SIEMENS

TIA Portal

SIMATIC
Creating the program

Getting Started

- Loading the block library
- Deleting program block Main [OB1]
- Copying program blocks
- Cyclic interrupt OB
- Copying tag tables
- Compiling a project
- Load project into the CPU
- Optimized block access
- Versioning a block
- Setting retentivity
- Activating the EN/ENO mechanism
- Using the comment function
- Local error handling
Legal information

Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent
damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert
symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are
graded according to the degree of danger.

⚠️ DANGER
indicates that death or severe personal injury will result if proper precautions are not taken.

⚠️ WARNING
indicates that death or severe personal injury may result if proper precautions are not taken.

⚠️ CAUTION
indicates that minor personal injury can result if proper precautions are not taken.

 Notices
indicates that property damage can result if proper precautions are not taken.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will
be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to
property damage.

Qualified Personnel

The product/system described in this documentation may be operated only by personnel qualified for the specific
task in accordance with the relevant documentation, in particular its warning notices and safety instructions.
Qualified personnel are those who, based on their training and experience, are capable of identifying risks and
avoiding potential hazards when working with these products/systems.

Proper use of Siemens products

Note the following:

⚠️ WARNING
Siemens products may only be used for the applications described in the catalog and in the relevant technical
documentation. If products and components from other manufacturers are used, these must be recommended
or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and
maintenance are required to ensure that the products operate safely and without any problems. The permissible
ambient conditions must be complied with. The information in the relevant documentation must be observed.

Trademarks

All names identified by ® are registered trademarks of Siemens AG. The remaining trademarks in this publication
may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software
described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the
information in this publication is reviewed regularly and any necessary corrections are included in subsequent
editions.
Table of contents

1 Loading the block library ........................................................................................................ 4
2 Deleting program block Main [OB1] ..................................................................................... 7
3 Copying program blocks ...................................................................................................... 9
4 Cyclic interrupt OB ............................................................................................................. 10
    4.1 Cyclic interrupt OB – Cycle time and phase ................................................................. 10
    4.2 Changing the cycle time ................................................................................................ 11
5 Copying tag tables .............................................................................................................. 12
6 Compiling a project ............................................................................................................ 13
7 Load project into the CPU .................................................................................................. 15
8 Optimized block access ...................................................................................................... 17
    8.1 Introduction .................................................................................................................. 17
    8.2 Expanding and reloading the optimized “Filling” data block ......................................... 18
9 Versioning a block .............................................................................................................. 23
10 Setting retentivity ............................................................................................................. 26
11 Activating the EN/ENO mechanism .................................................................................. 29
12 Using the comment function ............................................................................................ 31
13 Local error handling ........................................................................................................ 33
    13.1 Handle errors within block ......................................................................................... 33
    13.2 Loading blocks for local error handling ...................................................................... 35
    13.3 Generating errors without local error handling ........................................................... 37
    13.4 Generating errors with local error handling ............................................................... 38
Loading the block library

Introduction

In the following section, you will load the global library "ProgLib_ColorFillingStation". This library contains the blocks and tag tables that you need for the example project. This library is available as a ZIP file under "Getting Started S7-1500 / TIA V13 [http://www.automation.siemens.com/salesmaterial-as/interactive-manuals/getting-started_simatic-s7-1500/project/color_filling_station.zip]". You need to unzip this library before you import it to your project.

Global libraries

Global libraries are used to store elements that you want to reuse in other projects. You must create global libraries explicitly.

The following libraries are provided in the standard package:

- "Buttons and Switches"
  They offer a large selection of switches and buttons. The folders organize switches and buttons into categories. You can find the "System diagnostics indicator" object in the "DiagnosticsButtons" folder, for example. You use the "System diagnostics indicator" object for system diagnostics in your plant.

- "Monitoring and Control objects"
  This provides complex operator control and display objects in several designs as well as suitable control lights, buttons and switches.

Note

Library is write-protected

The "Open read-only" option is activated by default in the "Open global library" dialog. Click in the check box to open the library without write protection.
Procedure

1. Click on the "Libraries" tab.
2. Click "Open global library".
3. Select the "ProgLib_ColorFillingStation" file from the directory that contains the unzipped library folder and click "Open".
Result

The "ProgLib_ColorFillingStation" global library is open.
Deleting program block Main [OB1]

Introduction

In the following section, you will delete the automatically generated "Main [OB1]" program block from the project folder. A "Main [OB1]" program block is included in the program blocks of the example project.

