DATE TIME_OF_DAY	Data of this type represent the different time and date values within STEP 7 (for example, to set the date or to enter the time value for a timer).

You can find more information under **Help > Contents** in the topics “Programming Blocks“ and "Defining Symbols".

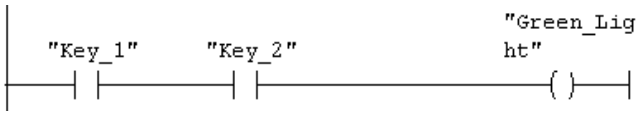
4 Creating a Program in OB1

4.1 Opening the LAD/STL/FBD Program Window

Choosing Ladder Logic, Statement List, or Function Block Diagram

With STEP 7, you create S7 programs in the standard languages Ladder Logic (LAD), Statement List (STL), or Function Block Diagram (FBD). In practice, and for this chapter too, you must decide which language to use.

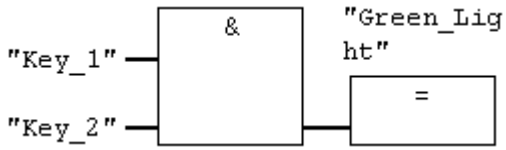
Ladder Logic (LAD)
Suitable for users from the electrical engineering industry, for example.



Statement List (STL)
Suitable for users from the world of computer technology, for example.

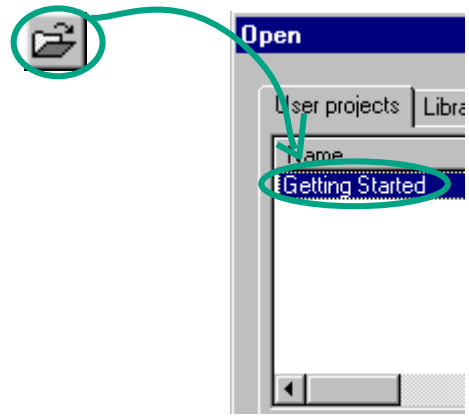
```
A     "Key_1"
A     "Key_2"
=     "Green_Light"
```

Function Block Diagram (FBD)
Suitable for users from the world of circuit engineering, for example.



The block OB1 will now be opened according to the language you chose when you created it in the project Wizard. However, you can change the default programming language again at any time.

Copying the Symbol Table and Opening OB1



If necessary, open your "Getting Started" project. To do this, click the **Open** button in the toolbar, select the "Getting Started" project you created, and confirm with **OK**.

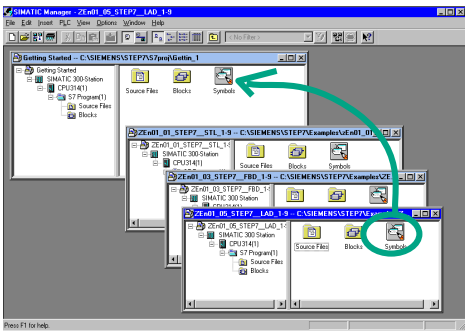
Depending on which programming language you have decided to use, open one of the following projects as well:

- ZEn01_05_STEP7__**LAD**_1-9
- ZEn01_01_STEP7__**STL**_1-9

or

- ZEn01_03_STEP7__**FDB**_1-9

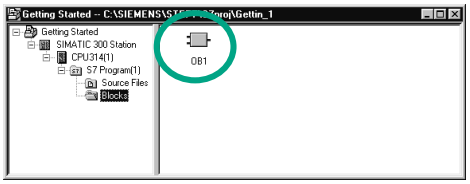
Here you can see all three sample projects displayed.



Navigate in the „ZEn01_XXX“ until you reach the **Symbols** component and copy this by dragging and dropping it to the **S7 Program** folder in your project window "Getting Started."

Then close the window „ZEn01_XXX“.

Drag and drop means that you click any object with the mouse and move it whilst keeping the mouse button depressed. When you release the mouse button, the object is pasted at the selected position.



Double-click **OB1** in the "Getting Started" project. The LAD/STL/FBD program window is opened.

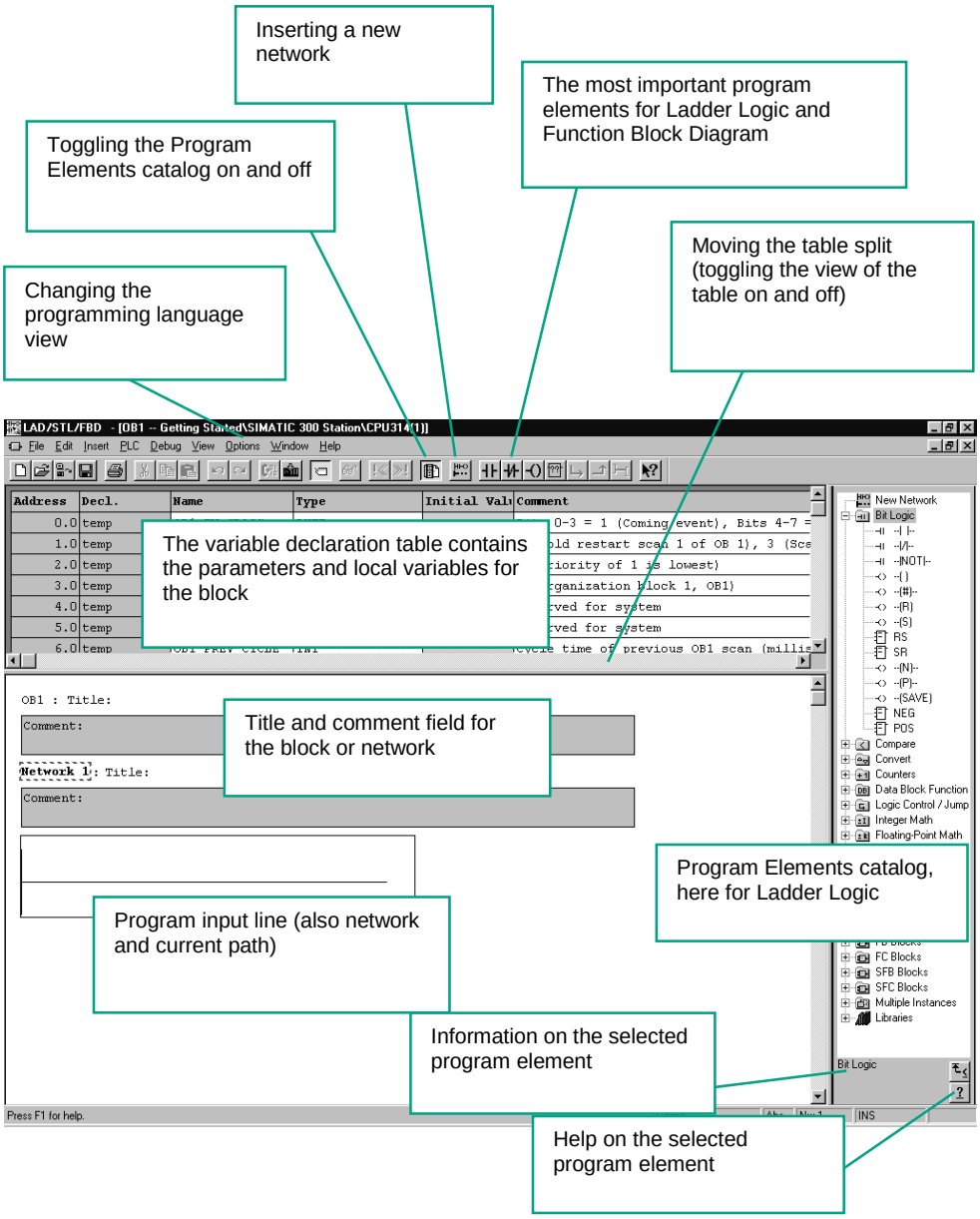
In STEP 7, OB1 is processed cyclically by the CPU. The CPU reads line by line and executes the program commands. When the CPU returns to the first program line, it has completed exactly one cycle. The time required for this is known as the scan cycle time.

Depending on which programming language you have selected, continue reading in either Section 4.2 for programming in Ladder Logic, Section 4.3 for Statement List, or Section 4.4 for Function Block Diagram.

You can find more information under **Help > Contents** in the topics “Programming Blocks“ and "Creating Blocks and Libraries.“

The LAD/STL/FBD Program Window

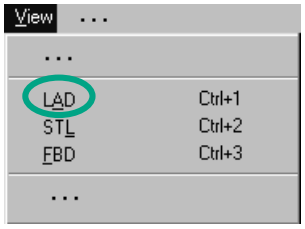
All blocks are programmed in the LAD/STL/FBD program window. Here, you can see the view for Ladder Logic.



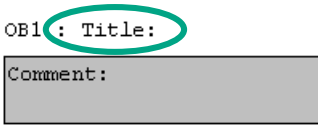
4.2 Programming OB1 in Ladder Logic

In the following section, you will program a series circuit, a parallel circuit, and the set / reset memory function in Ladder Logic (LAD).

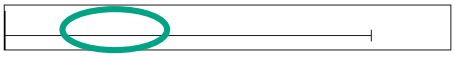
Programming a Series Circuit in Ladder Logic



If necessary, set **LAD** as the programming language in the **View** menu.



Click in the **title** area of OB1 and enter "Cyclically processed main program," for example.



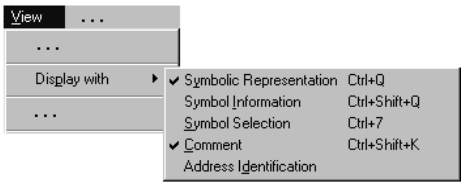
Select the current path for your first element.

Click the button in the toolbar and insert a normally open contact.

In the same way, insert a second normally open contact.

Insert a coil at the right-hand end of the current path.

The addresses of the normally open contacts and the coil are still missing in the series circuit.

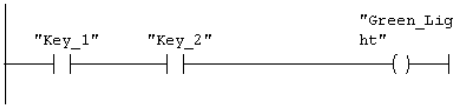


Check whether symbolic representation is activated.

Click the **??.?** sign and enter the symbolic name "Key_1" (in quotation marks).
Confirm with **Enter**.

Enter the symbolic name "Key_2" for the second normally open contact.

Enter the name "Green_Light" for the coil.



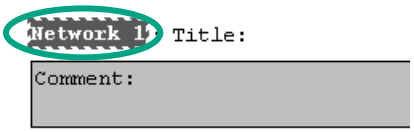
You have now programmed a complete series circuit.

Save the block if there are no more symbols shown in red.

Symbols are indicated in red if, for example, they do not exist in the symbol table, or if there is a syntax error.

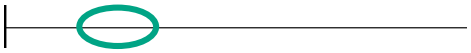
You can also insert the symbolic name directly from the symbol table. Click the **??.?** sign and then the menu command **Insert > Symbol**. Scroll through the pull-down list until you reach the corresponding name and select it. The symbolic name is added automatically.

Programming a Parallel Circuit in Ladder Logic



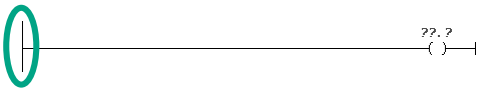
Select **Network 1**.

Insert a new network.



Select the current path again.

Insert a normally open contact and a coil.



Select the vertical line of the current path.

Insert a parallel branch.

Add another normally open contact in the parallel branch.

Close the branch (if necessary, select the lower arrow).

The addresses are still missing in the parallel circuit.

To assign symbolic addresses, proceed in the same way as for the series circuit.

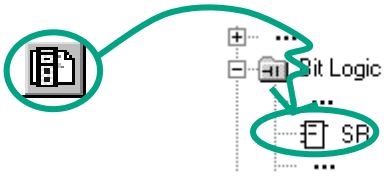
Overwrite the upper normally open contact with "Key_3," the lower contact with "Key_4," and the coil with "Red_Light."

Save the block.

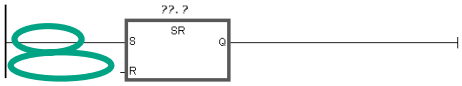
Programming a Memory Function in Ladder Logic

Select Network 2 and insert another network.

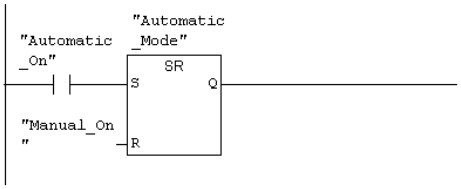
Select the current path again.



Navigate in the Program Elements catalog under **Bit Logic** until you reach the **SR** element. Double-click to insert the element.



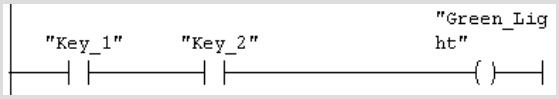
Insert a normally open contact in front of each of the inputs S and R.



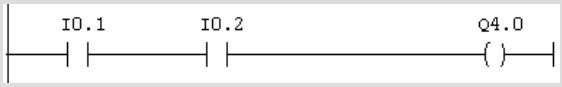
Enter the following symbolic names for the SR element:
Upper contact "Automatic_On"
Lower contact "Manual_On"
SR element "Automatic_Mode"

Save the block and close the window.

If you want to see the difference between absolute and symbolic addressing, deactivate the menu command **View > Display > Symbolic Representation**.



Example:
Symbolic addressing in LAD



Example:
Absolute addressing in LAD

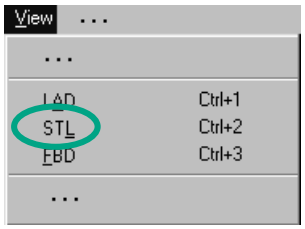
You can change the line break for symbolic addressing in the LAD/STL/FBD program window by using the menu command **Options > Customize** and then selecting "Width of address field" in the "LAD/FBD" tab. Here you can set the line break between 10 and 24 characters.

You can find more information under **Help > Contents** in the topics "Programming Blocks," "Creating Logic Blocks," and "Editing Ladder Instructions."

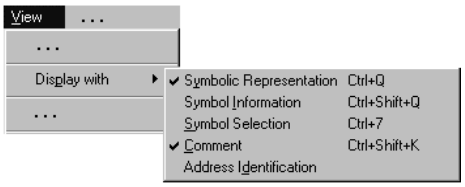
4.3 Programming OB1 in Statement List

In the following section, you will program an AND instruction, an OR instruction, and the memory instruction set/reset in Statement List (STL).

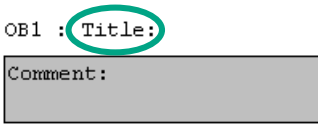
Programming an AND Instruction in Statement List



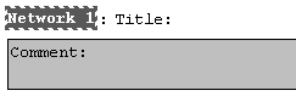
If necessary, set **STL** as the programming language in the **View** menu.



Check whether symbolic representation is activated.



Click in the **title** area of OB1 and enter "Cyclically processed main program," for example.



Select the area for your first statement.

Type an A (AND) in the first program line, a space, and then the symbolic name "Key_1" (in quotation marks).

Complete the line with **Enter**. The cursor jumps to the next line.

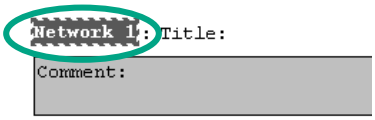
```
A     "Key_1"
A     "Key_2"
=     "Green_Light"
```

In the same way, complete the AND instruction as shown.

You have now programmed a complete AND instruction. Save the block if there are no more symbols shown in red.

> Symbols are indicated in red if, for example, they do not exist in the symbol table, or if there is a syntax error.
>
> You can also insert the symbolic name directly from the symbol table. Click the **??.?** sign and then the menu command **Insert > Symbol**. Scroll through the pull-down list until you reach the corresponding name and select it. The symbolic name is added automatically.

Programming an OR Instruction in Statement List



Select **Network 1**.

Insert a new network and select the input area again.

```
O     "Key_3"
```

Enter an O (OR) and the symbolic name "Key_3" (in the same way as for the AND instruction).

```
O     "Key_3"
O     "Key_4"
=     "Red_Light"
```

Complete the OR instruction and save it.

Programming a Memory Instruction in Statement List

Select Network 2 and insert another network.

```
A       "Automatic_On"
```

In the first line, type the instruction A with the symbolic name "Automatic_On."

```
A       "Automatic_On"
S       "Automatic_Mode"
A       "Manual_On"
R       "Automatic_Mode"
```

Complete the memory instruction and save it. Close the block.

If you want to see the difference between absolute and symbolic addressing, deactivate the menu command **View > Display > Symbolic Representation**.

```
A       "Key_1"
A       "Key_2"
=       "Green_Light"
```

Example:
Symbolic addressing in STL

```
A       I       0.1
A       I       0.2
=       Q       4.0
```

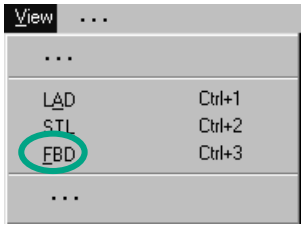
Example:
Absolute addressing in STL

You can find more information under **Help > Contents** in the topics "Programming Blocks," "Creating Logic Blocks," and "Editing STL Statements."

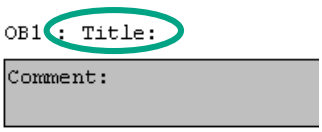
4.4 Programming OB1 in Function Block Diagram

In the following section, you will program an AND function, an OR function, and a memory function in Function Block Diagram (FBD).

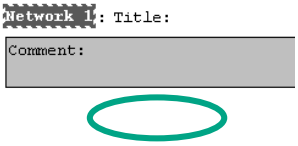
Programming an AND Function in Function Block Diagram



If necessary, set **FBD** as the programming language in the **View** menu.

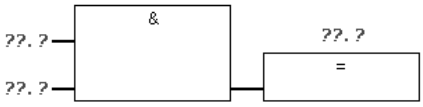


Click in the **title** area of OB1 and enter "Cyclically processed main program," for example.

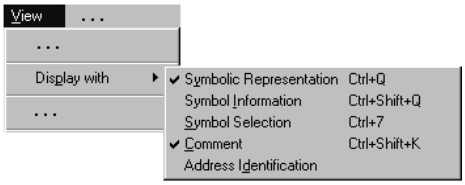


Select the input area for the AND function (below the comment field).

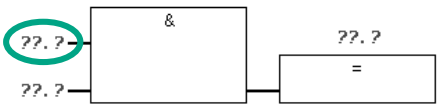
Insert an AND box (&) and an assignment (=).



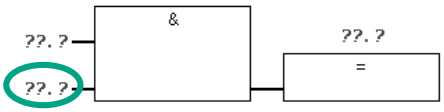
The addresses of the elements are still missing in the AND function.



Check whether symbolic representation is activated.

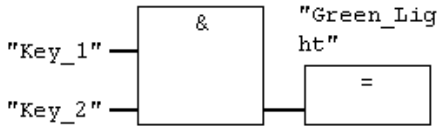


Click on the **??.?** sign and enter the symbolic name "Key_1" (in quotation marks). Confirm with **Enter**.



Enter the symbolic name "Key_2" for the second input.

Enter the name "Green_Light" for the assignment.



You have now programmed a complete AND function.

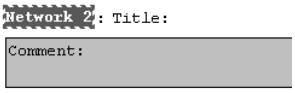
If there are no more symbols shown in red, you can save the block.

Symbols are indicated in red if, for example, they do not exist in the symbol table, or if there is a syntax error.

