Modicon M340 Discovery Kit
Start Up Guide

June 2009 eng

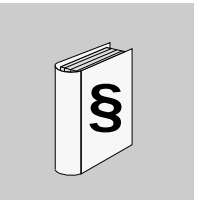
35015342 01

Schneider Electric

www.schneider-electric.com

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, serious injury, or equipment damage.

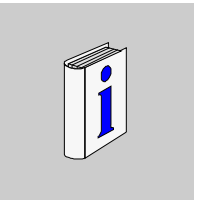
⚠ CAUTION

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

About the Book



At a Glance

Document Scope

This manual describes the hardware and software installation of the Modicon M340 discovery kit.

Validity Note

The data and illustrations found in this documentation are not contractually binding. Schneider Electric reserves the right to modify its products in line with its policy of continuous product development.
The information in this document is subject to change without notice and should not be considered as a commitment by Schneider Electric.

Product Related Warnings

Schneider Electric assumes no responsibility for any errors that may appear in this document. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.
No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying, without express written permission of Schneider Electric.
All pertinent state, regional, and local safety regulations must be observed when installing and using this product.
For reasons of safety and to ensure compliance with documented system data, only the manufacturer should perform repairs to components.
When controllers are used for applications with technical safety requirements, please follow the relevant instructions.
Failure to observe this product related warning can result in injury or equipment damage.

User Comments

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

Introduction

1

At a Glance

Subject of this Chapter

This chapter presents :

- The overall purpose of the discovery kit
- The summary of steps to be executed in order to learn about the hardware and software parts of the discovery kit
- The installation of the discovery kit

What's in this Chapter?

This chapter contains the following topics:

Purpose of Discovery Kit and Elements Provided

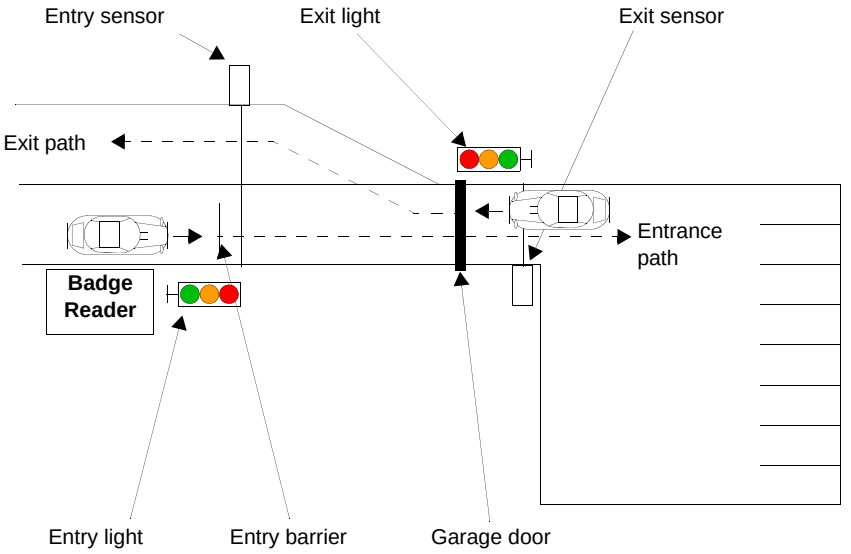
Purpose of Discovery Kit

The purpose of this kit is for discovering the possibilities of Unity Pro software associated with Modicon M340 PLCs.
The discovery kit is designed to be used by people who have never used Unity Pro software or Modicon M340 PLCs.
At different steps of this guide, you may refer to the Unity Pro documentation in order to have further information.
The discovery kit start up guide will make you discover certain Modicon M340 PLCs functions. For information about all the Modicon M340 PLCs' features, you will be able to see the directories *Modicon M340 Processors*, *Modicon M340 Expert Functions* and *Modicon M340 Communication*, in the Unity Pro documentation after you have installed and launched Unity Pro by pressing F1.
The following procedure describes the various discovery steps of this guide:

- Installation of the PLC configuration.
- Installation of the Unity Pro software.
- Test of the hardware configuration and Unity Pro software.
- Programmation of the application in Unity Pro.
- Debugging and simulation of the application.

Application Overview

The application example is a garage as follows:

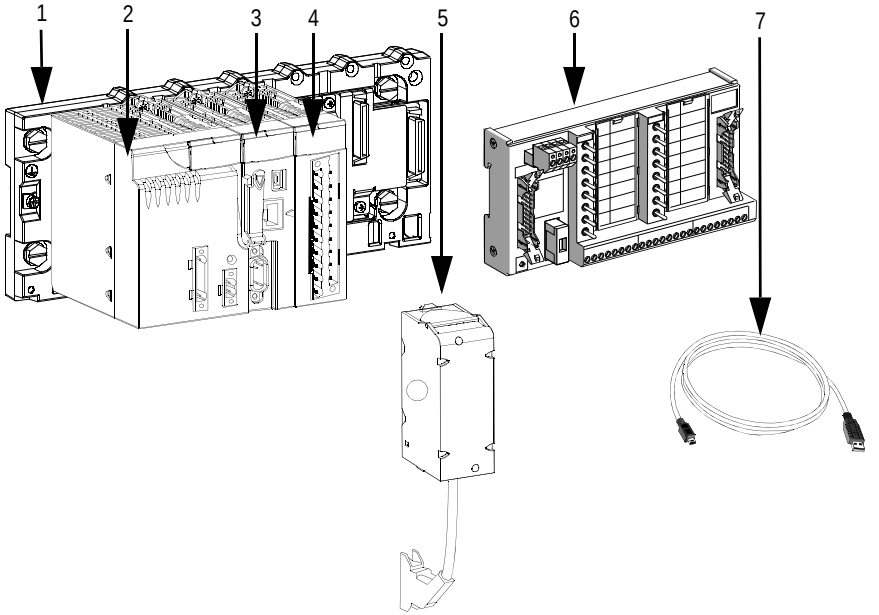


Provided Hardware Components

The discovery kit includes the following hardware components:

Label	Reference	Description
1	BMX XBP 0400	Rack with 4 slots.
2	BMX CPS 2000	Power supply.
3	BMX P34 2030	Processor including a memory card.
4	BMX DDM 16022	Discrete input/output module.
5	35014965_E53	Cable used to connect the discrete module BMX DDM 16022 to the TELEFAST module.
6	ABE7TES160	TELEFAST module.
7	BMX WCA USB H018	1.8-meter (6-foot) long USB cable.
	35014970_E53	2 wires (black and blue) to connect the TELEFAST module to the BMX CPS 2000 power supply module.

The illustration below shows the hardware configuration of the discovery kit:



Hardware Components Not Provided

The following hardware components are not included with the discovery kit but are required to install the PLC platform:

- One flat-tipped screwdriver with a maximum diameter of 3.5 mm. It will be used to mount the 20-pin terminal block (5) to the discrete module (4).
- One AWG 24 (0.34 sq.mm)...AWG 16 (1.5 sq.mm) and 79 inch (2 m) long wire that you will be able to split and strip, and an electrical plug. You will use it:
  - To connect the power supply module to the safety devices
  - To connect the safety devices to the alternating current network (power cable with an electrical plug at the extremity)
  - To ground the power supply module and the rack to the protective ground
- DZ5-CE type lugs can also be used but are optional because the power supply module's 5-pin terminal block can accomodate bare wires.
- An ohmmeter may be used to test the TELEFAST module's fuse. This test will be performed if the TELEFAST module's 24V LED will not be on when you will install and power up the PLC platform.
- A voltmeter may be used to test the input activated by the TELEFAST module. This test will be performed if the TELEFAST module does not activate one of the three discrete inputs of the application.

Provided Software Elements

The kit includes the following CD-ROMs:

- Unity Pro Small CD-ROM pack. This pack includes four CD-ROMs:
  - Unity Pro S V3.0 CD-ROM: This CD includes the Unity Pro V3.0 small version software. You will use the Unity Pro software to program and to simulate the PLC platform.
  - Driver Pack CD-ROM: This CD includes all the communication drivers. You will use the USB communication driver to connect your PC to the PLC platform.
  - Unity Loader CD-ROM: This CD includes the OSLoader software. The OSLoader software is used to upgrade/downgrade the operating systems of certain processors. You will not use this software for the discovery kit.
  - Unity Pro Documentation CD-ROM: This CD includes the Unity Pro online help. You will not use this CD for the discovery kit because the Unity Pro documentation will be installed on your PC along with the Unity Pro software.
- Modicon M340 design CD-ROM: This CD includes the Modicon M340 design software. This software guides you step by step through the design of a Modicon M340 PLC and reassures you as to the validity of your configuration. You will be able to use this software after implementing the discovery kit.
- Discovery Kit CD-ROM: This CD includes all the elements related to the discovery kit and the Modicon M340 PLCs.

The contents of the Discovery Kit CD-ROM is as follows:

- `Training guide` directory which consists of:
  - The learning guide `Discovery kit Start Up guide` to practice Unity Pro with Modicon M340 PLCs.
  - The subdirectory `Application files` which includes all the files to import when programming the discovery kit application.
- `Additionnal training` directory which consists of a learning guide `Hands on_Unity_Motion_CANopen`. This documentation guides you through setting up a Lexium 05 servodrive connected to a Modicon M340 processor's CANopen port.
- `Communication means` directory which consists of:
  - A Powerpoint presentation to discover the new PLC offer and key values.
  - Some brochures about: Modicon M340, Modicon M340 counting, Modicon M340 motion, winning association Modicon M340 Magelis, winning association Modicon M340 CANopen and winning association Modicon M340 Ethernet.
  - A Modicon M340 PLCs catalog.