Organization blocks (OBs) form the interface between the CPU operating system and the user program. These blocks are called by the operating system. At least one cycle OB must be available in an automation project.

![Diagram of OB1 execution](image)

OB1 (Main) is executed continuously by the operating system.

Possible interruption by other OBs for cyclic interrupts or errors
Deleting program block Main [OB1]

Procedure

1. Open the "Program blocks" folder in the project tree and then click the "Main [OB1]" program block.
2. Right-click to open the shortcut menu and then click "Delete".
3. Click "Yes" to confirm deletion of the block.

Result

The automatically generated "Main [OB1]" program block is deleted.
Copying program blocks

Introduction

In the following section, you will insert the program blocks from the "ProgLib_ColorFillingStation" global library into your project.

Procedure

1. Click on global library "ProgLib_ColorFillingStation".
2. Click the "Master copies" folder and then on "Programm_blocks".
3. Drag-and-drop the program block to be imported from the global library to the "Program blocks" folder.
4. Proceed as described in steps 2 and 3 for the other blocks.

Result

The program blocks are inserted in the project folder of the same name.
4.1 Cyclic interrupt OB – Cycle time and phase

Cycle time and phase offset can be changed

Main [OB35] is located below the program blocks inserted into the project. Main [OB35] is a cyclic interrupt organization block (cyclic interrupt OB). Cyclic interrupt OBs serve to start programs in periodic time intervals independently of the cyclic program execution. The start times of a cyclic interrupt OB are specified using the cycle time and the phase offset.

Cycle time

The cycle time determines the interval at which an OB is called. The cyclic interrupt OB has a cycle time of 100000 μs by default.

Phase offset

The phase offset is used to increase the accuracy of the processing intervals of cyclic interrupt programs. If an OB has the same or a common multiple clock pulse of another OB, both can be operated at a precise interval by a phase offset.
4.2 Changing the cycle time

Introduction

In the following section, you will change the cycle time for the "Main" program block.

Requirement

- The program block "Main" [OB35] is contained in the library
- The FB/FC calls exist

Procedure

1. Open the properties of the "Main" program block.
2. Select the "Cyclic interrupt" option under "General".
3. Enter the new value for the "Cycle time" and click "OK".

Result

The cycle time is changed.
Copying tag tables

Introduction

In the following section, you will insert the tag tables from the "ProgLib_ColorFillingStation" global library into your project.

Procedure

1. Open the "PLC tags" folder in the project navigation.
2. Open the "PLC_tags" folder.
3. Drag-and-drop the tag table to be imported from the global library to the "PLC tags" folder.
4. Proceed as described in step 3 for the other tag tables.

Result

The tag tables are inserted in the project folder of the same name.
Compiling a project

Introduction

In the next section, you will compile the "Color_Filling_Station" project.

Procedure

1. Select the "Color_Mixing_CPU" CPU in the project tree.
2. Right-click to open the shortcut menu and then select "Compile" > "Hardware and software (only changes)".
Compiling a project

Result

The project is compiled and ready for downloading.

Note

"Main" program block is updated

Open the "Main" program block after compilation. All instance data blocks have been created and the data blocks are updated.
Load project into the CPU

Introduction

In the next section, you will download the "Color_Filling_Station" project to the CPU.

Note

Displaying all compatible devices

If the desired CPU is not displayed after you have made the settings in the "Extended download to device" dialog, click the option "Show all compatible devices".

Procedure

1. Open the CPU shortcut menu and select "Download to device" > "Hardware and software (only changes)".
2. From the drop-down lists, select the PG/PC interface type, the interface and the connection with the subnet.
3. Select the CPU from the compatible devices in the subnet and click "Load".
4. Confirm the two "Assign IP address" dialogs with "Yes" and "OK".
5. In the "Load preview" dialog, select the alternative entry for all entries set to "No action" in the drop-down list and confirm open options.
6. Click "Load".
7. Confirm the "Start all" option and click "Finish".

Result

The project is downloaded to the CPU.
Optimized block access

8.1 Introduction

Operating principle

The "optimized data blocks" of the CPUs of the S7-1500 series are optimized for performance and are only programmed symbolically. By using the optimized data blocks, you make your program more efficient, because the declared tags are given symbolic names and no longer a fixed address.