You can also insert the symbolic name directly from the symbol table. Click the **??.?** sign and then the menu command **Insert > Symbol**. Scroll through the pull-down list until you reach the corresponding name and select it. The symbolic name is added automatically.

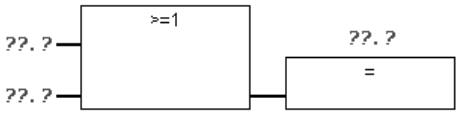
Programming an OR Function in Function Block Diagram

Insert a new network.

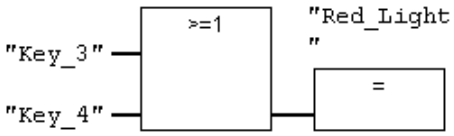


Select the input area again for the OR function.

Insert an OR box (≥1) and an assignment (=).



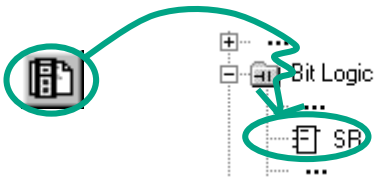
The addresses are still missing in the OR function. Proceed in the same way as for the AND function.



Enter "Key_3" for the upper input, "Key_4" for the lower input, and "Red_Light" for the assignment.

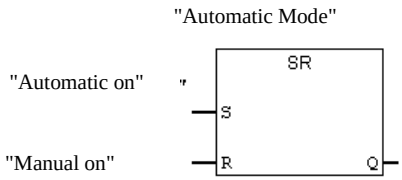
Save the block.

Programming a Memory Function in Function Block Diagram



Select Network 2 and insert another network. Select the input area again (below the comment field).

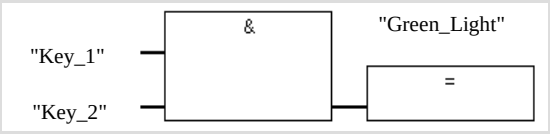
Navigate in the Program Elements catalog under **Bit Logic** until you reach the **SR** element. Double-click to insert the element.



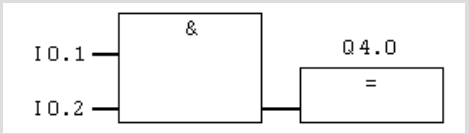
Enter the following symbolic names for the SR element:
Set "Automatic_On"
Reset "Manual_On"
Memory bit "Automatic_Mode"

Save the block and close the window.

If you want to see the difference between absolute and symbolic addressing, deactivate the menu command **View > Display > Symbolic Representation**.



Example:
Symbolic addressing in FBD



Example:
Absolute addressing in FBD

You can change the line break for symbolic addressing in the LAD/STL/FBD program window by using the menu command **Options > Customize** and then selecting "Address Field Width" in the "LAD/FB... ...d 24

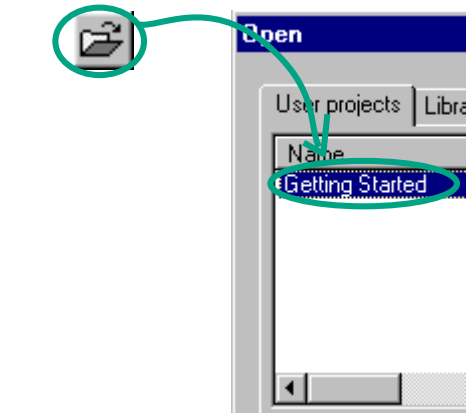
You can find more information under **Help > Contents** in the topics "Programming Blocks," "Creating Logic Blocks," and "Editing FBD Statements."

5 Creating a Program with Function Blocks and Data Blocks

5.1 Creating and Opening Function Blocks (FB)

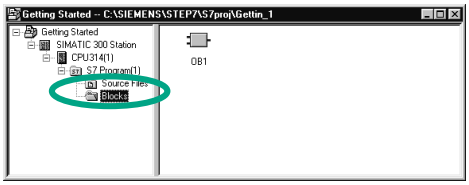
The function block (FB) is below the organization block in the program hierarchy. It contains a part of the program which can be called many times in OB1. All the formal parameters and static data of the function block are saved in a separate data block (DB), which is assigned to the function block.

You will program the function block (FB1, symbolic name "Engine"; see symbol table, page 3-3) in the LAD/STL/FBD program window, which you are now familiar with. To do this, you should use the same programming language as in Chapter 4 (programming OB1).



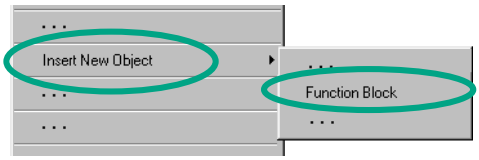
You should have already copied the symbol table into your project "Getting Started." If not, read how to do this on page 4-2, copying the symbol table, and then return to this section.

If necessary, open the "Getting Started" project.

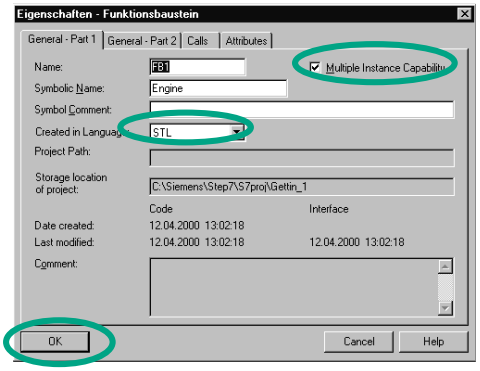


Navigate to the **Blocks** folder and open it.

Click in the right-hand half of the window with the right mouse button.

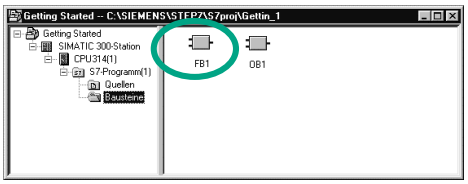


The pop-up menu for the right mouse button contains the most important commands from the menu bar. Insert a **function block** as a new object.



Double-click FB1 to open the LAD/STL/FBD program window.

In the "Properties – Function Block" dialog box, select the language in which you want to create the block, activate the check box "**Multiple instance FB**," and confirm the remaining settings with **OK**.



The function block **FB1** has been inserted in the Blocks folder.

Depending on which programming language you have selected, continue reading in either Section 5.2 for Ladder Logic, Section 5.3 for Statement List, or Section 5.4 for Function Block Diagram.

You can find more information under **Help > Contents** in the topics "Programming Blocks" and "Creating Blocks and Libraries."

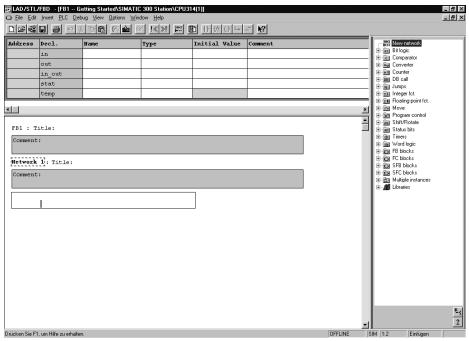
5.2 Programming FB1 in Ladder Logic

We will now show you how to program a function block which can, for example, control and monitor a petrol or diesel engine using two different data blocks.

All "engine-specific" signals are passed on as block parameters from the organization block to the function block and must therefore be listed in the variable declaration table as input and output parameters (declaration "in" and "out").

You should already know how to enter a series circuit, a parallel circuit, and a memory function with STEP 7.

1. Filling out the Variable Declaration Table



Your LAD/STL/FBD program window is open and the option **View > LAD** (programming language) is activated.

Note that FB1 is now in the header, because you double-clicked FB1 to open the program window.

Enter the following declarations in the variable declaration table.

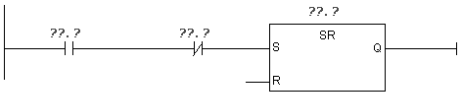
To do this, click a cell and use the corresponding name and the comment from the illustration below.

You can select the type with the pop-up menu command **Elementary Types** using the right mouse button. When you press **Enter**, the cursor jumps to the next column, or a new row is inserted.

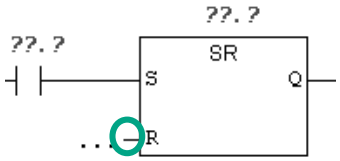
Address	Decl.	Name	Type	Initial Value	Comment
0.0	in	Switch_On	BOOL	FALSE	Switch on engine
0.1	in	Switch_Off	BOOL	FALSE	Switch off engine
0.2	in	Failure	BOOL	FALSE	Engine failure, causes the engine to switch off
2.0	in	Actual_Speed	INT	0	Actual engine speed
4.0	out	Engine_On	BOOL	FALSE	Engine is switched on
4.1	out	Preset_Speed_Reached	BOOL	FALSE	Preset speed reached
	in_out				
6.0	stat	Preset_Speed	INT	1500	Requested engine speed
	temp				

Only letters, numbers, and the underscore are permitted characters for the names of the block parameters in the variable declaration table.

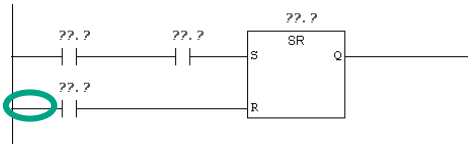
2. Programming an Engine to Switch On and Off



Insert a normally open contact, a normally closed contact, and an SR element in series in Network 1 using the corresponding buttons in the toolbar or the Program Elements catalog.

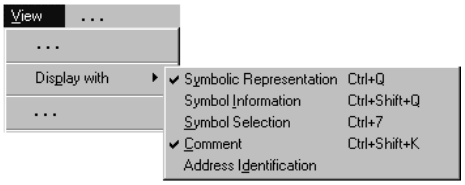


Then select the current path immediately before the input R.



Insert another normally open contact. Select the current path immediately before this contact.

Insert a normally closed contact parallel to the normally open contact.

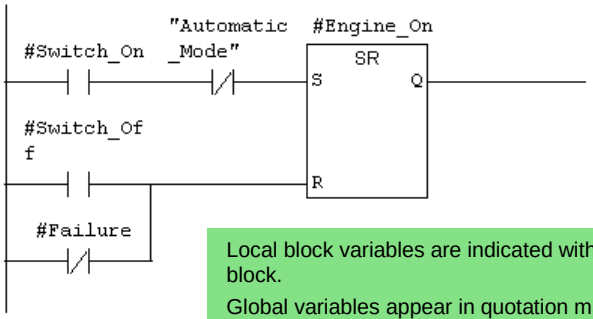


Check whether symbolic representation is activated.

Select the question marks and enter the corresponding names from the variable declaration table (the # sign is assigned automatically).

Enter the symbolic name "Automatic_Mode" for the normally closed contact in the series circuit.

Then save your program.

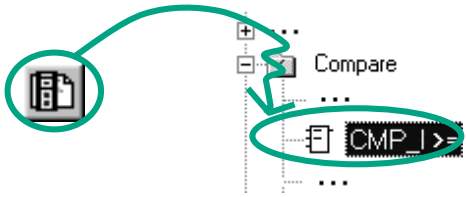


Local block variables are indicated with a # sign and are only valid in the block.

Global variables appear in quotation marks. These are defined in the symbol table and are valid for the entire program.

The signal state "Automatic_Mode" is defined in OB1 (Network 3; see page 4-7) by another SR element and now queried in FB1.

3. Programming Speed Monitoring



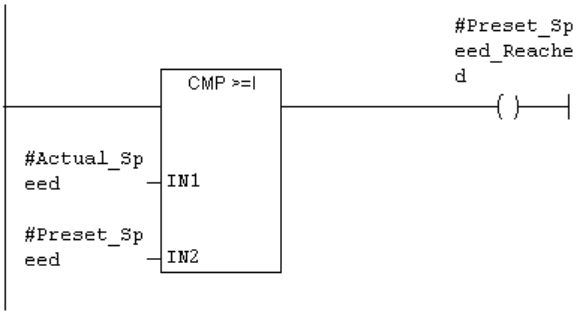
Insert a new network and select the current path.

Then navigate in the Program Elements catalog until you reach the **Compare** function and insert a **CMP>=I**.

Also insert a coil in the current path.

Select the question marks again and label the coil and the comparator with the names from the variable declaration table.

Then save your program.



When is the engine switched on and off?

When the variable #Switch_On has signal state "1" and the variable "Automatic_Mode" has signal state "0," the engine is switched on. This function is not enabled until "Automatic_Mode" is negated (normally closed contact).

When the variable #Switch_Off has signal state "1" or the variable #Fault has signal state "0," the engine is switched off. This function is achieved again by negating #Fault (#Fault is a "zero-active" signal and has the signal "1" in the normal state and "0" if a fault occurs).

How does the comparator monitor the engine speed?

The comparator compares the variables #Actual_Speed and #Setpoint_Speed and assigns the result of the variables to #Setpoint_Speed_Reached (signal state "1").

You can find more information under **Help > Contents** in the topics "Programming Blocks," "Creating Logic Blocks," and "Editing the Variable Declaration Table" or in "Editing LAD Instructions."

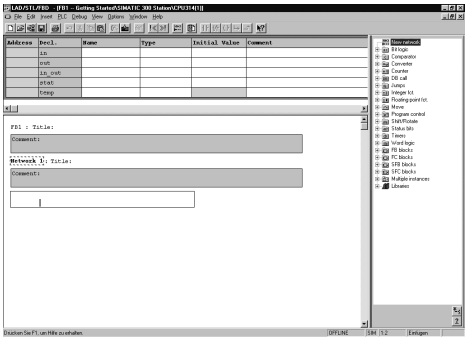
5.3 Programming FB1 in Statement List

We will now show you how to program a function block which can, for example, control and monitor a petrol or diesel engine using two different data blocks.

All "engine-specific" signals are passed on as block parameters from the organization block to the function block and must therefore be listed in the variable declaration table as input and output parameters (declaration "in" and "out").

You should already know how to enter an AND instruction, an OR instruction, and the set/reset memory instructions with STEP 7.

1. Filling out the Variable Declaration Table



Your LAD/STL/FBD program window is open and the option **View > STL** (programming language) is activated.

Note that FB1 is now in the header, because you double-clicked FB1 to open the program window.

Enter the following declarations in the variable declaration table.

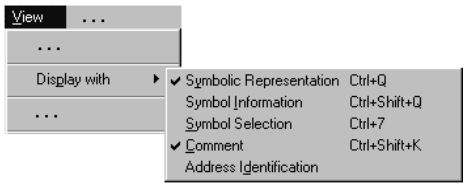
To do this, click a cell and use the corresponding name and the comment from the illustration below.

You can select the type with the pop-up menu command **Elementary Types** using the right mouse button. When you press **Enter**, the cursor jumps to the next column, or a new row is inserted.

Address	Decl.	Name	Type	Initial Value	Comment
0.0	in	Switch_On	BOOL	FALSE	Switch on engine
0.1	in	Switch_Off	BOOL	FALSE	Switch off engine
0.2	in	Failure	BOOL	FALSE	Engine failure, causes the engine to switch off
2.0	in	Actual_Speed	INT	0	Actual engine speed
4.0	out	Engine_On	BOOL	FALSE	Engine is switched on
4.1	out	Preset_Speed_Reached	BOOL	FALSE	Preset speed reached
	in_out				
6.0	stat	Preset_Speed	INT	1500	Requested engine speed
	temp				

Only letters, numbers, and the underscore are permitted characters for the names of the block parameters in the variable declaration table.

2. Programming an Engine to Switch On and Off



Check whether symbolic representation is activated.

```
A     #Switch_On
AN    "Automatic_Mode"
S     #Engine_On
O     #Switch_Off
ON    #Failure
R     #Engine_On
```

Enter the corresponding instructions in Network 1.

Local block variables are indicated with a # sign and are only valid in the block.

Global variables appear in quotation marks. These are defined in the symbol table and are valid for the entire program.

The signal state "Automatic_Mode" is defined in OB1 (Network 3; see page 4-10) by another SR element and now queried in FB1.

3. Programming Speed Monitoring

```
L     #Actual_Speed
L     #Preset_Speed
>=I
=     #Preset_Speed_Reached
```

Insert a new network and enter the corresponding instructions. Then save your program.

When is the engine switched on and off?

When the variable #Switch_On has signal state "1" and the variable "Automatic_Mode" has signal state "0," the engine is switched on. This function is not enabled until "Automatic_Mode" is negated (normally closed contact).

When the variable #Switch_Off has signal state "1" or the variable #Fault has signal state "0," the engine is switched off. This function is achieved again by negating #Fault (#Fault is a "zero-active" signal and has the signal "1" in the normal state and "0" if a fault occurs).

How does the comparator monitor the engine speed?

The comparator compares the variables #Actual_Speed and #Setpoint_Speed and assigns the result of the variables to #Setpoint_Speed_Reached (signal state "1").

You can find more information under **Help > Contents** in the topics "Programming Blocks," "Creating Logic Blocks," and "Editing the Variable Declaration Table" or in "Editing STL Statements."

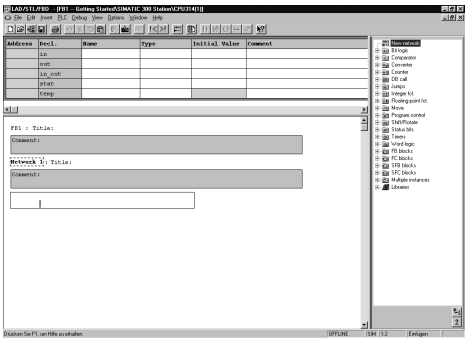
5.4 Programming FB1 in Function Block Diagram

We will now show you how to program a function block which can, for example, control and monitor a petrol or diesel engine using two different data blocks.

All "engine-specific" signals are passed on as block parameters from the organization block to the function block and must therefore be listed in the variable declaration table as input and output parameters (declaration "in" and "out").

You should already know how to enter an AND function, an OR function, and a memory function with STEP 7.

1. Filling out the Variable Declaration Table



Your LAD/STL/FBD program window is open and the option **View > FBD** (programming language) is activated.

Note that FB1 is now in the header, because you double-clicked FB1 to open the program window.

Enter the following declarations in the variable declaration table.

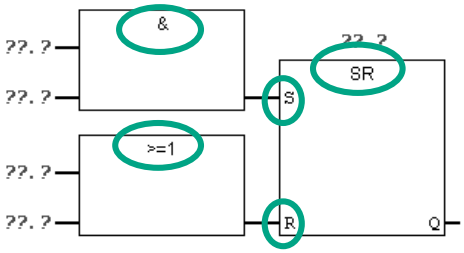
To do this, click a cell and use the corresponding name and the comment from the illustration below.

You can select the type with the pop-up menu command **Elementary Types** using the right mouse button. When you press **Enter**, the cursor jumps to the next column, or a new row is inserted.

Address	Decl.	Name	Type	Initial Value	Comment
0.0	in	Switch_On	BOOL	FALSE	Switch on engine
0.1	in	Switch_Off	BOOL	FALSE	Switch off engine
0.2	in	Failure	BOOL	FALSE	Engine failure, causes the engine to switch off
2.0	in	Actual_Speed	INT	0	Actual engine speed
4.0	out	Engine_On	BOOL	FALSE	Engine is switched on
4.1	out	Preset_Speed_Reached	BOOL	FALSE	Preset speed reached
	in_out				
6.0	stat	Preset_Speed	INT	1500	Requested engine speed
	temp				

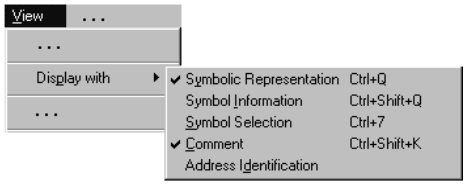
Only letters, numbers, and the underscore are permitted characters for the names of the block parameters in the variable declaration table.