Installing the Discovery Kit

At a Glance

You are going to perform the following steps:

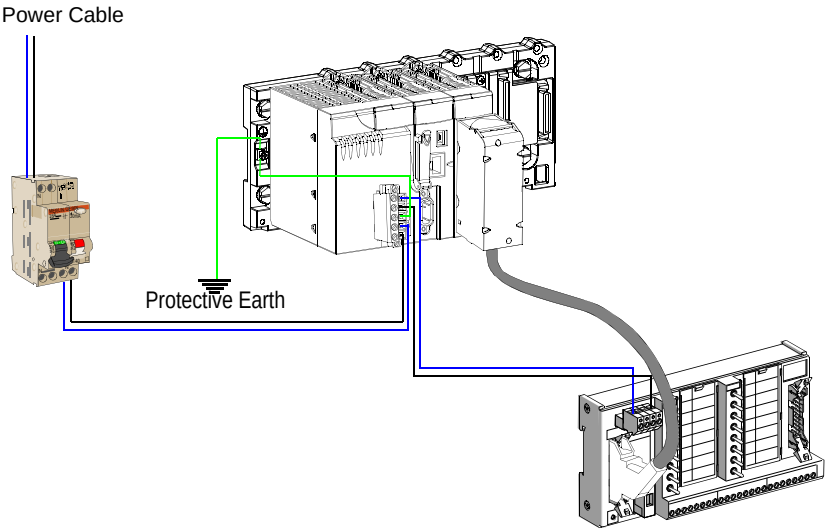
- Install the hardware part.
- Install Unity Pro software and communication drivers.
- Check the discovery kit configuration normally works.

Safety Rules for Connecting the Modules

	**DANGER** **HAZARD OF ELECTRIC SHOCK, EXPLOSION OR ARC FLASH** Do not connect the power supply module's power cable until the connection process, described in the following pages, is accomplished. **Failure to follow this instruction will result in death or serious injury.**

Installed Hardware Part

Once the entire connecting process will finish, the PLC platform will be connected as follows:

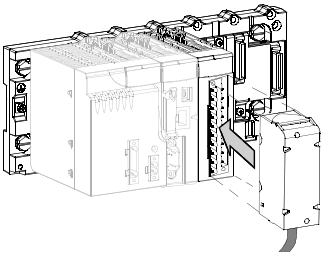


To perform the connecting process, follow the various steps in the order that they are described in the following pages.

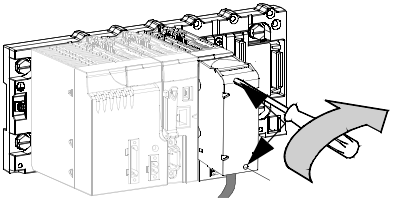
First Step: Connecting the TELEFAST Module to the Discrete Module

First, connect the TELEFAST module to the discrete module:

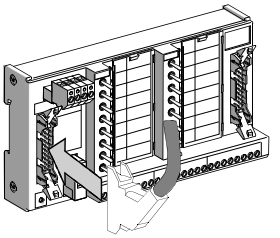
1. Insert the terminal block encoder into the discrete module encoder



2. Fix the terminal block to the discrete module by tightening the 2 mounting screws



3. Insert the HE10 connector into the TELEFAST module's connector

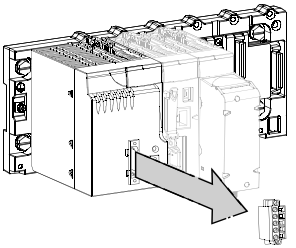


Second Step: Connecting TELEFAST Module to the Power Supply Module

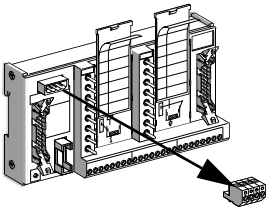
Note: To connect the TELEFAST module to the power supply module, we recommend to use the black wire to connect 0 Vdc and the blue wire to connect 24 Vdc.

Connect the TELEFAST module to the power supply module:

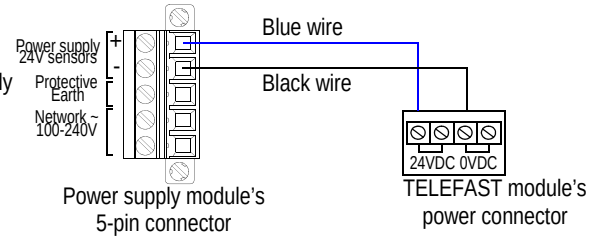
1 Disconnect the connector from the power supply module



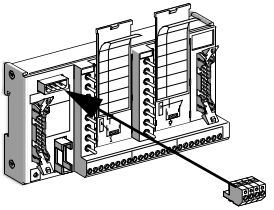
2 Disconnect the power connector from the TELEFAST module



3 Connect the power supply module's connector to TELEFAST module's power connector



4 Connect the power connector to the TELEFAST module



Third Step: Installing Safety Devices

You must install a safety device at the start of the line on the power supply network. The safety device should include the following elements:

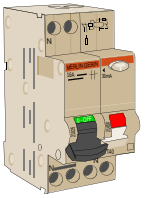
- Circuit breaker
- Fuse

The following table shows the characteristics of BMX CPS 2000 power supply module in order to help you to choose the safety devices:

Characteristic		**Current**
Nominal input current Irms	at 115 Vcc	0.61 A
	at 230 Vcc	0.31 A
Inrush current I (1)	at 115 Vcc	<30 A
	at 230 Vcc	<60 A
Current characteristic It	at 115 Vcc	0.03 As
	at 230 Vcc	0.06 As
Current characteristic I^2t	at 115 Vcc	0.5 A^2s
	at 230 Vcc	2 A^2s

(1) Values at initial power-up and at 25°C (77°F).

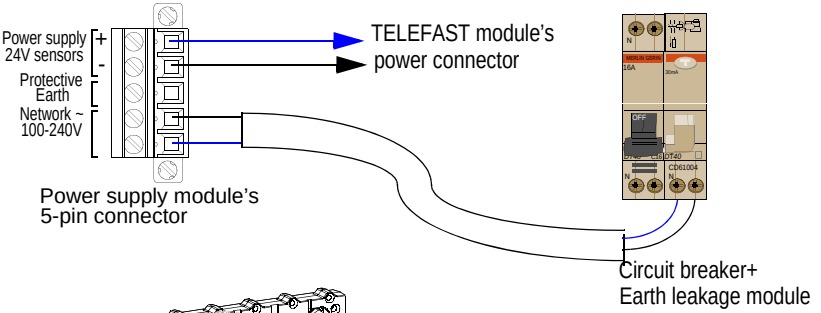
We recommend to use Merlin Gerin safety devices as a C16 16 A type circuit breaker mounted to a C16 Vigi 30 mA type ground leakage module:



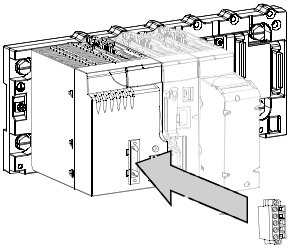
Fourth Step: Connecting the 5-Pin Terminal Block to the Alternating Current Network

Connect the 5-pin terminal block to the alternating current network:

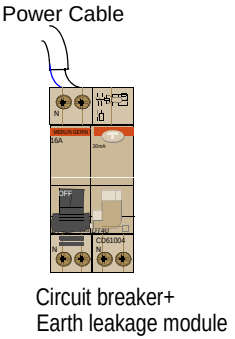
1 Connect the power supply module's connector to the circuit breaker or the earth leakage module



2 Connect the 5-pin terminal block to the power supply module



3 Connect the circuit breaker to the power cable

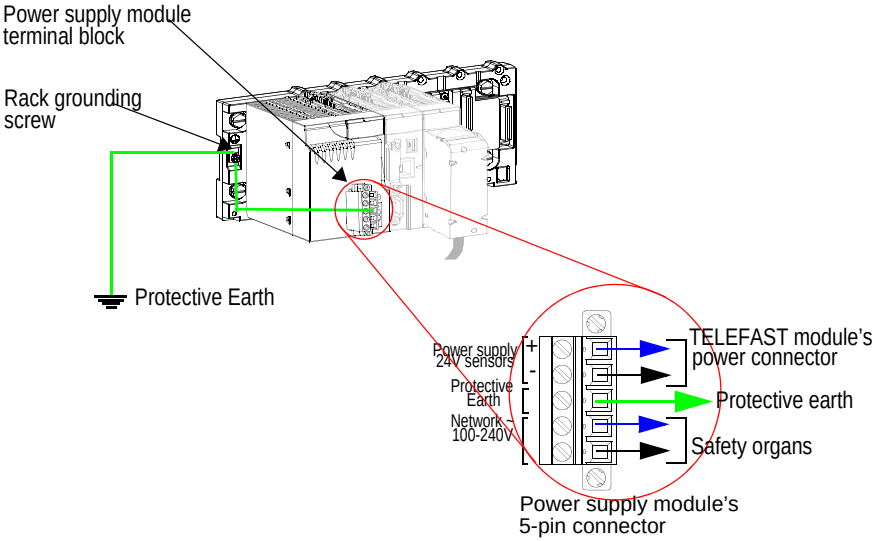


Note: The diagram above describes the connection to a C16 16 A type circuit breaker mounted to a C16 Vigi 30 mA type ground leakage module. If you use another circuit breaker and/or another ground leakage module, read the instruction sheets of these modules before connecting it.

Fifth Step: Grounding the Rack and the Power Supply Module

	DANGER
	HAZARD OF ELECTRIC SHOCK Power supply module must be grounded. Do not connect anything else to the power supply ground. **Failure to follow this instruction will result in death or serious injury.**

Connect the power supply module terminal block to the rack grounding screw and the rack grounding screw to the protective earth of the installation:



Note: The rack grounding screw is used to connect two cables (1.5 to 2.5 mm^2 or in AWG size, 16 to 13).

