SoMachine V3.0

M238 PLC Diagnostic

PLC_Diagnoctic.project

Example Guide

04/2012

Basic

Intermediate

Expert

EIO0000000901.00

www.schneider-electric.com

Schneider Electric

The information provided in this documentation contains general descriptions and/or technical characteristics of the performance of the products contained herein. This documentation is not intended as a substitute for and is not to be used for determining suitability or reliability of these products for specific user applications. It is the duty of any such user or integrator to perform the appropriate and complete risk analysis, evaluation and testing of the products with respect to the relevant specific application or use thereof. Neither Schneider Electric nor any of its affiliates or subsidiaries shall be responsible or liable for misuse of the information contained herein. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to help ensure compliance with documented system data, only the manufacturer should perform repairs to components.

When devices are used for applications with technical safety requirements, the relevant instructions must be followed.

Failure to use Schneider Electric software or approved software with our hardware products may result in injury, harm, or improper operating results.

Failure to observe this information can result in injury or equipment damage.

Table of Contents

Safety Information

Important Information

NOTICE

Read these instructions carefully, and look at the equipment to become familiar with the device before trying to install, operate, or maintain it. The following special messages may appear throughout this documentation or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.

The addition of this symbol to a Danger or Warning safety label indicates that an electrical hazard exists, which will result in personal injury if the instructions are not followed.

This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

⚠ DANGER

DANGER indicates an imminently hazardous situation which, if not avoided, **will result in** death or serious injury.

⚠ WARNING

WARNING indicates a potentially hazardous situation which, if not avoided, **can result in** death or serious injury.

⚠ CAUTION

CAUTION indicates a potentially hazardous situation which, if not avoided, **can result in** minor or moderate injury.

CAUTION

CAUTION, used without the safety alert symbol, indicates a potentially hazardous situation which, if not avoided, **can result in** equipment damage.

PLEASE NOTE

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and the installation, and has received safety training to recognize and avoid the hazards involved.

About the Book

At a Glance

Document Scope

This document describes one of the SoMachine examples.

Since the example described in this document is intended for learning purposes only, it must not be run, nor tested, on products that are part of a machine or process.

Validity Note

This document has been updated with the release of SoMachine V3.0.

The technical characteristics of the device(s) described in this manual also appear online. To access this information online:

Step	Action
1	Go to *www.schneider-electric.com*
2	In the **Search** box on the home page, type a model number. Do not type any blank spaces in the model number. To get information on a grouping of similar modules, you can use the characters **; do not use dots or xx's.
3	Under **All**, click **Products** → **Product Datasheets** and select the model number that interests you.
4	To save or print a data sheet as a .pdf file, click **Export to PDF**.

The characteristics presented in this manual should be the same as those that appear online. In line with our policy of constant improvement we may revise content over time to improve clarity and accuracy. In the event that you see a difference between the manual and online information, use the online information as your reference.

Related Documents

Title of Documentation	Reference Number
Modicon M238 Logic Controller Hardware Guide	EIO0000000016 (ENG); EIO0000000017 (FRE); EIO0000000018 (GER); EIO0000000019 (SPA); EIO0000000020 (ITA); EIO0000000021 (CHS)

Title of Documentation	Reference Number
M238 ExecLoader User Guide	EIO0000000374 (ENG); EIO0000000737 (FRE); EIO0000000738 (GER); EIO0000000739 (SPA); EIO0000000740 (ITA); EIO0000000741 (CHS)
Modicon M238 Logic Controller Programming Guide	EIO0000000384 (ENG); EIO0000000385 (FRE); EIO0000000386 (GER); EIO0000000387 (ITA); EIO0000000388 (SPA); EIO0000000389 (CHS)
Modicon M238 Logic Controller System Functions and Variables M238 PLCSystem Library Guide	EIO0000000364 (ENG); EIO0000000757 (FRE); EIO0000000758 (GER); EIO0000000759 (SPA); EIO0000000760 (ITA); EIO0000000761 (CHS)
SoMachine V3.0 PLC Time Example Guide	EIO0000000902 (ENG)

Product Related Information

This document and its related SoMachine project file focus on specific Functions and Function Blocks of the Schneider libraries provided with SoMachine, and on specific features available in SoMachine if these features are related to these libraries. They are intended to help you developing, testing, commissioning, and integrating applicative software of your own design on control systems.

It is intended for new SoMachine users who already have some degree of expertise in the design and programming of control systems.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Only use software approved by Schneider Electric for use with this equipment.
- Update your application program every time you change the physical hardware configuration.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

⚠ WARNING

LOSS OF CONTROL

- The designer of any control scheme must consider the potential failure modes of control paths and, for certain critical control functions, provide a means to achieve a safe state during and after a path failure. Examples of critical control functions are emergency stop and overtravel stop, power outage and restart.
- Separate or redundant control paths must be provided for critical control functions.
- System control paths may include communication links. Consideration must be given to the implications of unanticipated transmission delays or failures of the link.
- Observe all accident prevention regulations and local safety guidelines.[1]
- Each implementation of this equipment must be individually and thoroughly tested for proper operation before being placed into service.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

[1] For additional information, refer to NEMA ICS 1.1 (latest edition), "Safety Guidelines for the Application, Installation, and Maintenance of Solid State Control" and to NEMA ICS 7.1 (latest edition), "Safety Standards for Construction and Guide for Selection, Installation and Operation of Adjustable-Speed Drive Systems" or their equivalent governing your particular location.

Before You Begin

The products specified in this document have been tested under actual service conditions. Of course, your specific application requirements may be different from those assumed for this and any related examples described herein. In that case, you will have to adapt the information provided in this and other related documents to your particular needs. To do so, you will need to consult the specific product documentation of the hardware and/or software components that you may add or substitute for any examples specified in this documentation. Pay particular attention and conform to any safety information, different electrical requirements and normative standards that would apply to your adaptation.

⚠ WARNING

REGULATORY INCOMPATIBILITY

Be sure that all equipment applied and systems designed comply with all applicable local, regional and national regulations and standards.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

The use and application of the information contained herein require expertise in the design and programming of automated control systems. Only the user or integrator can be aware of all the conditions and factors present during installation and setup, operation, and maintenance of the machine or process, and can therefore determine the automation and associated equipment and the related safeties and interlocks which can be effectively and properly used. When selecting automation

and control equipment, and any other related equipment or software, for a particular application, the user or integrator must also consider any applicable local, regional or national standards and/or regulations.

Some of the major software functions and/or hardware components used in the proposed architectures and examples described in this document cannot be substituted without significantly compromising the performance of your application. Further, any such substitutions or alterations may completely invalidate any proposed architectures, descriptions, examples, instructions, wiring diagrams and/or compatibilities between the various hardware components and software functions specified herein and in related documentation. You must be aware of the consequences of any modifications, additions or substitutions. A residual risk, as defined by EN/ISO 12100-1, Article 5, will remain if:

- it is necessary to modify the recommended logic and if the added or modified components are not properly integrated in the control circuit.
- you do not follow the required standards applicable to the operation of the machine, or if the adjustments to and the maintenance of the machine are not properly made (it is essential to strictly follow the prescribed machine maintenance schedule).
- the devices connected to any safety outputs do not have mechanically-linked contacts.

⚠ CAUTION

EQUIPMENT INCOMPATIBILITY

Read and thoroughly understand all device and software documentation before attempting any component substitutions or other changes related to the application examples provided in this document.

Failure to follow these instructions can result in injury or equipment damage.

Start-up and Test

Before using electrical control and automation equipment after design and installation, the application and associated functional safety system must be subjected to a start-up test by qualified personnel to verify correct operation of the equipment. It is important that arrangements for such testing be made and that enough time is allowed to perform complete and satisfactory testing.

⚠ CAUTION

EQUIPMENT OPERATION HAZARD

- Verify that all installation and set up procedures have been completed.
- Before operational tests are performed, remove all blocks or other temporary holding means used for shipment from all component devices.
- Remove tools, meters and debris from equipment.

Failure to follow these instructions can result in injury or equipment damage.

Verify that the completed system, including the functional safety system, is free from all short circuits and grounds, except those grounds installed according to local regulations. If high-potential voltage testing is necessary, follow the recommendations in equipment documentation to help prevent injury or equipment damage.

Operation and Adjustments

Regardless of the care exercised in the design and manufacture of equipment or in the selection and ratings of components, there are hazards that can be encountered if such equipment is improperly installed and operated.

In some applications, such as packaging machinery, additional operator protection such as point-of-operation guarding must be provided. This is necessary if the hands and other parts of the body are free to enter the pinch points or other hazardous areas where serious injury can occur. Software products alone cannot protect an operator from injury. For this reason, the software cannot be substituted for or take the place of point-of-operation protection.

⚠ WARNING

UNGUARDED MACHINERY CAN CAUSE SERIOUS INJURY

- Do not use this software and related automation equipment on equipment which does not have point-of-operation protection.
- Do not reach into machinery during operation.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Ensure that appropriate safeties and mechanical/electrical interlocks related to point-of-operation protection have been installed and are operational before placing the equipment into service. All interlocks and safeties related to point-of-operation protection must be coordinated with the related automation equipment and software programming.

NOTE: Coordination of safeties and mechanical/electrical interlocks for point-of-operation protection is outside the scope of the examples and implementations suggested herein.

It is sometimes possible to adjust the equipment incorrectly and this produce unsatisfactory or unsafe operation. Always use the manufacturer instructions as a guide to functional adjustments. Personnel who have access to these adjustments must be familiar with the equipment manufacturer instructions and the machinery used with the electrical equipment.

Only those operational adjustments actually required by the machine operator should be accessible to the operator. Access to other controls should be restricted to help prevent unauthorized changes in operating characteristics.

User Comments

We welcome your comments about this document. You can reach us by e-mail at techcomm@schneider-electric.com.

1. Description

1.1. Presentation

This example presents the system functions of the M238 logic controller. These functions are located in the **M238 PLCSystem** library.

The controller's program is created using SoMachine software.

In this example, these system functions are used to diagnose the status a M238 controller in normal operation mode.

Related SoMachine project: **PLC_Diagnoctic.project**

Supported SoMachine Languages:

☑ CFC	☐ IL
☑ ST	☐ FBD
☑ LD	☐ SFC

Key features: System functions of the M238 logic controller

Requirements: To use this example, the user must have:

- installed SoMachine V3.0 on a PC.

PC with SoMachine

M238 logic controller

Note: Because system functions of the controller are used in this example's program, **do not run this example in SIMULATION mode**.

1.2. Functions Used in this Example

The Functions used in this example are listed below, grouped by library:

- **M238 PLCSystem** library (Schneider Electric)

  These system Functions are specific to each controller. In the case of some of these functions, each controller's **PLCSystem** library features its own versions of these functions; e.g. the **M238 PLCSystem** library has a **IsFirstMastColdCycle** Function Block that has the same purpose than the **HMI_IsFirstMastColdCycle** Function of the **XBT PLCSystem** library.

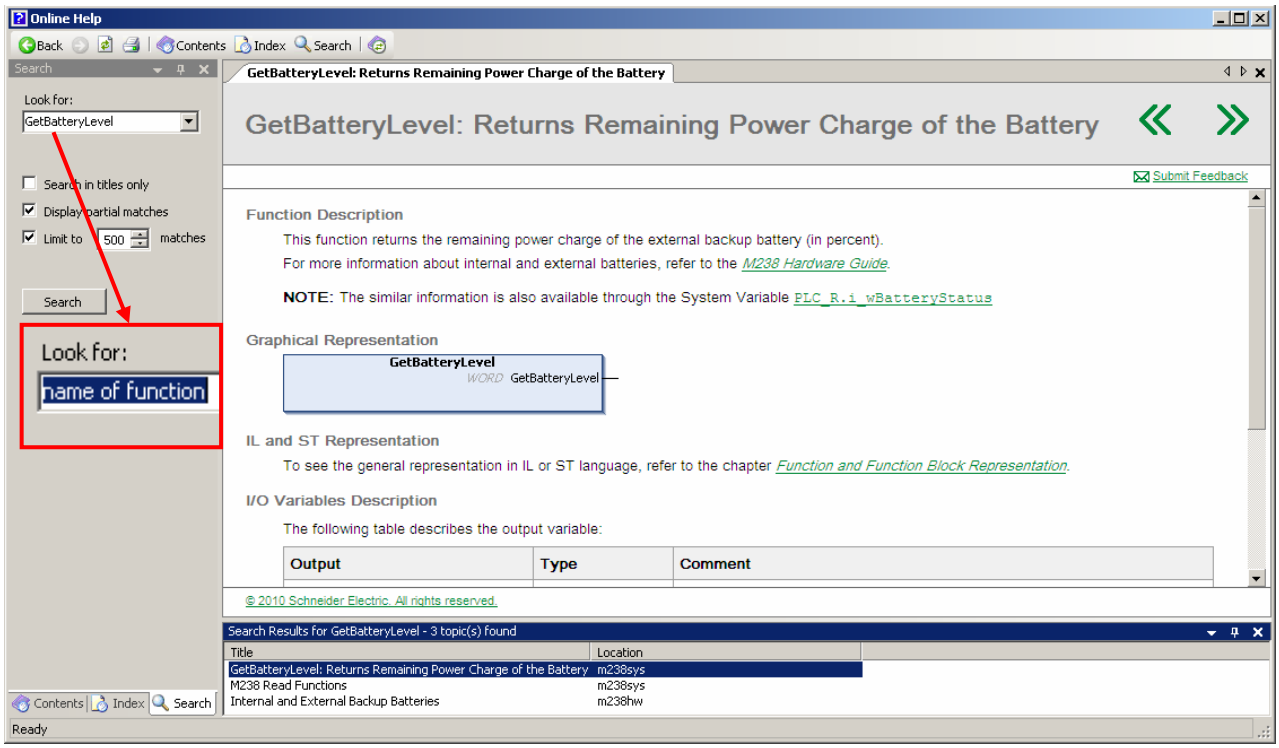
Function	Description	Location in the Input Assistant
GetBatteryLevel	Returns remaining power charge of the battery	Module Calls ➔ { } SEC ➔ Read
GetBootProjectStatus	Returns the boot project status	
GetEventsNumber	Returns the number of external events detected	
GetFirmwareVersion	Returns information about the firmware, the boot and the coprocessor versions	
GetHardwareVersion	Returns the hardware version	
GetLastStopCause	Returns the cause last stop	
GetLocalIOStatus	Returns the embedded I/O status	
GetPlcFault	Returns detected errors on the controller I/O	
GetRightBusStatus	Returns the status of the expansion bus	
GetSerialNumber	Returns the serial number of the controller	
GetShortCutStatus	Returns the short-circuit status on the embedded outputs	
IsFirstMastColdCycle	Indicates if cycle is the first MAST cold start cycle	
IsFirstMastCycle	Indicates if cycle is the first MAST start cycle	
IsFirstMastWarmCycle	Indicates if cycle is the first MAST warm start cycle	
InhibitBatLowLed	Disables or re-enables the low battery LED	Module Calls ➔ { } SEC ➔ Write
ResetEventsNumber	Resets events number	

Note: The other functions of the **M238 PLCSystem** library are used in the example described in the *SoMachine V3.0 PLC Time Example Guide*.

Please refer to the SoMachine online help.

Note: In the rest of this document, the former sentence instructs you to refer to the online help of SoMachine, accessible through its upper-right help button.

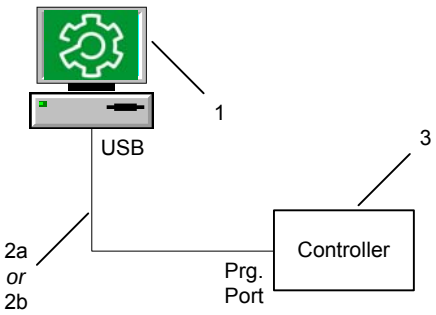
Please refer to the SoMachine online help for detailed information on these Functions: Function description, Graphical representation, I/O Variables description, and more.



To install these libraries in your own project, please refer to *Library Manager (see page 25)*.

1.3. Hardware Installation

Required Devices



N°	Designation	Reference	Use or Description
1	**SoMachine Software**	MSD CHNLMUA	SoMachine Software, 1-station license, installed on a PC
2a	**Terminal port/USB port cordset**	TCS XCN AM UM3P	From the mini B USB port on the Modicon M238 base to the type A USB port on the PC terminal for programming and updating firmware; length: 3 m (10 ft)
2b	**Programming cable**	BMX XCA USB H018	Same as TCS XCN AM UM3P, but with two ground connections along the cable; length: 1.8 m (6 ft)
3	**M238 controller**	TM238 ••••••••	Compact base logic controller with 24 I/O (removable battery to be ordered separately: TSX PLP 01)

Please refer to the *Modicon M238 Logic Controller Hardware Guide* for the hardware setup of this device.