2. Programming an Engine to Switch On and Off



Insert an SR function in Network 1 using the Program Elements catalog (Bit Logic folder).

Add an AND box at input S (Set), and an OR box at input R (Reset).



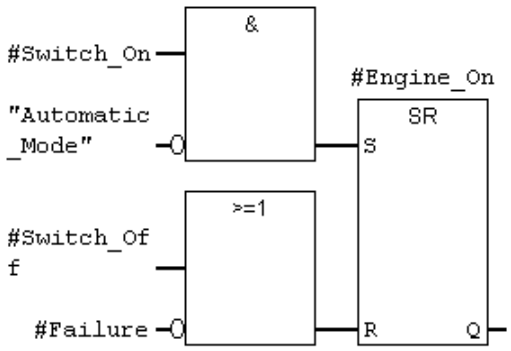
Check whether symbolic representation is activated.

Click the **??.?** sign and enter the corresponding names from the declaration table (the # sign is assigned automatically).

Make sure that one input of the AND function is addressed with the symbolic name "Automatic_Mode."

Negate the inputs "Automatic_Mode" and #Fault with the corresponding button from the toolbar.

Then save your program.

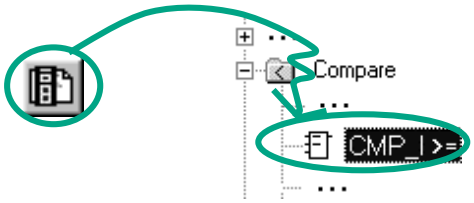


Local block variables are indicated with a # sign and are only valid in the block.

Global variables appear in quotation marks. These are defined in the symbol table and are valid for the entire program.

The signal state "Automatic_Mode" is defined in OB1 (Network 3; see page 4-14) by another SR element and now queried in FB1.

3. Programming Speed Monitoring

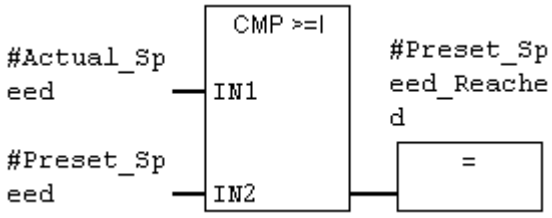


Insert a new network and select the input area.

Then navigate in the Program Elements catalog under you reach the **Compare** function, and insert a **CMP>=I**.

Append an output assignment to the comparator and address the inputs with the names from the variable declaration table.

Then save your program.



When is the engine switched on and off?

When the variable #Switch_On has signal state "1" and the variable "Automatic_Mode" has signal state "0," the engine is switched on. This function is not enabled until "Automatic_Mode" is negated (normally closed contact).

When the variable #Switch_Off has signal state "1" or the variable #Fault has signal state "0," the engine is switched off. This function is achieved again by negating #Fault (#Fault is a "zero-active" signal and has the signal "1" in the normal state and "0" if a fault occurs).

How does the comparator monitor the engine speed?

The comparator compares the variables #Actual_Speed and #Setpoint_Speed and assigns the result of the variables to #Setpoint_Speed_Reached (signal state "1").

You can find more information under **Help > Contents** in the topics "Programming Blocks," "Creating Logic Blocks," and "Editing the Variable Declaration Table" or in "Editing FBD Instructions."

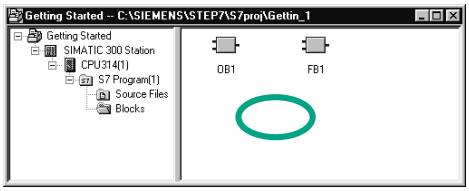
5.5 Generating Instance Data Blocks and Changing Actual Values

You have just programmed the function block FB1 ("Engine") and defined, among other things, the engine-specific parameters in the variable declaration table.

In order for you to be able to program the call for the function block in OB1 later on, you must generate the corresponding data block. An instance data block (DB) is always assigned to a function block.

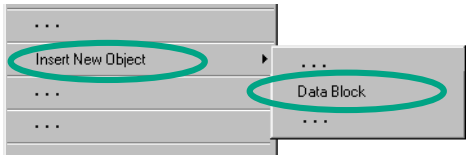
The function block is to control and monitor a petrol or diesel engine. The different setpoint speeds of the engines are stored in two separate data blocks, in which the actual value (#Setpoint_Speed) is changed.

By centrally programming the function block only once, you can cut down on the amount of programming involved.

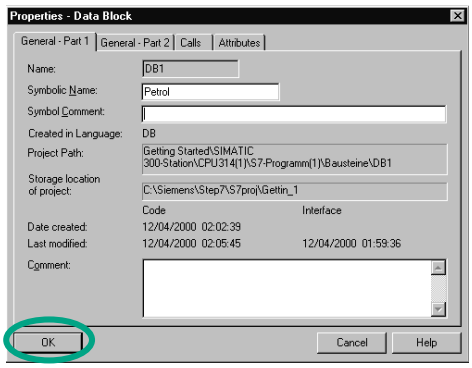


The "Getting Started" project is open in the SIMATIC Manager.

Navigate to the **Blocks** folder and click in the right half of the window with the right mouse button.



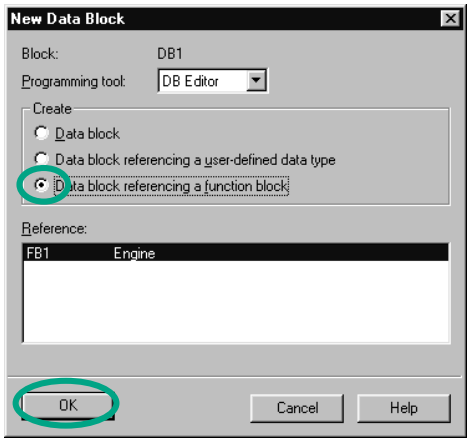
Insert a **data block** using the pop-up menu with the right mouse button.



Accept all the settings displayed in the "Properties" dialog box with **OK**.

The data block **DB1** is added to the "Getting Started" project.

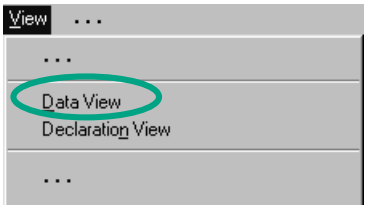
Double-click to open **DB1**.



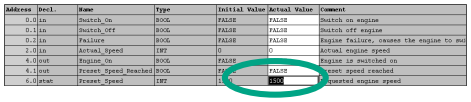
Activate the option **Data block referencing a function block** in the "New Data Block" dialog box.

Confirm the assignment "FB1, Engine" with **OK**.

The LAD/STL/FBD program window opens with the data from the variable declaration table for FB1.

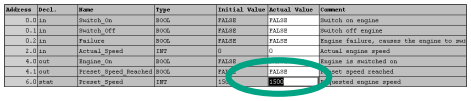


DB1 is now to contain the data specific to a petrol engine. You still have to enter these data. First set the **Data View**.



Next enter the value "1500" for the petrol engine in the Actual Value column (in the row "Setpoint_Speed). You have now defined the maximum speed for this engine.

Save DB1 and close the program window.



In the same way as for DB1, generate another data block, DB2, for FB1.

Now enter the actual value "1200" for the diesel engine.

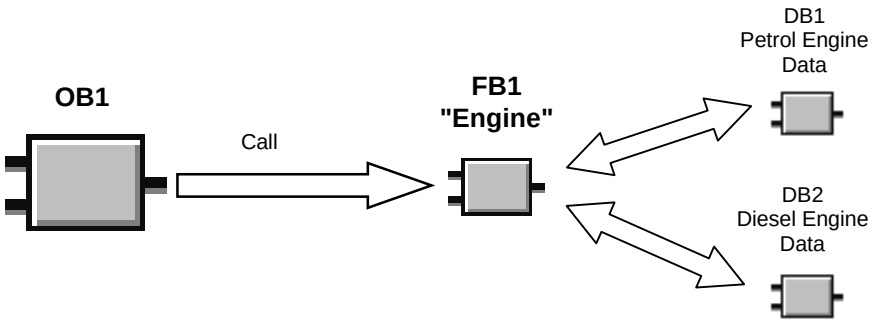
By changing the actual values, you have finished your preparations for controlling two engines with just one function block. To control more engines, all you have to do is generate additional data blocks.

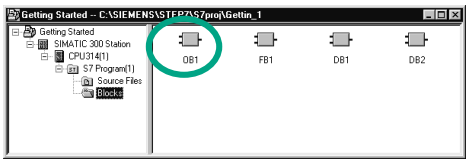
The next thing you have to do is program the call for the function block in OB1. To do this, continue reading in Section 5.6 for Ladder Logic, Section 5.7 for Statement List, or Section 5.8 for Function Block Diagram, depending on the programming language you are using.

You can find more information under **Help > Contents** in the topics "Programming Blocks" and "Creating Data Blocks."

5.6 Programming a Block Call in Ladder Logic

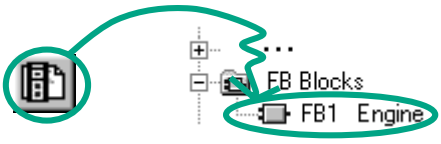
All the work you have done programming a function block is of no use unless you call this block in OB1. A data block is used for each function block call, and in this way, you can control both engines.



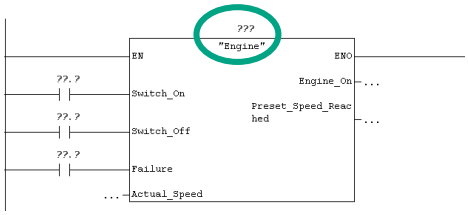


The SIMATIC Manager is open with your "Getting Started" project.

Navigate to the **Blocks** folder and open **OB1**.

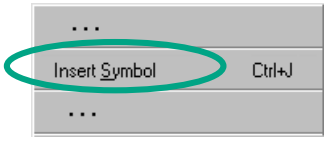


Insert Network 4 in the LAD/STL/FBD program window. Then navigate in the Program Elements catalog until you reach **FB1** and insert this block.

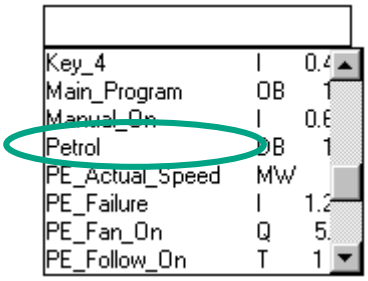


Insert a normally open contact in front of each of the following: Switch_On, Switch_Off, and Fault.

Click the **???** sign above "Engine" and then, keeping the cursor in the same position, click in the input frame with the right mouse button.

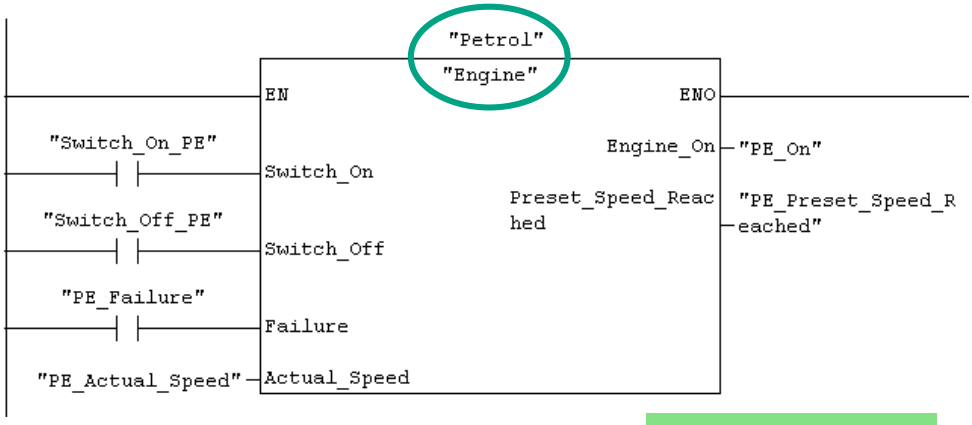


Select **Insert Symbol** in the pop-up menu using the right mouse button. A pull-down list will appear. The first time you do this, the procedure may take some time.



Click the data block **Petrol**. This block is then entered automatically in the input frame in quotation marks.

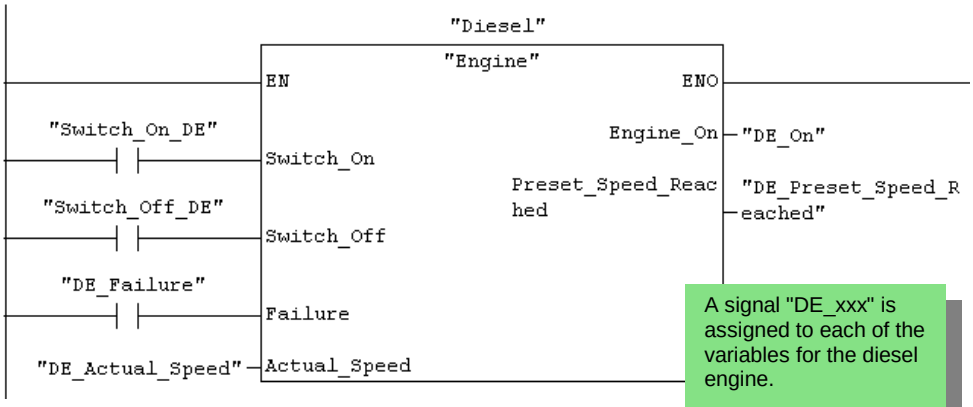
Click the question marks and address all the other parameters of the function block using the corresponding symbolic names in the pull-down list.



The engine-specific input and output variables (declaration "in" and "out") are displayed in the FB "Engine."

A signal "PE_xxx" is assigned to each of the variables for the petrol engine.

Program the call for the function block "Engine" (FB1) with the data block "Diesel" (DB2) in a new network and use the corresponding addresses from the pull-down list.



A signal "DE_xxx" is assigned to each of the variables for the diesel engine.

Save your program and close the block.

When you create program structures with organization blocks, function blocks, and data blocks, you must program the call for subordinate blocks (such as FB1) in the block above them in the hierarchy (for example, OB1). The procedure is always the same.

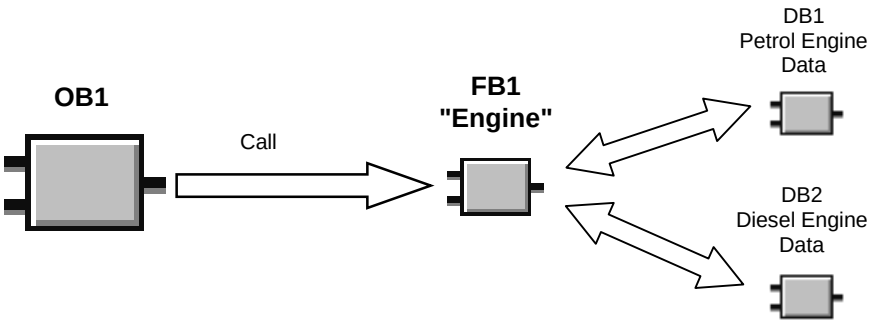
You can also give the various blocks symbolic names in the symbol table (for example, FB1 has the name "Engine" and DB1 the name "Petrol").

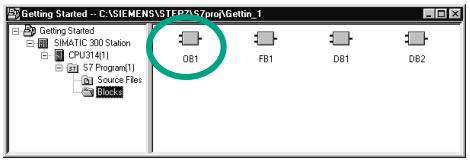
You can archive or print out the programmed blocks at any time. The corresponding functions can be found in the SIMATIC Manager under the menu commands **File > Archive** or **File > Print**.

You can find more information under **Help > Contents** in the topics "Calling Reference Helps," "Language Description: LAD," and "Program Control Instructions."

5.7 Programming a Block Call in Statement List

All the work you have done programming a function block is of no use unless you call this block in OB1. A data block is used for each function block call, and in this way, you can control both engines.





The SIMATIC Manager is open with your "Getting Started" project.

Navigate to the **Blocks** folder and open **OB1**.

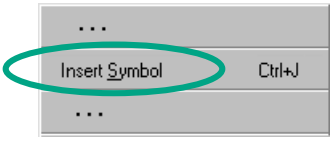
Insert Network 4 in the LAD/STL/FBD program window.

```
CALL  "Engine" , "Petrol"
  Switch_On            :=
  Switch_Off           :=
  Failure              :=
  Actual_Speed         :=
  Engine_On            :=
  Preset_Speed_Reached:=
```

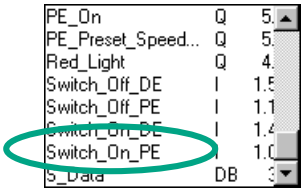
Type **CALL "Engine", "Petrol"** in the code section and then press **Enter**.

All the parameters of the function block "Petrol" are displayed.

Position the cursor after the equals sign of Switch_On and press the right mouse button.



Select **Insert Symbol** in the pop-up menu using the right mouse button. A pull-down list will appear. The first time you do this, the procedure may take some time.



Click the name **Switch_On_PE**. This is taken from the pull-down list and added automatically in quotation marks.

```
CALL  "Engine" , "Petrol"
 Switch_On           :="Switch_On_PE"
 Switch_Off          :="Switch_Off_PE"
 Failure             :="PE_Failure"
 Actual_Speed        :="PE_Actual_Speed"
 Engine_On           :="PE_On"
 Preset_Speed_Reached:="PE_Preset_Speed_Reached"
```

Assign all the required addresses to the variables of the function block using the pull-down list.

A signal "PE_xxx" is assigned to each of the variables for the petrol engine.

```
CALL  "Engine" , "Diesel"
 Switch_On           :="Switch_On_DE"
 Switch_Off          :="Switch_Off_DE"
 Failure             :="DE_Failure"
 Actual_Speed        :="DE_Actual_Speed"
 Engine_On           :="DE_On"
 Preset_Speed_Reached:="DE_Preset_Speed_Reached"
```

Program the call for the function block "Engine" (FB1) with the data block "Diesel" (DB2) in a new network. Proceed in the same way as for the other call.

Save your program and close the block.

When you create program structures with organization blocks, function blocks, and data blocks, you must program the call for subordinate blocks (such as FB1) in the block above them in the hierarchy (for example, OB1). The procedure is always the same.

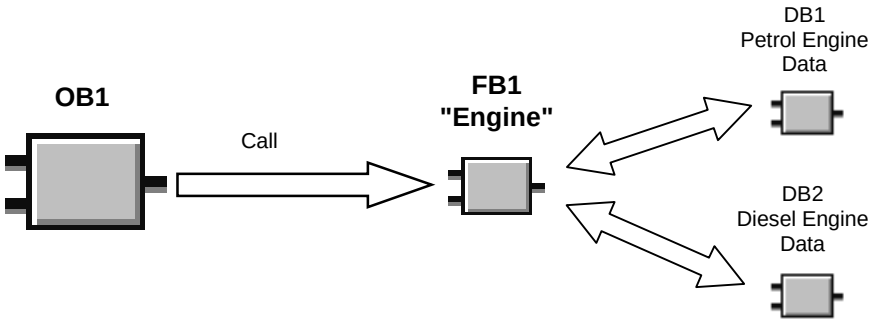
You can also give the various blocks symbolic names in the symbol table (for example, FB1 has the name "Engine" and DB1 the name "Petrol").