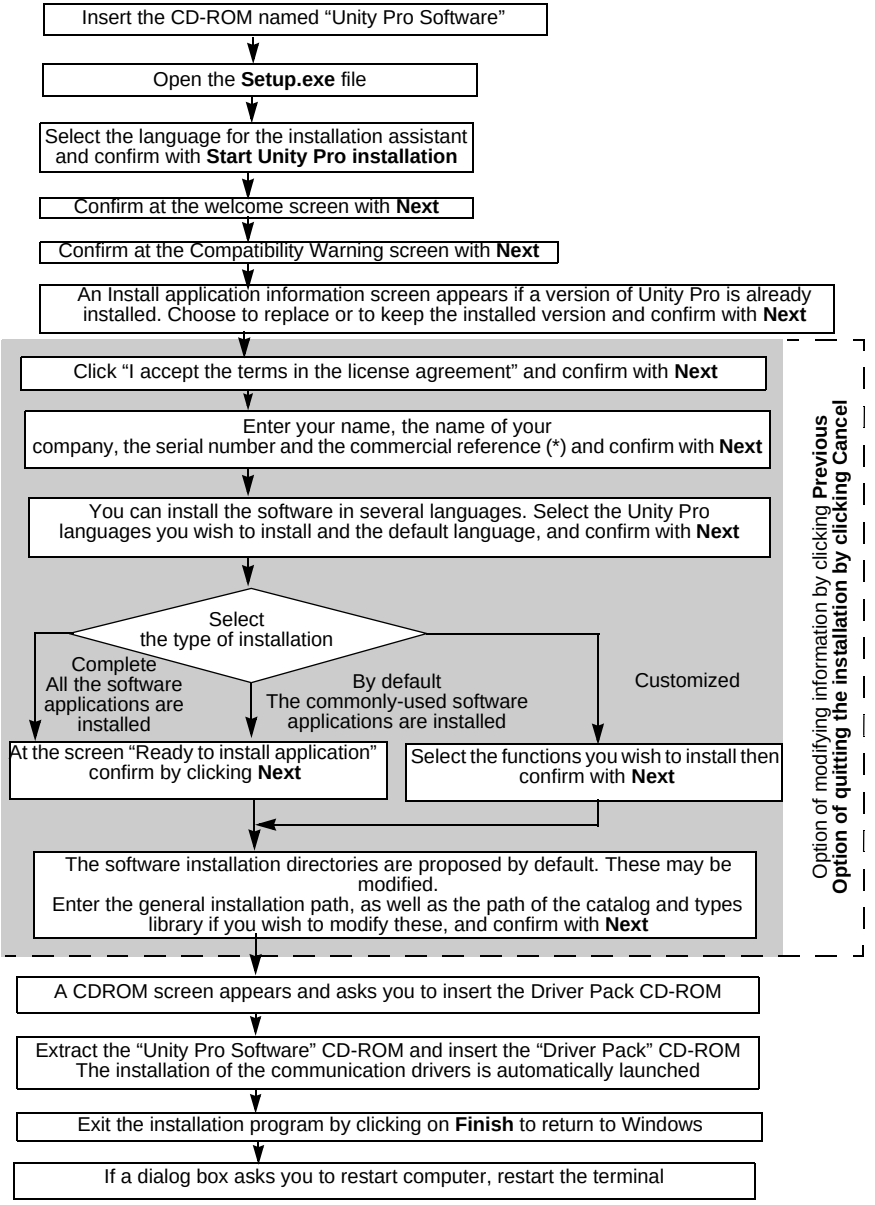
The hardware part installation is now complete. The next step is to install Unity Pro software on the terminal.

Installing Unity Pro

Note: Do not install Unity Pro available on the CD if Unity Pro version 3.0 S, M, L or XL is already installed because it would replace the installed version of Unity Pro with Unity Pro 3.0 S. Install Unity Pro 3.0 S available on CD only if Unity Pro is not installed or a version lower than 3.0 is installed.

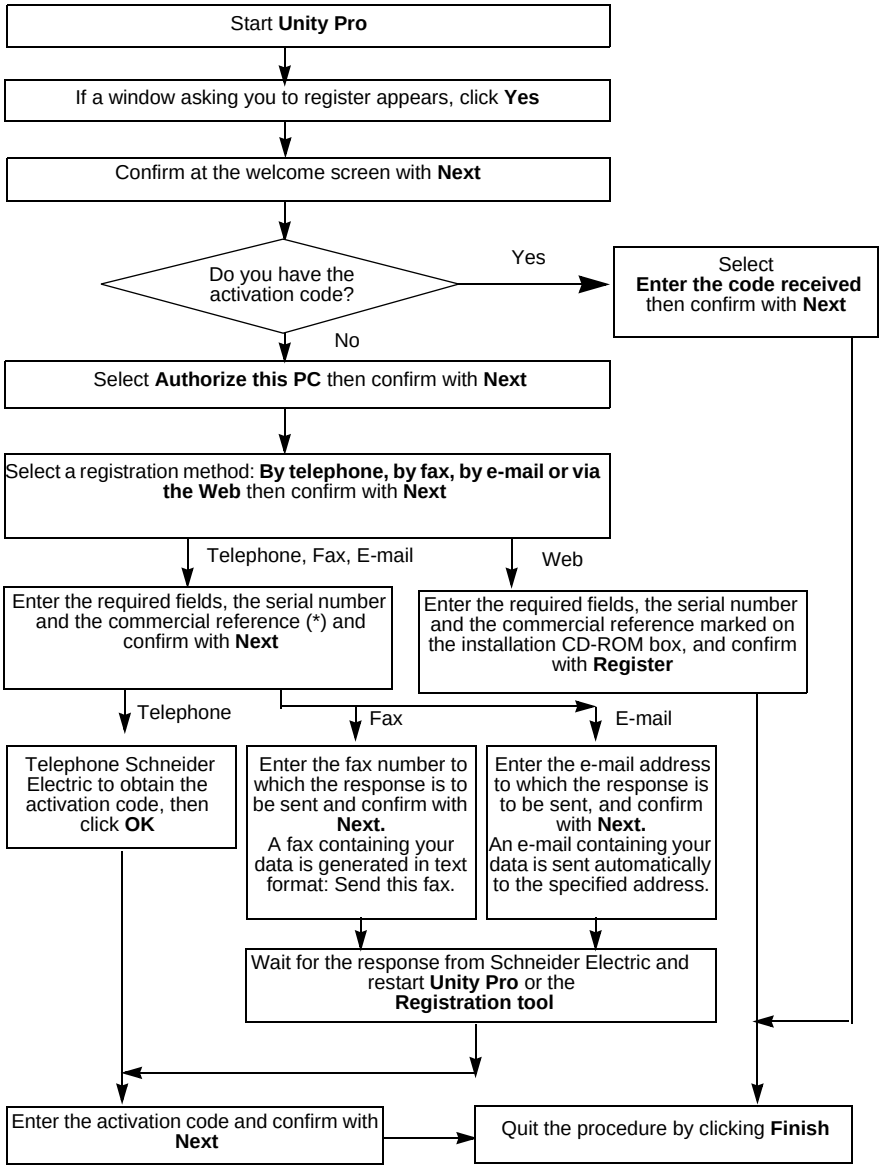
Note: The time needed for installation differs according to your PC's configuration.

Install Unity Pro software by using the following procedure:



(*) These numbers are marked on the label inside the box containing the software CD-ROMs.

When you are installing Unity Pro or once you have installed Unity Pro, register Unity Pro in order to use it for an undetermined amount of time:



(*) These numbers are marked on the label inside the box containing the software CD-ROMs.

Checking the Discovery Kit Configuration Normally Works

Once the entire connecting process described above has finished and you have installed Unity Pro, check the entire configuration normally works:

- Check the PLC platform normally works.
- Check the terminal can be connected to the PLC platform.

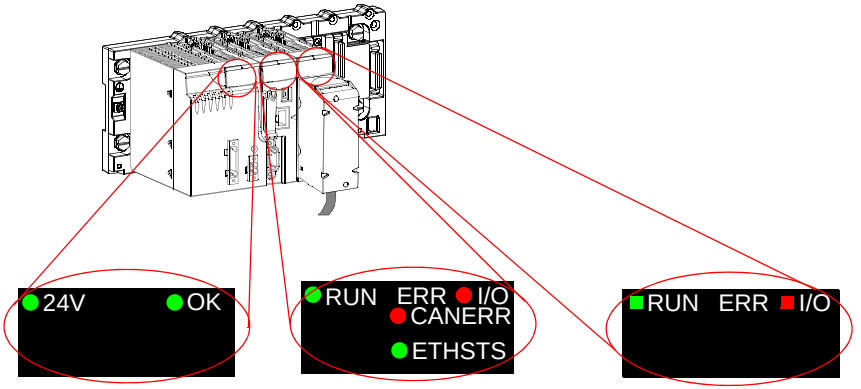
To check the PLC platform normally works, perform the following steps:

- Turn on the PLC platform.
- Check the first three inputs are simulated by the TELEFAST module.

To turn on the PLC platform:

- Connect the power cable to the alternating current network.
- Turn on the general isolator and the safety devices (e.g. the circuit breaker).

As a result, the display panel of the following modules must be of the same status than described below:



Check status of the following LEDs:

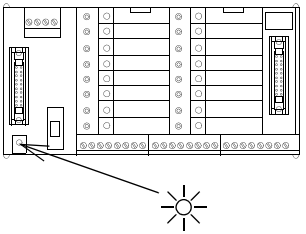
- 24V LED of the power supply module is on.
- OK LED of the power supply module is on.
- RUN LED of the processor is on or is flashing (on in RUN mode or flashing in STOP mode).
- I/O LED of the processor is on.
- CAN ERR LED of the processor is on because no device is connected to the CANopen port.
- ETH STS LED of the processor is flashing because no device is connected to the Ethernet port.
- RUN LED of the discrete module is on.
- I/O LED of the discrete module is on.

If the power supply module OK or/and 24V LEDs do not comply with the LEDs status described above, see the troubleshooting chapter (see *Discovery Kit Troubleshooting, p. 105*).

Note: For the processor and discrete module I/O LEDs, the troubleshooting chapter describes how to disable the discrete module supply monitoring. You must perform this procedure later, when associating the inputs and the outputs to the discrete module.