2. Description of the Example's Content

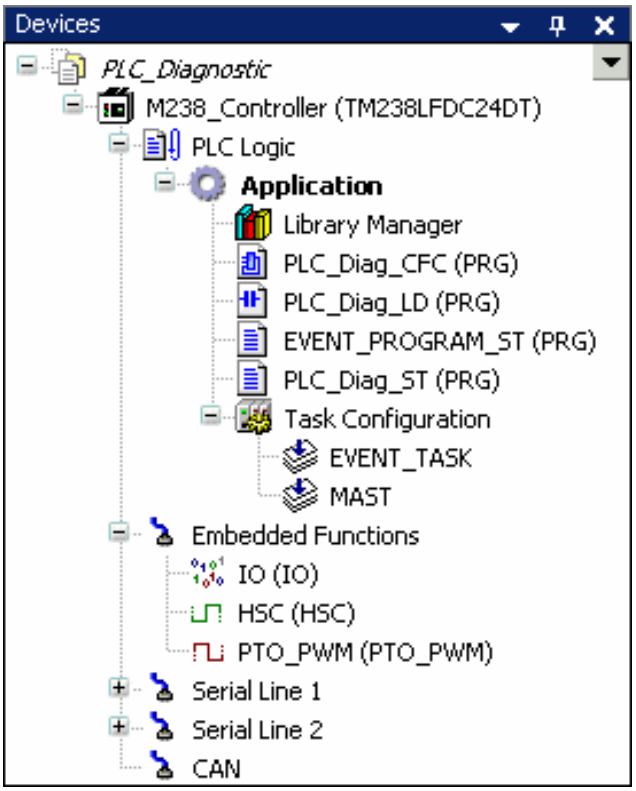
In SoMachine, the configuration of the example is made with the following device:

- **1 Logic Controller:** TM238LFDC24DT

The program of the M238 controller is made of the following specific items:

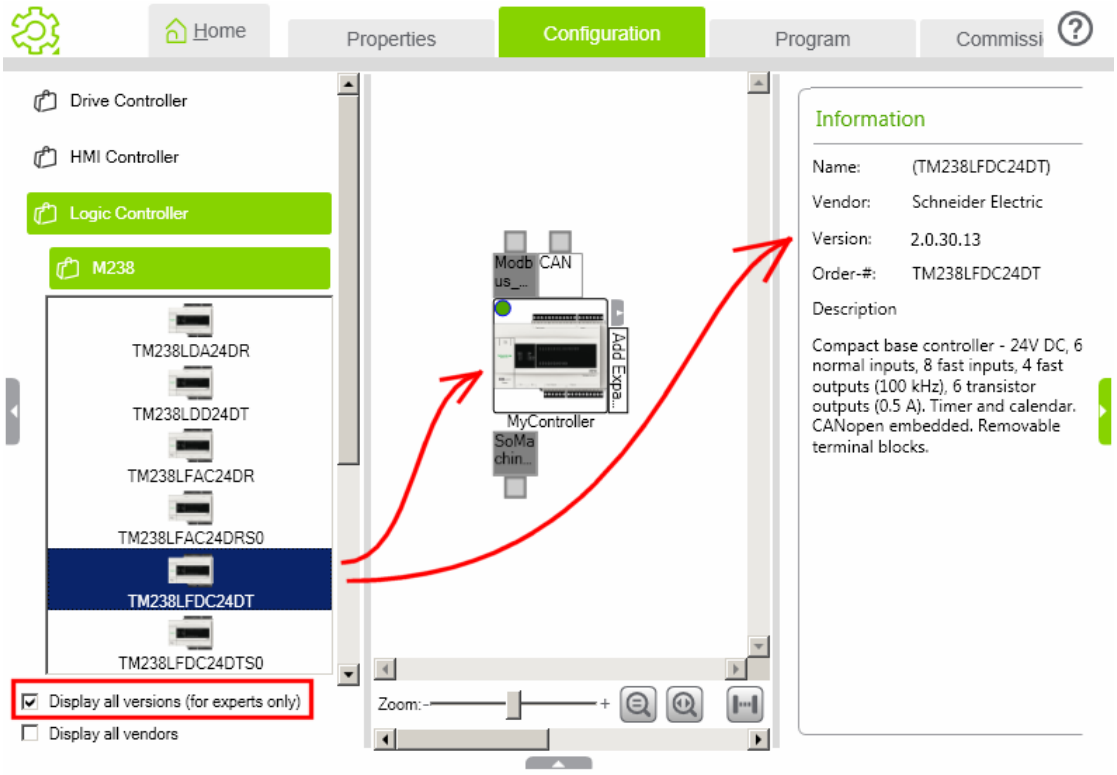
- **Library Manager:** List of the libraries linked to the programs of this example.
- **CFC program:** Contains the source code that uses the system functions for diagnosing the Logic Controller. This is the default program since it is called by the MAST task of the controller.
- **LD program:** Translation of the CFC program into LD language. To run this program on the controller, change the POU called by the MAST task of the controller from **PLC_Diag_CFC** to **PLC_Diag_LD**.
- **EVENT_PROGRAM_ST program:** This empty program, in ST language, is only intended to be run by the EVENT_TASK task.
- **ST program:** Translation of the CFC program into ST language. To run this program on the controller, change the POU called by the MAST task of the controller from **PLC_Diag_CFC** to **PLC_Diag_ST**.
- **Task Configuration**: The standard MAST task, cyclically called every 20ms, plus an EVENT_TASK task, triggered by the external event configured for the **I0** fast input.
- **Embedded Functions:** Configuration of an external event on the rising and falling edges of the **I0** fast input.

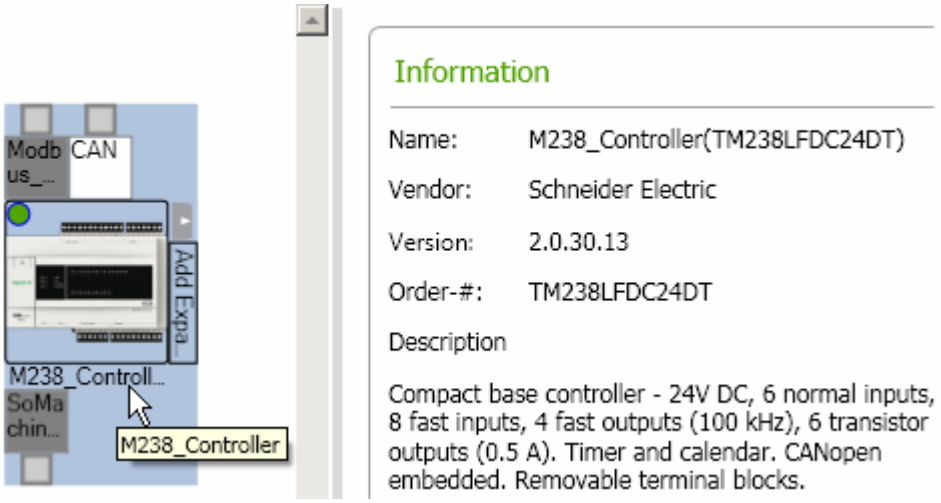
Example's content, visible in the **Devices** panel of the **Program** tab:



3. Creation of the Project

The steps listed in the following table describe how to create the SoMachine project, and how to set up the device(s) used in this example. No details are given here since it is assumed that you already know the Basic commands of SoMachine.

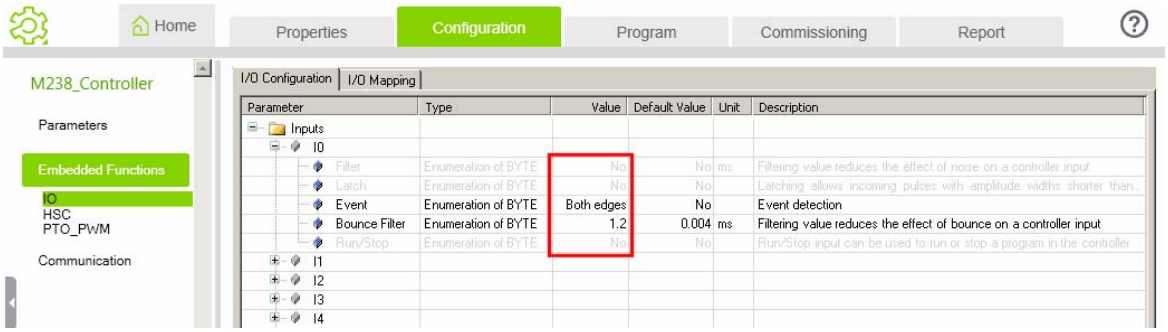
Step	Action
1	In the **Create new machine** part of the **Home** tab, select **Start with empty project** to create a new SoMachine project. Give this new project the following name: **PLC_Diagnostic**.
2	In the **Configuration** tab, add a TM238LFDC24DT Logic Controller.  **Note:** Details on the selected controller are displayed in the **Information** section of SoMachine. SoMachine controller version: Defines the version of the selected controller; it is displayed in the **Information** section of SoMachine. Target controller firmware version: Defines the firmware version of your controller. This version is shown when you select your controller's node, as shown in *Downloading the Example to the Controller (see page 51)*. For compatibility purposes between a SoMachine controller version and a target controller firmware version, only the first three numbers of a version must be identical. In the preceding picture, the **2.0.30.13** SoMachine controller version is compatible with any **2.0.30.••** target controller firmware version. For each controller model, SoMachine only presents the latest available version. If you check the ☑ **Display all versions (for expert only)** option, SoMachine will list all supported controller firmware versions. However, a good practice consists in using the latest available version and updating the firmware of your controller, if required. Please refer to the *M238 ExecLoader User Guide*.

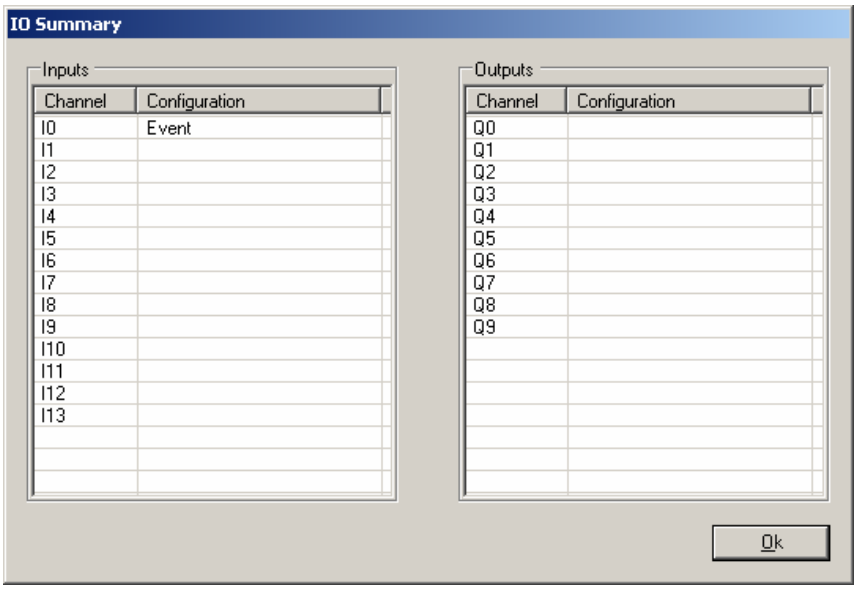
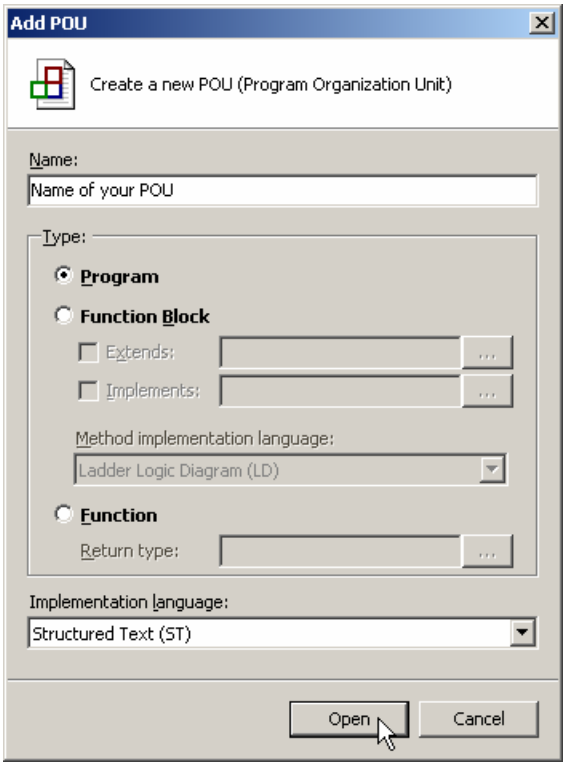
Step	Action
3	Rename this controller to **M238_Controller**. 
4	Save your new project.

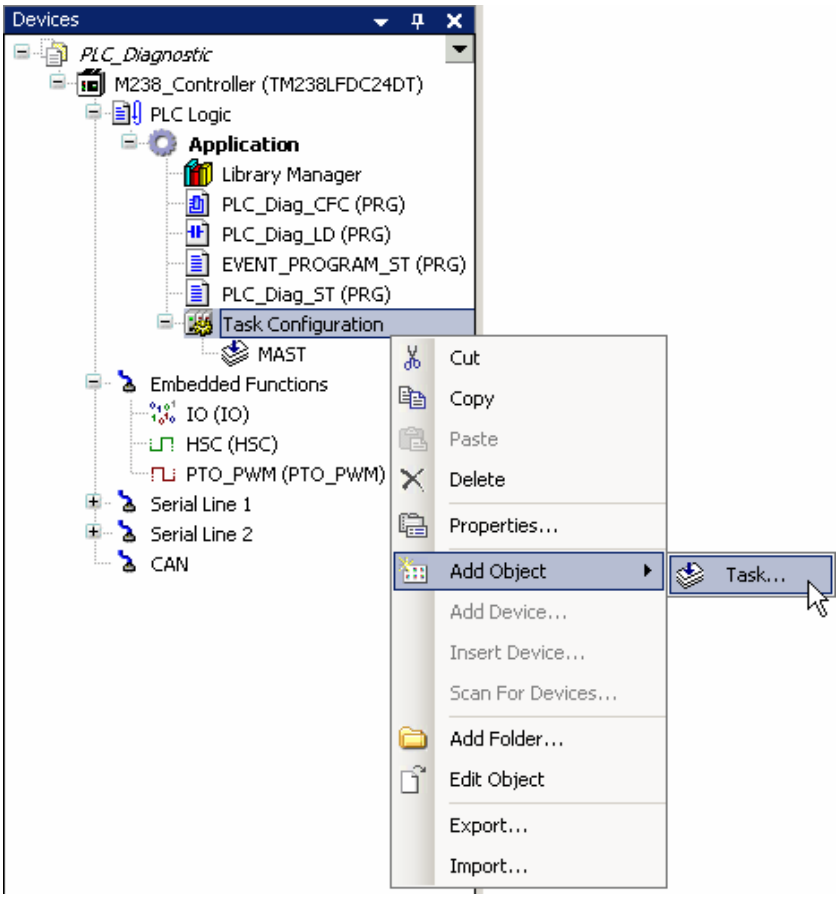
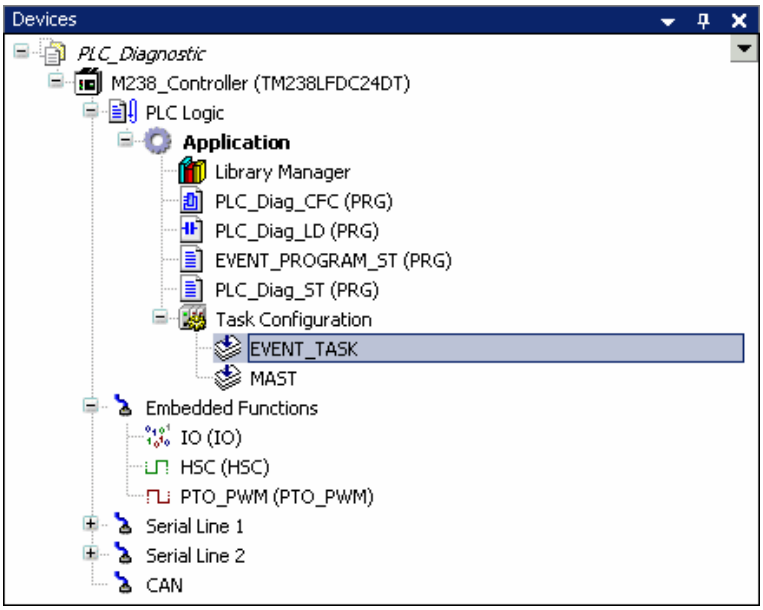
4. Event Task Creation and Configuration

The steps listed in the following table describe how to create and configure an **Event Task**. This task is required for testing the **GetEventsNumber** and **ResetEventsNumber** functions.

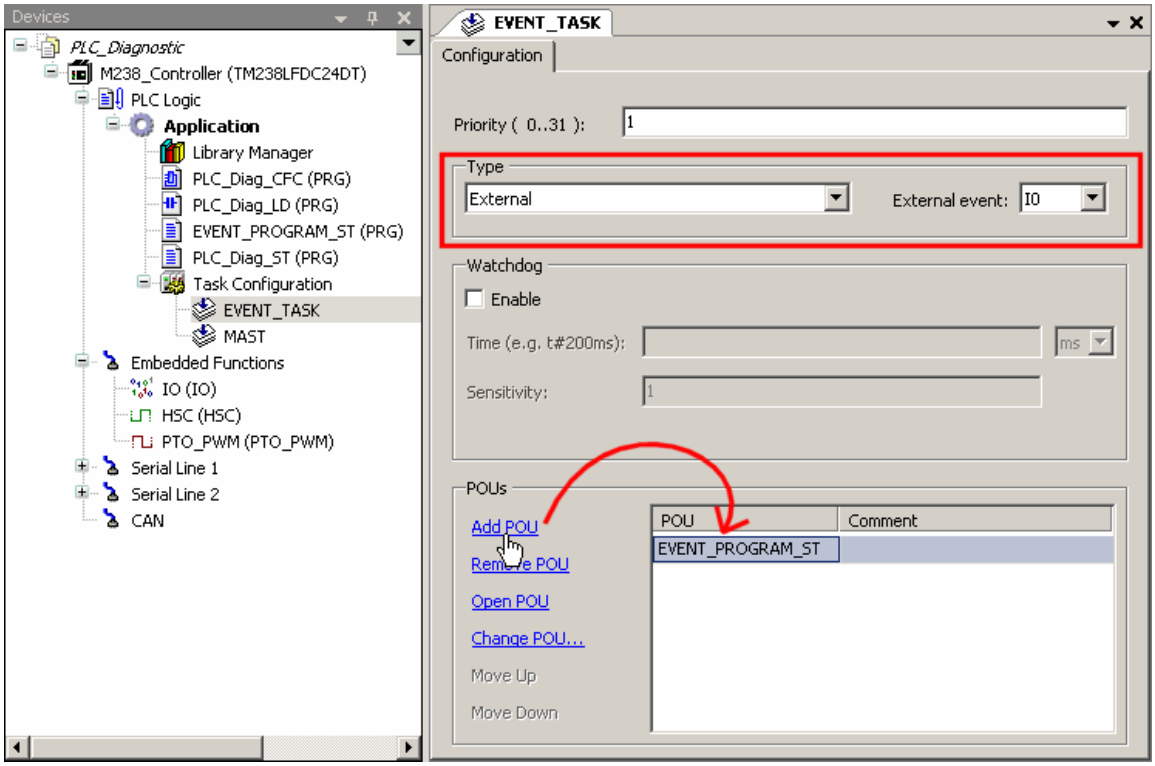
To trigger this **Event Task**, an external event is configured on the rising and falling edges of the **I0** fast input of the M238 controller.