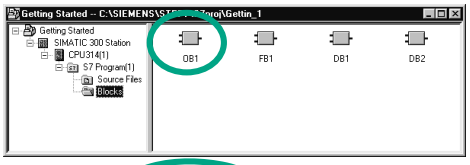
You can archive or print out the programmed blocks at any time. The corresponding functions can be found in the SIMATIC Manager under the menu commands **File > Archive** or **File > Print**.

You can find more information under **Help > Contents** in the topics "Calling Reference Helps," "Language Description: STL," and "Program Control Instructions."

5.8 Programming a Block Call in Function Block Diagram

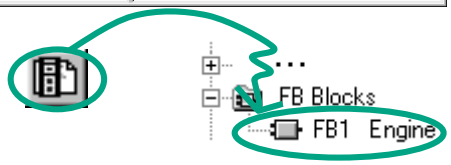
All the work you have done programming a function block is of no use unless you call this block in OB1. A data block is used for each function block call, and in this way, you can control both engines.



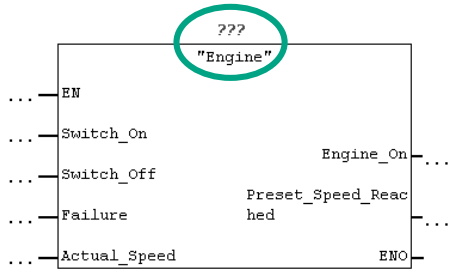


The SIMATIC Manager is open with your "Getting Started" project.

Navigate to the **Blocks** folder and open **OB1**.

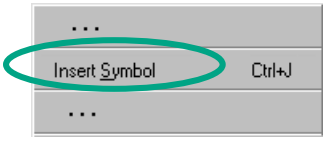


Insert Network 4 in the LAD/STL/FBD program window. The navigate in the Program Elements catalog until you reach **FB1** and insert this block.

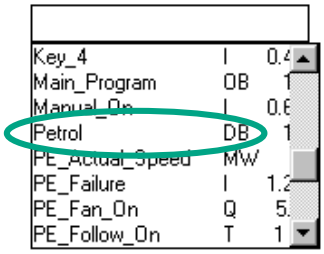


All the engine-specific input and output variables are displayed.

Click the **???** sign above "Engine" and then, keeping the cursor in the same position, click in the input frame with the right mouse button.

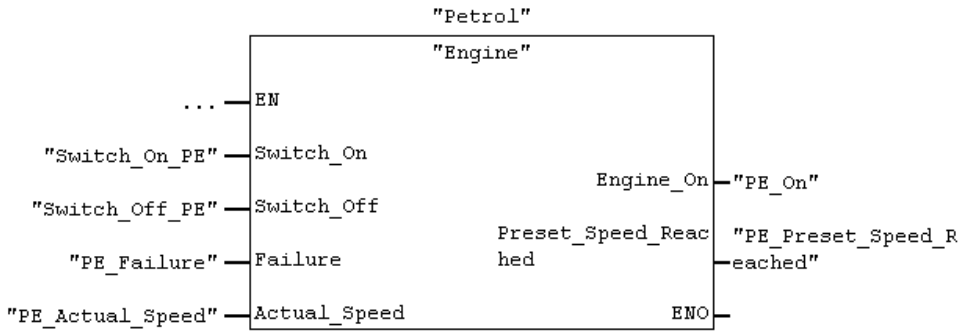


Select **Insert Symbol** in the pop-up menu using the right mouse button. A pull-down list will appear. The first time you do this, the procedure may take some time.



Click the data block **Petrol**. It is taken from the pull-down list and entered automatically in the input frame in quotation marks.

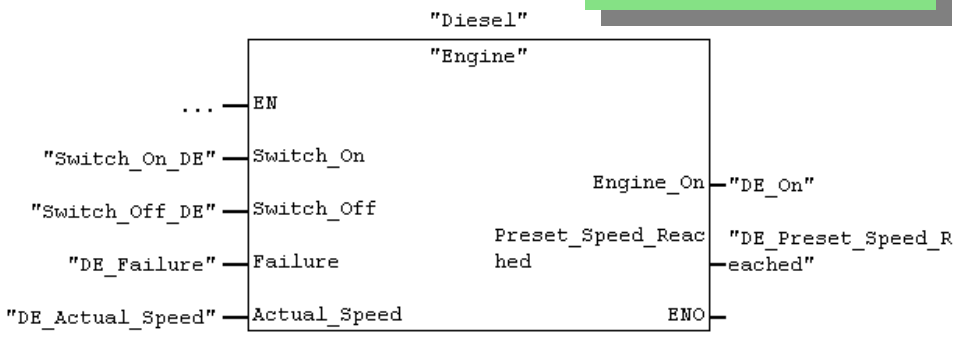
Address all the other parameters of the function block using the corresponding symbolic names in the pull-down list.



A signal "PE_xxx" is assigned to each of the variables for the petrol engine.

Program the call for the function block "Engine" (FB1) with the data block "Diesel" (DB2) in a new network and use the corresponding addresses from the pull-down list.

A signal "DE_xxx" is assigned to each of the variables for the diesel engine.



Save your program and close the block.

When you create program structures with organization blocks, function blocks, and data blocks, you must program the call for subordinate blocks (such as FB1) in the block above them in the hierarchy (for example, OB1). The procedure is always the same.

You can also give the various blocks symbolic names in the symbol table (for example, FB1 has the name "Engine" and DB1 the name "Petrol").

You can archive or print out the programmed blocks at any time. The corresponding functions can be found in the SIMATIC Manager under the menu commands **File > Archive** or **File > Print**.

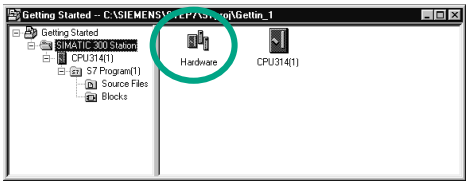
You can find more information under **Help > Contents** in the topics "Calling Reference Helps," "Language Description: FBD," and "Program Control Instructions."

6 Configuring the Central Rack

6.1 Configuring Hardware

You can configure the hardware once you have created a project with a SIMATIC station. The project structure which was created with the STEP 7 Wizard in Section 2.1 meets all the requirements for this.

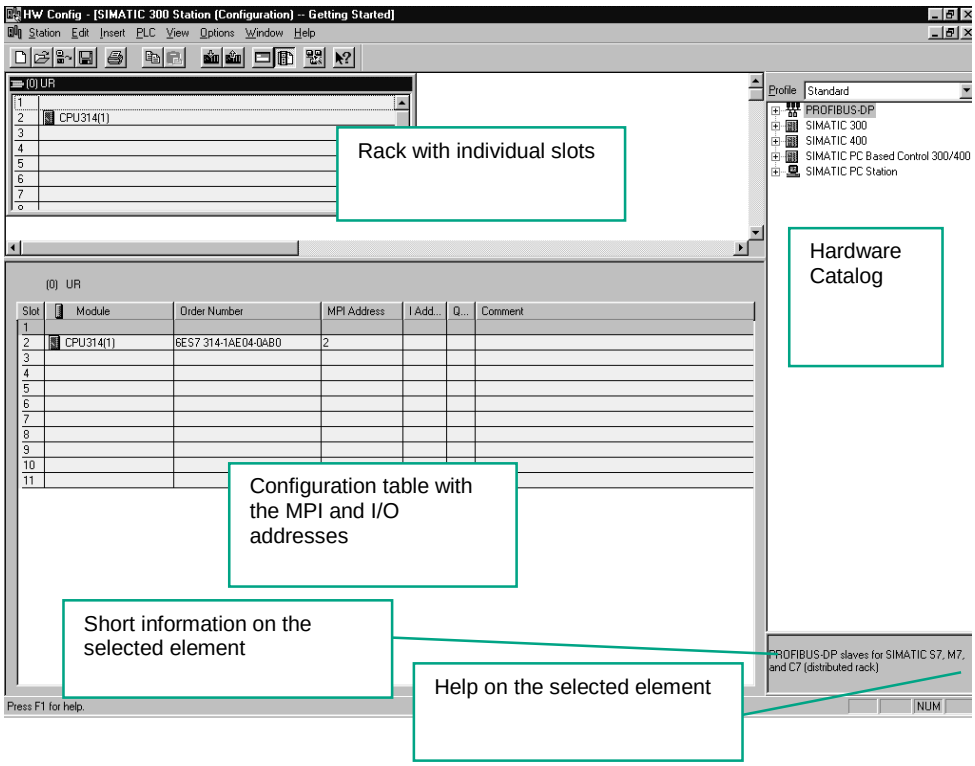
The hardware is configured with STEP 7. These configuration data are transferred to the programmable controller later on "downloading" (see Chapter 7).

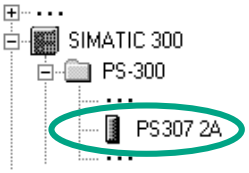


The starting point is the open SIMATIC Manager together with the "Getting Started" project.

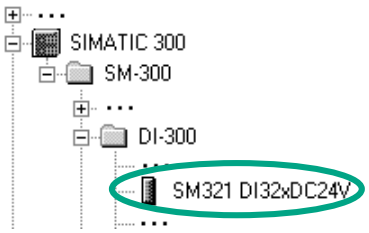
Open the **SIMATIC 300 Station** folder and double-click the **Hardware** symbol.

The "HW Config" window opens. The CPU you selected on creating the project is displayed. For the "Getting Started" project, this is CPU 314.

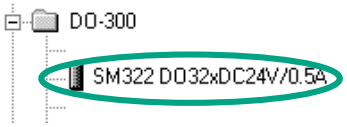




First you require a power supply module. Navigate in the catalog until you reach the **PS307 2A** and drag and drop this onto slot 1.



Navigate until you find the input module (DI, Digital Input) **SM321 DI32xDC24V** and insert this in slot 4. Slot 3 remains empty.

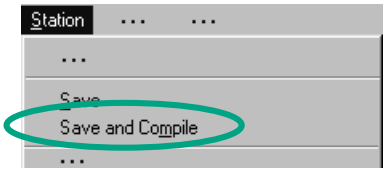


In the same way, insert the output module **SM322 DO32xDC24V/0.5A** in slot 5.

In order to change the parameters (for example, address) of a module within a project, double-click the module. However, you should only change the parameters if you are sure you know what effects the changes will have on your programmable controller.

No changes are necessary for the "Getting Started" project.

Slot	Module	Order Number	MPI Address	I Add...	Q...	Comment
1	PS307 2A	6ES7 307-1BA00-0AA0				
2	CPU314(1)	6ES7 314-1AE04-0AB0	2			
3						
4	DI32xDC24V	6ES7 321-1BL00-0AA0		0...3		
5	DO32xDC24V/0.5A	6ES7 322-1BL00-0AA0			4...7	
6						
7						
8						
9						
10						
11						



The data are prepared for transfer to the CPU using the menu command **Save and Compile**.

Once you close the "HW Config" application, the System Data symbol will appear in the Blocks folder.

You can also check your configuration for errors using the menu command **Station > Consistency Check**. STEP 7 will provide you with possible solutions to any errors which may have occurred.

You can find more information under **Help > Contents** in the topics "Configuring theHardware" and "Configuring Central Racks."

7 Downloading and Debugging the Program

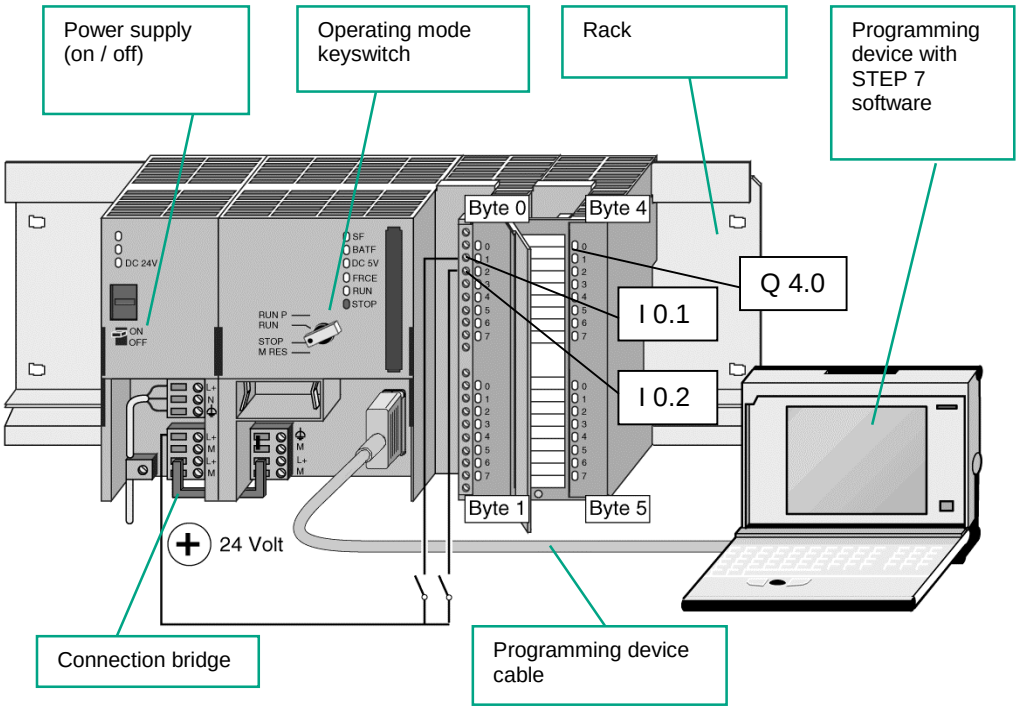
7.1 Establishing an Online Connection

Using the supplied project "GS-LAD_Example" or the "Getting Started" project you have created and a simple test configuration, we will show you how to download the program to the programmable logic controller (PLC) and then debug it.

You should have:

- Configured the hardware for the "Getting Started" project (see Chapter 6)
- Set up the hardware according to the installation manual

Example of a series circuit (AND function):
Output Q 4.0 is not to light up (diode Q 4.0 lights up on the digital output module) unless both Key I 0.1 **and** Key I 0.2 are pressed. Set up the test configuration below using wires and your CPU.



Configuring the Hardware

To assemble a module on the rail, proceed in the order given below:

- Attach the module onto the bus connector
- Hang the module on the rail and swing it downwards
- Screw the module in place
- Assemble the remaining modules
- Insert the key in the CPU once you have finished assembling all the modules.

You can still carry out the test even if you are using different hardware to that shown in the diagram. You simply have to keep to the addressing of the inputs and outputs.
STEP 7 offers you various ways of debugging your program; for example, using the program status or by means of the variable table.

You can find more information on configuring the central rack in the manuals "S7-300, Hardware and Installation / Module Specifications" and "S7-400 / M7-400 – Hardware."

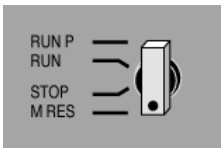
7.2 Downloading the Program to the Programmable Controller

You must have already established an online connextion in order tp download the program.

Switch on the power supply using the ON/OFF switch. The diode "DC 5V" will light up on the CPU.

Turn the operating mode switch to the STOP position (if not already in STOP). The red "STOP" LED will light up.

Resetting the CPU and Switching it to RUN



Turn the operating mode switch to the **MRES** position and hold it there for at least 3 seconds until the red "STOP" LED starts flashing slowly.

Release the switch and, after a maximum of 3 seconds, turn it to the **MRES** position again. When the "STOP" LED flashes quickly, the CPU has been reset.

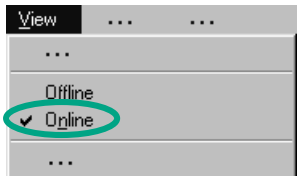
If the "STOP" LED does not start flashing quickly, repeat the procedure.

A memory reset deletes all the data on the CPU. The CPU is then in the initial state.

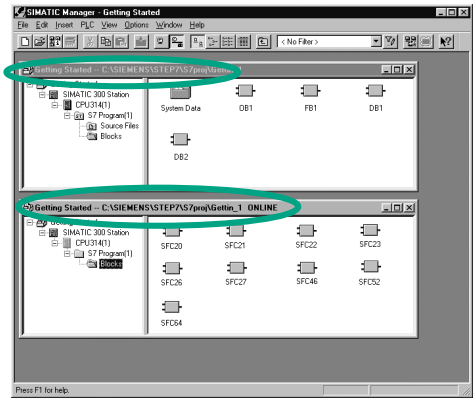
Downloading the Program to the CPU

Now turn the operating mode switch to "STOP" again to download the program.

Start the SIMATIC Manager and open the "Getting Started" project in the "Open" dialog box (if it is not already open).



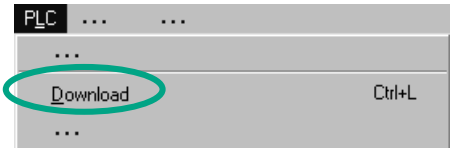
In addition to the "Getting Started Offline" window, open the "Getting Started ONLINE" window. The online or offline status is indicated by the different colored headers.



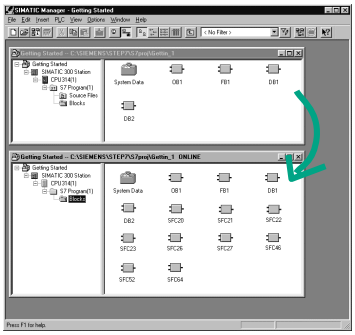
Navigate in both windows to the **Blocks** folder.

The offline window shows the situation on the programming device; the online window shows the situation on the CPU.

The system functions (SFCs) remain in the CPU even though you have carried out a memory reset. The CPU provides these functions of the operating system. They do not have to be downloaded, but they cannot be deleted.



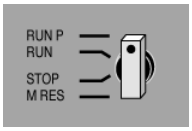
Select the **Blocks** folder in the offline window and then download the program to the CPU using the menu command **PLC > Download**. Confirm the prompt with **OK**.



The program blocks are displayed in the online window when you download them.

You can also call the menu command **PLC > Download** using the corresponding button in the toolbar or from the pop-up menu using the right mouse button.

Switching on the CPU and Checking the Operating Mode



Turn the operating mode switch to **RUN-P**. The green "RUN" LED lights up and the red "STOP" LED goes out. The CPU is ready for operation.

When the green LED lights up, you can start testing the program.

If the red LED remains lit, an error has occurred. You would then have to evaluate the diagnostic buffer in order to diagnose the error.

Downloading individual blocks

In order to react to errors quickly in practice, blocks can be transferred individually to the CPU using the drag and drop function.
When you download blocks, the operating mode switch on the CPU must be in either "RUN-P" or "STOP" mode. Blocks downloaded in "RUN-P" mode are activated immediately. You should therefore remember the following:

- If error-free blocks are overwritten with faulty blocks, this will lead to a plant failure. You can avoid this by testing your blocks before you download them.
- If you do not observe the order in which blocks are to be downloaded – first the subordinate blocks and then the higher-level blocks – the CPU will go into "STOP" mode. You can avoid this by downloading the entire program to the CPU.

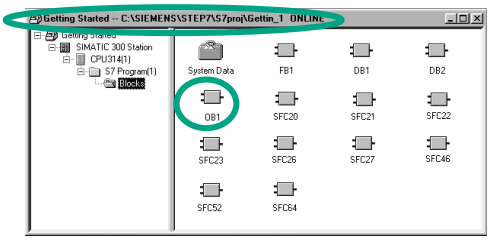
Programming online

In practice, you may need to change the blocks already downloaded to the CPU for test purposes. To do this, double-click the required block in the online window to open the LAD/STL/FBD program window. Then program the block as usual. Note that the programmed block immediately becomes active in your CPU.