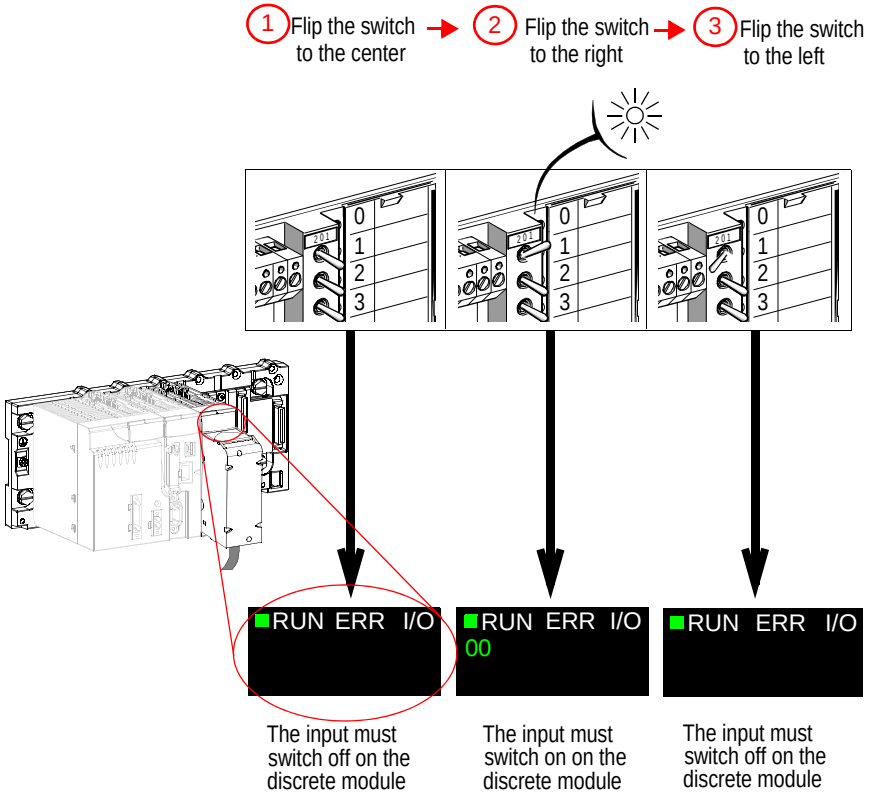
Note: All the discrete module's LEDs corresponding to the discrete inputs/outputs (marked 00, 01...15) are flashing when you power up the PLC platform and stop flashing after few seconds.

For further information about the power supply module LEDs, see Modicon M340 Processors/Modicon M340 Hardware/BMX CPS xxxx Power Supply Modules/BMX CPS xxxx Power Supply Module Diagnostics/BMX CPS xxxx Power Supply Modules Display, in the Unity Pro documentation.
For further information about the processor LEDs, see Modicon M340 Processors/ Modicon M340 Hardware/BMX P34 xxxx Processors/BMX P34 xxxx Processors Diagnostics/Display, in the Unity Pro documentation.
As a result, on the TELEFAST module, 24V LED must switch on:



If the TELEFAST module's LED 24V is not switched on, see the troubleshooting chapter (see *Discovery Kit Troubleshooting, p. 105*) to solve the problem.

Check the first three inputs are simulated by the TELEFAST module by viewing the display panel of the discrete module:



Note: The first three inputs are marked 0, 1 and 2 on the TELEFAST module and 00, 01, 02 on the display panel of the discrete module.
If the discrete inputs do not switch on, see the troubleshooting chapter (see *Discovery Kit Troubleshooting, p. 105*) to solve the problem.

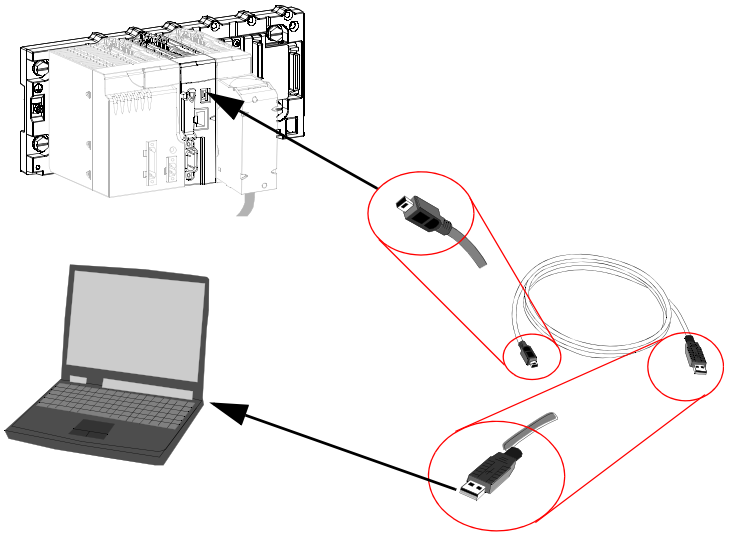
The hardware part is checked. The next step is to check the terminal can be connected to the PLC platform.
To do this, perform the following steps:

- Connect the terminal to the PLC platform by using the USB cable.
- Launch Unity Pro.
- Set the PLC address.
- Connect the terminal to the PLC.

First of all, connect the terminal to the BMX P34 2030 processor by using the provided BMX XCA USB 018 cable:

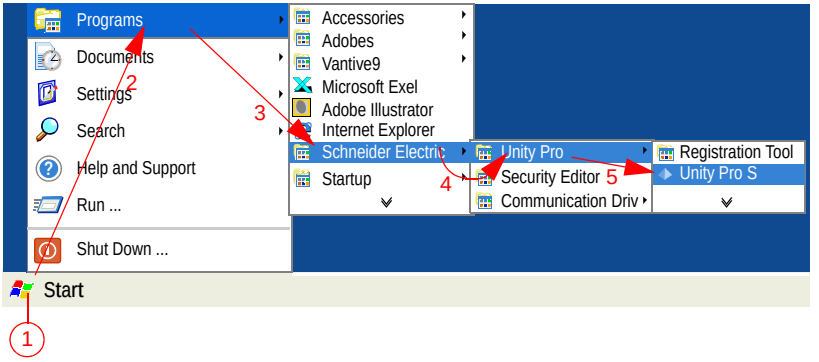
1 Connect the mini B USB port to the processor's USB port

2 Connect the A USB port to the terminal's USB port

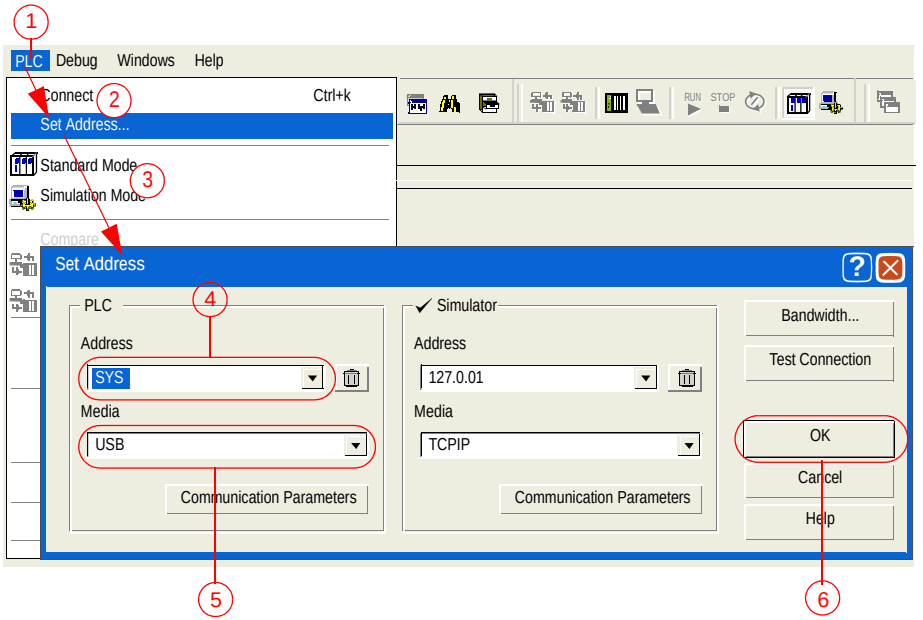


Launch Unity Pro. To do this, if a Unity Pro shortcut exists double click it:

If no Unity Pro shortcut exists, click on the menu elements in the following order:

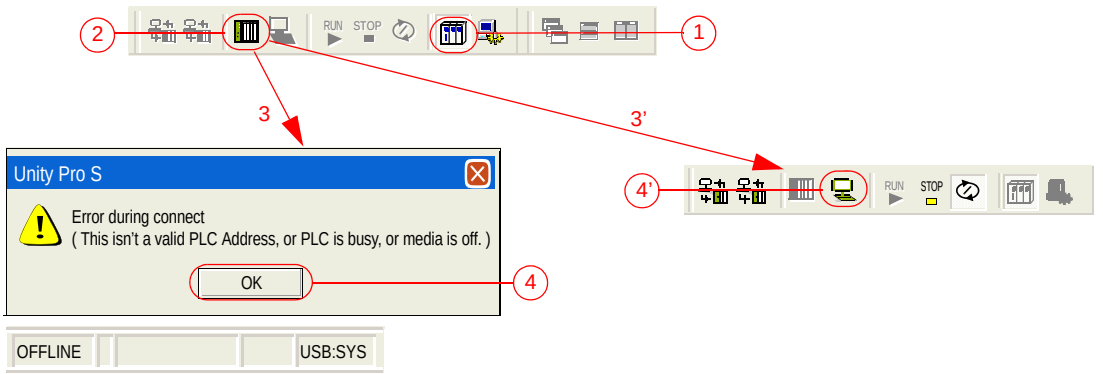


Once Unity Pro is launched, set the PLC address:



1- Click on PLC in the menu bar
2- Click on Set Address
3- The Set Address screen appears
4- Enter SYS in the Address field of the PLC zone
5- Select USB in the Media field of the PLC zone
5- Confirm the settings by clicking on OK

The PLC address is set. Now, connect the terminal to the PLC platform:



There are 2 modes, simulator mode (use of the PLC simulator) and standard mode (use of the PLC processor)
1- Click on the icon Standard mode in the toolbar to select the standard mode (corresponding to the real PLC)
2- Click on the icon Connect
3- If the terminal cannot connect to the PLC, the error screen appears and OFFLINE is written in the status bar.
4- Click on the button OK to acknowledge the error message.
See the troubleshooting chapter to solve the problem.

3'- If the terminal can connect to the PLC, no error message appears and the terminal is connected to the PLC
4'- Click on the icon Disconnect to disconnect the terminal to the PLC.
The purpose is to program the application in offline mode.

Modicon M340 Discovery Kit Application

2

At a Glance

At a Glance

This chapter deals with the implementation of the discovery kit application in the Unity Pro software.