Step	Action
1	In the **Configuration** tab, double-click on the **M238_Controller**. This opens the **Parameters** configuration panel of this controller.
2	In the left-hand panel: ➢ Select the **Embedded Functions** group. ➢ Select the **IO** item.
3	In the **I/O configuration** tab, configure the **I0** input as follows:  To achieve this configuration, please follow these steps: ➢ Double-click on the **No** value of the **Inputs ► I0 ► Event** parameter. ➢ In the drop-down list that appears, select **Both edges**. ➢ Click outside this new value, or press the **ENTER** key, to validate it. **Note:** Because this input is configured in **Both edges**, any rising edge or falling edge on the **I0** fast input will trigger an external event on the controller. ➢ Double-click on the **0.004** value of the **Inputs ► I0 ► Bounce Filter** parameter. ➢ In the drop-down list that appears, select **1.2**. ➢ Click outside this new value, or press the **ENTER** key, to validate it. **Note:** This **Bounce Filter** parameter, whose value is expressed in milliseconds, represents the duration of the anti-bounce filter applied to this fast input.

Step	Action
4	Click on the IO Summarize… button located in the lower right corner of the central panel. The **IO Summary** window that appears shows that the **I0** input of the controller has been affected to an **Event**. 
5	Click on **OK** to close the **IO Summary** window.
6	Go to the **Devices** panel of the **Program** tab.
7	Create a new POU, in ST language, called **EVENT_PROGRAM_ST**.  Upon creation of this POU, it is automatically opened by SoMachine. **Note:** At first, this program is empty. In this example, it will remain empty because its only purpose consists in providing an **Event Task** with a program to run.

Step	Action
8	Create a new task: ➢ Expand the contents of the **M238_Controller (TM238LFDC24DT)** item. ➢ Right-click on the **Task Configuration** item. ➢ In the **Add Object** part of the contextual menu that appears, select the **Task…** command.  ➢ In the **Add Task** window that appears, write the **Name** of this new task: **EVENT_TASK**. ➢ Click on **OK** to validate the creation of this new task. 

Step	Action
9	Double-click on the **EVENT_TASK** task and configure it as follows:



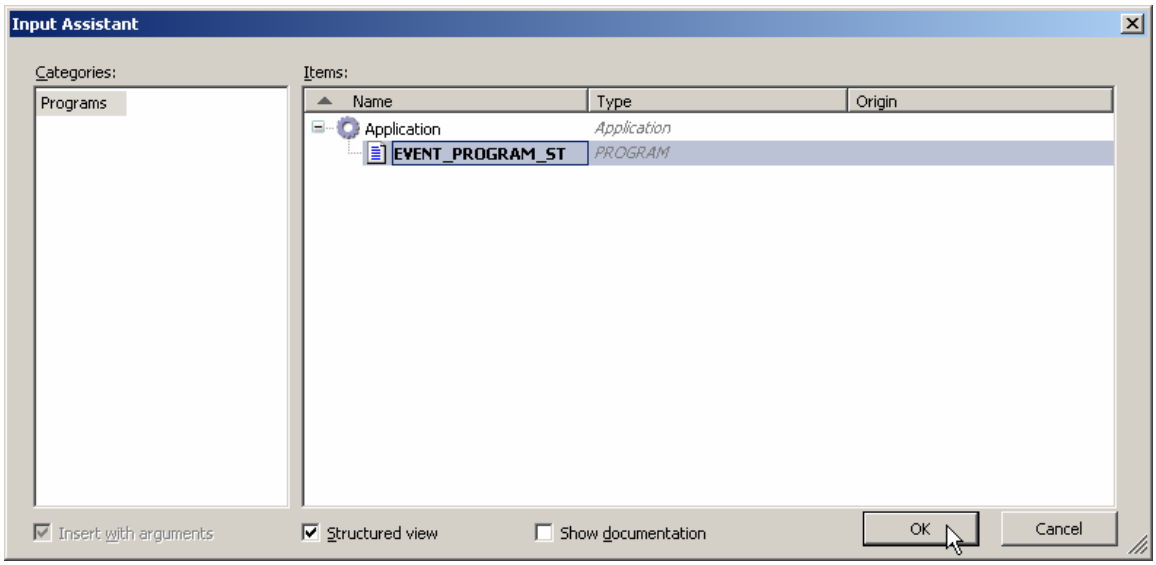
To achieve this configuration, please follow these steps:

- Change the **Type** of this task from **Cyclic** to **External**.
- Check that the **External event** is equal to **I0**; if this not the case, change it to **I0**.

  Note: With this configuration, the **EVENT_TASK** task will be run each time the event configured for the **I0** fast input of the controller *(see page 20)* is triggered.
- Click on the **Add POU** command.

In the **Input Assistant** window that appears:

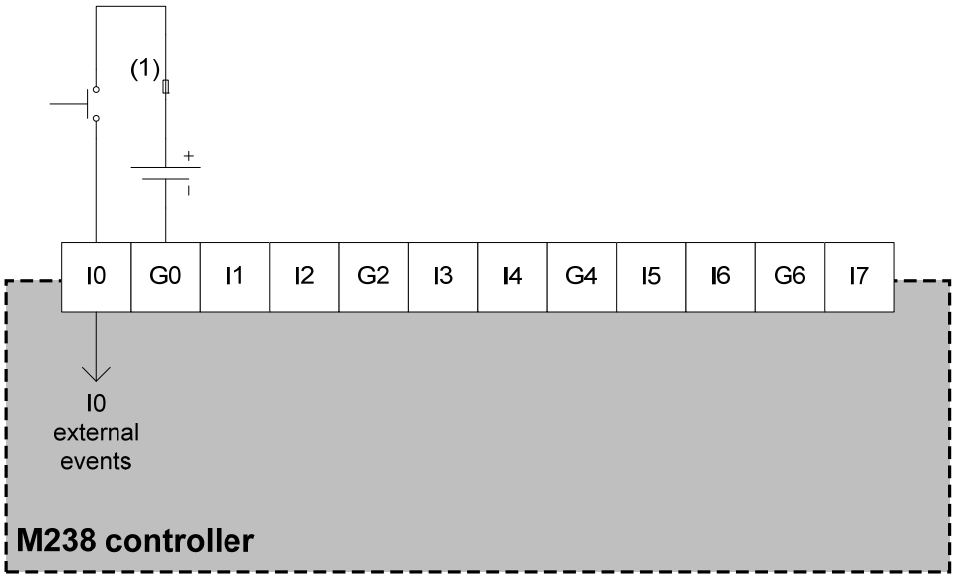
- Select the **EVENT_PROGRAM_ST** program.
- Click on **OK**.



5. Wiring the Controller’s I0 Fast Input

Before wiring the controller’s **I0** fast input, you must first refer to the *Modicon M238 Logic Controller Hardware Guide*.

The following diagram is an adaptation of the **Fast Input Wiring Diagram**, given in the *Modicon M238 Logic Controller Hardware Guide*, to the usage made by this example of the controller’s fast inputs.

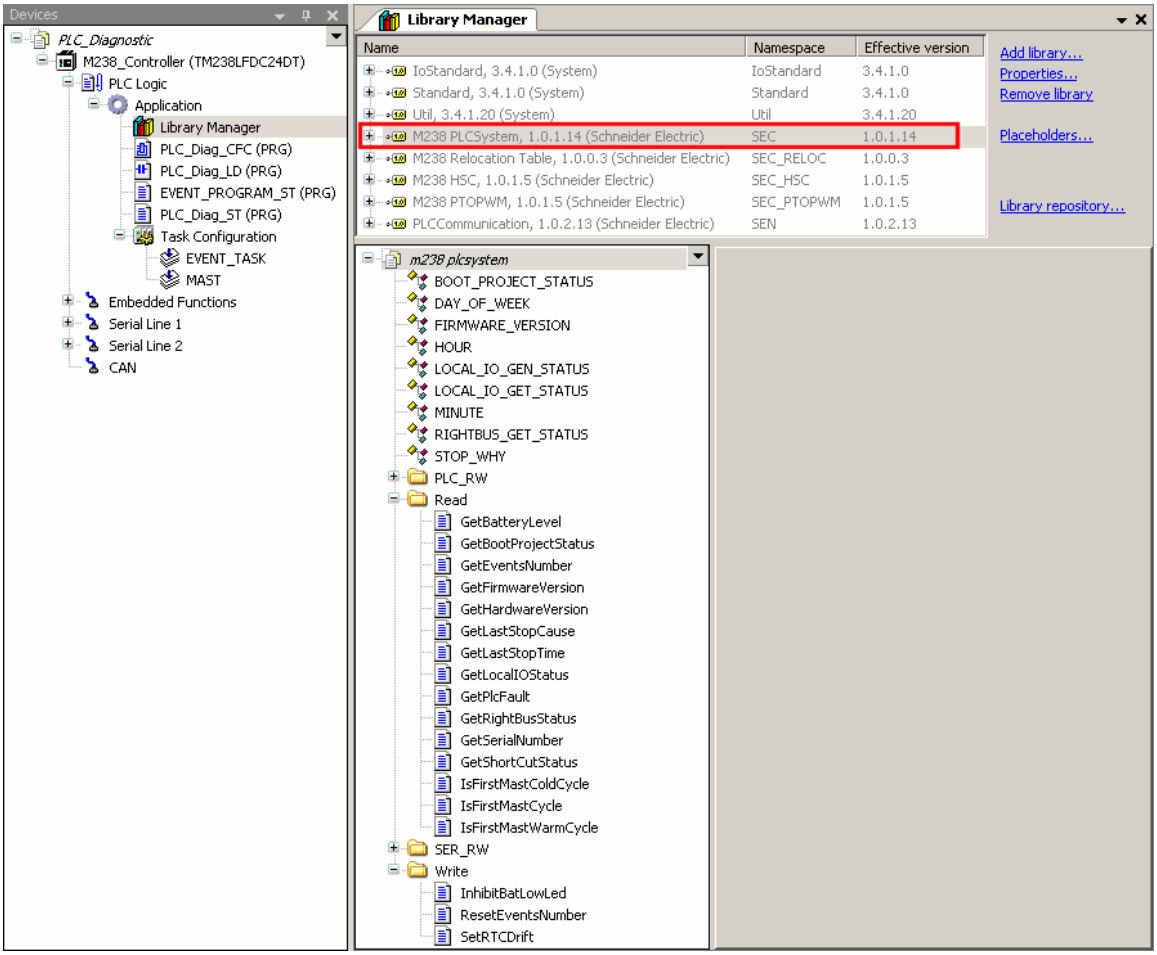


(1) Fast-blow fuse 0.5 A

In this diagram, the push-button wired to the **I0** fast input is used to generate pulses. Each rising or falling edge of these pulses will be transformed into an **I0** external event.

6. Library Manager

The steps listed in the following table describe how to add and/or check the list of the libraries linked to this example.

Step	Action
1	Select the **Program** tab. In the **Devices** tree view, double-click on the **Library Manager** to open the list of the libraries linked to the **Application** software of this example.
2	Check that the **M238 PLCSystem** library is already linked, as shown below:  **Note:** This library is **grayed** to inform that it has been automatically linked to the program upon addition of the M238 Controller to the project and that it cannot be removed.

7. CFC, LD, or ST Program

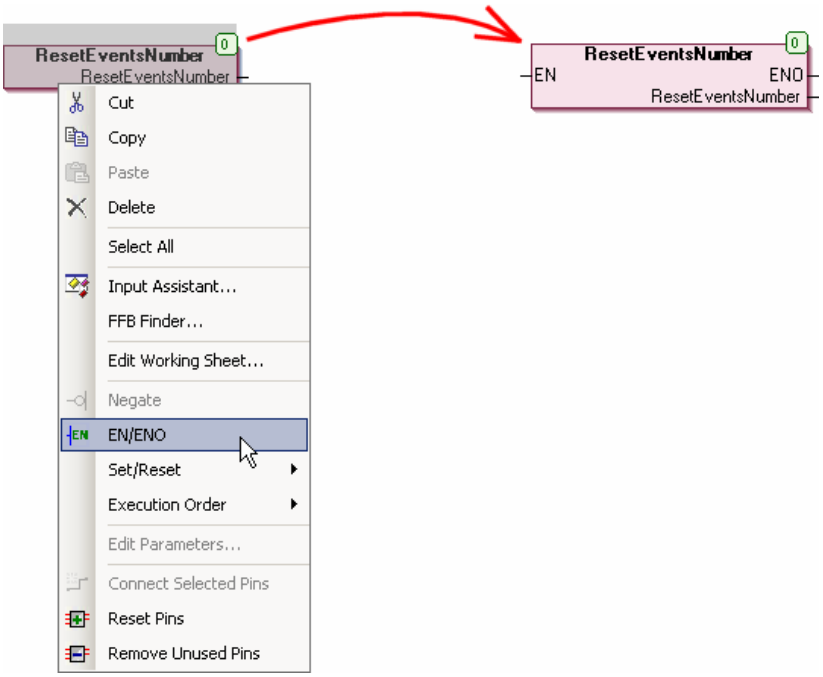
Each of the following three chapters describes how to create the program used in the example. Choose the language of your program (CFC, LD, or ST) and go to the corresponding chapter:

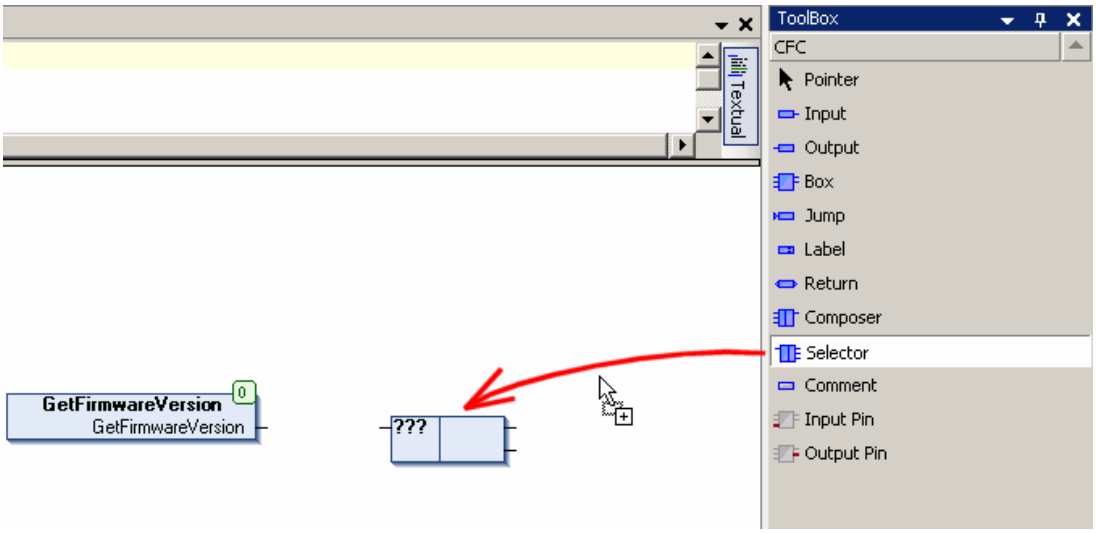
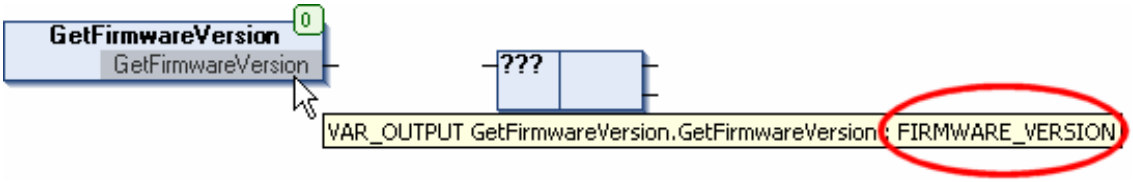
You only need to write your SoMachine program in one of these three languages.

In addition, each of these three chapters begins with explanations on the difficulties you may face, if any, in the form of **optional steps**.

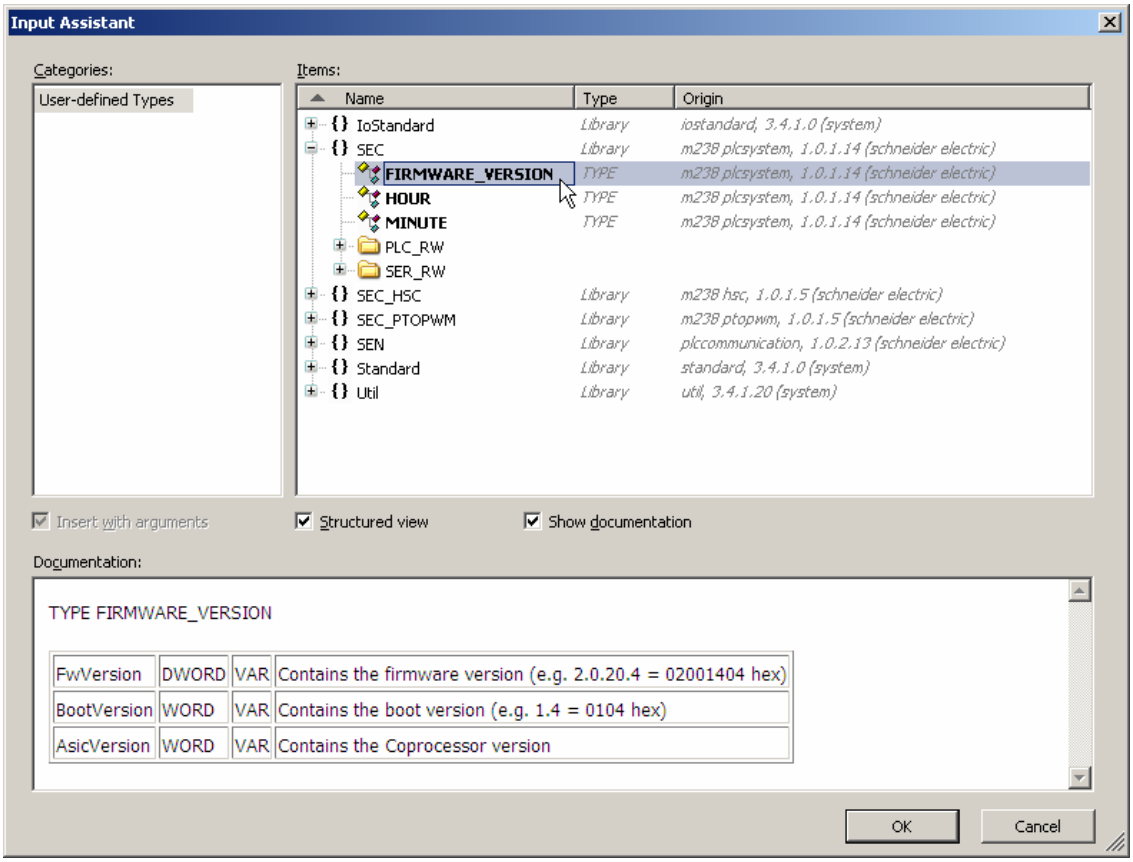
7.1. CFC Program

Before instructing you how to write down the CFC program used in this example, the following table presents optional steps that give you information on how to program in CFC language.