You can find more information under **Help > Contents** in the topics "Establishing an Online Connection and Making CPU Settings" and "Downloading from the PG / PC to the Programmable Controller."

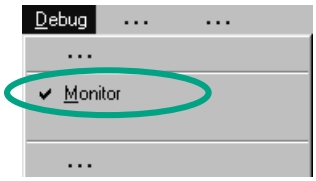
7.3 Testing the Program with Program Status

Using the program status function, you can test the program in a block. The requirement for this is that you have established an online connection to the CPU, the CPU is in RUN or RUN-P mode, and the program has been downloaded.



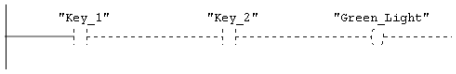
Open **OB1** in the project window "Getting Started ONLINE."

The LAD/STL/FBD program window is opened.



Activate the function **Debug > Monitor**.

Debugging with Ladder Logic

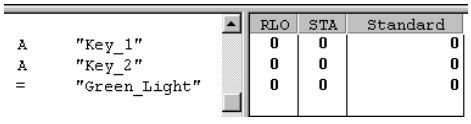


The series circuit in Network 1 is displayed in Ladder Logic. The current path is represented as a full line up to Key 1 (I 0.1); this means that power is already being applied to the circuit.

Debugging with Function Block Diagram

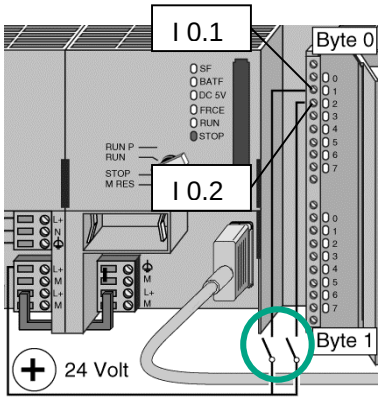
The signal state is indicated by "0" and "1." The dotted line means that there is no result of logic operation.

Debugging with Statement List



For Statement List the following is displayed in tabular form:
– Result of logic operation (RLO)
– Status bit (STA)
– Standard status (STANDARD)

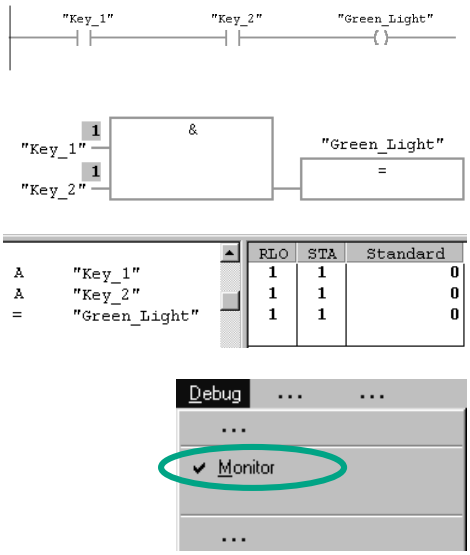
Using **Options > Customize** you can change the way in which the programming language is represented during testing.



Now press both keys in your test configuration.

The diodes for input I 0.1 and I 0.2 light up on the input module.

The diode for output Q 4.0 lights up on the output module.



In the graphic programming languages Ladder Logic and Function Block Diagram, you can trace the test result by following the change in color in the programmed network. This color change shows that the result of logic operation is fulfilled up to this point.

With the Statement List programming language, the display in the STA and RLO columns changes when the result of logic operation is fulfilled.

Deactivate the function **Debug > Monitor** and close the window.

Then close the online window in the SIMATIC Manager.

We recommend you do not completely download extensive programs onto the CPU to run them, because diagnosing errors is more difficult due to the number of possible sources of an error. Instead, you should download blocks individually and then test them in order to obtain a better overview.

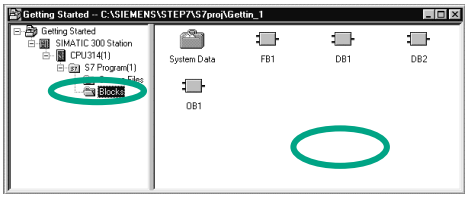
You can find more information under **Help > Contents** in the topics "Debugging" and "Testing with Program Status."

7.4 Testing the Program with the Variable Table

You can test individual program variables by monitoring and modifying them. The requirement for this is that you have established an online connection to the CPU, the CPU is in RUN-P mode, and the program has been downloaded.

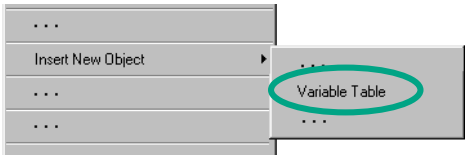
As with testing with program status, you can monitor the inputs and outputs in Network 1 (series circuit or AND function) in the variable table. You can also test the comparator for the engine speed in FB1 by presetting the actual speed.

Creating the Variable Table

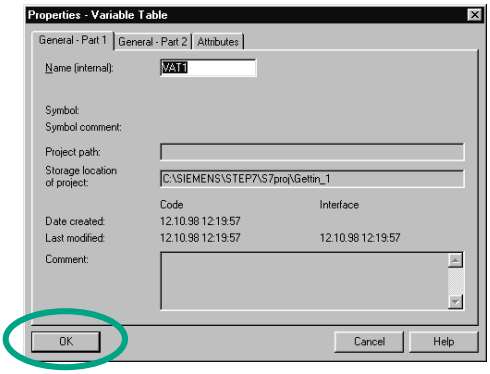


The starting point is the SIMATIC Manager again with the open project window "Getting Started Offline."

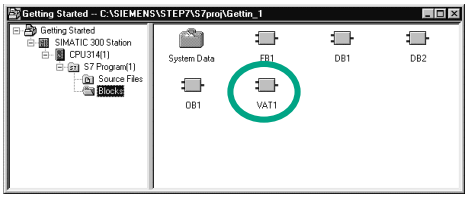
Navigate to the **Blocks** folder and click in the right half of the window with the right mouse button.



Use the right mouse button to insert a **Variable Table** from the pop-up menu.



Accept the default settings by closing the "Properties" dialog box with **OK**.

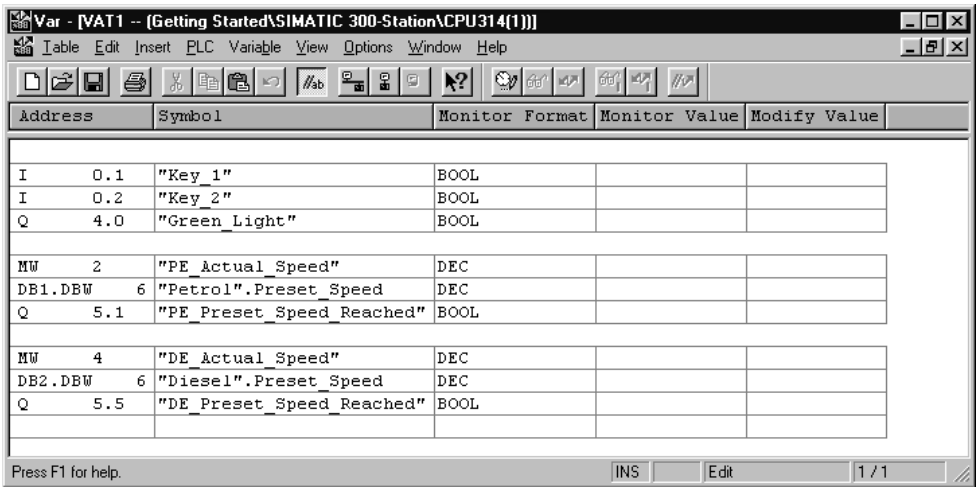


A VAT1 (variable table) is created in the Blocks folder.

Double-click to open **VAT1**; the "Monitoring and Modifying Variables" window will open.

At first, the variable table is empty. Enter the symbolic names or the addresses for the "Getting Started" example according to the illustration below. The remaining details will be added when you complete your entry with **Enter**.

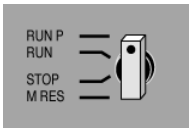
Change the status format of all the speed values to DEC (decimal) format. To do this, click the corresponding cell in the header (the cursor will change to an arrow over the Status Format column) and select DEC format using the right mouse button.



Save your variable table.

Switching the Variable Table Online

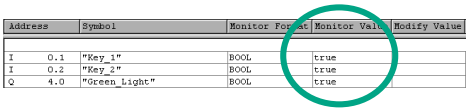
Click the **ON** button in the toolbar of the "Monitoring and Modifying Variables" window to establish a connection to the configured CPU. The word "ONLINE" will appear in the status bar.



Set the keyswitch of the CPU to **RUN-P** (if you have not already done so).

Monitoring Variables

Click the **Monitor Variables** button in the toolbar. The operating mode of the CPU is displayed in the status bar.



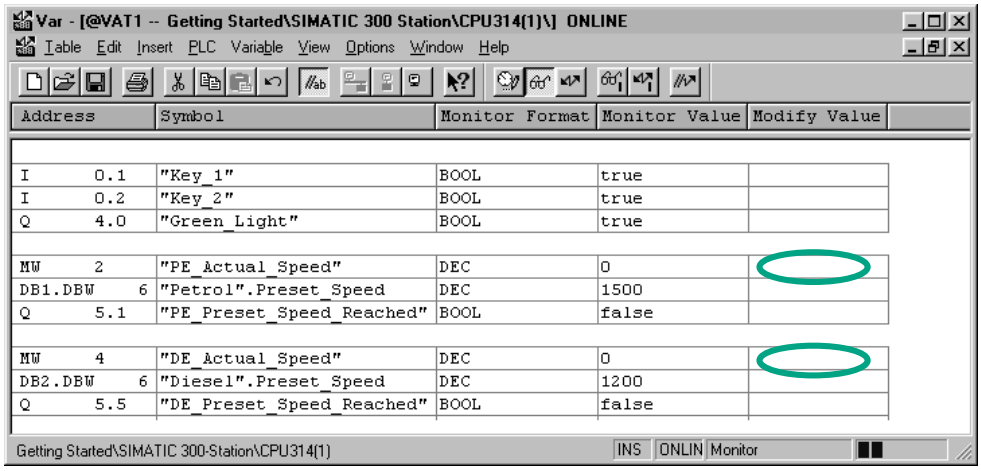
Address	Symbol	Monitor Format	Monitor Value	Modify Value
I 0.1	"Key_1"	BOOL	true	
I 0.2	"Key_2"	BOOL	true	
Q 4.0	"Green_Light"	BOOL	true	

Press Key 1 and Key 2 in your test configuration and monitor the result in the variable table.

The status values in the variable table will change from false to true.

Modifying Variables

Enter the value "1500" for the address MW2 in the Modify Value column and "1300" for the address MW4.

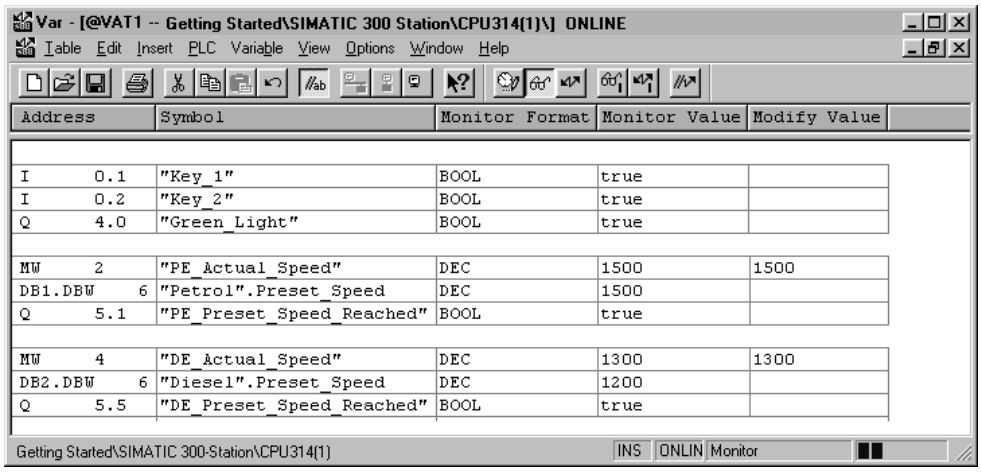


Address	Symbol	Monitor Format	Monitor Value	Modify Value
I 0.1	"Key_1"	BOOL	true	
I 0.2	"Key_2"	BOOL	true	
Q 4.0	"Green_Light"	BOOL	true	
MW 2	"PE_Actual_Speed"	DEC	0	
DB1.DBW 6	"Petrol".Preset_Speed	DEC	1500	
Q 5.1	"PE_Preset_Speed_Reached"	BOOL	false	
MW 4	"DE_Actual_Speed"	DEC	0	
DB2.DBW 6	"Diesel".Preset_Speed	DEC	1200	
Q 5.5	"DE_Preset_Speed_Reached"	BOOL	false	

Transfer the modify values to your CPU.

Following transfer, these values will be processed in your CPU. The result of the comparison becomes visible.

Stop monitoring the variables (click the button in the toolbar again) and close the window. Acknowledge any queries with **Yes** or **OK**.



Very large variable tables often cannot be displayed fully due to the limited screen space. If you have large variable tables, we recommend you create several tables for one S7 program using STEP 7. You can adapt the variable tables to precisely match your own test requirements.
You can assign individual names to variable tables in the same way as for blocks (for example, the name OB1_Network1 instead of VAT1). Use the symbol table to assign new names.

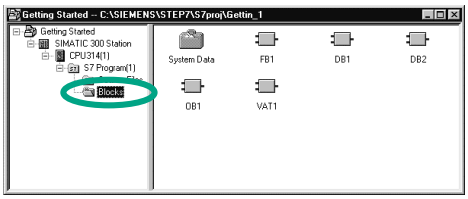
You can find more information under **Help > Contents** in the topics "Debugging" and "Testing with the Variable Table."

7.5 Evaluating the Diagnostic Buffer

If, in an extreme case, the CPU goes into STOP while processing an S7 program, or if you cannot switch the CPU to RUN after you have downloaded the program, you can determine the cause of the error from the events listed in the diagnostic buffer.

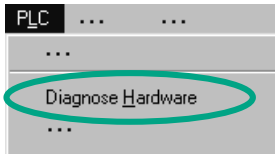
The requirement for this is that you have established an online connection to the CPU and the CPU is in STOP mode.

First turn the operating mode switch on the CPU to STOP.

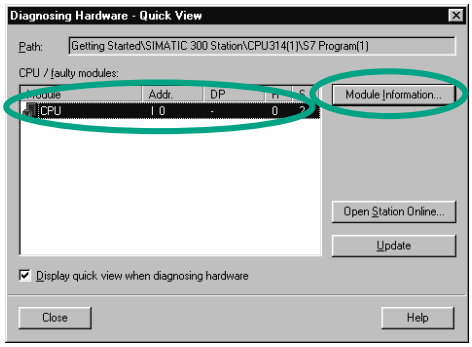


The starting point is the SIMATIC Manager again with the open project window "Getting Started Offline."

Select the **Blocks** folder.



If there are several CPUs in your project, first determine which CPU has gone into STOP.



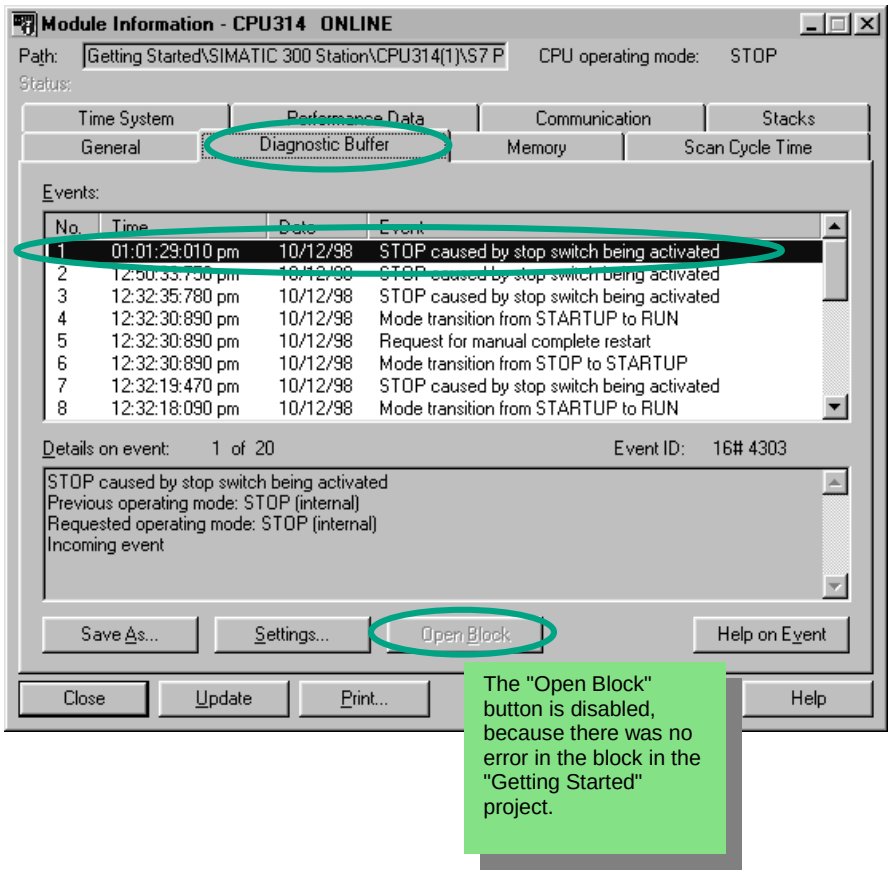
All the accessible CPUs are listed in the "Diagnosing Hardware" dialog box. The CPU with the STOP operating mode is highlighted.

The "Getting Started" project only has one CPU which is displayed.

Click **Module Information** to evaluate the diagnostic buffer of this CPU.

If only one CPU is connected, you can query the module information for this CPU directly using the menu command **PLC > Module Information**.

The "Module Information" window provides you with information on the properties and parameters of your CPU. Now select the "Diagnostic Buffer" tab to determine the cause of the STOP state.



The "Open Block" button is disabled, because there was no error in the block in the "Getting Started" project.

The latest event (number 1) is at the top of the list. The cause of the STOP state is displayed. Close all windows except for the SIMATIC Manager.

If a programming error caused the CPU to go into STOP mode, select the event and click the "Open Block" button.

The block is then opened in the familiar LAD/STL/FBD program window and the faulty network is highlighted.

With this chapter you have successfully completed the "Getting Started" sample project, from creating a project through to debugging the finished program. In the next chapters, you can extend your knowledge further by working through selected exercises.

You can find more information under **Help > Contents** in the topics "Calling the Module Information."

8 Programming a Function

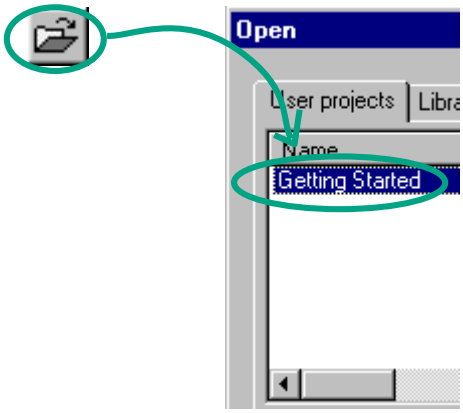
8.1 Creating and Opening Functions (FC)

Functions, like function blocks, are below the organization block in the program hierarchy. In order for a function to be processed by the CPU, it must also be called in the block above it in the hierarchy. In contrast to the function block, however, no data block is necessary.