Note: To program the discovery kit application, the Unity Pro software must be in offline mode because some functions, dialog boxes, menus etc. are not available in online mode. To set the offline mode, click on the icon Disconnect if the icon
appears in color. You will set the online mode when tranferring the discovery kit application from the terminal to the PLC.

Note: At any moment while programming the application example, you can save your project by using the command **File** → **Save** or by pressing CTRL+S. Then you can save your project as an STU file.
To open your saved project, use the command **File** → **Open** or press CTRL+O then open the STU file.

What's in this Chapter?

This chapter contains the following topics:

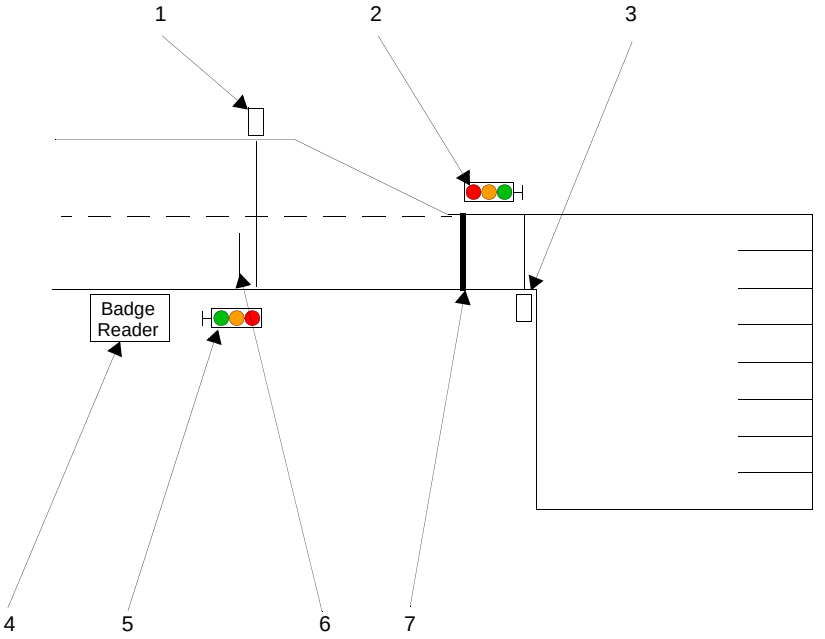
Application Operating Modes

At a Glance

The application example is the programmation of a garage. The following pages describe all the application operating modes.

Garage Overview

Following is an illustration of the main elements of the garage:



The following table describes the main elements of the garage:

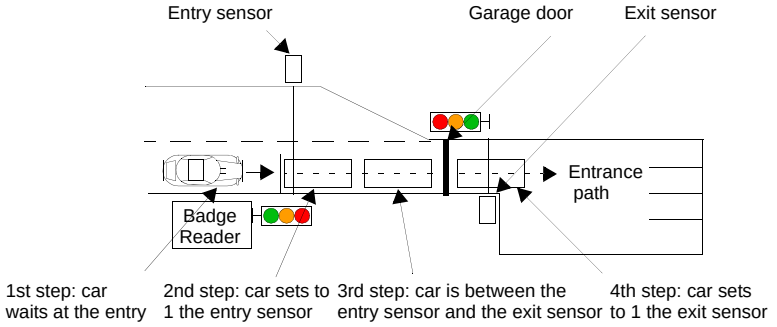
Label	Description
1	Entry sensor
2	Exit light
3	Exit sensor
4	Badge reader
5	Entry light
6	Entry barrier
7	Garage door

Car Coming Into the Garage

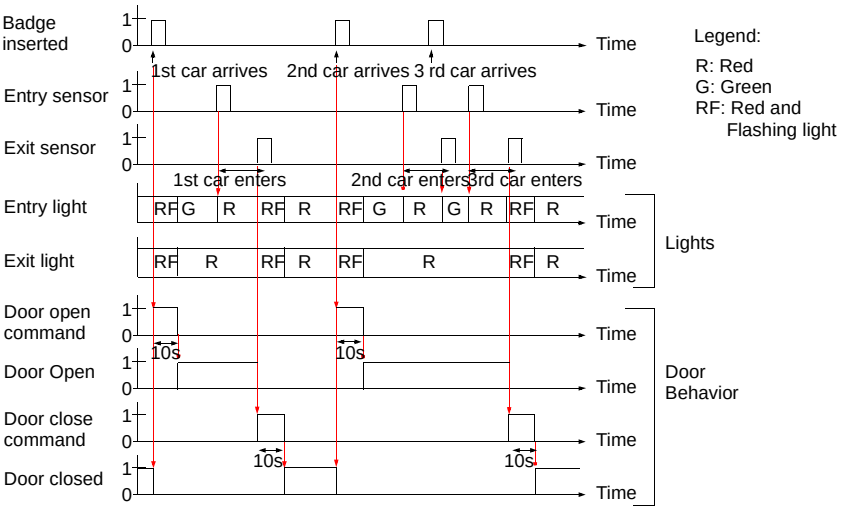
The following sequence takes place when a car wants to come into the garage:

- The car waiting at the entry inserts a badge into the badge reader (4).
- The door (7) opens. The entry light (5) remains red and is flashing.
- Once the door (7) is open, the entry light (5) turns green and the entry barrier (6) is open.
- The car activates the entry sensor (3) and the entry light (5) turns red.
- The car goes under the door (7).
- The car activates the exit sensor (3). The door (7) closes (if no other car is waiting at the entry). The entry light (5) remains red and is flashing.
- The door (7) is closed.

Following is an illustration of a car which comes into the garage:



The following timing diagram shows 2 cars which come into the garage and a third one which arrives while the second car comes into the garage:

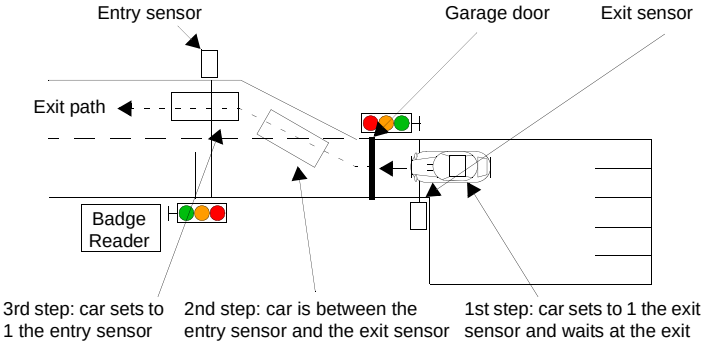


Car Coming Out of the Garage

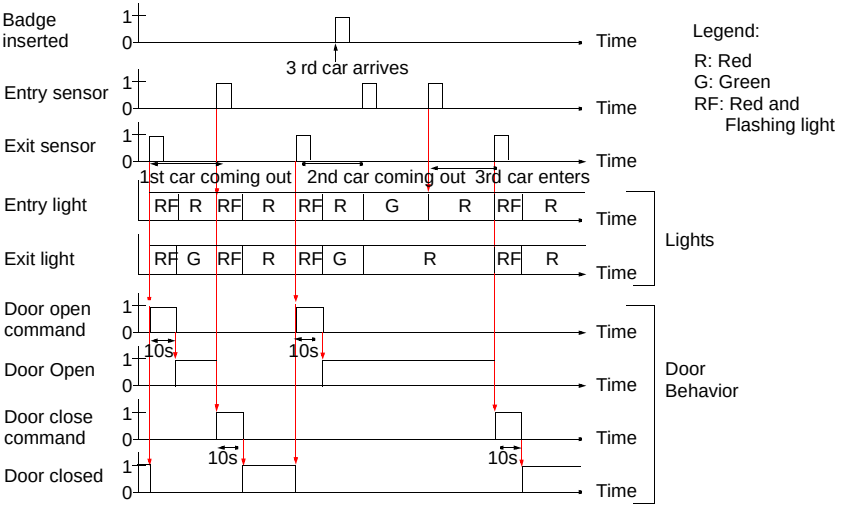
The following sequence takes place when a car wants to come out of the garage:

- The car waiting at the exit activates the exit sensor (3).
- The door (7) opens. The exit light (2) remains red and is flashing.
- Once the door(7) is open, the exit light (2) turns green.
- The car goes under the door (7).
- The car activates the entry sensor (1). The exit light (2) turns red.
- The door (7) closes (if no other car is waiting at the exit and at the entry). The exit light (5) remains red and is flashing.
- The door (7) is closed.

Following is an illustration of a car which comes into the garage:



The following timing diagram shows 2 cars which come into the garage and a third one which arrives while the second car comes into the garage:



Cars Waiting At Both the Entry and Exit

When a car is waiting at the entry and another car is waiting at the exit, the system makes all the cars waiting at the exit come out of the garage. The car at the entry barrier can enter only when there are no other cars exiting the garage. The timing diagram above illustrates this case.

Sensors: Door Open and Door Closed

The "door open" sensor is set to 1 if the door open command is on for 10 seconds. The "door closed" sensor is set to 1 if the door close command is on for 10 seconds.

Note: These 2 sensors are simulated by the user program. In reality, the sensors must not be simulated for safety reasons, and must be physical inputs which are set to 1 or 0 by sensors.