Step	Action
1	**Optional step** **How to add the EN/ENO pins on a Function Block** In this example, these pins are used on three functions: **IsFirstMastWarmCycle**, **IsFirstMastColdCycle**, and **ResetEventsNumber**. This optional step indicates how to append the additional Boolean enable input EN and Boolean output ENO (Enable Out) to any Function or Function Block. These two pins are used to enable (TRUE) or disable (FALSE) the execution of the Functions or Function Blocks on which they are used. Please refer to the SoMachine online help: search for **additional Boolean ENO**. ➢ Right-click, in your CFC program, on a Function (or a Function Block). ➢ In the contextual menu that appears, select the **EN/ENO** command.  **Note:** If you want to remove these EN/ENO pins, select the **EN/ENO** command again.

Step	Action
2	**Optional step** **How to use a Selector element to display structure components** In this example, the **Selector** element is used to access the three components of a variable whose type is the **FIRMWARE_VERSION** structure: **FwVersion**, **BootVersion**, and **AsicVersion**. This optional step indicates how to use a **Selector** element. Its input is a variable whose type must be a structure. The number of its outputs depends on the number of components of this structure: the **Selector** element has one output per component of the input structure. Please refer to the SoMachine online help: search for **Selector**. In the following steps, a **Selector** element is used to access the various components of the structured output of the **GetFirmwareVersion** Function (output type = **FIRMWARE_VERSION** structure): ➢ From the **ToolBox** panel, create a **Selector** element into the central worksheet.  ➢ Determine the name of the structure you want to use for this **Selector** element. In the given example, this element will be used with the **GetFirmwareVersion** output of the **GetFirmwareVersion** Function. The type of this output is a structure whose name is **FIRMWARE_VERSION**.  ➢ Click on the **???** of the **Selector** element. ➢ Click on the white button that appears next to the **???**; this will display the **Input Assistant**. 

Step	Action
	➢ In the **Input Assistant**, select the **FIRMWARE_VERSION** data type.



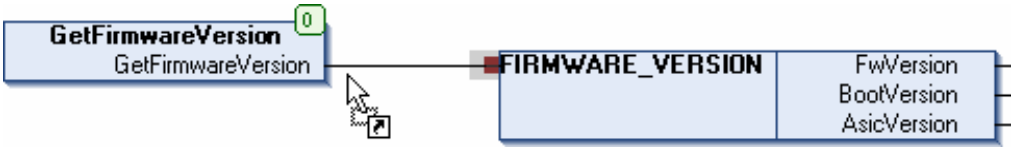
Note: The **Documentation** section displays details on the components of the selected data type.

Note: As a general rule, a data type used by a Function (or a Function Block) is located in the very same library than the Function (or Function Block). In the given example, the **GetFirmwareVersion** Function and the **FIRMWARE_VERSION** data type, used by this Function, are both located in the **M238 PLCSystem** library.

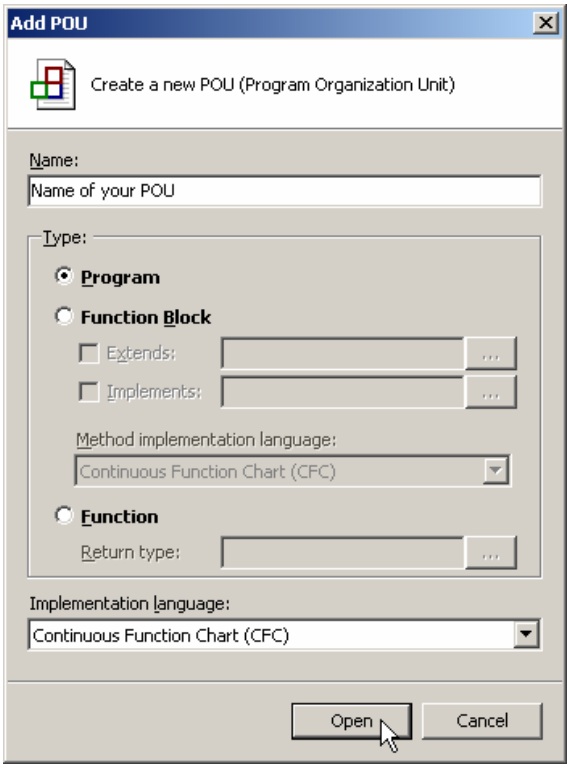
➢ Click on **OK**.

➢ Press the **ENTER** key to validate this choice.

➢ Connect the **Selector** element's input to the **GetFirmwareVersion** output of the **GetFirmwareVersion** Function.



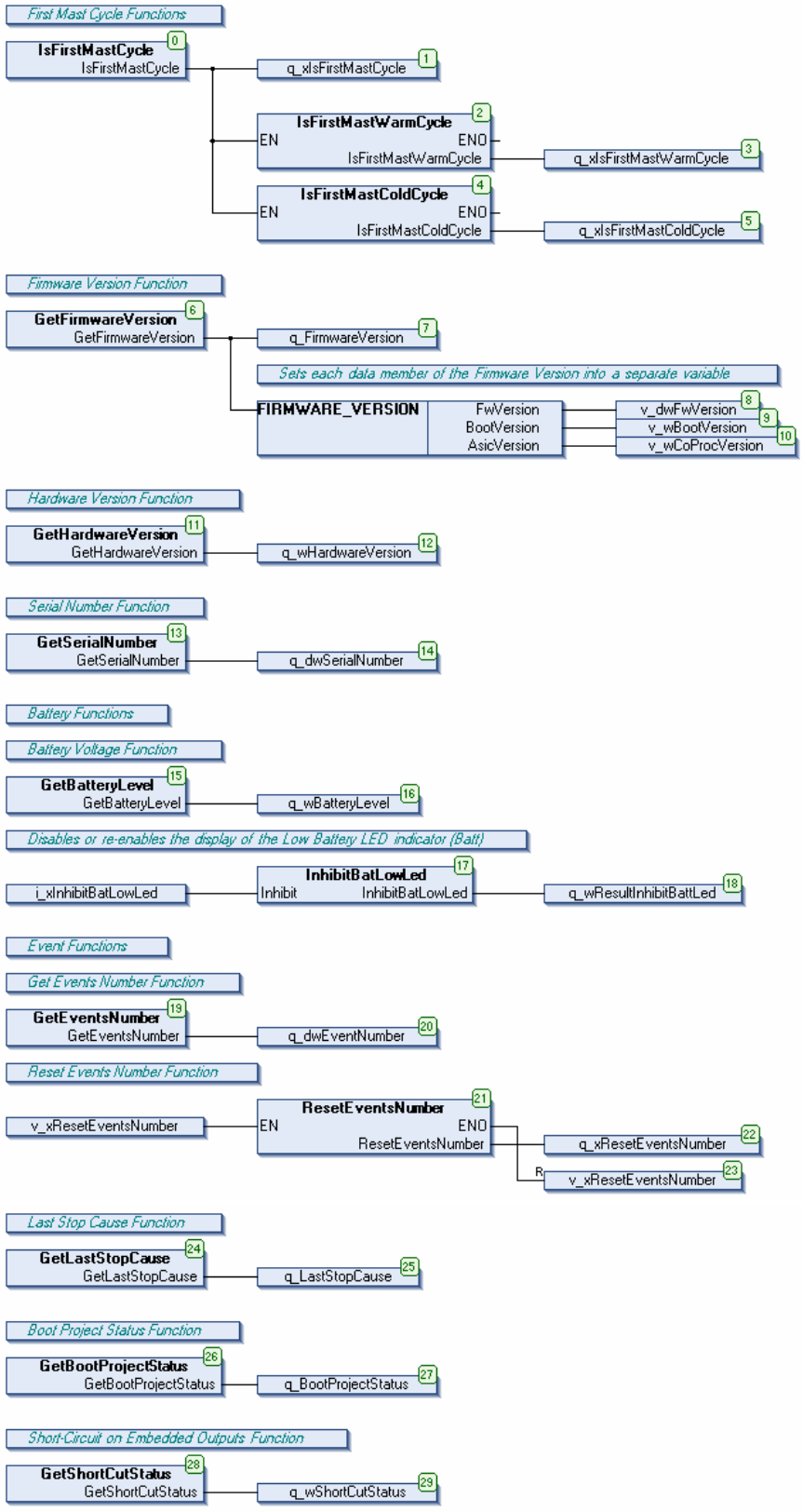
Now, each component of the **GetFirmwareVersion** Function's structured output is available as one of the outputs of the **Selector** element.

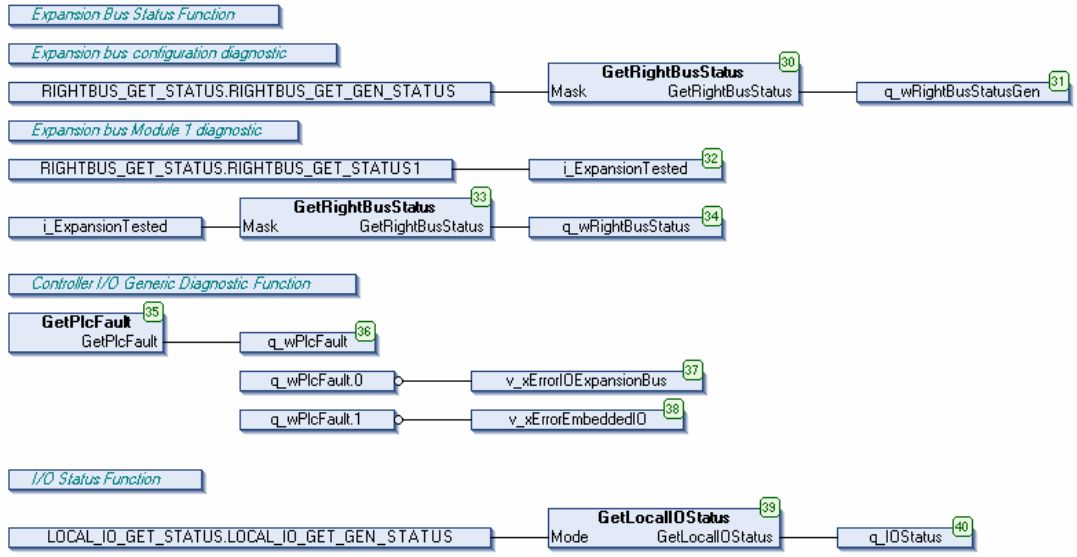
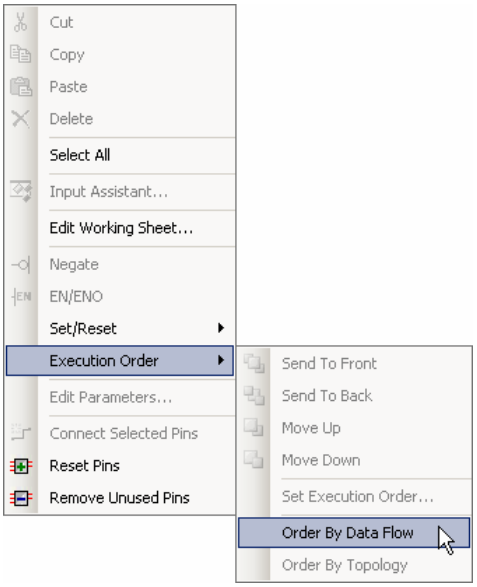
Step	Action
3	**Creation of the POU:** Create a new POU in CFC language, called **PLC_Diag_CFC**.  Upon creation of this POU, it is automatically opened by SoMachine.

Step	Action
4	**CFC variables:** In the upper part of the CFC editor, declare the following variables:

```
PROGRAM PLC_Diag_CFC
VAR
(*First Mast Cycle Functions*)
    q_xIsFirstMastCycle         : BOOL;
    q_xIsFirstMastWarmCycle     : BOOL;
    q_xIsFirstMastColdCycle     : BOOL;

(*Firmware Version Function*)
    q_FirmwareVersion           : FIRMWARE_VERSION;
    v_dwFwVersion               : DWORD;
    v_wBootVersion              : WORD;
    v_wCoProcVersion            : WORD;

(*Hardware Version Function*)
    q_wHardwareVersion          : WORD;

(*Serial Number Function*)
    q_dwSerialNumber            : DWORD;

(*Battery Functions*)
    q_wBatteryLevel             : WORD;
    i_xInhibitBatLowLed         : BOOL  := FALSE;
    q_wResultInhibitBattLed     : WORD;

(*Event Functions*)
    q_dwEventNumber             : DWORD;
    v_xResetEventsNumber        : BOOL  := FALSE;
    q_xResetEventsNumber        : BOOL;

(*Last Stop Cause Function*)
    q_LastStopCause             : STOP_WHY;

(*Boot Project Status Function*)
    q_BootProjectStatus         : BOOT_PROJECT_STATUS;

(*Short-Circuit Status on Embedded Outputs Function*)
    q_wShortCutStatus           : WORD;

(*Expansion Bus Status Function*)
    i_ExpansionTested           : RIGHTBUS_GET_STATUS;
    q_wRightBusStatusGen        : WORD;
    q_wRightBusStatus           : WORD;

(*Controller I/O Generic Diagnostic Function*)
    q_wPlcFault                 : WORD;
    v_xErrorIOExpansionBus      : BOOL;
    v_xErrorEmbeddedIO          : BOOL;

(*I/O Status Function*)
    q_IOStatus                  : LOCAL_IO_GEN_STATUS;
END_VAR
```

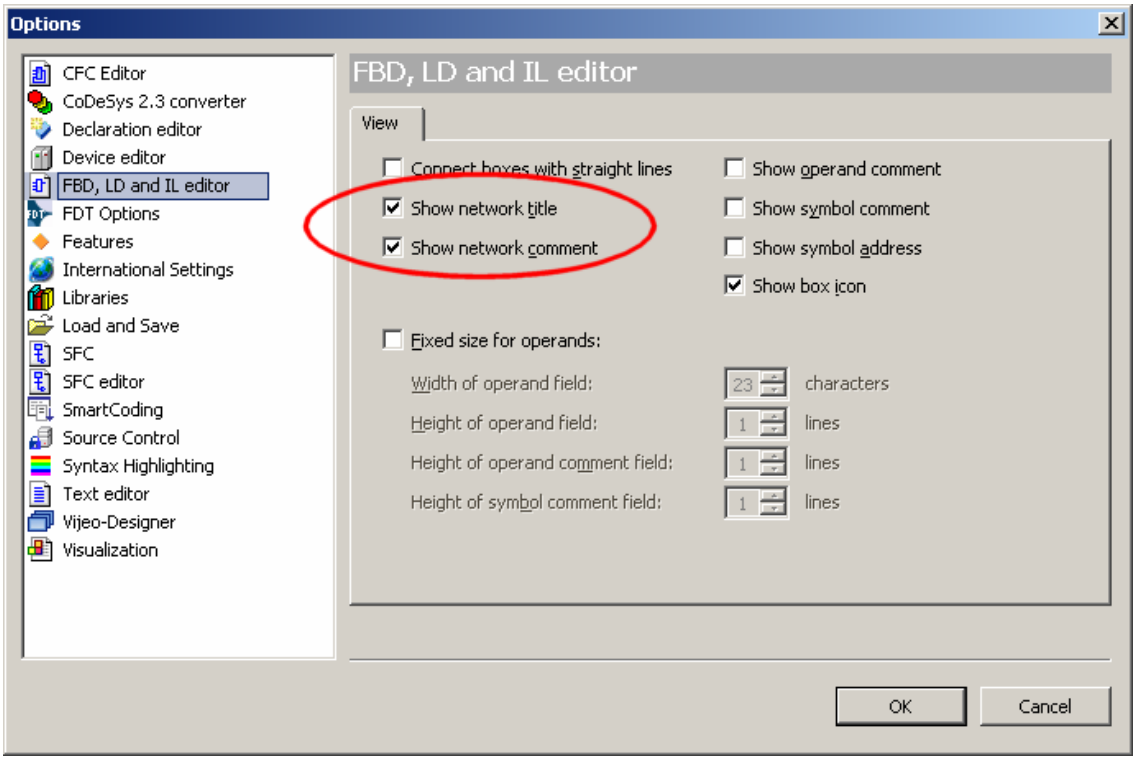
Step	Action
5	**CFC program:** In the lower part of the CFC editor, implement the following program:

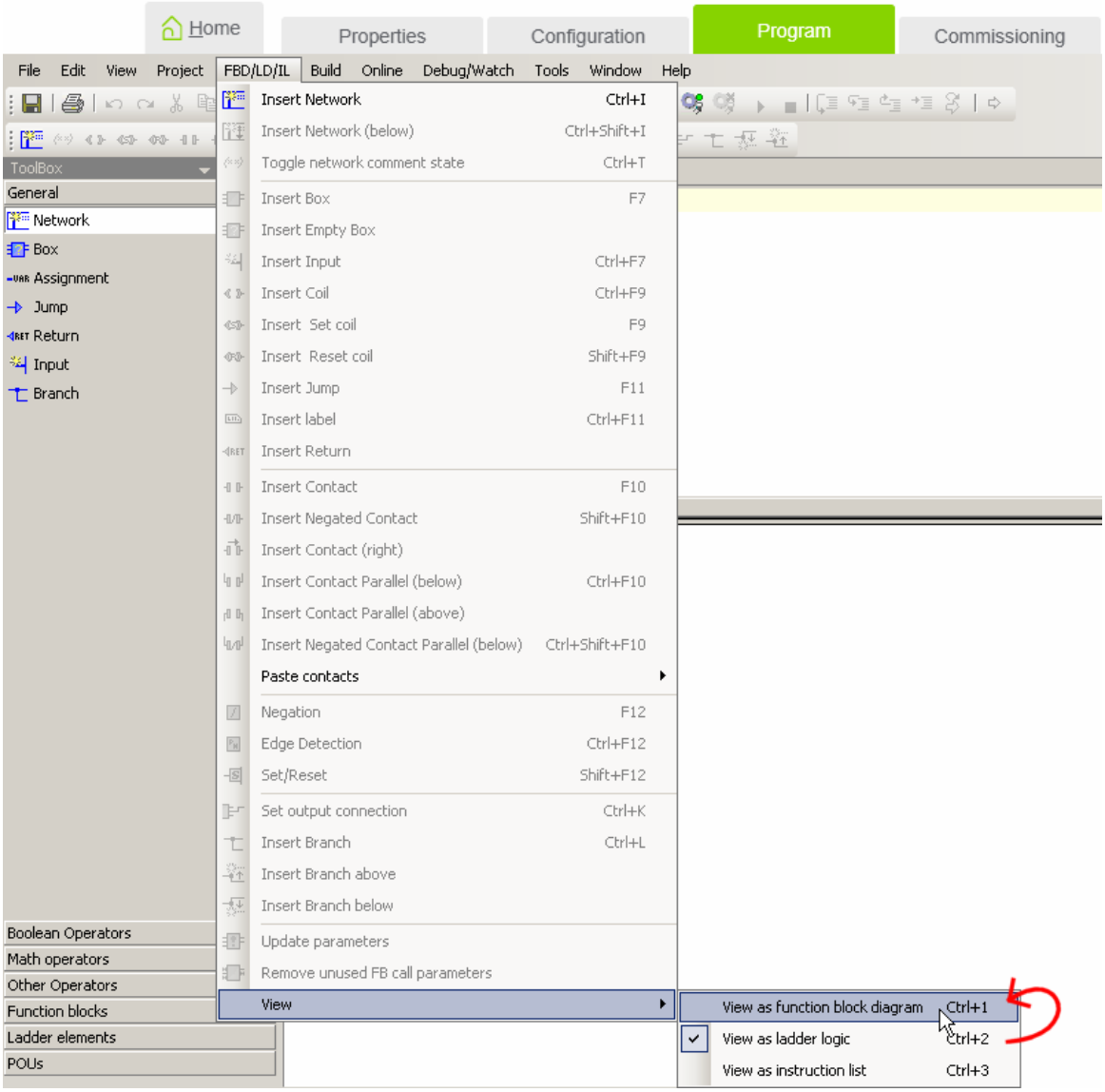


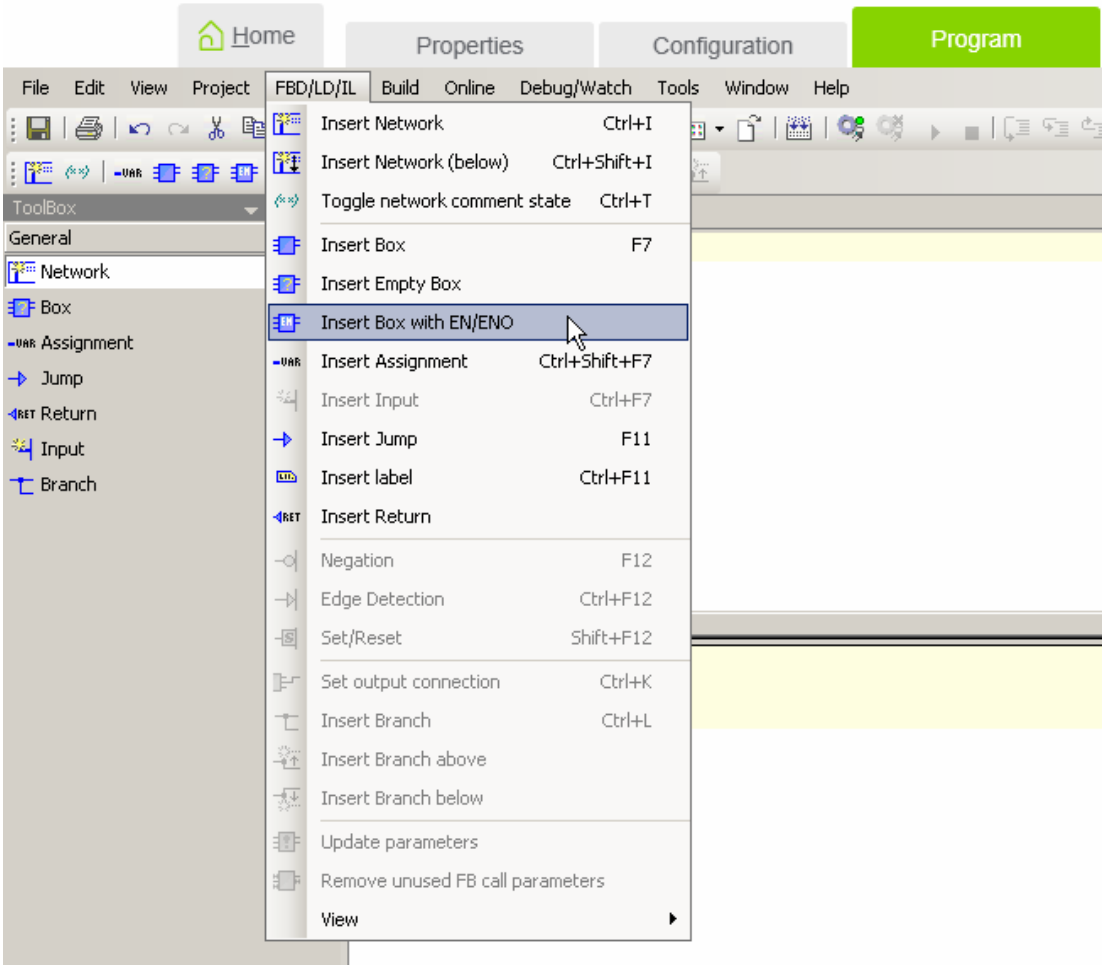
Step	Action
	
6	**Order By Data Flow** When you have finished implementing this CFC program, perform the following steps to correctly set the execution order of its blocks: ➢ Right-click on an empty spot of the central worksheet. ➢ In the **Execution Order** part of the contextual menu, execute the **Order By Data Flow** command. 

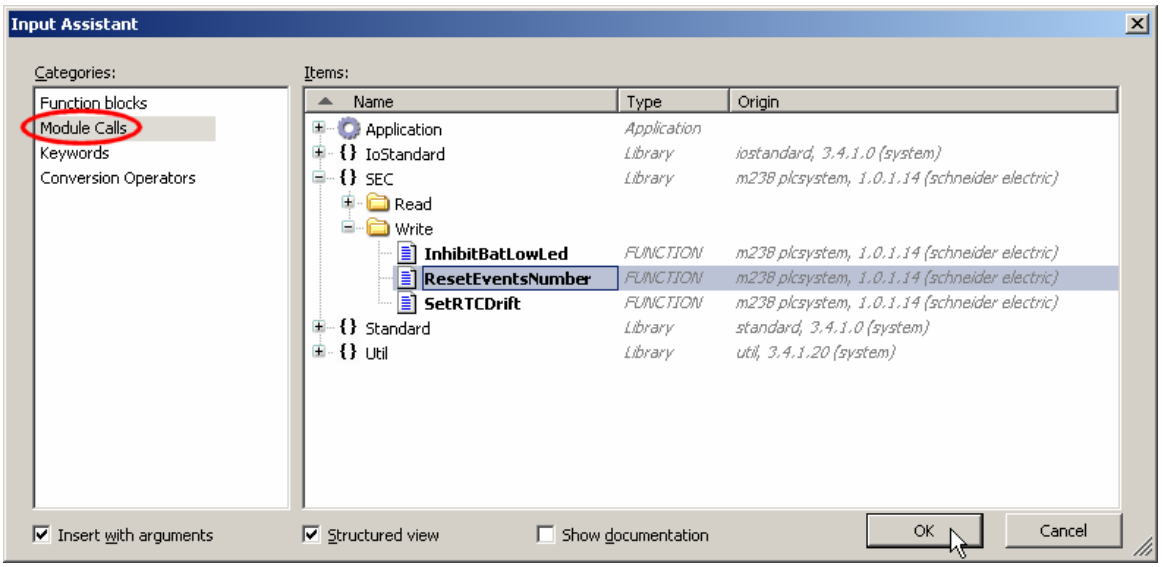
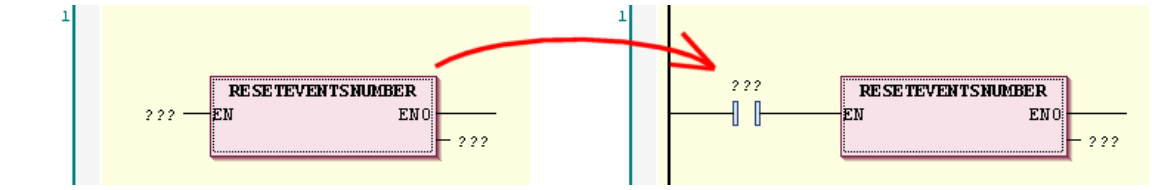
7.2. LD Program

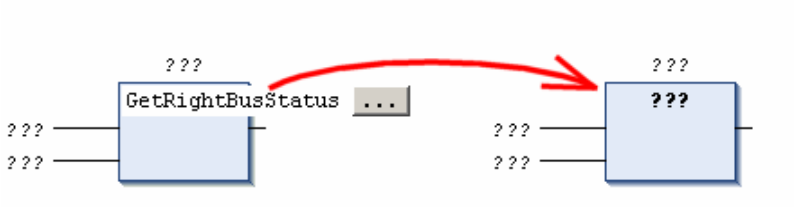
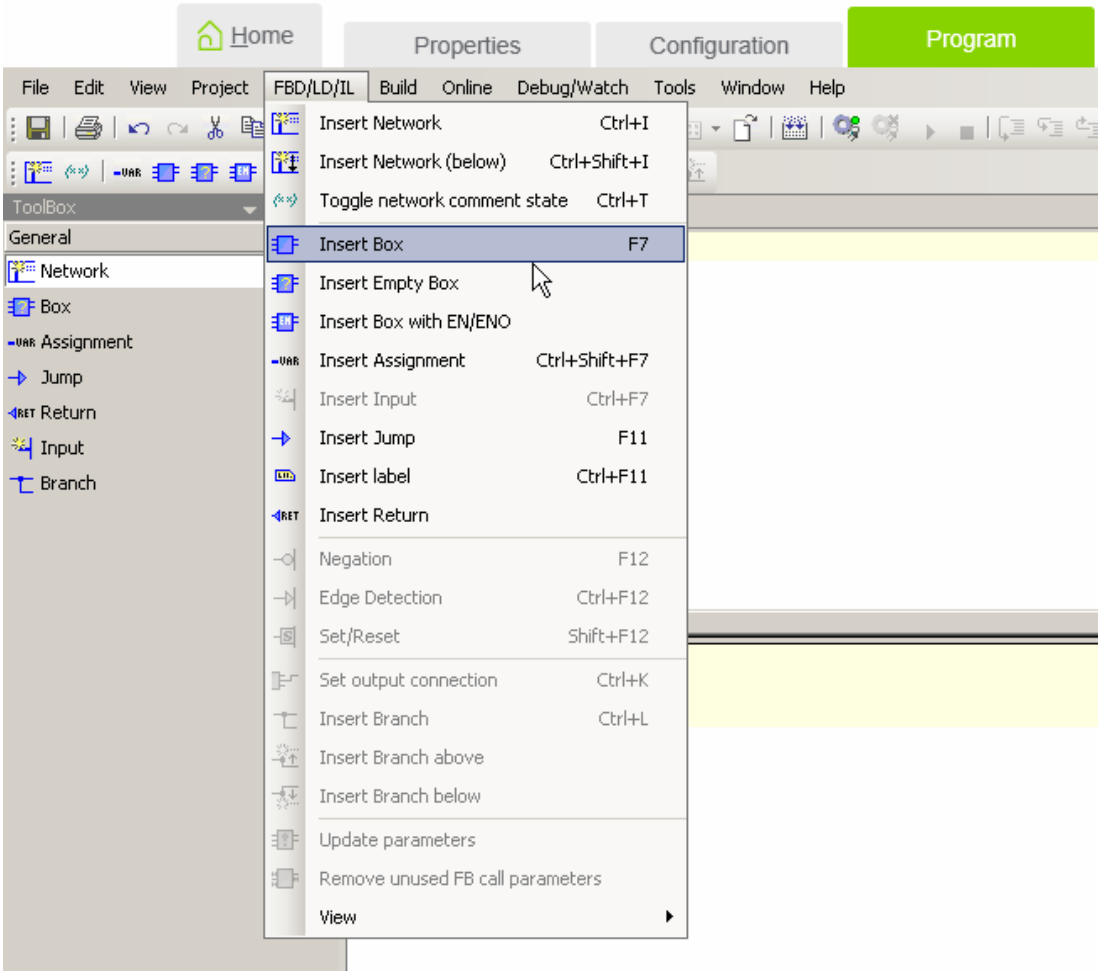
Before instructing you how to write down the LD program used in this example, the following table presents optional steps that give you information on how to program in LD language.

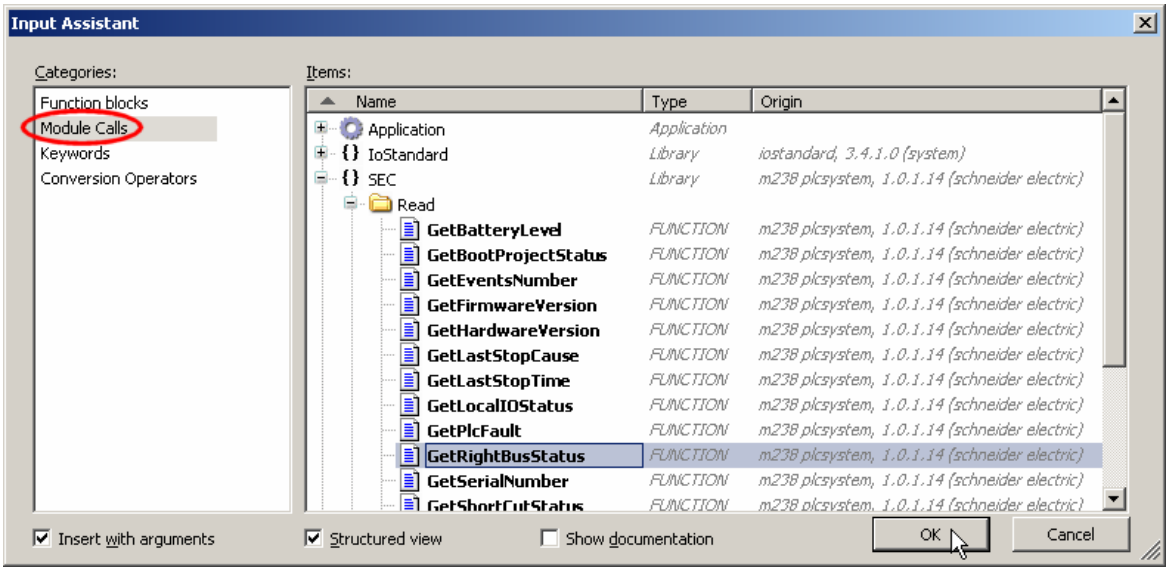
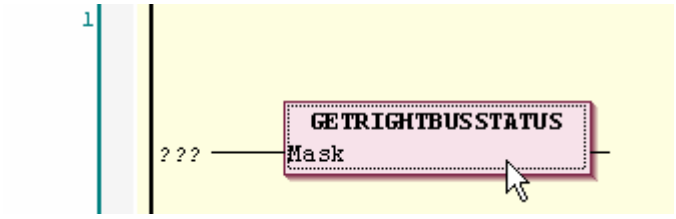
Step	Action
1	**Optional step** **How to display titles and comments in LD language** ➢ Select the **Options…** command of the **Tools** menu. ➢ Select, in the **Options** window, the **FBD, LD and IL editor** section. ➢ If you wish to add a title and/or a comment for each LD network, check the ☑ **Show network title** and/or the ☑ **Show network comment** options, as shown below: 

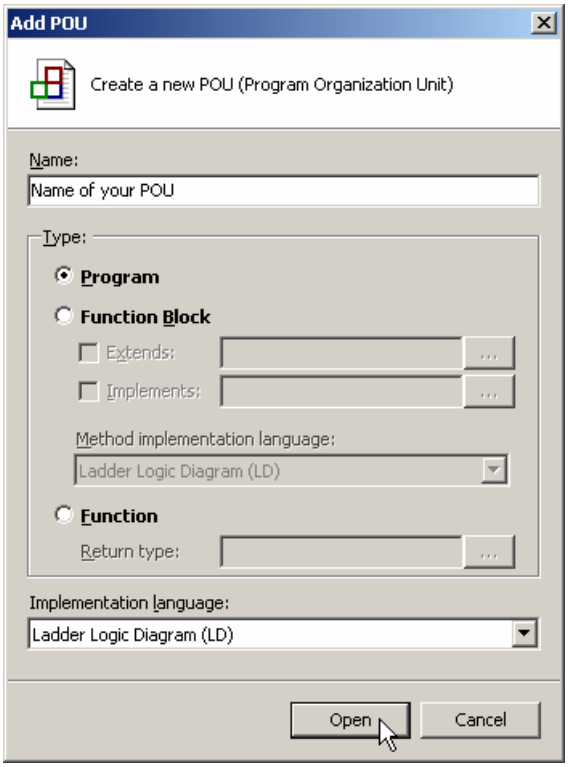
Step	Action
2	**Optional step** **How to add the EN/ENO pins on a Function Block** In this example, these pins are used on three functions: **IsFirstMastWarmCycle**, **IsFirstMastColdCycle**, and **ResetEventsNumber**. This optional step indicates how to append the additional Boolean enable input EN and Boolean output ENO (Enable Out) to any Function or Function Block. These two pins are used to enable (TRUE) or disable (FALSE) the execution of the Functions or Function Blocks on which they are used. Please refer to the SoMachine online help: search for **additional Boolean ENO**. ➢ In the **View** part of the **FBD/LD/IL** menu, select the **View as function block diagram** mode. 

Step	Action
	➢ Select, in your LD program, the LD network where you want to add a Function Block (or a Function). ➢ In the **FBD/LD/IL** menu, execute the **Insert Box with EN/ENO** command. 

Step	Action
	➢ In the **Input Assistant** window that appears, select the **Function blocks** category, if you wish to add a Function Block, or the **Module Calls** category, if you wish to add a Function. ➢ Select the Function Block (or the Function) you wish to add in your program. In the following picture, the **ResetEventsNumber** Function of the **M238 PLCSystem** library is selected. The location of this Function is given in *Functions Used in this Example (see page 14)*:  ➢ Click on **OK**. This adds the selected Function Block (or Function) to your program, with the two EN and ENO pins. ➢ In the **View** part of the **FBD/LD/IL** menu, you can now select the **View as ladder logic** mode to return to the default view mode. 

Step	Action
3	**Optional step** **How to use the Insert Empty Box Command** Sometimes, the Empty Box created by the **Insert Empty Box** command does not work with a Function or Function Block. In this example, the Empty Box is not compatible with the **GetRightBusStatus** and the **InhibitBatLowLed** Functions. In this case, you must use the **Insert Box** command, instead of the **Insert Empty Box** command. Example: In the following picture, the **GetRightBusStatus** Function is rejected by an Empty Box; in this case, the **Insert Box** command must be used:  ➢ Select, in your LD program, the LD network where you want to add a Function Block (or a Function). ➢ In the **FBD/LD/IL** menu, execute the **Insert Box** command. 