With functions, the parameters are also listed in the variable declaration table, but static local data are not permitted.

You can program a function in the same way as a function block using the LAD/STL/FBD program window.

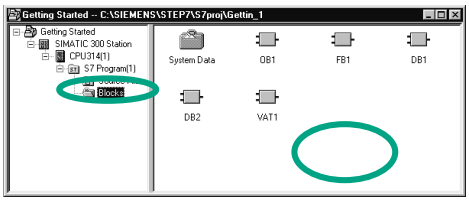
You should already be familiar with programming in Ladder Logic, Function Block Diagram, or Statement List (see Chapters 4 and 5) and also symbolic programming (see Chapter 3).



If you have worked through the "Getting Started" sample project in Chapters 1 to 7, open this now.

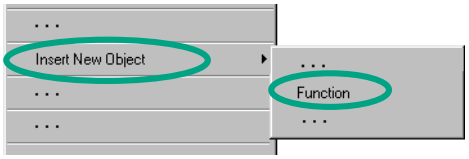
If not, create a new project in the SIMATIC Manager using the menu command **File > "New Project" Wizard**. To do this, follow the instructions in Section 2.1 and rename the project "Getting Started Function."

We will continue with the "Getting Started" project. However, you can still carry out each step using a new project.

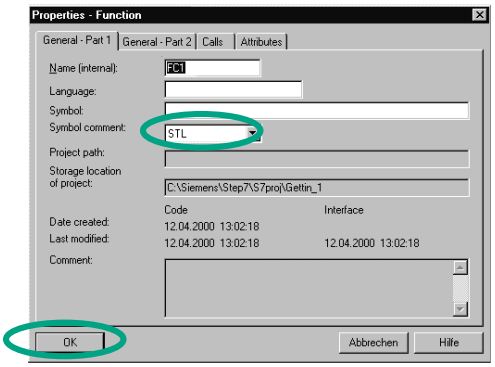


Navigate to the **Blocks** folder and open it.

Click in the right half of the window with the right mouse button.

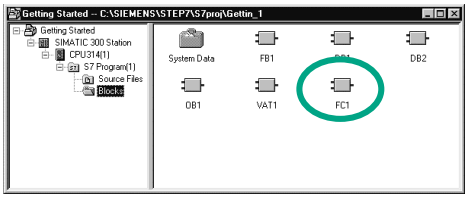


Insert a **Function** (FC) from the pop-up menu.



In the "Properties – Function" dialog box, accept the name FC1 and select the required programming language.

Confirm the remaining default settings with **OK**.



The function FC1 is added to the Blocks folder.

Double-click to open **FC1**.

In contrast to the function block, no static data can be defined in the variable declaration table for a function.

The static data defined in a function block are retained when the block is closed. Static data can be, for example, the memory bits used for the "Speed" limit values (see Chapter 5).

To program the function, you can use the symbolic names from the symbol table.

You can find more information under **Help > Contents** in the topics "Working Out the Automation Concept," "Basics of Designing a Program Structure," and "Blocks in the User Program."

8.2 Programming Functions

In this section, you will program a timer function in our example. The timer function enables a fan to switch on as soon as an engine is switched on (see Chapter 5), and the fan then continues running for four seconds after the engine is switched off (off-delay).

As mentioned earlier, you must specify the input and output parameters of the function ("in" and "out" declaration) in the variable declaration table.

The LAD/STL/FBD program window is open. You work with this variable declaration table in the same way as with the table for the function block (see Chapter 5).

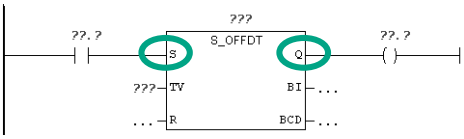
Enter the following declarations:

Address	Decl.	Name	Type	Initial Valu	Comment
0.0	in	Engine_On	BOOL		Signal for switching on the engine
2.0	in	Timer_Function	TIMER		Timer function used for the switch-off delay
4.0	out	Fan_On	BOOL		Signal for switching on the fan
	in_out				
	temp				

Programming the Timer Function in Ladder Logic

Select the current path for entering the Ladder instruction.

Navigate in the Program Elements catalog until you reach the element **S_OFFDT** (start off-delay timer), and select the element.

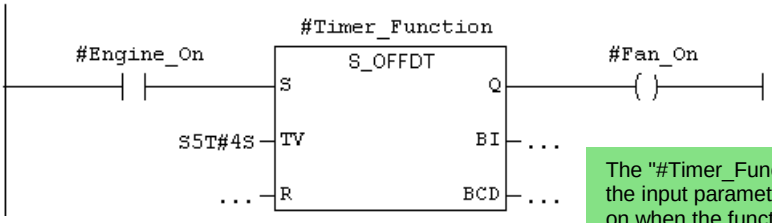


Insert a normally open contact in front of input **S**.
Insert a coil after output **Q**.

Select the question marks and enter the corresponding names from the variable declaration table (the # sign is assigned automatically).

Set the delay time at input TV of S_OFFDT. Here, S5T#4s means that a constant has been defined with the data type S5Time#(S5T#), lasting four seconds (4s).

Then save the function and close the window.



The "#Timer_Function" is started with the input parameter "#Engine_On." Later on when the function is called in OB1, it will be supplied once with the parameters for the petrol engine and once with the parameters for the diesel engine (for example, T1 for "PE_Follow_on"). You will enter the symbolic names of these parameters later in the symbol table.

Programming the Timer Function in Statement List

```
A     #Engine_On
L     S5T#4S
SF    #Timer_Function
A     #Timer_Function
=     #Fan_On
```

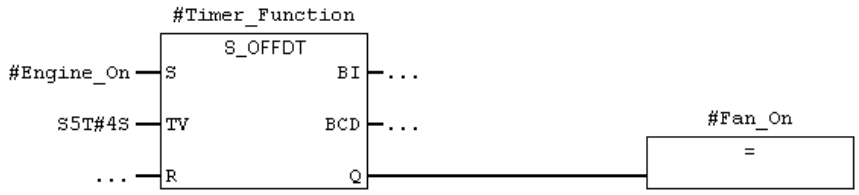
If you are programming in Statement List, select the input area below the network and enter the statement as shown here.

Then save the function and close the window.

Programming the Timer Function in Function Block Diagram

If you are programming in Function Block Diagram, select the input area below the network and enter the FBD program below for the timer function.

Then save the function and close the window.



In order for the timer function to be processed, you need to call the function in a block which is higher up in the block hierarchy (in our example, in OB1).

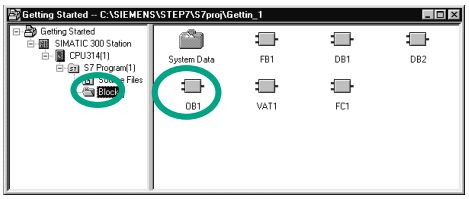
You can find more information under **Help > Contents** in the topics "Calling Reference Helps," "The STL, FBD, or LAD Language Description," and "Timer Instructions."

8.3 Calling the Function in OB1

The call for the function FC1 is carried out in a similar way to the call for the function block in OB1. All the parameters of the function are supplied in OB1 with the corresponding addresses of the petrol or diesel engine.

Since these addresses are not yet defined in the symbol table, the symbolic names of the addresses will now be added.

An address is part of a STEP 7 statement and specifies what the processor should execute the instruction on. Addresses can be absolute or symbolic.



The SIMATIC Manager is open with the "Getting Started" project or your new project.

Navigate to the **Blocks** folder and open **OB1**.

The LAD/STL/FBD program window opens.

If you copied the symbol table from a sample project (GS-LAD_Example, GS-STL_Example, or GS-FBD_Example) to your "Getting Started" project in Chapter 4, you do not need to add any symbols now.

Adding Symbolic Names at a Later Stage

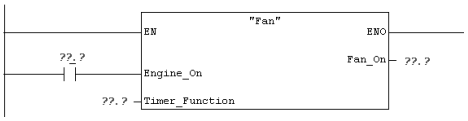
Open the symbol table from the LAD/STL/FBD program window using the menu command **Options > Symbol Table** and use the scroll bars at the right-hand edge of the window to scroll to the end of the symbol table.

Now add the following symbols to the symbol table:

Symbol	Address	Data Type	Comment
DE_Follow_On	T 2	TIMER	Follow-on time for diesel engine fan
PE_Follow_On	T 1	TIMER	Follow-on time for petrol engine fan
Fan	FC 1	FC 1	Fan control
PE_Fan_On	Q 5.2	BOOL	Command for switching on petrol engine fan
DE_Fan_On	Q 5.6	BOOL	Command for switching on diesel engine fan

Programming the Call in Ladder Logic

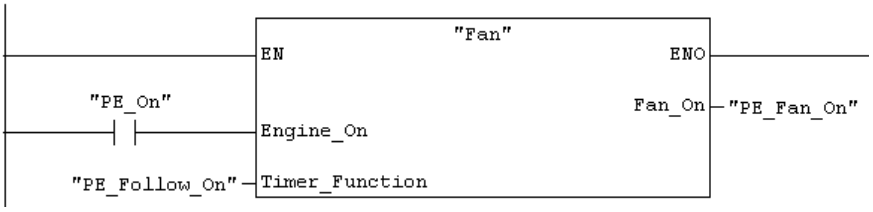
You are in **LAD** view. Insert a new network (No. 6). Then navigate in the Program Elements catalog until you reach FC1 and insert the function.



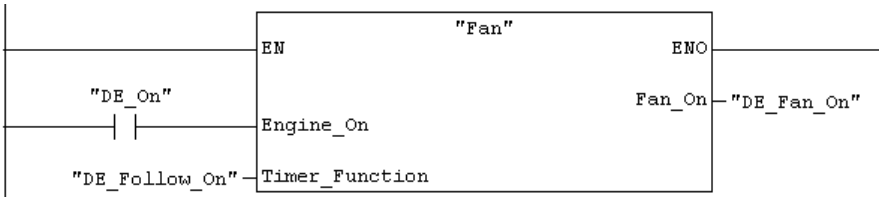
Insert a normally open contact in front of "Engine_On."

Using the menu command **View > Display >Symbolic Representation**, you can toggle between symbolic and absolute addresses.

Click the question marks for the FC1 call and insert the symbolic names.



Program the call for the function FC1 in Network 7 using the addresses for the diesel engine. You can do this in the same way as for the previous network (you have already added the addresses for the diesel engine to the symbol table).



Save the block and then close the window.

Activate the menu command **View >Display > Symbol Information** to view the information on individual addresses in each network.

To display several networks on the screen, deactivate the menu command **View > Display > Comment** and, if necessary **View > Display > Symbol Information**.

Using the menu command **View > Zoom Factor**, you can change the size of the networks displayed.

Programming the Call in Statement List

```
Network 6 : Controlling the Fan for the Petrol Engine
    CALL  "Fan"
     Engine_On      :="PE_On"
     Timer_Function:="PE_Follow_On"
     Fan_On         :="PE_Fan_On"

Network 7 : Controlling the Fan for the Diesel Engine
     CALL  "Fan"
      Engine_On      :="DE_On"
      Timer_Function:="DE_Follow_On"
      Fan_On         :="DE_Fan_On"
```

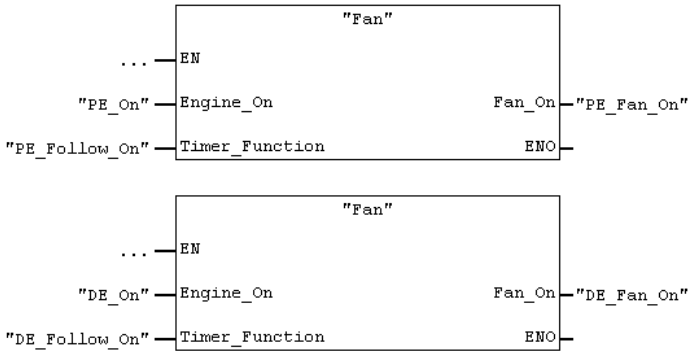
If you are programming in Statement List, select the input area below a new network and enter the STL statements shown here.

Then save the call and close the window.

Programming the Call in Function Block Diagram

If you are programming in Function Block Diagram, select the input area below a new network and enter the FBD instructions shown below.

Then save the call and close the window.



The call for the functions was programmed as an unconditional call in our example; that is, the function will always be processed.

Depending on the requirements of your automation task, you can make the call for a function or function block dependent on certain conditions; for example, an input or a preceding logic operation. The EN input and the ENO output are provided in the box for programming conditions.

You can find more information under **Help > Contents** in the topics "Calling Reference Helps," "The LAD, FBD, or STL Language Description," or "Program Control Instructions."

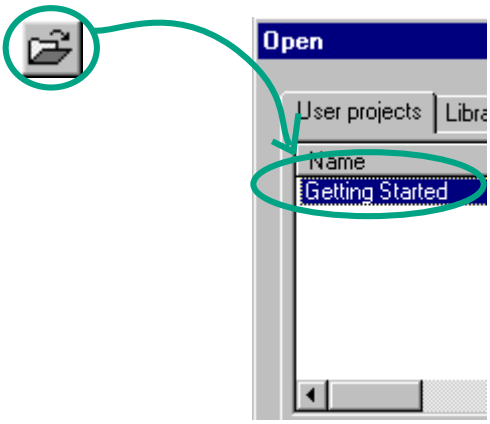
9 Programming a Shared Data Block

9.1 Creating and Opening Shared Data Blocks

If there are not enough internal memory bits in a CPU to save all the data, you can store specific data in a shared data block.

The data in a shared data block are available to every other block. An instance data block, on the other hand, is assigned to one specific function block, and its data are only available locally in this function block (see Section 5.5).

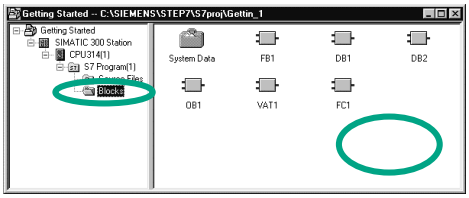
You should already be familiar with programming in Ladder Logic, Function Block Diagram, or Statement List (see Chapters 4 and 5) and also symbolic programming (see Chapter 3).



If you have worked through the "Getting Started" sample project in Chapters 1 to 7, open this now.

If not, create a new project in the SIMATIC Manager using the menu command **File > "New Project" Wizard**. To do this, follow the instructions in Section 2.1 and rename the project "Getting Started Function."

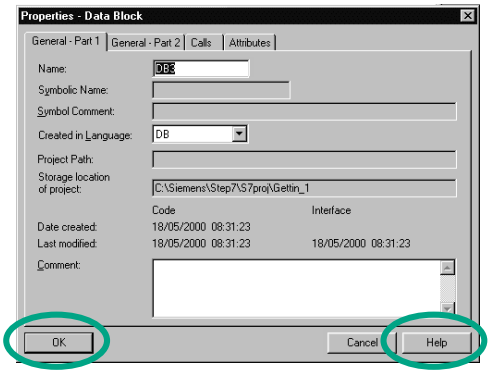
We will continue with the "Getting Started" project. However, you can still carry out each step using a new project.



Navigate to the **Blocks** folder and open it.

Click in the right half of the window with the right mouse button.

Insert a **Data Block** (DB) from the pop-up menu.

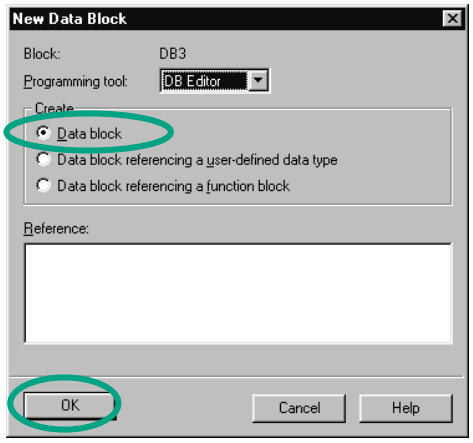


In the "Properties – Data Block" dialog box, accept all the default settings with **OK**.

Use the "Help“ Button for further information.

The data block DB3 has been added to the **Blocks** folder.

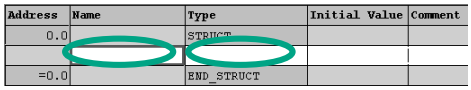
Double-click to open **DB3**.



In the "New Data Block" dialog box which then appears, activate the option **Data block**.
Close the dialog box with **OK**.

Remember: In Section 5.5, you generated an instance data block by activating the option "Data block referencing a function block." In contrast, using "Data block" you create a shared data block.

Programming Variables in the Data Block



Address	Name	Type	Initial Value	Comment
0.0		STRUCT		
=0.0		END_STRUCT		

Enter "PE_Actual_Speed" in the Name column.

Click with the right mouse button to select the type using the menu command **Elementary Types > INT** from the pop-up menu.

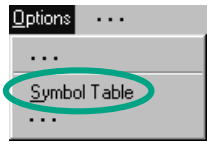
In the example below, three shared data are defined in DB3. Enter these data accordingly in the variable declaration table.

Address	Name	Type	Initial Value	Comment
0.0		STRUCT		
+0.0	PE_Actual_Speed	INT	0	Actual speed for petrol engine
+2.0	DE_Actual_Speed	INT	0	Actual speed for diesel engine
+4.0	Preset_Speed_Reached	BOOL	FALSE	Both engines have reached the preset speed
=6.0		END_STRUCT		

The variables for the actual speeds in the data block "PE_Actual_Speed" and "DE_Actual_Speed" are treated in the same way as the memory words MW2 (PE_Actual_Speed) and MW4 (DE_Actual_Speed). This can be seen in the next chapter.

Save the shared data block.

Assigning Symbols



You can also assign symbolic names to data blocks.

Open the **Symbol Table** and enter the symbolic name "S_Data" for the data block DB3.

If you copied the symbol table from a sample project (zEn01_02_STEP7__STL_1-10, zEn01_06_STEP7__LAD_1-10 or zEn01_04_STEP7__FBD_1-10) to your "Getting Started" project in Chapter 4, you do not need to add any symbols now.

Symbol	Address	Data Type	Comment
...	...	...	...
S_Data	DB 3	DB 3	Shared data block

Save the symbol table and close the "Symbol Editor" window.

Also close the variable declaration table for the shared data block.

Shared data blocks in the variable declaration table:

Using the menu command **View > Data View**, you can change the actual values of the data type INT in the table for the shared data block (see Section 5.5).

Shared data blocks in the symbol table:

In contrast to the instance data block, the data type for the shared data block in the symbol table is always the absolute address. In our example, the data type is "DB3." With the instance data block, the corresponding function block is always specified as the data type.

You can find more information under **Help > Contents** in the topics "Programming Blocks" and "Creating Data Blocks."