Car Counter

The car counter behaves as follows:

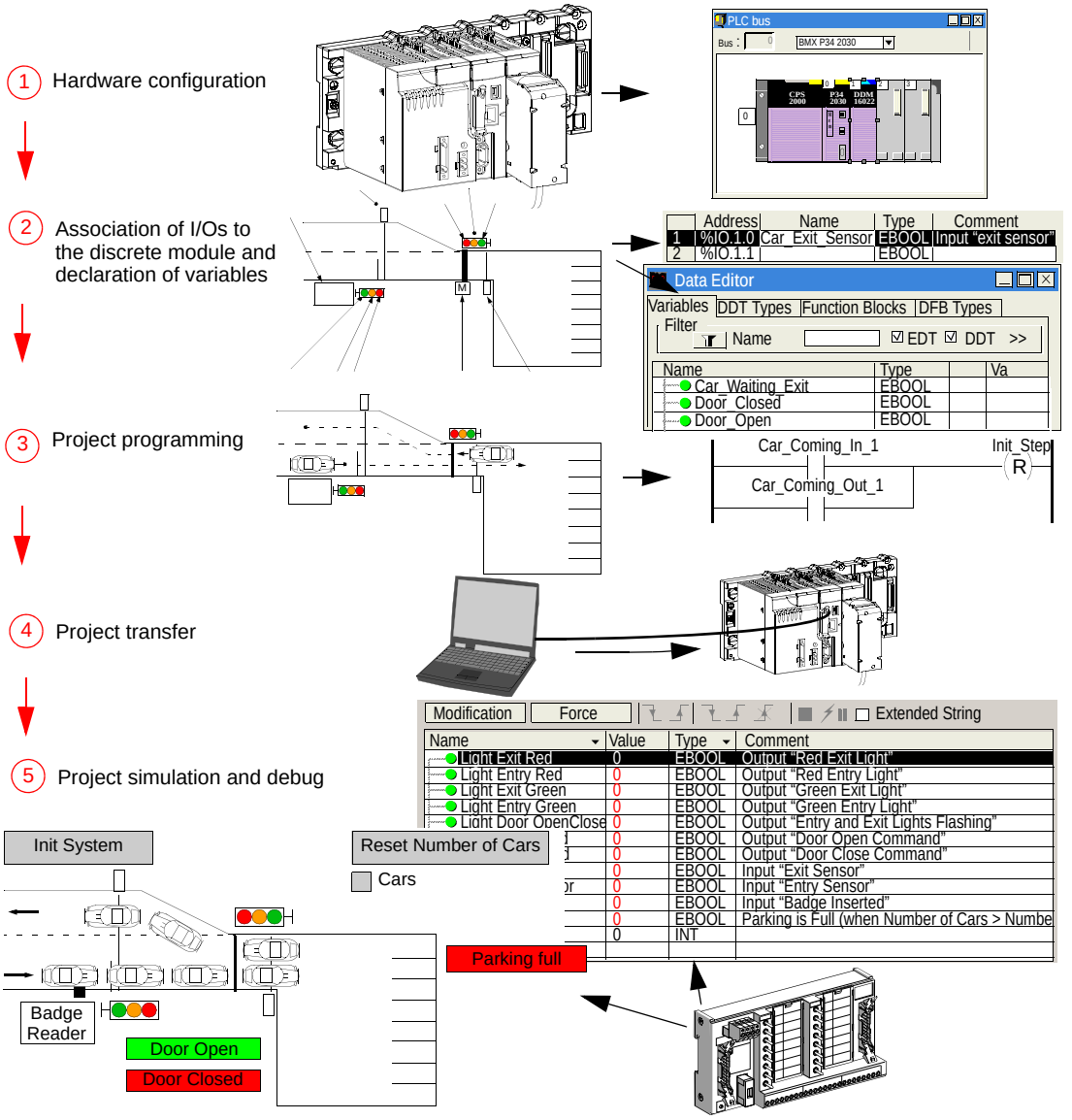
- Counting: A car has entered the garage.
- Downcounting: A car has exited the garage.
- Reset to 0: The user pushed the "Reset Number of Cars" button, which is present in the operator screen.

The garage is full when the number of cars is greater than or equal to 20.

Programming Methodology

Programming Methodology

The implementation of the application example will be performed as follows:



Creating the Project and Configuring its Hardware in Unity Pro

Introduction

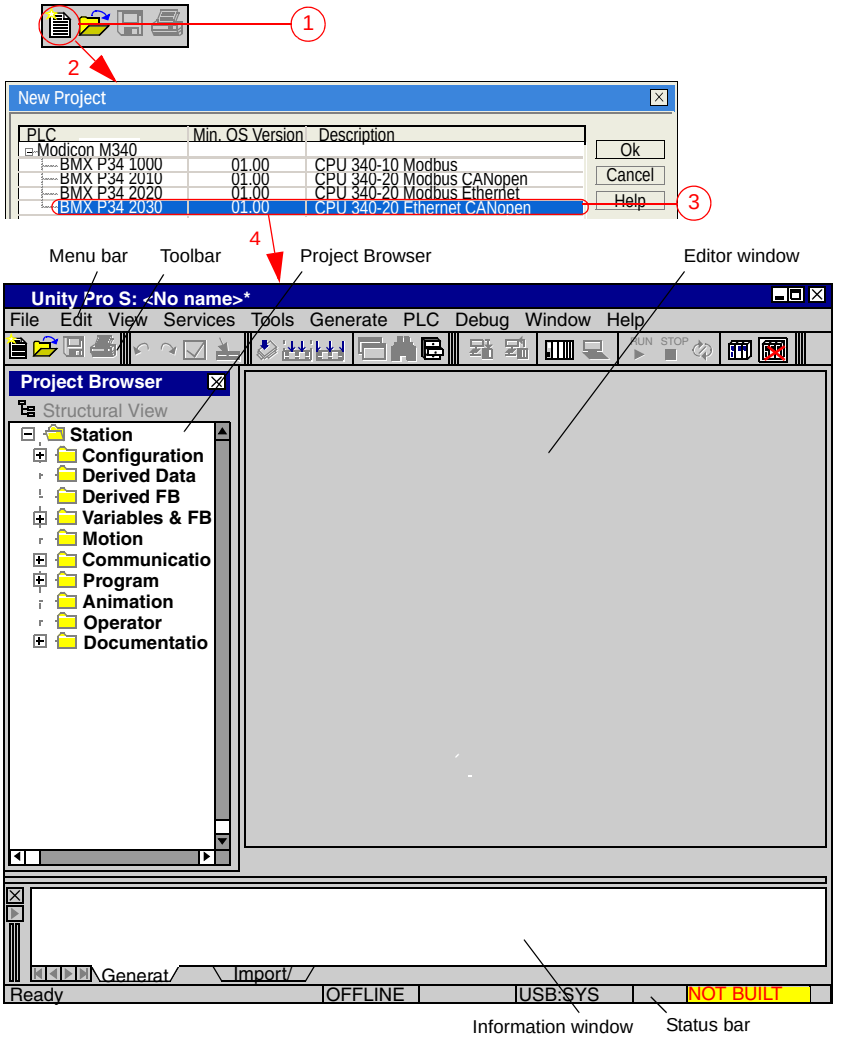
You are going to perform the following 2 steps:

- Create the project in Unity Pro.
- Implement the hardware configuration in Unity Pro.

These 2 steps are necessary for every Unity Pro application.

Creating the Project

Create the project:



1- Click on the New Project icon in the menu bar
2- The New Project screen appears
3- Double click the processor BMX P34 2030
4- The Unity Pro user interface appears

Further Information About the User Interface

At the various programming steps, you will use the following directories of the project browser:

- **Configuration**, to configure and access the hardware configuration in Unity Pro.
- **Derived FB Types**, to program the DFB type `Door_Management`.
- **Variables & FB instances**, to declare all the variables of the discovery kit application.
- **Program**, to program the section `Garage_Management` in LD language and the section `Car_Counting` in ST language.
- **Animation Tables**, to program the animation table `Animation`.
- **Operator Screens**, to program the operator screen `Garage_Control`.

For further information about the Unity Pro user interface, see *Unity Pro Software/Operating Modes/Appendix/User Interface*, in the Unity Pro documentation.

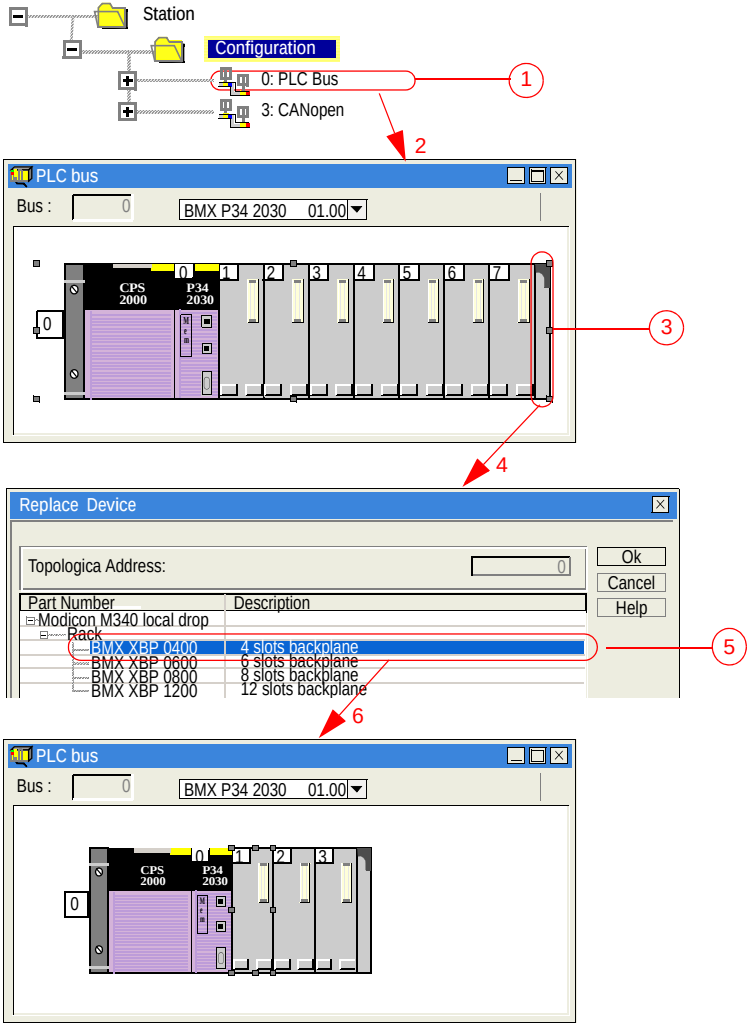
Accessing the Unity Pro Documentation

At different steps of this guide, you may refer to the Unity Pro documentation in order to have further information.
To access the Unity Pro documentation, click on the icon Help in the menu bar or press F1:

Then the Unity Pro help appears and you can browse the path which is given to you (*Unity Pro Software/Operating Modes/Appendix/User Interface*, as above, for instance).