You can create data blocks with any structure without paying attention to the physical arrangement of the individual tags. Quick access to the optimized data is always available because the data storage is optimized and managed by the system.

Changing data types increases the risk of error in the standard block. In the optimized block, changes lead to a reorganization of the data storage. Addressing remains unique.

To enable the subsequent editing of user programs that are already running in a CPU, the S7-1500 CPUs support the option of extending the interfaces of function or data blocks during runtime. You can download the modified blocks without setting the CPU to STOP and without affecting the actual values of tags already loaded.

In addition: You can define in the data block itself, which the values in the CPU are read-only for an HMI device ("Visible in HMI") or which can be written ("Accessible from HMI").
8.2 Expanding and reloading the optimized "Filling" data block

Introduction

In the following section, you will supplement the "Filling" data block with the date and time of the last filling and reload the data block. To do this, create a block for recording the date and time and enable the function "Download without reinitialization".

Note: The "Download without reinitialization" function protects the actual parameters of the data block from being overwritten during download to the CPU.

Advantages of symbolic addressing: The use of universally applied and meaningful symbols in the entire project makes the program code easier to read and understand. This gives you the following advantages:

- You do not have to write detailed comments.
- Data access is faster.
- No errors occur when accessing data.
- You no longer have to work with absolute addresses.
- The assignment of the symbol to the memory address is monitored by STEP 7, which means that all points of use are automatically updated when the name or the address of a tag changes.

Requirement

- The library has been loaded
- The project has been compiled and loaded into the CPU
8.2 Expanding and reloading the optimized "Filling" data block

Procedure

1. Open the "Filling" data block and the "Main" program block.
2. Enable the "Monitoring on/off" function for the "Main" program block.
3. In the "Main" program block, open the shortcut menu of the "FILLING' FillingLevel_CMYK_C" I/O in the 3 network with a right-click and select "Modify > Modify operand".
4. Enter a new value and click "OK".
5. Enable the "Download without reinitialization" function and the "Monitor all" function in the "Filling" data block.
6. Create a new parameter named "DT_Loc-T_Last_Filling" and select "Date_And_Time" as the data type.
7. Insert a normally closed contact into the "Main" program block in the 5 network, and interconnect it with the "FILLING_DONE" parameter.

8. Open the "Date & time" folder from the "Instructions" tab and insert the "RD_Loc_T" block in the "Main" program block.
9. Interconnect the "OUT" output with the "DT_Loc-T_Last_Filling" parameter and the "RED_VAL" output with the newly created "RED_VAL_Loc-T" parameter. Use the "LAD_Tanks_Filling_Process" data block as the storage location for the "RED_VAL_Loc-T" parameter.

10. Compile and download the project.
Result

The date and time of the last filling are reloaded. The actual parameters of the "Filling" data block are not overwritten.
Versioning a block

Introduction

The use of block types ensures a high degree of standardization in your projects. You can easily integrate function extensions to the block type into existing projects. Change tracking is ensured by versioning. In this example, you create a "LAD_Tanks_Filling" block as a type in the project library. As a function extension, replace the three instructions for the level calculation with CalculateBox, which performs all arithmetic functions. This optimization means that fewer temporary tags are required and that the switch between blocks with various programming languages is no longer necessary.

Procedure

1. Compile the "LAD_Tanks_Filling" block and then insert it in the project library under "Types".
2. Create a new block version with "Edit type".
3. Insert the CALCULATE instruction from the "Basic instructions > Mathematical functions" library.
4. Delete the MUL, DIV and SUB instructions from the block.
5. Insert two inputs into the `CALCULATE` instruction and interconnect the inputs.

6. Define the calculation formula and then interconnect the output.

7. Release the block version.
The revised version of the block type is saved in the library with a new version number.
Setting retentivity

Introduction

All tags are initialized with their configured start values during CPU startup, for example, after a power failure. The most recent values the tags had immediately before the interruption are overwritten with the initial values. To prevent this, define the tag as retentive. Retentive tags retain their values even after a restart.

In this example, the levels of paint storage tanks are backed up in the retentive memory area of the CPU.

Procedure

1. Connect to the CPU online.
2. Enable the retentivity for the "Cyan" entry in the "Filling" data block.
3. Load the change to the CPU.
4. Drag the "Watchtable" object from the library into the project. This object contains the fill level tags included a control value.
5. Transfer the control values to the CPU with "Modify now".