10 Programming a Multiple Instance

10.1 Creating and Opening a Higher-Level Function Block

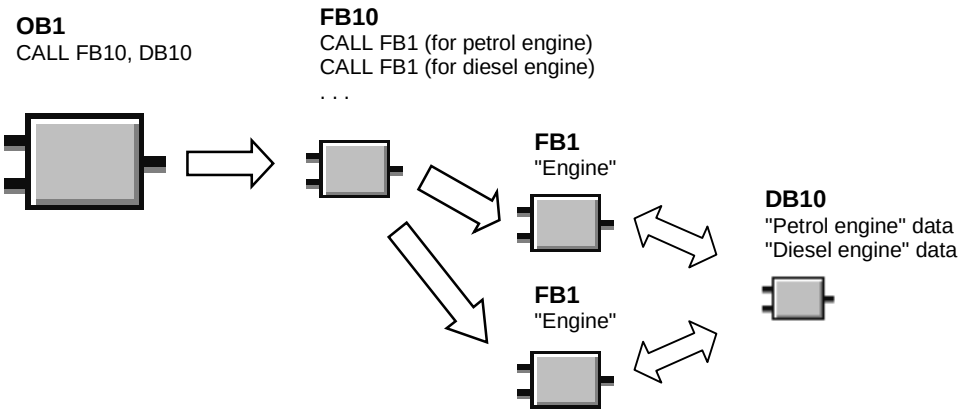
In Chapter 5 you created a program for controlling an engine with the function block "Engine" (FB1). When the function block FB1 was called in the organization block OB1, it used the data blocks "Petrol" (DB1) and "Diesel" (DB2). Each data block contained the different data for the engines (for example, #Setpoint_Speed).

Now imagine that you require other programs to control the engine for your automation task; for example, a control program for a rapeseed oil engine, or a hydrogen engine, etc.

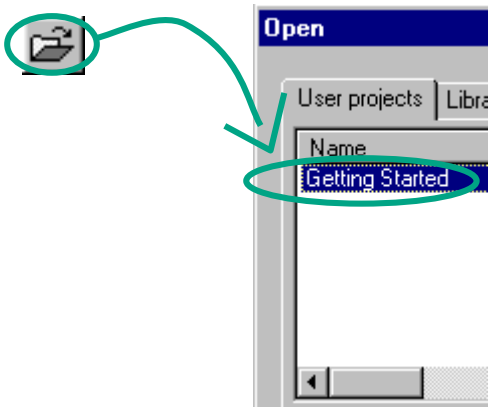
Following the procedure you have learned so far, you would now use FB1 for each additional engine control program and assign a new data block each time with the data for this engine; for example, FB1 with DB3 to control the rapeseed oil engine, FB1 with DB4 for the hydrogen engine, etc. The number of blocks would increase significantly as you created new engine control programs.

By working with multiple instances, on the other hand, you can reduce the number of blocks. To do this, you create a new, higher-level function block (in our example, FB10), and call the unchanged FB1 in it as a "local instance." For each call, the subordinate FB1 stores its data in data block DB10 of the higher-level FB10. This means that you do not have to assign any data blocks to FB1. All the function blocks refer back to a single data block (here DB10).

The data blocks DB1 and DB2 are integrated in DB10. To do this, you must declare FB1 in the static local data of FB10.

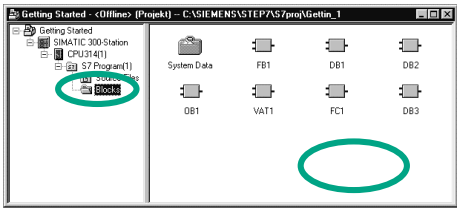


You should already be familiar with programming in Ladder Logic, Function Block Diagram, or Statement List (see Chapters 4 and 5) and also symbolic programming (see Chapter 3).



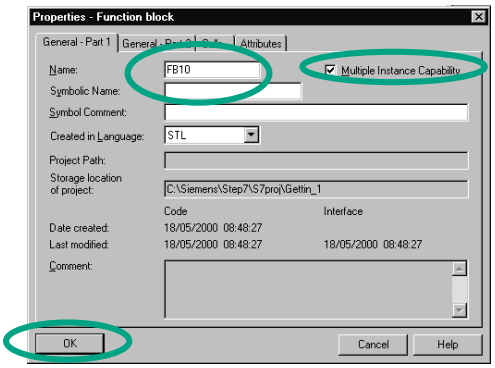
If you have worked through the "Getting Started" example in Chapters 1 to 7, open the "Getting Started" project.

If not, open one of the following projects in the SIMATIC Manager:
ZEn01_05_STEP7__**LAD**_1-9 for Ladder Logic,
ZEn01_01_STEP7__**STL**_1-9 for Statement List
ZEn01_03_STEP7__**FBD**_1-9 for Function Block Diagram.



Navigate to the **Blocks** folder and open it.

Click with the right mouse button in the right half of the window and insert a function block using the pop-up menu.



Change the name of the block to FB10 and select the required programming language.

Activate **Multiple instance FB** (if necessary) and accept the remaining default settings with **OK**.

FB10 has been added to the Blocks folder. Double-click to open **FB10**.

You can create multiple instances for any function block, even for valve control programs, for example. If you want to work with multiple instances, note that both the calling and the called function blocks must have multiple instance capability.

You can find more information under **Help > Contents** in the topics "Programming Blocks" and "Creating Blocks and Libraries."

10.2 Programming FB10

To call FB1 as a "local instance" of FB10, a static variable must be declared with a a different name for each planned call of FB1. Here, the data type is FB1 ("Engine").

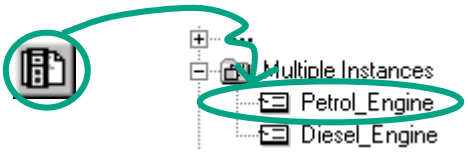
Filling out the Variable Declaration Table

The LAD/STL/FBD program window is open. Declare the following variables for the FB1 call:

Address	Decl.	Name	Type	Initial Valu	Comment
	in				
0.0	out	Preset_Speed_Reached	BOOL	FALSE	Both engines have reached the preset speed
	in_out				
2.0	stat	Petrol_Engine	"Engine"		First local instance of FB1 "Engine"
10.0	stat	Diesel_Engine	"Engine"		Second local instance of FB1 "Engine"
0.0	temp	PE_Preset_Speed_Reached	BOOL		Preset speed reached (petrol engine)
0.1	temp	DE_Preset_Speed_Reached	BOOL		Preset speed reached (diesel engine)

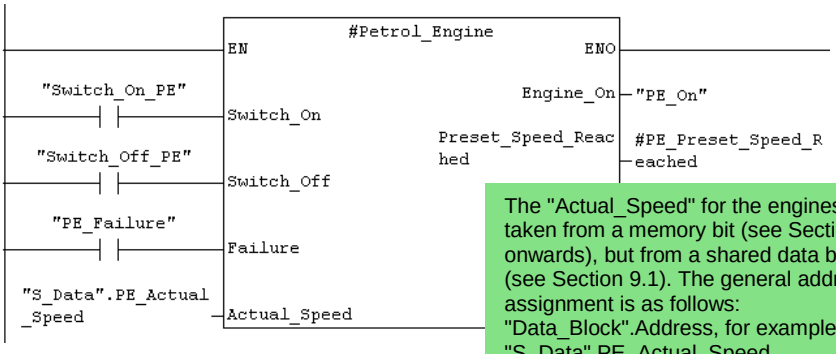
The declared local instances will then appear in the Program Elements catalog under "Multiple Instances."

Programming FB10 in Ladder Logic



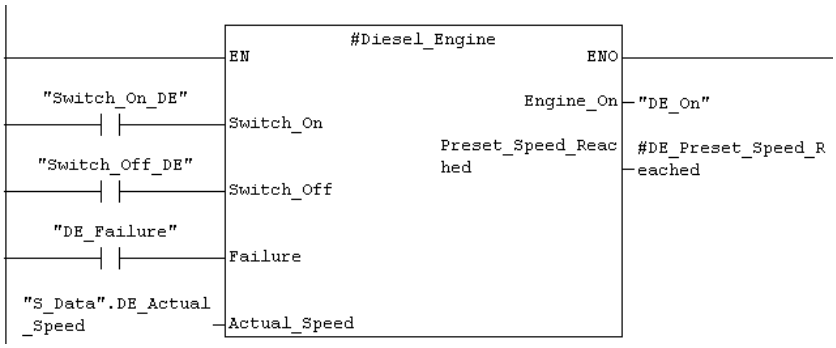
Insert the call "Petrol_Engine" as the multiple-instance block "Petrol_Engine" in Network 1.

Then insert the required normally open contacts and complete the call with the symbolic names.

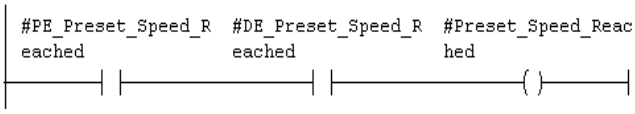


The "Actual_Speed" for the engines is not taken from a memory bit (see Section 5.6 onwards), but from a shared data block (see Section 9.1). The general address assignment is as follows: "Data_Block".Address, for example: "S_Data".PE_Actual_Speed.

Insert a new network and program the call for the diesel engine. Proceed in the same way as for Network 1.



Insert a new network and program a series circuit with the corresponding addresses. Then save your program and close the block.



The temporary variables ("PE_Setpoint_Reached" and "DE_Setpoint_Reached") are supplied to the output parameter "Setpoint_Reached," which is then processed further in OB1.

Programming FB10 in Statement List

```
CALL  #Petrol_Engine
 Switch_On            :="Switch_On_PE"
 Switch_Off           :="Switch_Off_PE"
 Failure              :="PE_Failure"
 Actual_Speed         :="S_Data".PE_Actual_Speed
 Engine_On            :="PE_On"
 Preset_Speed_Reached:=#PE_Preset_Speed_Reached

CALL  #Diesel_Engine
 Switch_On            :="Switch_On_DE"
 Switch_Off           :="Switch_Off_DE"
 Failure              :="DE_Failure"
 Actual_Speed         :="S_Data".DE_Actual_Speed
 Engine_On            :="DE_On"
 Preset_Speed_Reached:=#DE_Preset_Speed_Reached

A     #PE_Preset_Speed_Reached
A     #DE_Preset_Speed_Reached
=     #Preset_Speed_Reached
```

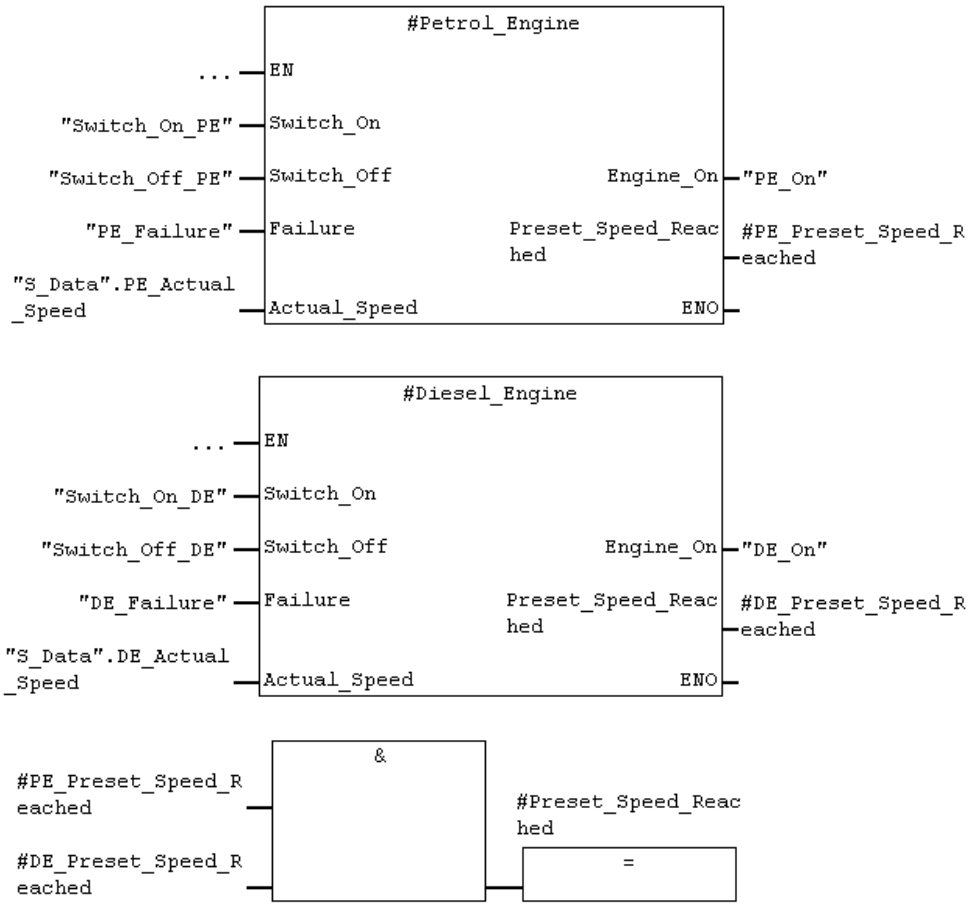
If you are programming in Statement List, select the input area under a new network and enter the STL instructions shown here.

Then save your program and close the block.

Programming FB10 in Function Block Diagram

If you are programming in Function Block Diagram, select the input area under a new network and enter the FBD instructions below.

Then save your program and close the block.



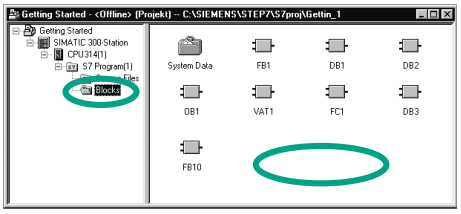
To edit both calls for FB1 in FB10, FB10 must be called itself.

Multiple instances can only be programmed for function blocks. Creating multiple instances for functions (FCs) is not possible.

You can find more information under **Help > Contents** in the topics "Programming Blocks," "Creating Logic Blocks," and "Multiple Instances in the Variable Declaration Table."

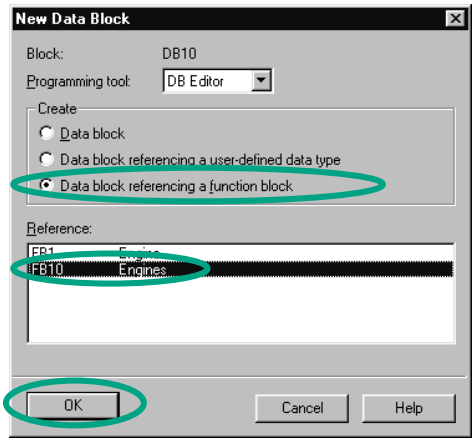
10.3 Generating DB10 and Adapting the Actual Value

The new data block DB10 will replace the data blocks DB1 and DB2. The data for the petrol engine and the diesel engine are stored in DB10 and will be required later for calling FB10 in OB1 (see "Calling FB1 in OB1" from Section 5.6 onwards).



Create the data block DB10 in the **Blocks** folder of the "Getting Started" project in the SIMATIC Manager using the pop-up menu.

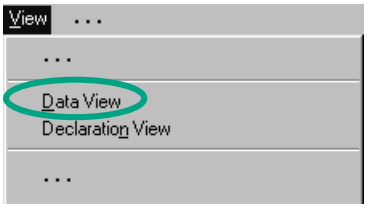
To do this, change the name of the data block to DB10 in the dialog box which appears and confirm the remaining settings with **OK**.



The data block DB10 has been added. Open this block to view the "New Data Block" dialog box.

Activate the option **Data block referencing a function block** and select FB10.

Confirm the settings with **OK**.



The data block DB10 is opened. Select the menu command **View > Data View**.

The data view displays each individual variable in DB10, including the "internal" variables of the two calls for FB1 ("local instances").

The declaration view displays the variables as they are declared in FB10.

Change the actual value of the diesel engine to "1300," save the block, and then close it.

Address	Decl.	Name	Type	Initial Value	Actual Value	Comment
0.0	out	Preset_Speed_Reached	BOOL	FALSE	FALSE	Both engines have reached
2.0	stat:in	Petrol_Engine.Switch_On	BOOL	FALSE	FALSE	Switch on engine
2.1	stat:in	Petrol_Engine.Switch_Off	BOOL	FALSE	FALSE	Switch off engine
2.2	stat:in	Petrol_Engine.Failure	BOOL	FALSE	FALSE	Engine failure, causes the
4.0	stat:in	Petrol_Engine.Actual_Speed	INT	0	0	Actual engine speed
6.0	stat:out	Petrol_Engine.Engine_On	BOOL	FALSE	FALSE	Engine is switched on
6.1	stat:out	Petrol_Engine.Preset_Speed_Reached	BOOL	FALSE	FALSE	Preset speed reached
8.0	stat	Petrol_Engine.Preset_Speed	INT	1500	1500	Requested engine speed
10.0	stat:in	Diesel_Engine.Switch_On	BOOL	FALSE	FALSE	Switch on engine
10.1	stat:in	Diesel_Engine.Switch_Off	BOOL	FALSE	FALSE	Switch off engine
10.2	stat:in	Diesel_Engine.Failure	BOOL	FALSE	FALSE	Engine failure, causes the
12.0	stat:in	Diesel_Engine.Actual_Speed	INT	0	0	Actual engine speed
14.0	stat:out	Diesel_Engine.Engine_On	BOOL	FALSE	FALSE	Engine is switched on
14.1	stat:out	Diesel_Engine.Preset_Speed_Reached	BOOL	FALSE	FALSE	Preset speed reached
16.0	stat	Diesel_Engine.Preset_Speed	INT	1500	1300	Requested engine speed

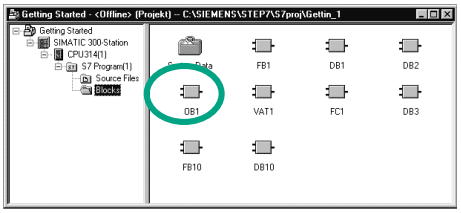
All the variables are now contained in the variable declaration table of DB10. In the first half, you can see the variables for calling the function block "Petrol_Engine" and in the second half the variables for calling the function block "Diesel_Engine" (see Section 5.5).

The "internal" variables of FB1 retain their symbolic names; for example, "Switch_On." The name of the local instance is now placed in front of these names; for example, "Petrol_Engine.Switch_On."

You can find more information under **Help > Contents** in the topics "Programming Blocks" and "Creating Data Blocks."

10.4 Calling FB10 in OB1

The call for FB10 is made in OB1 in our example. This call represents the same function which you have learned while programming and calling FB1 in OB1 (see Section 5.6 onwards.). Using multiple instances, you can replace Networks 4 and 5 programmed from Section 5.6 onwards.



Open **OB1** in the project in which you have just programmed FB10.

If you copied the symbol table from a sample project (ZEn01_01_STEP7__STL_1-10, ZEn01_05_STEP7__LAD_1-10 or ZEn01_03_STEP7__FBD_1-10) to your "Getting Started" project in Chapter 4, you do not need to add any symbols now.

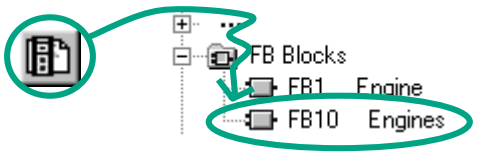
Defining Symbolic Names

The LAD/STL/FBD program window is open. Open the symbol table using the menu command **Options > Symbol Table** and enter the symbolic names for the function block FB10 and the data block DB10 in the symbol table.

Then save the symbol table and close the window.