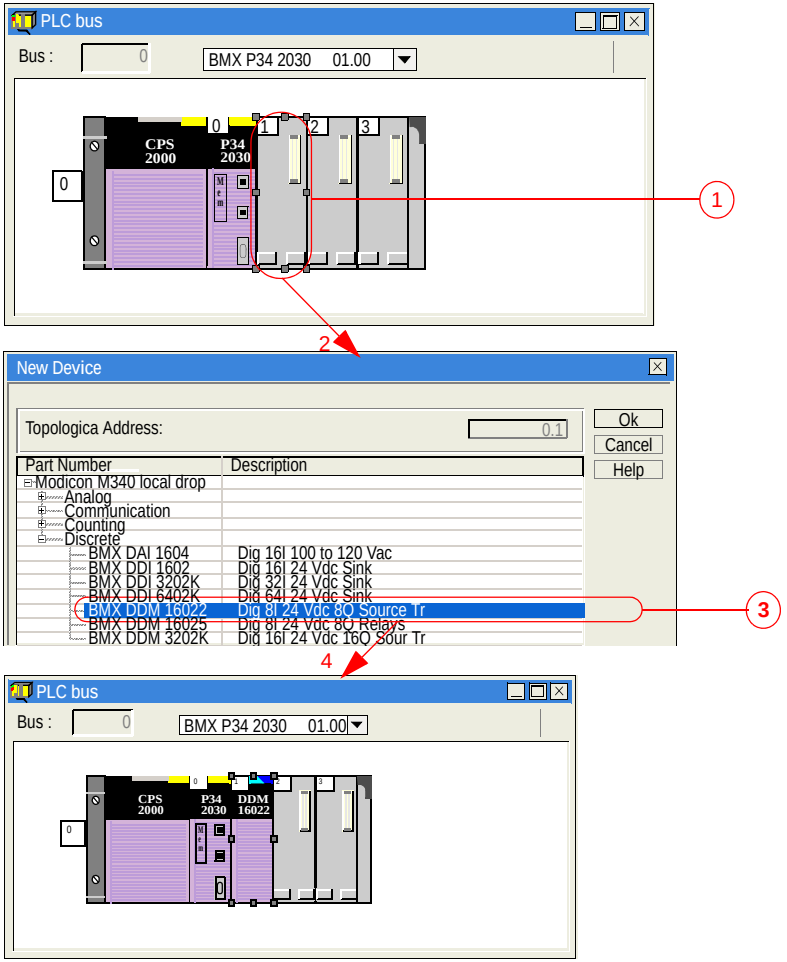
Configuring the Hardware Configuration

You have already configured the processor BMX P34 2030 when you have created the project. Now, you have to replace the rack BMX XBP 0800 with a rack BMX XBP 0400:



1- Double click the PLC Bus directory in the project browser
2- The bus editor is displayed
3- Double click the rack
4- The Replace Device screen is displayed
5- Double click the rack BMX XBP 0400
6- The rack BMX XBP 0400 is configured

The processor BMX P34 2030 and the rack BMX XBP 0400 are configured. Now, you have to insert the discrete module BMX DDM 16022:



1- Double click the slot 1 of the rack
2- The New Device screen is displayed
3- Double click the module BMX DDM 16022
4- The module BMX DDM 16022 is configured

Result: The project is created and the entire hardware configuration is implemented in Unity Pro.

Other Ways to Insert a Module in the Hardware Configuration

For other ways to insert a module in the hardware configuration, see *Unity Pro Software/Operating Modes/Project Configuration/Bus Editors/Configuration of the modules* in the PLC station, in the Unity Pro documentation.

Associating the Inputs and the Outputs to the Discrete Module

Introduction

The inputs/outputs are the logic images of the electrical states of the discrete module's inputs/outputs.
When you will program the sections in LD, ST and FBD languages you will need all the discrete inputs/outputs. It is the reason why you must declare all the inputs and the outputs before to program the application.

Implementation Methodology

You may declare the inputs/outputs by using one of the following procedures:

- Declare the inputs/outputs by using the discrete module's configuration screen.
- Declare the inputs/outputs by using the data editor.
- Import all the inputs/outputs of the discovery kit application. Declare the first input `Badge_Inserted` before importing all the inputs/outputs.

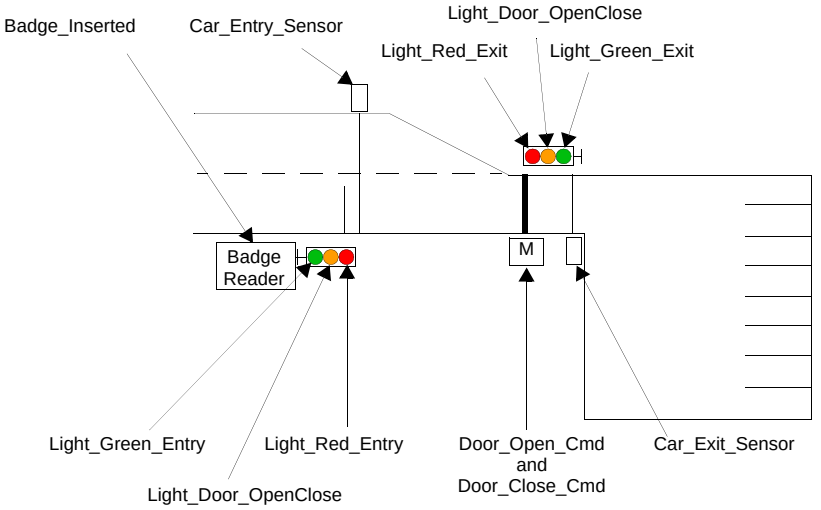
The following pages only describe how to declare all the inputs/outputs by using the discrete module's configuration screen and do not describe how to declare all the inputs/outputs by using the data editor.

List of Discrete Inputs/Outputs

The following table presents all the discrete inputs/outputs to implement:

Address	Name	Comment
%I0.1.0	Badge_Inserted	Input "badge inserted"
%I0.1.1	Car_Entry_Sensor	Input "entry sensor"
%I0.1.2	Car_Exit_Sensor	Input "exit sensor"
%Q0.1.16	Light_Door_OpenClose	Output "entry and exit lights flashing"
%Q0.1.17	Light_Red_Exit	Output "exit light red"
%Q0.1.18	Light_Green_Exit	Output "exit light green"
%Q0.1.19	Light_Red_Entry	Output "entry light red"
%Q0.1.20	Light_Green_Entry	Output "entry light green"
%Q0.1.21	Door_Open_Cmd	Output "Door Open Command"
%Q0.1.22	Door_Close_Cmd	Output "Door Close Command"

The following diagram shows the location of inputs/outputs in the garage:

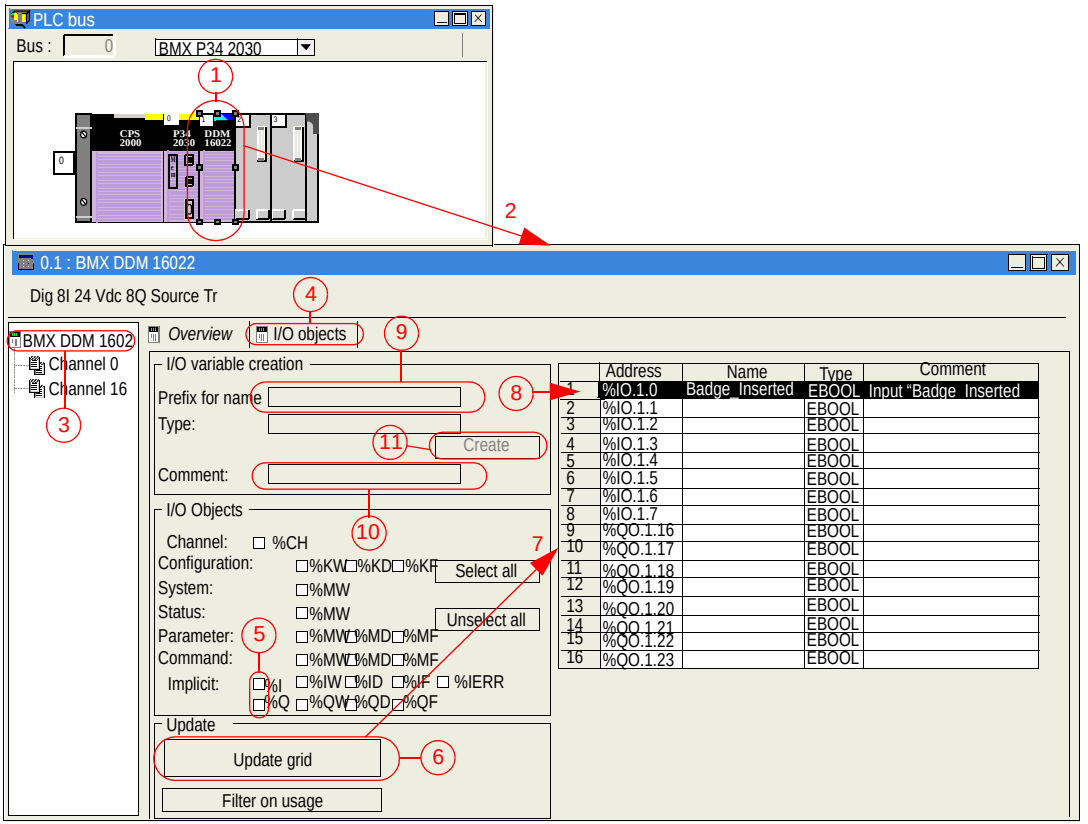


The topological addresses of inputs are of %I0.1.c-type and the topological addresses of outputs are of %Q0.1.c-type:

- 0 is the rack number. The rack number is always 0 when the PLC configuration consists of one rack.
- 1 is the module number. The reason is the discrete module is connected to the slot 1 of the rack.
- c represents the channel number. For this discrete module, the channel number of inputs is 0...7 range and the channel number of outputs is 16...23 range.