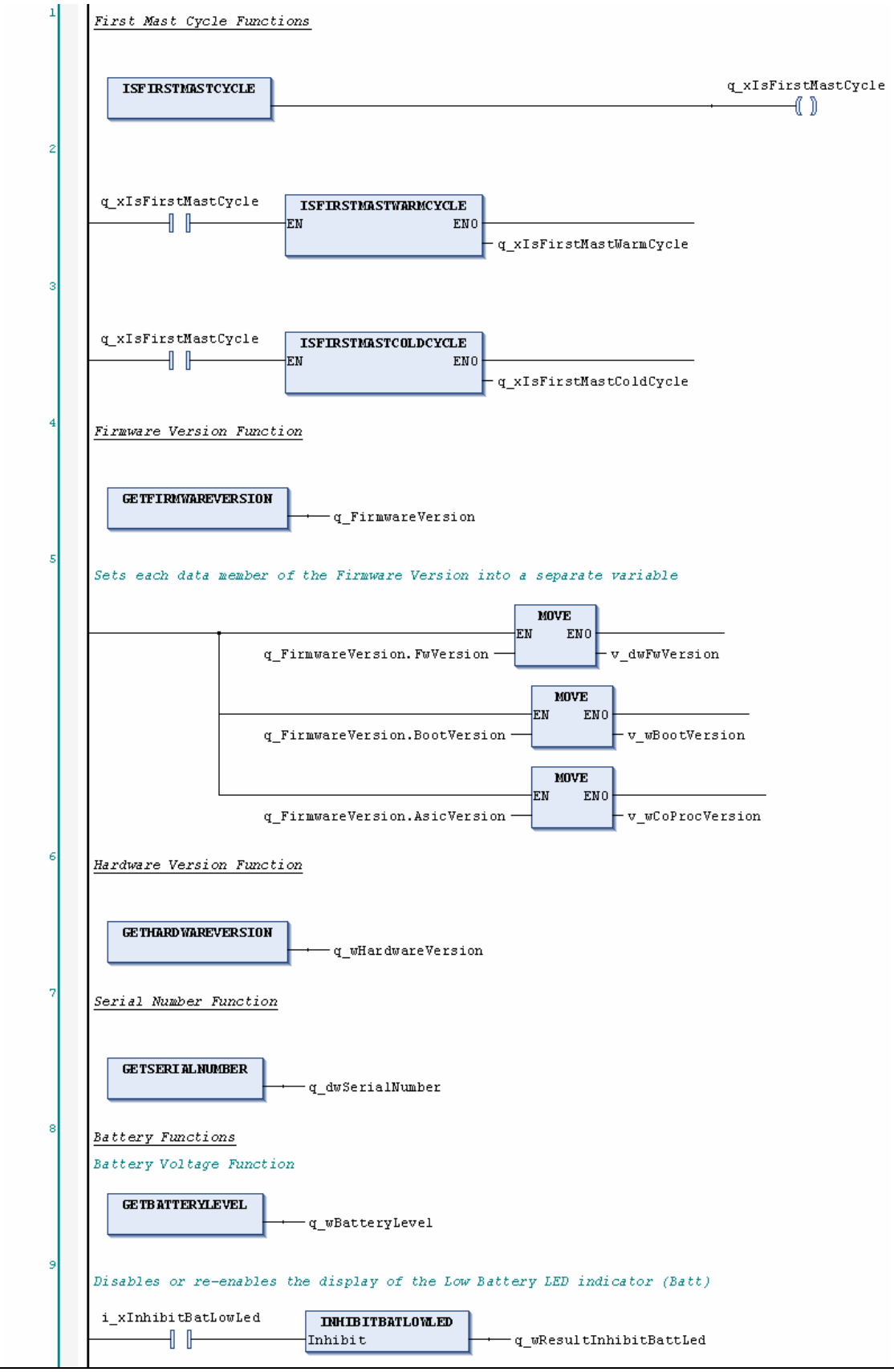
Step	Action
	➢ In the **Input Assistant** window that appears, select the **Function blocks** category, if you wish to add a Function Block, or the **Module Calls** category, if you wish to add a Function. ➢ Select the Function Block (or the Function) you wish to add in your program. In the following picture, the **GetRightBusStatus** Function of the **M238 PLCSystem** library is selected:  ➢ Click on **OK**. This adds the selected Function Block (or Function) to your program. 

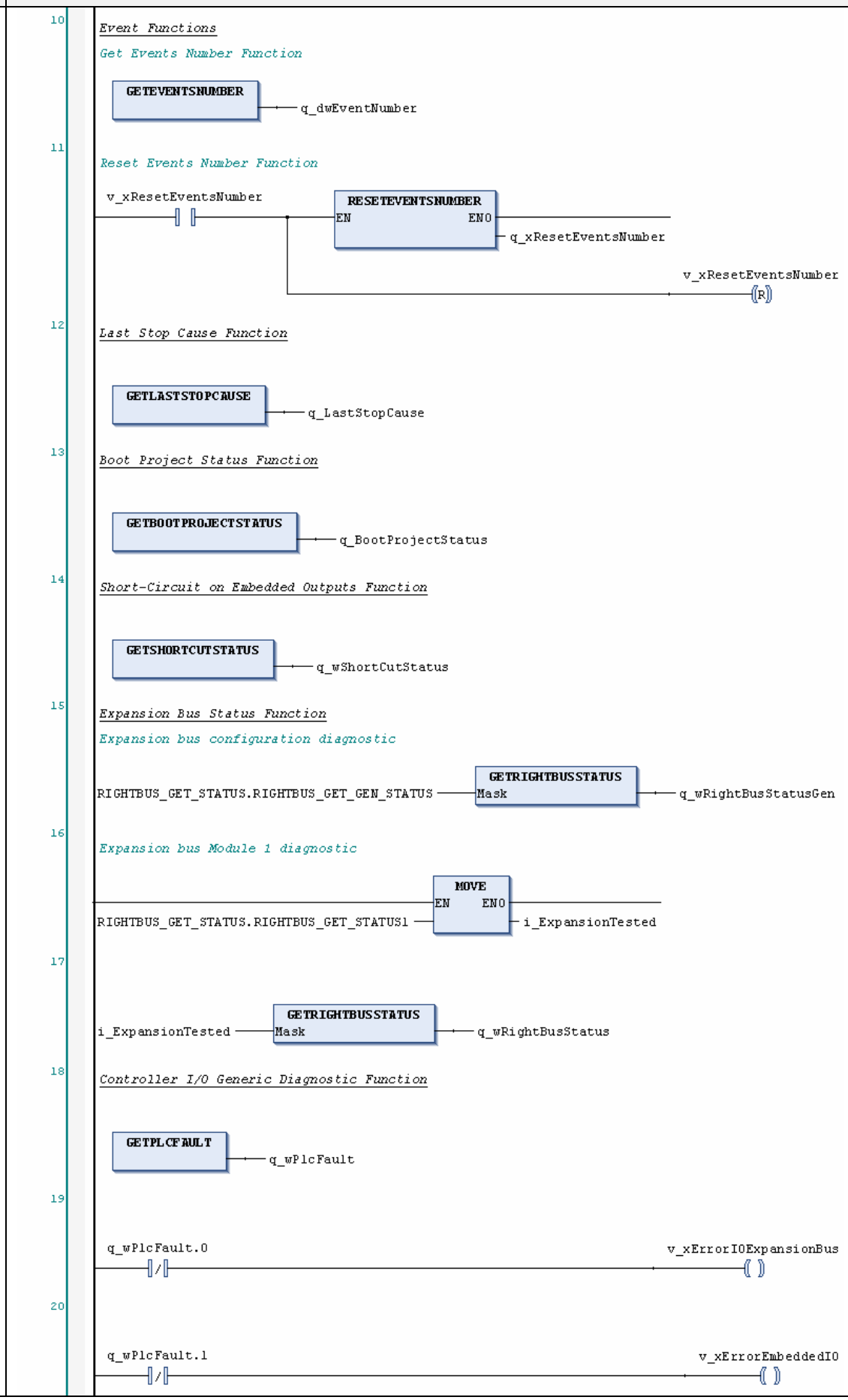
Step	Action
4	**Creation of the POU:** Create a new POU in LD language, called **PLC_Diag_LD**.  Upon creation of this POU, it is automatically opened by SoMachine.

Step	Action
5	**LD variables:** In the upper part of the LD editor, declare the following variables:

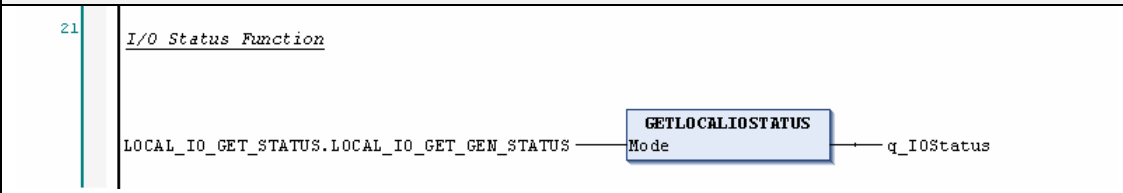
```
PROGRAM PLC_Diag_LD
VAR
(*First Mast Cycle Functions*)
    q_xIsFirstMastCycle         : BOOL;
    q_xIsFirstMastWarmCycle     : BOOL;
    q_xIsFirstMastColdCycle     : BOOL;

(*Firmware Version Function*)
    q_FirmwareVersion           : FIRMWARE_VERSION;
    v_dwFwVersion               : DWORD;
    v_wBootVersion              : WORD;
    v_wCoProcVersion            : WORD;

(*Hardware Version Function*)
    q_wHardwareVersion          : WORD;

(*Serial Number Function*)
    q_dwSerialNumber            : DWORD;

(*Battery Functions*)
    q_wBatteryLevel             : WORD;
    i_xInhibitBatLowLed         : BOOL  := FALSE;
    q_wResultInhibitBattLed     : WORD;

(*Event Functions*)
    q_dwEventNumber             : DWORD;
    v_xResetEventsNumber        : BOOL  := FALSE;
    q_xResetEventsNumber        : BOOL;

(*Last Stop Cause Function*)
    q_LastStopCause             : STOP_WHY;

(*Boot Project Status Function*)
    q_BootProjectStatus         : BOOT_PROJECT_STATUS;

(*Short-Circuit Status on Embedded Outputs Function*)
    q_wShortCutStatus           : WORD;

(*Expansion Bus Status Function*)
    i_ExpansionTested           : RIGHTBUS_GET_STATUS;
    q_wRightBusStatusGen        : WORD;
    q_wRightBusStatus           : WORD;

(*Controller I/O Generic Diagnostic Function*)
    q_wPlcFault                 : WORD;
    v_xErrorIOExpansionBus      : BOOL;
    v_xErrorEmbeddedIO          : BOOL;

(*I/O Status Function*)
    q_IOStatus                  : LOCAL_IO_GEN_STATUS;
END_VAR
```

Step	Action
6	**LD program:** In the lower part of the LD editor, implement the following program:



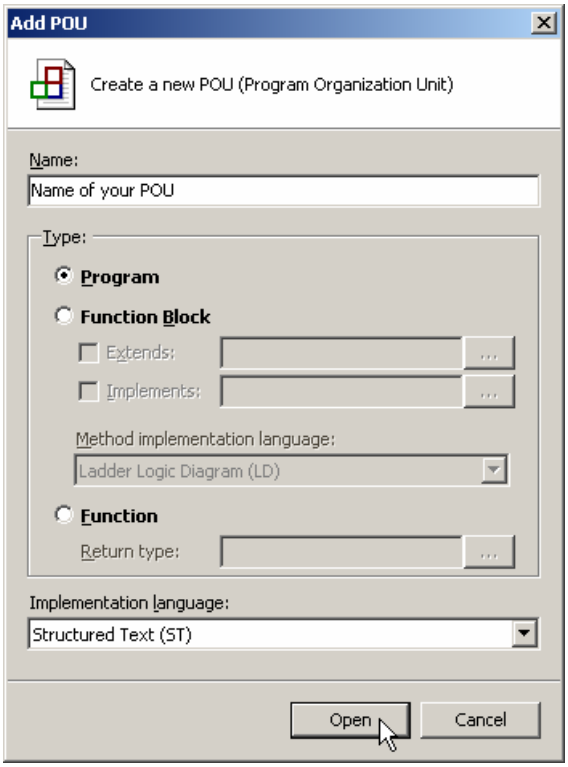
Step	Action
	

10
Event Functions
Get Events Number Function
GETEVENTSNUMBER
q_dwEventNumber
11
Reset Events Number Function
v_xResetEventsNumber
RESETEVENTSNUMBER
EN
ENO
q_xResetEventsNumber
v_xResetEventsNumber
R
12
Last Stop Cause Function
GETLASTSTOPCAUSE
q_LastStopCause
13
Boot Project Status Function
GETBOOTPROJECTSTATUS
q_BootProjectStatus
14
Short-Circuit on Embedded Outputs Function
GETSHORTCUTSTATUS
q_wShortCutStatus
15
Expansion Bus Status Function
Expansion bus configuration diagnostic
RIGHTBUS_GET_STATUS.RIGHTBUS_GET_GEN_STATUS
GETRIGHTBUSSTATUS
Mask
q_wRightBusStatusGen
16
Expansion bus Module 1 diagnostic
MOVE
EN
ENO
RIGHTBUS_GET_STATUS.RIGHTBUS_GET_STATUS1
i_ExpansionTested
17
i_ExpansionTested
GETRIGHTBUSSTATUS
Mask
q_wRightBusStatus
18
Controller I/O Generic Diagnostic Function
GETPLCFAULT
q_wPlcFault
19
q_wPlcFault.0
v_xErrorIOExpansionBus
20
q_wPlcFault.1
v_xErrorEmbeddedIO

Step	Action
	

21
I/O Status Function
LOCAL_IO_GET_STATUS.LOCAL_IO_GET_GEN_STATUS
GETLOCALIOSTATUS
Mode
q_IOStatus

7.3. ST Program

Step	Action
1	**Creation of the POU:** Create a new POU in ST language, called **PLC_Diag_ST**.  Upon creation of this POU, it is automatically opened by SoMachine.

Step	Action
2	**ST variables:** In the upper part of the ST editor, declare the following variables:

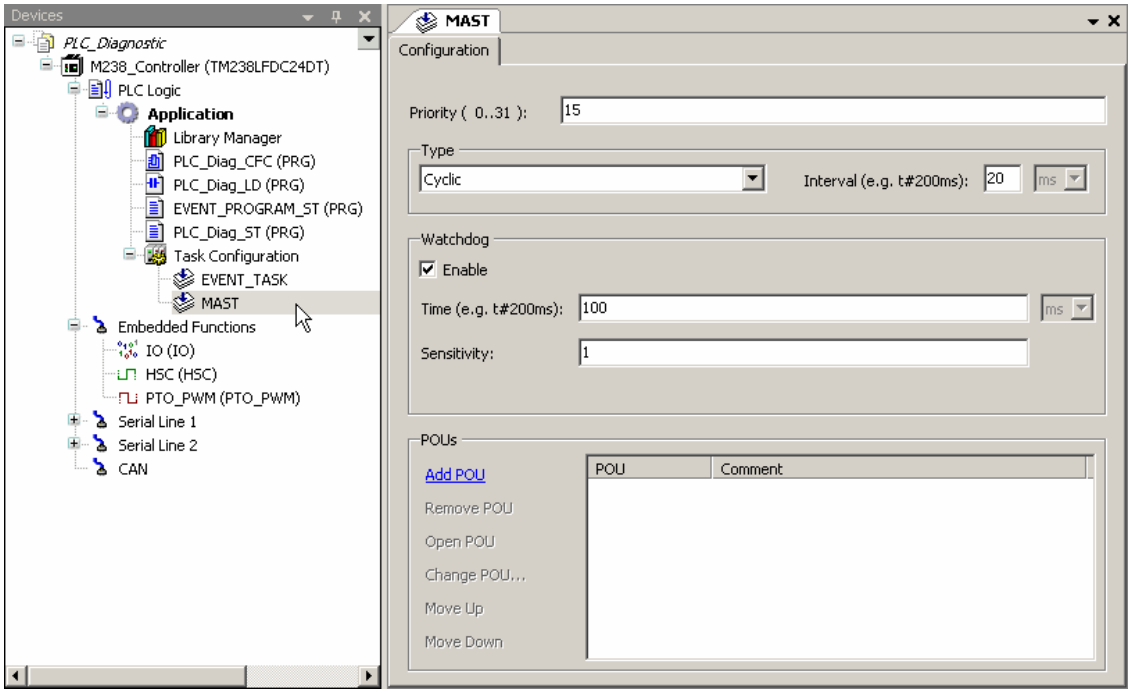
```
PROGRAM PLC_Diag_ST
VAR
(*First Mast Cycle Functions*)
    q_xIsFirstMastCycle         : BOOL;
    q_xIsFirstMastWarmCycle     : BOOL;
    q_xIsFirstMastColdCycle     : BOOL;

(*Firmware Version Function*)
    q_FirmwareVersion           : FIRMWARE_VERSION;
    v_dwFwVersion               : DWORD;
    v_wBootVersion              : WORD;
    v_wCoProcVersion            : WORD;

(*Hardware Version Function*)
    q_wHardwareVersion          : WORD;

(*Serial Number Function*)
    q_dwSerialNumber            : DWORD;

(*Battery Functions*)
    q_wBatteryLevel             : WORD;
    i_xInhibitBatLowLed         : BOOL  := FALSE;
    q_wResultInhibitBattLed     : WORD;

(*Event Functions*)
    q_dwEventNumber             : DWORD;
    v_xResetEventsNumber        : BOOL  := FALSE;
    q_xResetEventsNumber        : BOOL;

(*Last Stop Cause Function*)
    q_LastStopCause             : STOP_WHY;

(*Boot Project Status Function*)
    q_BootProjectStatus         : BOOT_PROJECT_STATUS;

(*Short-Circuit Status on Embedded Outputs Function*)
    q_wShortCutStatus           : WORD;

(*Expansion Bus Status Function*)
    i_ExpansionTested           : RIGHTBUS_GET_STATUS;
    q_wRightBusStatusGen        : WORD;
    q_wRightBusStatus           : WORD;

(*Controller I/O Generic Diagnostic Function*)
    q_wPlcFault                 : WORD;
    v_xErrorIOExpansionBus      : BOOL;
    v_xErrorEmbeddedIO          : BOOL;

(*I/O Status Function*)
    q_IOStatus                  : LOCAL_IO_GEN_STATUS;
END_VAR
```