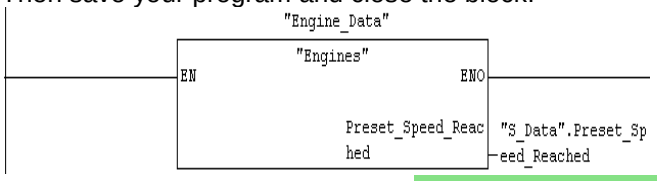
Symbol	Address	Data Type	Comment
...	...	...	...
Engines	FB 10	FB 10	Example of multiple instances
Engine_Data	DB 10	FB 10	Instance data block for FB10 10
...	...	...	...

Programming the Call in Ladder Logic



Insert a new network at the end of OB1 and add the call for **FB10** ("Engines").

Complete the call below with the corresponding symbolic names.
Delete the call for FB1 in OB1 (Networks 4 and 5 from Section 5.6 onwards), since we are now calling FB1 centrally via FB10.
Then save your program and close the block.



The output signal "Setpoint_Reached" for FB10 ("Engines") is passed on to the variable in the shared data block.

Programming the Call in Statement List

If you are programming in Statement List, select the input area under the new network and enter the STL instructions below. To do this, use the **FB Blocks > FB10 Engines** in the Program Elements catalog.

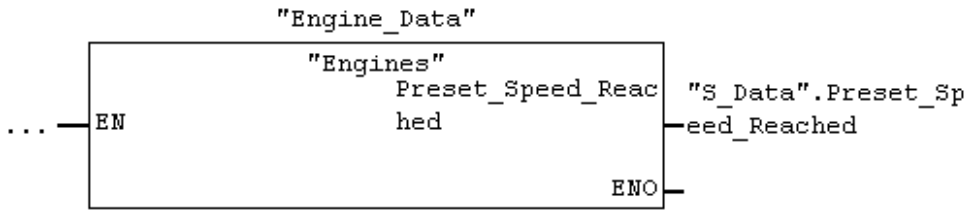
Delete the call for FB1 in OB1 (Networks 4 and 5 from Section 5.6 onwards), since we are now calling FB1 centrally via FB10.

Then save your program and close the block.

```
CALL  "Engines" , "Engine_Data"
 Preset_Speed_Reached:="S_Data".Preset_Speed_Reached
```

Programming the Call in Function Block Diagram

If you are programming in Function Block Diagram, select the input area under the new network and enter the FBD instructions below. To do this, use the **FB Blocks > FB10 Engines** in the Program Elements catalog.
Delete the call for FB1 in OB1 (Networks 4 and 5 from Section 5.6 onwards), since we are now calling FB1 centrally via FB10.
Then save your program and close the block.



If you require additional engine control programs for your automation task; for example, for gas engines, hydrogen engines, etc., you can program these as multiple instances in the same way and call them from FB10.

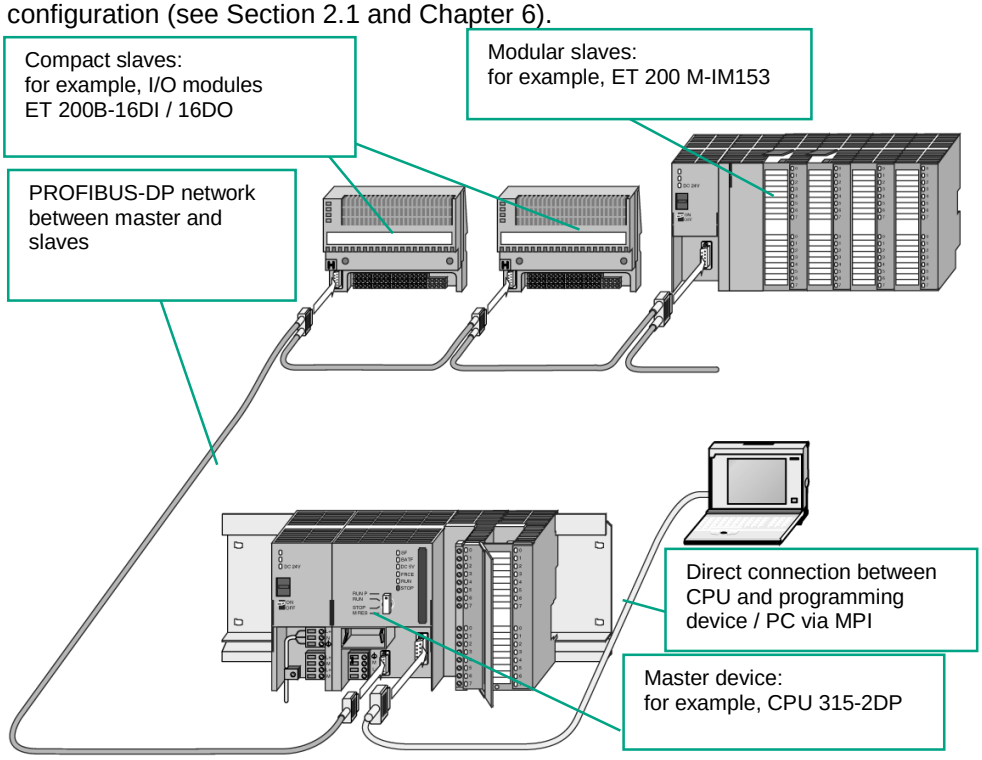
To do this, declare the additional engines as shown in the variable declaration table of FB10 ("Engines") and program the call for FB1 in FB10 (multiple instance in the Program Elements catalog). You can then define the new symbolic names; for example, for the switch-on and switch-off procedures in the symbol table.

You can find more information under **Help > Contents** in the topics "Calling References Helps," "The STL, FBD, or LAD Language Description," and "Program Control Instructions."

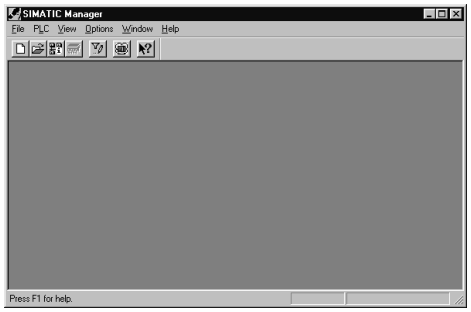
11 Configuring the Distributed I/O

11.1 Configuring the Distributed I/O with PROFIBUS DP

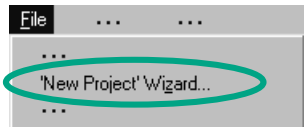
Automation systems with conventional configurations have the cable connections to the sensors and actuators inserted directly into the I/O modules of the central programmable logic controller. This often means a considerable amount of wiring is involved.
Using a distributed configuration, you can considerably reduce the amount of wiring involved by placing the input and output modules close to the sensors and actuators. You can establish the connection between the programmable logic controller, the I/O modules, and the field devices using the PROFIBUS DP.
You can find out how to program a conventional configuration in Chapter 6. It makes no difference whether you create a central configuration or a distributed configuration. You select the modules to be used from the hardware catalog, arrange them in the rack, and adapt their properties according to your requirements.
It would be an advantage when reading this chapter if you have already familiarized yourself with creating a project and programming a central configuration (see Section 2.1 and Chapter 6).



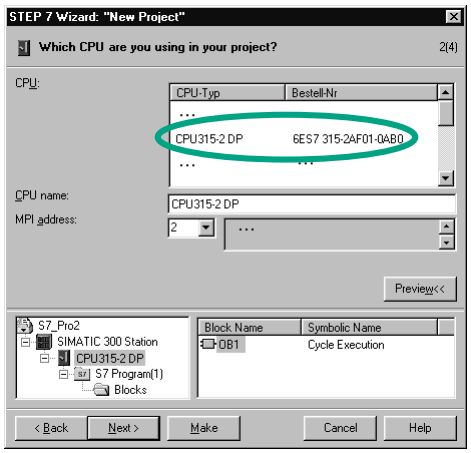
Creating a New Project



The starting point is the SIMATIC Manager. To make things easier to follow, close any open projects.



Create a **new project**.

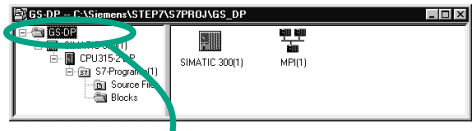


Select the **CPU 315-2DP** in the corresponding dialog box (CPU with PROFIBUS-DP network).

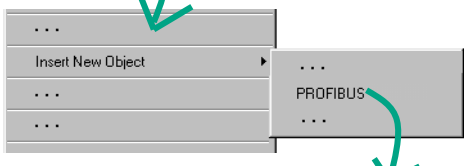
Now proceed in the same way as for Section 2.1 and assign the project the name "GS-DP" (Getting Started – Distributed I/O).

If you want to create your own configuration at this point, specify your CPU now. Note that your CPU must support distributed I/Os.

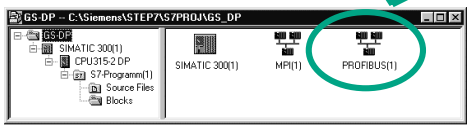
Inserting the PROFIBUS Network



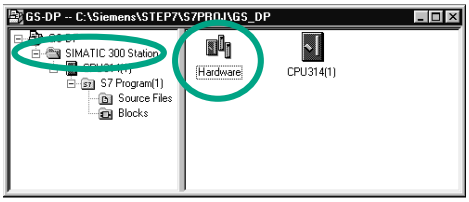
Select the folder **GS-DP**.



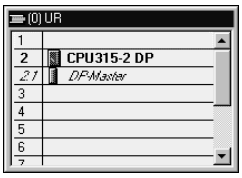
Insert the **PROFIBUS** network using the right mouse button in the right half of the window.



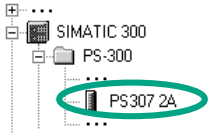
Configuring the Station



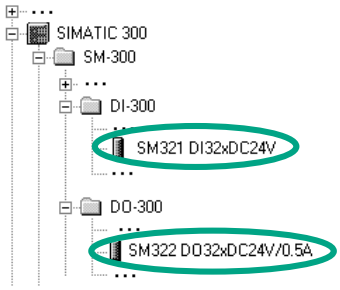
Select the folder **SIMATIC 300 Station** and double-click **Hardware**.
The "HW Config" window is opened (see Section 6.1).



The CPU 315-2 DP already appears in the rack. If necessary, open the Hardware catalog using the menu command **View > Hardware Catalog** or the corresponding button in the toolbar.



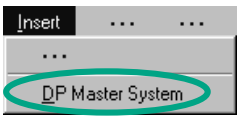
Drag and drop the power supply module **PS307 2A** into slot 1.



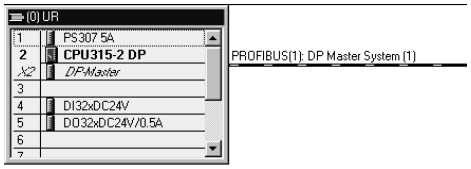
In the same way, insert the I/O modules **DI32xDC24V** and **DO32xDC24V/0.5A** in slots 4 and 5.

In addition to the CPU which supports the distributed I/O, you can also place other CPUs in the same rack (not described here).

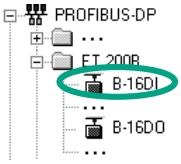
Configuring the DP-Master System



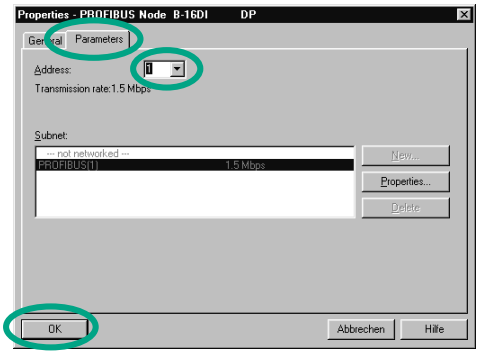
Select the DP master in slot 2.1 and insert a **DP-master system**.



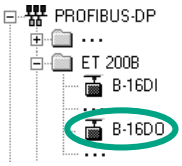
You can now move any objects which you place in the master system by dragging them with the left mouse button held down.



Navigate in the Hardware catalog until you reach the module **B-16DI** and insert this module in the master system (drag the object to the master system until the cursor changes to a "+" sign; then drop the object).

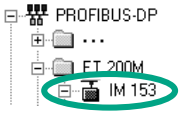


You can change the node address of the module you have inserted in the "Network Connection" tab of the "Properties" dialog box.
Confirm the suggested address **1** with **OK**.



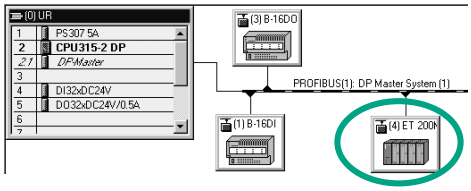
In the same way, drag and drop the module **B-16DO** onto the master system.

The node address is automatically adapted in the dialog box. Confirm this entry with **OK**.

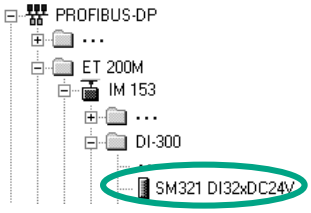


Drag and drop the interface module **IM153** onto the master system and confirm the node address again with **OK**.

In our example, we are using the default node addresses. However, you can change these addresses at any time to meet your requirements.



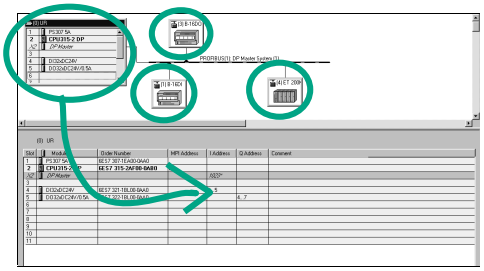
Select the **ET200M** in the network. The free slots for the ET200M are displayed in the lower configuration table. Select slot **4** here.



The ET200M itself can have additional I/O modules. Select, for example, the module **DI32xDC24V** for slot 4 and double-click this module to insert it.

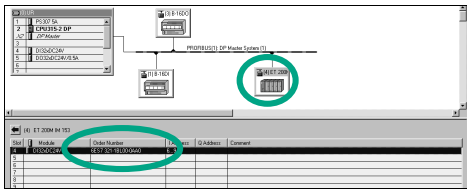
You should always make sure that you are in the right folder when using the Hardware catalog. For example, navigate to the ET200M folder to select modules for the ET200M.

Changing the Node Address



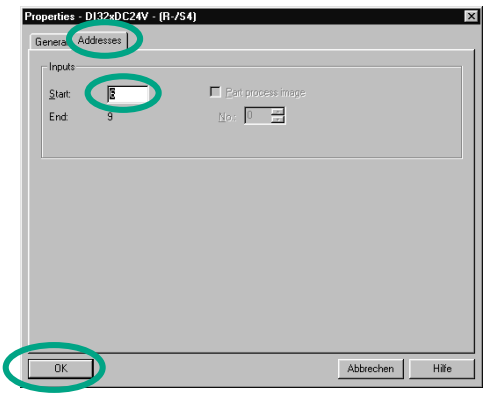
In our example, we do not need to change the node address. In practice, however, this is often necessary.

Select the other nodes one after another and check the input and output addresses. The "Configuring Hardware" application has adapted all the addresses, so there are no double assignments.



Let us imagine that you want to change the address of the ET200M:

Select the **ET200M** and double-click **DO32xDC24V/0.4A** (slot 4).



Now change the input addresses in the "Addresses" tab of the "Properties" dialog box from 6 to **12**.
Close the dialog box with **OK**.

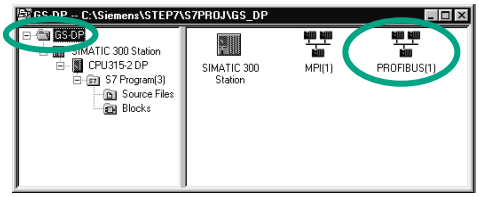
Finally, **save and compile** the distributed I/O configuration.

Close the window.

The menu command **Save and Compile** means that the configuration is automatically checked for consistency. If there are no errors, the system data are generated and can be downloaded to the programmable controller.

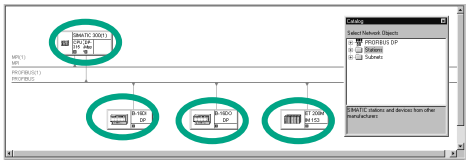
With **Save**, you can save the configuration even if it contains errors. However, you cannot then download the configuration to the programmable controller.

Optional: Configuring Networks



You can also configure the distributed I/O using the optional package "Configuring Networks."

Double-click the network **PROFIBUS (1)** in the SIMATIC Manager.



The "NETPRO" window is opened.

You can drag and drop additional DP slaves onto the PROFIBUS DP from the catalog of network objects.

Double-click any element to configure it. The "Configuring Hardware" window is opened.

Using the menu commands **Station > Consistency Check** ("Configuring Hardware" window) and **Network > Consistency Check** ("Configuring Networks" window), you can check the configuration for errors before saving. Any errors are displayed and STEP 7 will suggest possible solutions.

You can find more information under **Help > Contents** in the topics "Configuring the Hardware" and "Configuring the Distributed I/O."

Congratulations! You have worked through the Getting Started manual and learned the most important terms, procedures, and functions of STEP 7. Now you can get started on your first project.

If, while working on future projects, you are looking for specific functions or have forgotten any of the operating instructions in STEP 7, you can use our comprehensive Help on STEP 7.

If you want to extend your knowledge of STEP 7, there are a number of specialized training courses available. Your local Siemens representative will be happy to help you.

We wish you lots of success with your projects!

Siemens AG

Appendix A

Overview of the Sample Projects for the Getting Started Manual

- **ZEn01_02_STEP7__STL_1-10:**
  The programmed Chapters 1 to 10 including the symbol table in the STL programming language.
- **ZEn01_01_STEP7__STL_1-9:**
  The programmed Chapters 1 to 9 including the symbol table in the STL programming language.
- **ZEn01_06_STEP7__LAD_1-10:**
  The programmed Chapters 1 to 10 including the symbol table in the LAD programming language.
- **ZEn01_05_STEP7__LAD_1-9:**
  The programmed Chapters 1 to 9 including the symbol table in the LAD programming language.
- **ZEn01_04_STEP7__FBD_1-10:**
  The programmed Chapters 1 to 10 including the symbol table in the FBD programming language.
- **ZEn01_03_STEP7__FBD_1-9:**
  The programmed Chapters 1 to 9 including the symbol table in the FBD programming language.
- **ZEn01_07_STEP7__Dist_IO:**
  The programmed Chapter 11 with the distributed I/O.

Index

M

N

O

P

R

S

V

Siemens AG
A&D AS E 81
Oestliche Rheinbrueckenstr. 50
76181 Karlsruhe

From:

Your Name: .....

Your Title: .....

Company Name: .....

Street: .....

Country: .....

Phone: .....

Please check any industry that applies to you:

- ❒ Automotive
- ❒ Chemical
- ❒ Electrical Machinery
- ❒ Food
- ❒ Instrument and Control
- ❒ Nonelectrical Machinery
- ❒ Petrochemical
- ❒ Pharmaceutical
- ❒ Plastic
- ❒ Pulp and Paper
- ❒ Textiles
- ❒ Transportation
- ❒ Other .....

Remarks Form

Your comments and recommendations will help us to improve the quality and usefulness of our publications. Please take the first available opportunity to fill out this questionnaire and return it to Siemens.

Please give each of the following questions your own personal mark within the range from 1 (very good) to 5 (poor).

1. Do the contents meet your requirements? ☐
2. Is the information you need easy to find? ☐
3. Is the text easy to understand? ☐
4. Does the level of technical detail meet your requirements? ☐
5. Please rate the quality of the graphics/tables: ☐

Additional comments:

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....