6. Close the online connection to the CPU. To simulate a power failure, disconnect the power supply to the CPU.

7. Reconnect the power supply and go online to the CPU. Enable "Monitor all" for the "Filling" DB.
**Result**

The fill level for "Cyan" is read from the retentive memory area. All other fill levels are re-initialized with their start value.
Activating the EN/ENO mechanism

Introduction

The EN/ENO mechanism in various instructions enables you to detect runtime errors and avoid a program crash. Newly inserted ENO instructions are disabled by default. You can then activate the ENO enable output. You can use this in a new network that has the fill level of all paint storage tanks reset to the start value (1000) at the same time.

Procedure

1. Open the Main[OB35] program block and insert the MOVE instruction into network 10.
2. Expand the instruction to a total of four outputs.
3. Insert a normally open contact before the MOVE instruction.
4. Insert a reset coil after the MOVE instruction.
5. Interconnect the inputs and outputs of the MOVE instruction.
6. Generate the instruction with the ENO shortcut menu.
Result

The EN/ENO mechanism is interconnected for this block. If there are no errors during execution, the ENO enable output has the signal state "1". If there are errors during execution, the ENO enable output has the signal state "0".
Using the comment function

Introduction

The MOVE and Reset instructions should be expanded with detailed commentary.

Procedure

1. Insert a comment using the shortcut menu.
2. Enter the comment text.

Result

The comments for the instruction and the coil are entered.
Local error handling

13.1 Handle errors within block

Procedure

Unlike the CPUs of the S7-300/400, CPUs of the S7-1500 go to STOP with errors much less often. If an error occurs, it is entered in the diagnostics buffer of the CPU. You avoid the CPU STOP by using local error handling at each block. You should preferably enable local error handling during development of the user program.
Local error handling

13.1 Handle errors within block

You can precisely evaluate the information and, for example, program the error handling in the block with STL/FBD/LAD and SCL programs. The block generates an error ID that is evaluated by the "GET_ERROR_ID" instruction. You can call the "GET_ERROR_ID" instruction in both the MAIN block and in the function blocks. The CPU remains in RUN mode.
13.2 Loading blocks for local error handling

Introduction

To illustrate the local error handling, load the blocks of the "ProgLib_LEH" library in the project. The blocks are used only to demonstrate the local error handling and are otherwise not used in the project.

Procedure

1. Open the global library, "ProgLib_LEH".
2. Copy the blocks from the master copies into the project.
3. Call the "LAD_Local_Error_Handling" function block in an empty network of the "Main" block.
4. Interconnect the parameters of the "LAD_Local_Error_Handling" function block with tags of the "LEH_InOutValues" data block.

5. Connect to the CPU online.

6. Compile and load the changes to the CPU.

**Result**

Use the "LEH_INDEX" tag at the "INDEX[0..100]" input parameter to trigger a programming error in the following. For example, if you set the input parameter to "101", an error at the output parameters is reported.
13.3 Generating errors without local error handling

Introduction

Perform the following steps to trigger a programming error without using the local error handling or creating a corresponding OB.

Procedure

1. Activate the "Monitor" function.
2. Set the value of the "LEH_INDEX" tag to an invalid value, for example, "101". In the Testing dialog, the ERROR LED flashes briefly and the CPU goes from RUN to STOP.
3. Switch to the diagnostics buffer. The error and the error response is displayed in the diagnostics buffer.
4. Set the CPU back to RUN.

Result

The transition from STOP in RUN resets the "LEH_INDEX" tag to the start value "0". This automatically solves the problem.
13.4 Generating errors with local error handling

Introduction

Perform the following steps to use "GET_ERR_ID" instruction and its ENO bit for the local error handling to respond to the error with an error message. This means the CPU remains in RUN mode.

Procedure

1. Open the "LAD_Local_Error_Handling" function block.
2. Insert the "GET_ERR_ID" instruction in the second network and interconnect the "ID" output.
3. Call the "ErrorID_to_ErrorText" function from the project tree.
4. Interconnect the parameters of the "ErrorID_to_ErrorText" function so that they can convert the error code into an error message.
5. Load the changes to the CPU.
6. Trigger an error in the "Main" organization block by entering an invalid value, for example, "101". An error message is output at the "ERROR_MESSAGE" parameter.
13.4 Generating errors with local error handling

Result

The error message is output as long as the error is not corrected. To correct the error, assign the "LEH_INDEX" tag a valid value or restart the CPU.