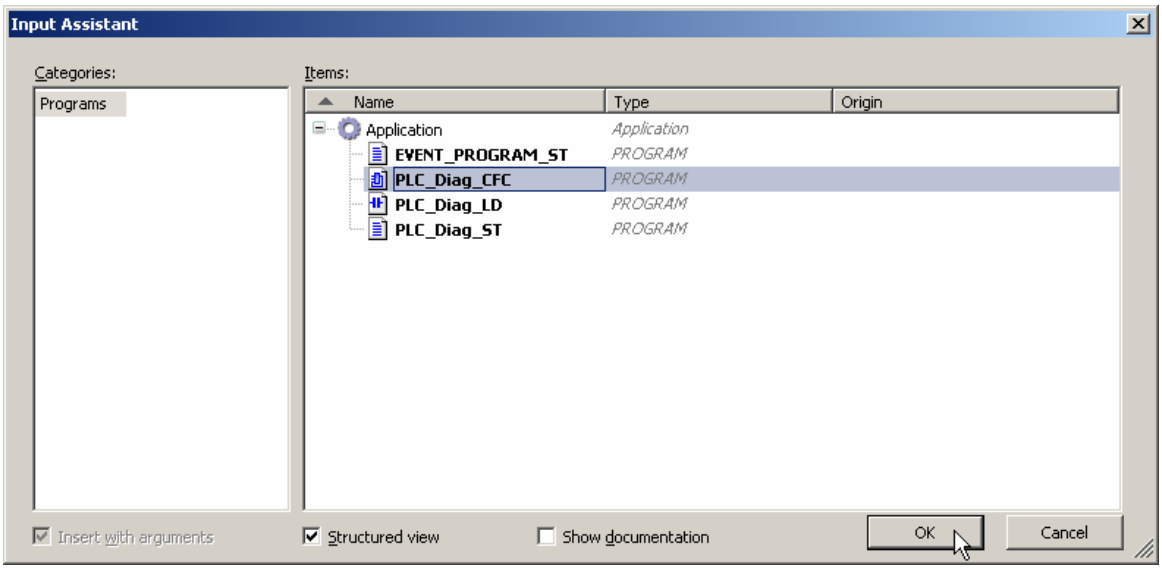
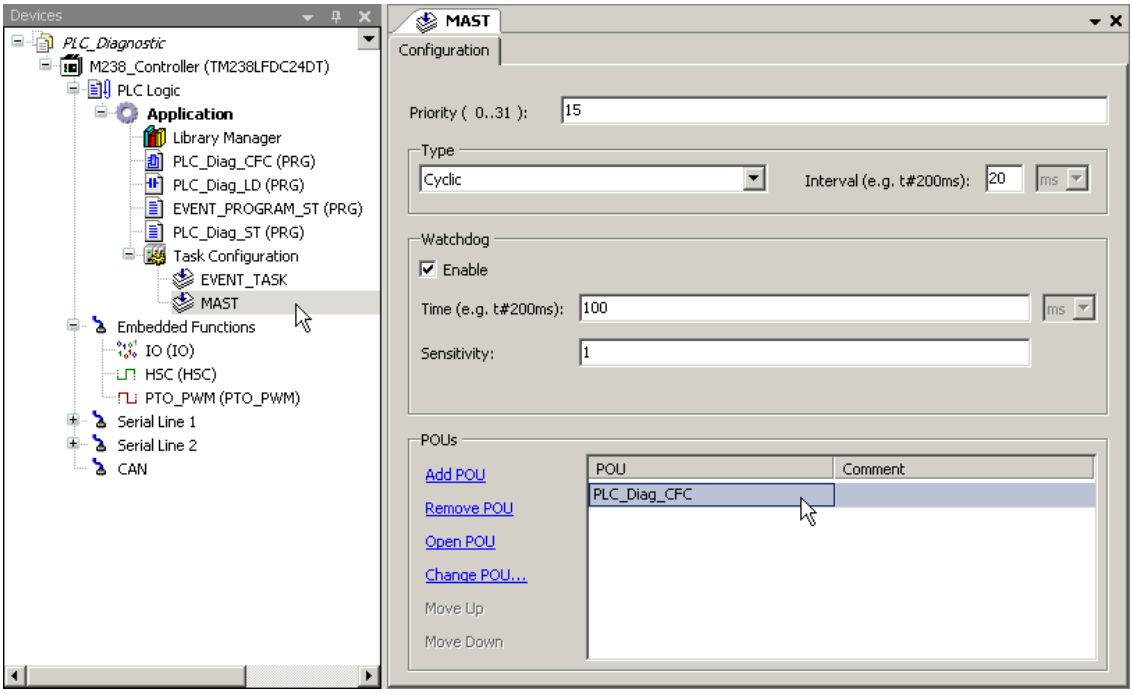
Step	Action
3	**ST program:** In the lower part of the ST editor, implement the following program:

```
(*First Mast Cycle Functions*)
    q_xIsFirstMastCycle := IsFirstMastCycle();
    IF q_xIsFirstMastCycle THEN
        q_xIsFirstMastWarmCycle := IsFirstMastWarmCycle();
        q_xIsFirstMastColdCycle := IsFirstMastColdCycle();
    END_IF

(*Firmware Version Function*)
    q_FirmwareVersion := GetFirmwareVersion();
    (*Sets each data member of the Firmware Version into a separate variable*)
    v_dwFwVersion     := q_FirmwareVersion.FwVersion;
    v_wBootVersion    := q_FirmwareVersion.BootVersion;
    v_wCoProcVersion  := q_FirmwareVersion.AsicVersion;

(*Hardware Version Function*)
    q_wHardwareVersion := GetHardwareVersion();

(*Serial Number Function*)
    q_dwSerialNumber := GetSerialNumber();

(*Battery Functions*)
    (*Battery Voltage Function*)
    q_wBatteryLevel := GetBatteryLevel();
    (*Disables or re-enables the display of the Low Battery LED indicator (Batt)*)
    q_wResultInhibitBattLed := InhibitBatLowLed(Inhibit := i_xInhibitBatLowLed);

(*Event Functions*)
    (*Get Events Number Function*)
    q_dwEventNumber := GetEventsNumber();
    (*Reset Events Number Function*)
    IF v_xResetEventsNumber THEN
        q_xResetEventsNumber := ResetEventsNumber();
        v_xResetEventsNumber := FALSE;
    END_IF

(*Last Stop Cause Function*)
    q_LastStopCause := GetLastStopCause();

(*Boot Project Status Function*)
    q_BootProjectStatus := GetBootProjectStatus();

(*Short-Circuit on Embedded Outputs Function*)
    q_wShortCutStatus := GetShortCutStatus();

(*Expansion Bus Status Function*)
    (*Expansion bus configuration diagnostic*)
    q_wRightBusStatusGen := GetRightBusStatus(Mask :=
RIGHTBUS_GET_STATUS.RIGHTBUS_GET_GEN_STATUS);
    (*Expansion bus Module 1 diagnostic*)
    i_ExpansionTested := RIGHTBUS_GET_STATUS.RIGHTBUS_GET_STATUS1;
    q_wRightBusStatus := GetRightBusStatus(Mask := i_ExpansionTested);

(*Controller I/O Generic Diagnostic Function*)
    q_wPlcFault := GetPlcFault();
    v_xErrorIOExpansionBus := NOT q_wPlcFault.0;
    v_xErrorEmbeddedIO     := NOT q_wPlcFault.1;

(*I/O Status Function*)
    q_IOStatus := GetLocalIOStatus(Mode :=
LOCAL_IO_GET_STATUS.LOCAL_IO_GET_GEN_STATUS);
```

8. Running the Example

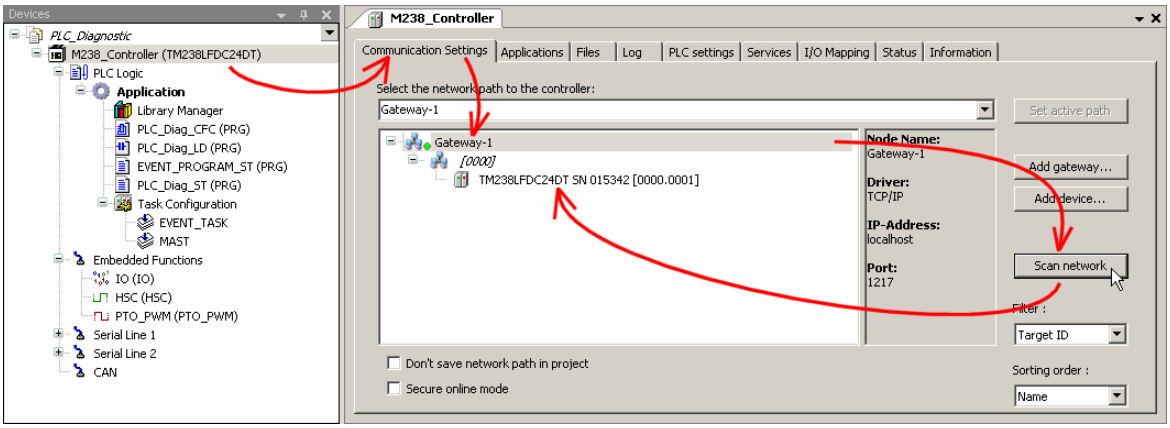
8.1. MAST Task Configuration

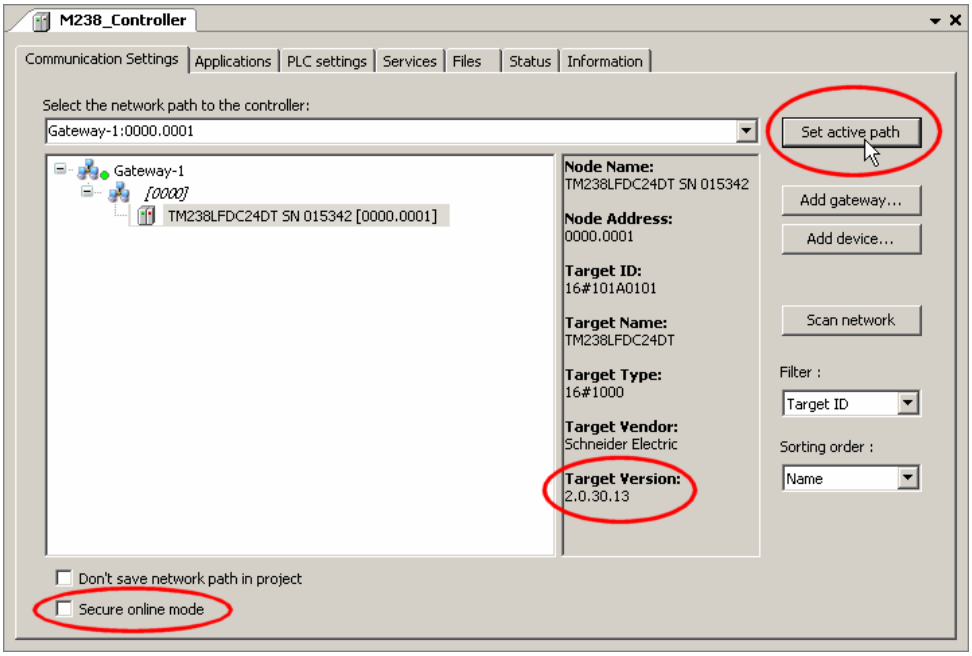
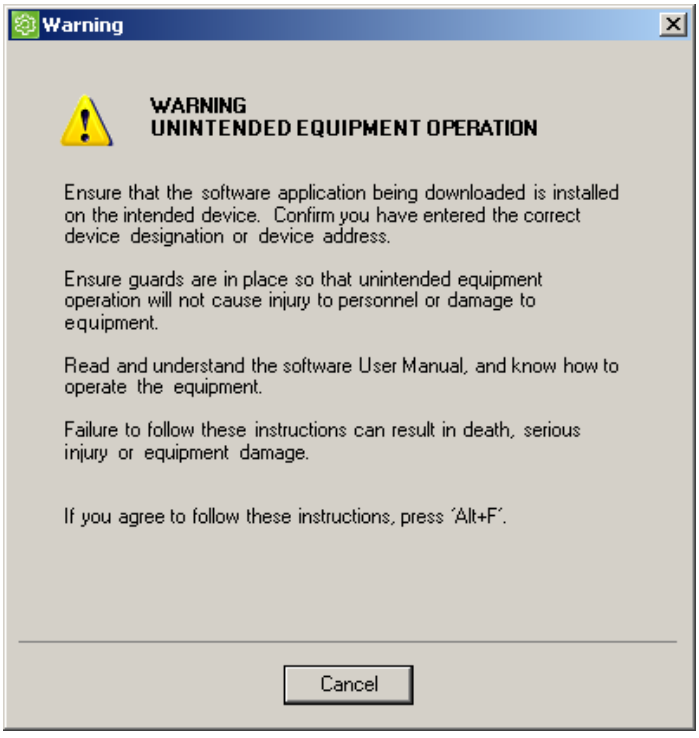
Step	Action
1	In the **Devices** tree view: ➢ Expand the contents of the **M238_Controller (TM238LFDC24DT)** item. ➢ Double-click on the **MAST** task of the **Task Configuration** item. ➢ Click on the **Add POU** command. 

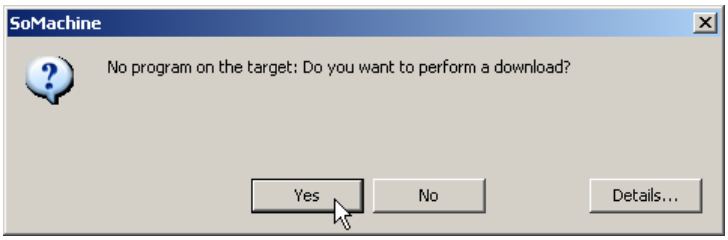
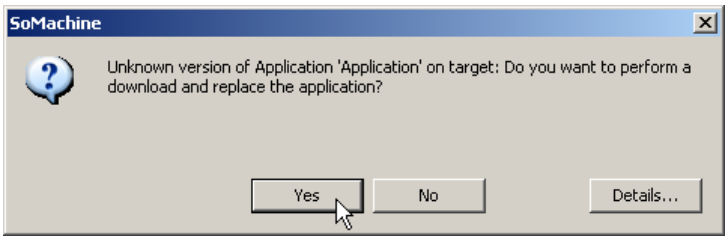
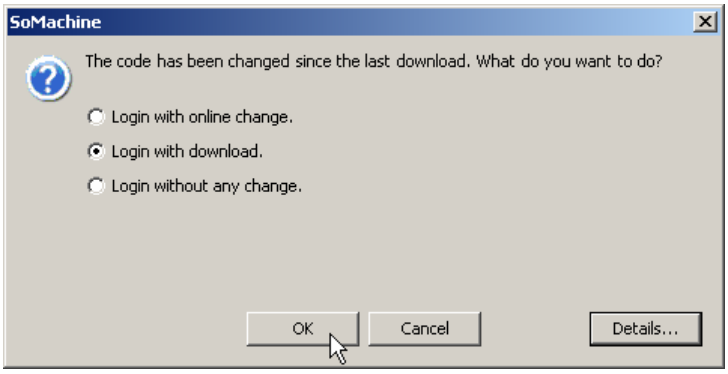
Step	Action
2	In the **Input Assistant** window that appears: ➢ Select your program. In the following picture, the CFC program described in *CFC Program (see page 27)*, **PLC_Diag_CFC**, is selected: 
3	Click on **OK**. This adds the selected POU to the list of programs run by the MAST task of the controller. 

8.2. Downloading the Example to the Controller

The steps listed in the following table describe how to download the example to the Logic Controller. If needed, please refer to the SoMachine online help for further information on these steps: search for **Communication Settings**.

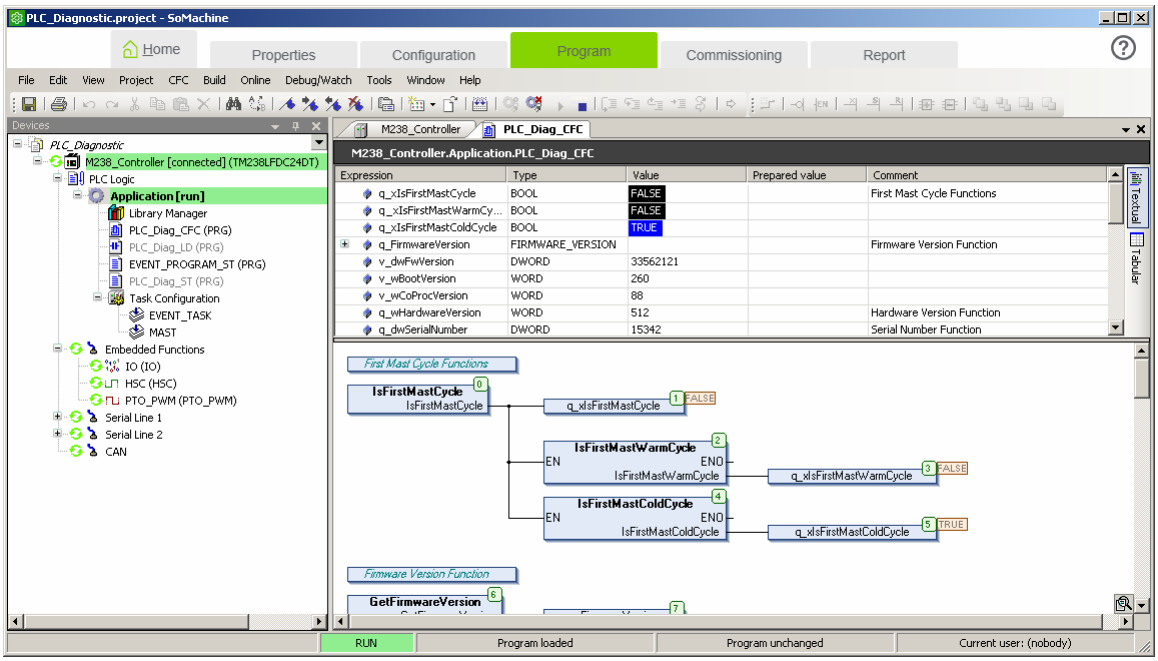
Step	Action
1	Connect the USB programming cable between your PC and the M238 Controller. Please refer to *Hardware Installation (see page 16)*, for the reference and usage of this cable.
2	In the **Devices** tree view, double-click on the **M238_Controller (TM238LFDC24DT)** item to open its configuration panel.
3	In the **Communication Settings** tab of this panel. ➢ Click on the **Gateway-1** node ➢ Click on the Scan network button If the controller is switched on and connected to your PC with the USB programming cable, it will be detected by SoMachine as shown below:  In this example, the **TM238LFDC24DT SN 015342** controller has been detected.

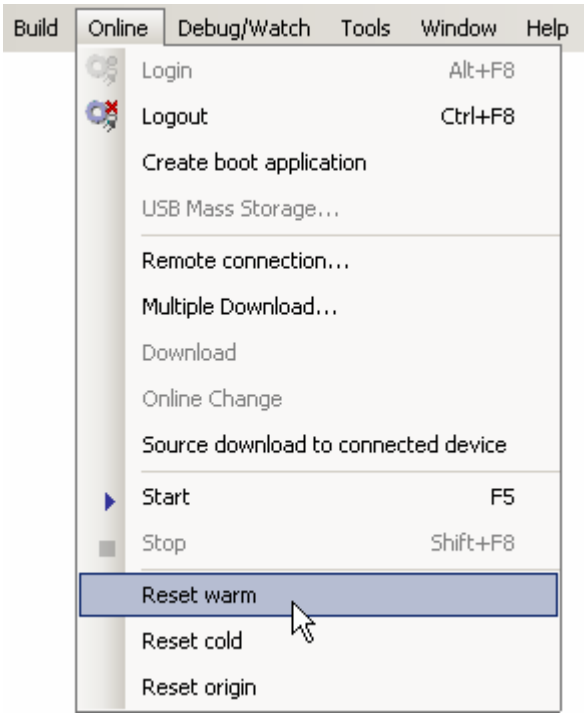
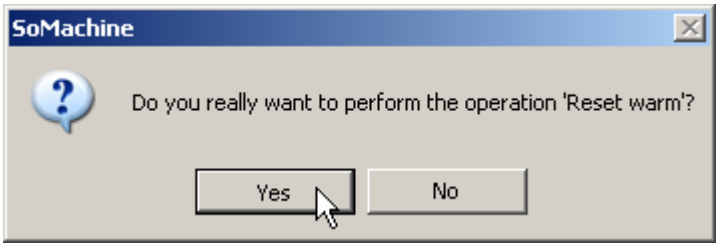
Step	Action
4	Select this controller's node and click on the Set active path button.  This will show SoMachine the way to reach your controller. **Option:** Uncheck the ☐ **Secure online mode** box to avoid validation messages during future online modifications. **Note:** The firmware version of your controller is displayed as the **Target Version**.
5	The following window will appear and ask you to confirm your choice: ➢ Read the hazard message; ➢ Press on both **<Alt>** and **<F>** keys to validate your choice and close this window; ➢ Or, click on the **Cancel** button if you can not comply with the statements in the hazard message. 

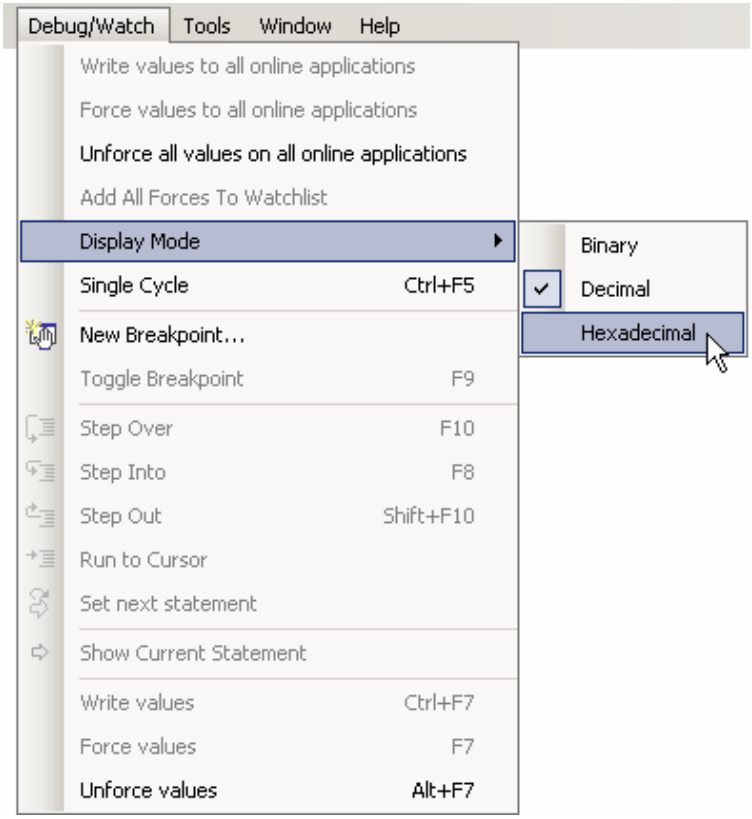
Step	Action
6	Click on the **Login** button of the toolbar to establish a connection from SoMachine to the controller.
7a	If you download the current project to the controller for the first time and if there is no project on the controller, the following window appears:  Click on **Yes** to download the **Application** software to the controller.
7b	If you download the current project to the controller for the first time and if there is another project on the controller, the following window appears:  Click on **Yes** to download the **Application** software to the controller.
7c	If you already have downloaded the current project to the controller, the following window appears if you have brought new modifications to this project:  ➢ Select the ⊙ **Login with download** choice. Click on **OK** to download the **Application** software to the controller.
7d	If you already have downloaded the current project to the controller, the connection is immediate if you brought no modification to this project.
8	Wait for the completion of the download operation. Once it is finished, the status bar of SoMachine displays the state of the controller: STOP
9	Click on the **Start** button of the toolbar to run the **Application** software on the controller. The state of the controller switches from STOP to RUN

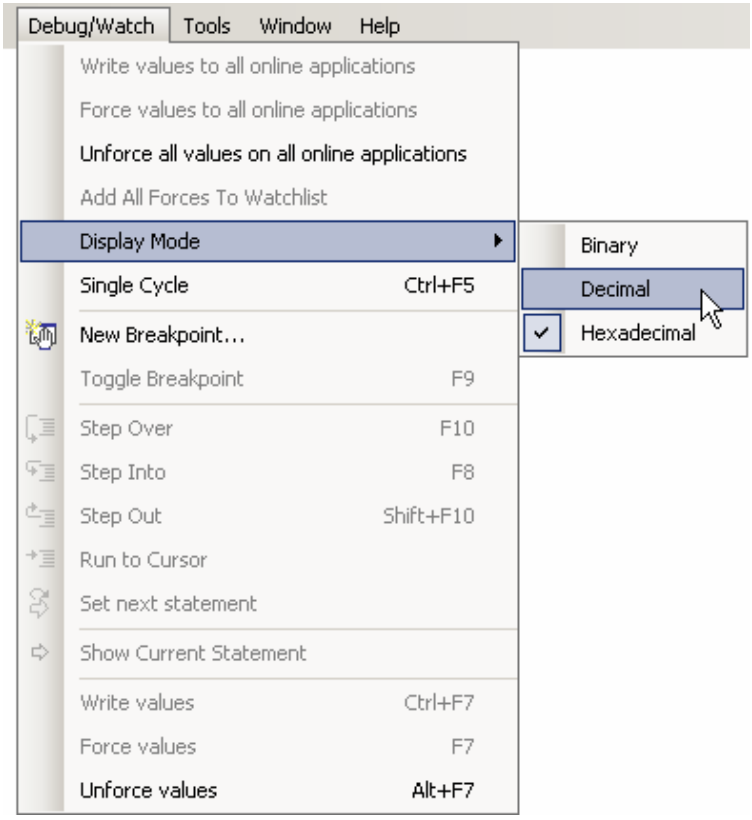
8.3. Running the Example on the Controller

The steps listed in the following table describe how to use the example, once it has been downloaded to the controller.

Step	Action
1	In the **Devices** panel, double-click on your **PLC_Diag** program to open it (e.g. **PLC_Diag_CFC** if you previously selected this program as the program run by the **MAST** task of your controller). This command opens the program in the central panel of SoMachine; the online values of the variables used by this program are also displayed, as shown below: 
2	**Detecting the first cycle of the MAST task using the IsFirstMastCycle, IsFirstMastWarmCycle and IsFirstMastColdCycle Functions:** In this example, these three Functions are used in the following manner: • The **IsFirstMastCycle** Function is continuously called to detect the first cycle of the MAST task. This Function updates the `q_xIsFirstMastCycle` variable: it is set to TRUE only during the first cycle of the MAST task, then it is reset to FALSE. • The **IsFirstMastWarmCycle** Function is only called during the first cycle of the MAST task (the previous `q_xIsFirstMastCycle` variable is used to enable or disable its call). This Function updates the `q_xIsFirstMastWarmCycle` variable: it is set to TRUE if the controller started in **warm start** mode. • Likewise, the **IsFirstMastColdCycle** Function is only called during the first cycle of the MAST task. This Function updates the `q_xIsFirstMastColdCycle` variable: it is set to TRUE if the controller started in **cold start** mode.

Step	Action
	To test these functions, proceed as follows: ➢ Check that the value of `q_xIsFirstMastWarmCycle` is equal to FALSE. ➢ Check that the value of `q_xIsFirstMastColdCycle` is equal to TRUE: the controller started in **cold start** mode (normal behavior after a download operation). ➢ Click on the **Stop** button of the toolbar to stop the **Application** software on the controller. ➢ In the **Online** menu, click on the **Reset warm** command to force a **warm start** of the controller.  ➢ Click on **Yes** to confirm this command.  ➢ Click on the **Start** button of the toolbar to run the **Application** software on the controller. ➢ Check that the value of `q_xIsFirstMastWarmCycle` is equal to TRUE: the controller started in **warm start** mode because of the **Reset warm** command. ➢ Check that the value of `q_xIsFirstMastColdCycle` is equal to FALSE. For a description of the **cold start** and **warm start** modes, please refer to the SoMachine online help: search for **Reset Warm**, **Reset Cold**, or **Commanding State Transitions**.

Step	Action
3	**Getting the controller's firmware, boot, coprocessor, and hardware versions using the GetFirmwareVersion and GetHardwareVersion Functions:** To test these functions, proceed as follows: ➢ In the **Debug/Watch** menu, execute the **Hexadecimal** command to switch the numeric display mode from **Decimal** to **Hexadecimal**.  ➢ Split the value of the `v_dwFwVersion` DWORD variable (32-bit) into four BYTE data (4 × 8-bit). ➢ Convert each of these bytes in Decimal to get the controller's firmware version. An example is given below: `v_dwFwVersion` .....16#02001E0D 4-byte data .....16#02 / 16#00 / 16#1E / 16#0D 4-byte version.....2 / 0 / 30 / 13 Firmware version.....v2.0.30.13 **Note:** Compare this version with the firmware version displayed when you select your controller's node, as shown in *Downloading the Example to the Controller (see page 51)*. They are identical. ➢ Split the value of the `v_wBootVersion` WORD variable (16-bit) into two BYTE data (2 × 8-bit). ➢ Convert each of these bytes in Decimal to get the controller's boot version. An example is given below: `v_wBootVersion` .....16#0104 2-byte data .....16#01 / 16#04 2-byte version.....1 / 4 Boot version v1.4

Step	Action
	➢ The value of `v_wCoProcVersion` indicates the controller's coprocessor version. Example: 16#0058. ➢ The value of `q_wHardwareVersion` indicates the controller's Hardware Version. Example: 16#0200.
4	**Getting the controller's Serial Number using the GetSerialNumber Function:** To test this function, proceed as follows: ➢ In the **Debug/Watch** menu, execute the **Hexadecimal** command to switch back the numeric display mode from **Hexadecimal** to **Decimal**.  ➢ The value of `q_dwSerialNumber` indicates the controller's Serial Number. Example: 15342. **Note:** Compare this value with the Serial Number (notation: "**S/N**") written on the controller's identification sticker. They are identical. Example: "**S/N :015342**".
5	**Getting the battery level using the GetBatteryLevel Function:** To test this function, proceed as follows: ➢ Check the value of `q_wBatteryLevel`: it is a percentage that ranges from 0 (empty battery) to 100 (fully charged battery). **Note:** If no external backup battery (ref. TSX PLP 01) has been installed on your controller, then `q_wBatteryLevel` is equal to 0.

Step	Action
6	**Disabling and re-enabling the Batt LED using the InhibitBatLowLed Function:** If an external backup battery is installed on your controller, remove it as instructed below: ➢ Click on the  **Logout** button of the toolbar to disconnect SoMachine from the controller. ➢ Power off the controller. ➢ Read the hazard message given in the **Installing and Replacing the External backup battery** section of the *Modicon M238 Logic Controller Hardware Guide*. ➢ Remove the external backup battery as instructed in this very same section of the *Modicon M238 Logic Controller Hardware Guide*. ➢ Power on the controller. ➢ Click on the  **Login** button of the toolbar to establish a connection from SoMachine to the controller. To test this function, proceed as follows: ➢ Check that the red **Batt** LED of the controller is ON. ➢ Change the value of `i_xInhibitBatLowLed` from FALSE to TRUE. ➢ Check that the red **Batt** LED of the controller is OFF. ➢ Check that the value of `q_wResultInhibitBattLed` is equal to TRUE (operation successful). ➢ Change the value of `i_xInhibitBatLowLed` from TRUE to FALSE. ➢ Check that the red **Batt** LED of the controller is ON. ➢ Check that the value of `q_wResultInhibitBattLed` is equal to TRUE (operation successful). If you previously removed an external backup battery from your controller, install it as instructed below: ➢ Click on the  **Logout** button. ➢ Power off the controller. ➢ Install the external backup battery as instructed in the **Installing and Replacing the External backup battery** section of the *Modicon M238 Logic Controller Hardware Guide*. ➢ Power on the controller. ➢ Click on the  **Login** button.

Step	Action
7	**Reading and resetting the number of events using the GetEventsNumber and ResetEventsNumber Functions:** To test these functions, proceed as follows: ➢ Using the push-button connected to the **I0** fast input of the controller, generate a few pulses on this input. **Note:** This push-button is described in *Wiring the Controller's I0 Fast Input (see page 24)*. ➢ Observe the value of `q_dwEventNumber`: each pulse increased this value by two (+1 for the rising edge of the pulse and +1 for its falling edge). ➢ Set the `v_xResetEventsNumber` variable to TRUE. Since this variable is connected to the **EN** (enable) pin of the **ResetEventsNumber** Function, this Function is now called. Then, this variable resets the `v_xResetEventsNumber` variable to FALSE. **Note:** In the case of the CFC program described in *CFC Program (see page 27)*, this variable is transmitted through the **ENO** (enable output) pin before resetting `v_xResetEventsNumber`. This is intended to obtain an execution order in which the **ResetEventsNumber** Function is called before the reset of `v_xResetEventsNumber`, provided that the **Order By Data Flow** command *(see page 33)* has been performed. Because `v_xResetEventsNumber` is reset, the **ResetEventsNumber** Function is called during only one task cycle. ➢ Check that the value of `q_dwEventNumber` is equal to 0.
8	**Reading the cause of the last stop using the GetLastStopCause Function:** To test this function, proceed as follows: ➢ Click on the **Stop** button of the toolbar. ➢ Click on the **Start** button of the toolbar. ➢ Check that the value of `q_LastStopCause` is equal to `STOP_FROM_STOP_REQUEST`. ➢ Click on the **Logout** button of the toolbar. ➢ Power off the controller. ➢ Wait during two seconds. **Note:** Since the M238 controller has been designed to continue operation during short transient power interruptions, it is necessary to wait for the complete shutdown of the controller. Please refer to the **AC Power Supply Specifications** section of the *Modicon M238 Logic Controller Hardware Guide* for detailed information. ➢ Power on the controller. ➢ Click on the **Login** button of the toolbar. ➢ Check that the value of `q_LastStopCause` is equal to `STOP_FROM_POWER_FAIL`.

Step	Action
9	**Reading the status of the boot project using the GetBootProjectStatus Function:** To test this function, proceed as follows: ➢ Check that the value of `q_BootProjectStatus` is equal to `VALID_BOOT_PROJECT`.
10	**Reading the short-circuit (or overload) diagnostic on the controller's embedded outputs using the GetShortCutStatus Function:** To test this function, proceed as follows: ➢ Check that the value of `q_wShortCutStatus` is equal to 0: bits 0, 1, and 2 of this WORD are equal to FALSE because there is no short-circuit on the controller's embedded outputs. **Note:** Please refer to the **Regular Output Wiring Diagram** (or **Transistor Output Wiring Diagram**) and **Short-circuit or Over-current on Outputs** sections of the *Modicon M238 Logic Controller Hardware Guide*. These sections contain several hazard messages and instructions that you must read concerning the M238 controller's protections against short-circuits and overloads.
11	**Reading the status of the controller's expansion bus using the GetRightBusStatus Function:** To test this function for a configuration diagnostic of the expansion bus (`RIGHTBUS_GET_GEN_STATUS`), proceed as follows: ➢ Check the value of `q_wRightBusStatusGen`: it is equal to 0 (bits 1 to 7 are equal to FALSE) because there is no expansion bus configuration mismatch. **Note:** In the case of this example, no TM2 module has been configured in SoMachine and no TM2 module has been physically installed on the controller. To test this function for a diagnostic of the first (analog) expansion module (`RIGHTBUS_GET_STATUS1`), proceed as follows: ➢ Check the value of `q_wRightBusStatus`: it is equal to 0 because no diagnostic is available for this analog expansion module. **Note:** In the case of this example, no analog module has been physically installed on the controller.
12	**Detecting errors on the controller I/O using the GetPlcFault Function:** To test this function, proceed as follows: ➢ Check the value of `v_xErrorIOExpansionBus` (bit 0 of `q_wPlcFault`): it is equal to FALSE because the I/O expansion bus is healthy. ➢ Check the value of `v_xErrorEmbeddedIO` (bit 1 of `q_wPlcFault`): it is equal to FALSE because the embedded I/O of the controller are healthy. **Note:** As pointed out in the SoMachine online help, these two data could be used as conditions for calling the **GetRightBusStatus** and the **GetLocalIOStatus** Functions. Please refer to the SoMachine online help: search for **GetPlcFault**.
13	**Reading the status of the controller's embedded I/O using the GetLocalIOStatus Function:** To test this function, proceed as follows: ➢ Check the value of `q_IOStatus`: it is equal to `LOCAL_IO_OK` because the inputs / outputs of the controller are operational.