Declaring a Discrete Input

Associate the discrete input `Badge_Inserted` to the first channel of the discrete module:



1- Double click the discrete module BMX DDM 16022 in the bus editor
2- The BMX DDM 16022 configuration screen appears
3- Click on the BMX DDM 16022 node
4- Click on the I/O objects tab
5- Activate the %I and %Q checkboxes to list I/O objects only
6- Click on the button Update grid
7- The list of all the inputs/outputs is displayed
8- Select the line corresponding to the input %I0.1.0
9- Enter “Badge_Inserted” in the Prefix for name field
10- Enter “Input “badge inserted”” in the Comment field
11- Click on the button Create
Result: The input Badge_Inserted is implemented.

Once you declare the input `Badge_Inserted`, you can either declare all the inputs/outputs one by one or import all the inputs/outputs of the discovery kit application (see *Importing the Variables, p. 116*).
For declaring all the inputs/outputs one by one, repeat the steps 8 to 11 of the procedure above.

How to Disable the Discrete Module Supply Monitoring

When powering the PLC platform, the processor and discrete module I/O LEDs are switched on. The reason is the checkbox **Supply monitoring** is checked by default for the channel group 16 of the discrete module. However, there is no external power supply for this channel group which corresponds to the discrete outputs. Therefore, you have to disable the supply monitoring for the discrete channel group 16 (see *How to Disable the Discrete Module Supply Monitoring, p. 111*).

Note: We recommend to disable the supply monitoring at this step because the BMX DDM 16022 configuration screen is displayed. However, you can perform the procedure at any moment while programming the application.

Declaring the Variables

Introduction

The variables of the discovery kit application are of the following types:

- Internal bits: These bits are used to store intermediary states during execution of the program.
- Internal words: These words are used to store values during execution of the program. They are arranged inside the data space in the same memory field.

When you will program the sections in LD, ST and FBD languages you will need variables. It is the reason why you must declare all the variables before to program the application.

Note: All the variables of this application are unlocated. It means that you do not assign any address to these variables. An unlocated variable is a variable for which it is impossible to know its position in the PLC memory.

Implementation Methodology

You may perform one of the following procedures to declare the variables:

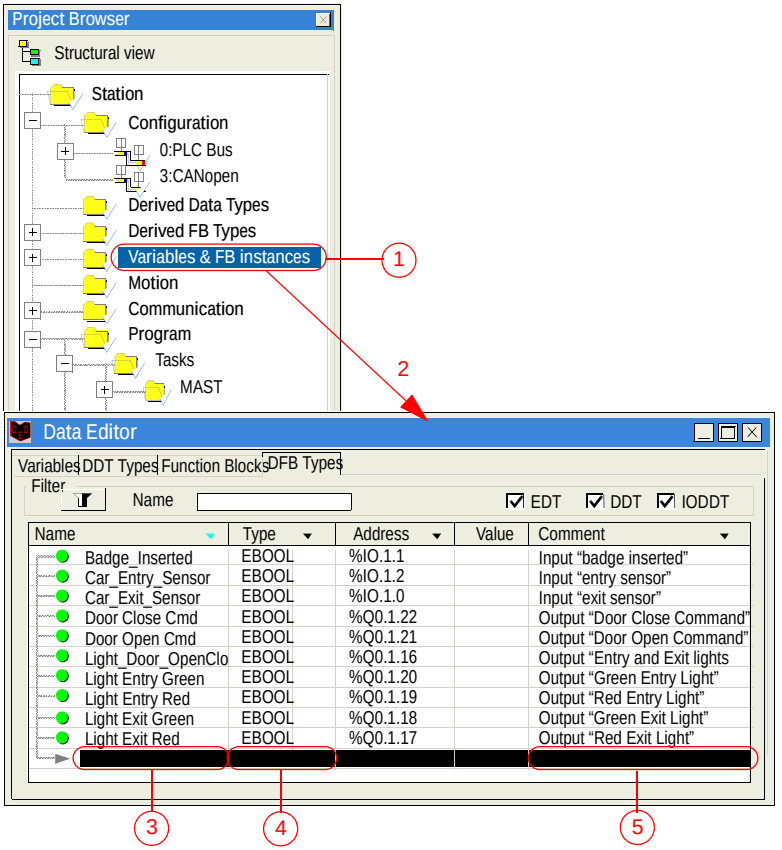
- Declare all the variables by using data editor.
- Import all the variables of the application. Declare the variable `Init` before importing all the variables.

List of Variables The following table presents all the variables to implement:

Name	Type	Comment
Anim_Car_Enter_Betw_Entry_Exit	EBOOL	Op. screen animation: car is coming into the garage and is between the entry and the exit
Anim_Car_Enter_Entry_Sensor	EBOOL	Op. screen animation: car is coming into the garage and activates the entry sensor
Anim_Car_Enter_Exit_Sensor	EBOOL	Op. screen animation: car is coming into the garage and activates the exit sensor
Anim_Car_Exit_Entry_Sensor	EBOOL	Op. screen animation: car is coming out of the garage and activates the entry sensor
Car_Coming_In_1	EBOOL	Car has inserted the badge and door is opening
Car_Coming_In_2	EBOOL	Door is open and entry light is green
Car_Coming_In_3	EBOOL	Car is coming into the garage
Car_Coming_In_4	EBOOL	Car has entered the garage and door is closing
Car_Coming_Out_1	EBOOL	Car is waiting behind the door (exit side) and the door is opening
Car_Coming_Out_2	EBOOL	Door is open and exit light is green
Car_Coming_Out_3	EBOOL	Car is coming out of the garage and is between the exit sensor and the entry sensor
Car_Coming_Out_4	EBOOL	Car has come out of the garage and door is closing
Car_Waiting_Entry	EBOOL	A car is waiting at the entry
Car_Waiting_Exit	EBOOL	A car is waiting at the exit
Door_Closed	EBOOL	Sensor "door closed" simulated by user program
Door_Open	EBOOL	Sensor "door open" simulated by user program
Init	EBOOL	Sets the garage to Init_Step
Init_Step	EBOOL	Init step of garage management
Initialization	EBOOL	System initialization (access from the operator screen)
Number_Of_Cars	INT	Number of cars in the parking (counter)
Number_Of_Cars_Max	INT	Maximum number of cars in the parking
Garage_Full	EBOOL	Garage is full (when Number_Of_Cars > Number_Of_Cars_Max)
RAZ_Number_Of_Cars	EBOOL	Sets to 0 the Number_Of_Cars. Accessible from the operator screen

Declaring a Variable

Declare the variable `Init`:



1- Double click the Variables & FB instances directory in the project browser
2- The data editor is displayed
3- Enter “Init” in the Name field
4- Select EBOOL in the Type field
5- Enter “Sets the garage to the init step” in the Comment field

Result: The variable Init is implemented. Repeat the steps 3 to 5 to declare each other variable.

Once you declare the variable `Init`, you can either declare all the variables one by one or import all the variables of the discovery kit application (see *Importing the Variables, p. 116*).
For declaring all the variables one by one, repeat the steps 3 to 5 of the procedure above.

Programming the Application

Application Overview

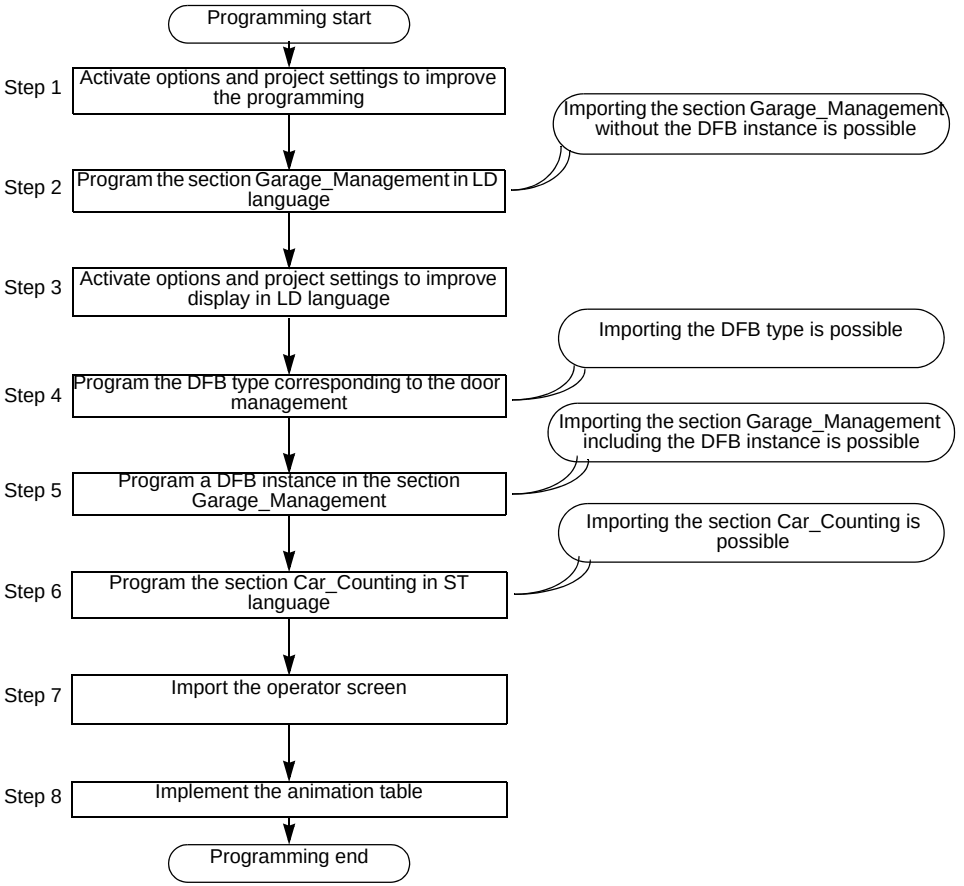
The application consists of the following elements:

- 2 sections:
  - `Garage_Management`, in LD language, which manages all the functions related to the garage.
  - `Car_Counting`, in ST language, which counts the number of cars.
- An animation table `Animation` which displays the status of all discrete inputs/outputs and two variables related to the car counting.
- An operator screen `Garage_Control` which is a graphical representation of the garage.

Note: Another section in FBD language, `Garage_Door_Management`, is associated with the DFB type `Door_Management`, which implements the garage door behavior. This DFB type is instantiated in the section `Garage_Management`.

Programming Steps

You are going to program the PLC platform as follows:



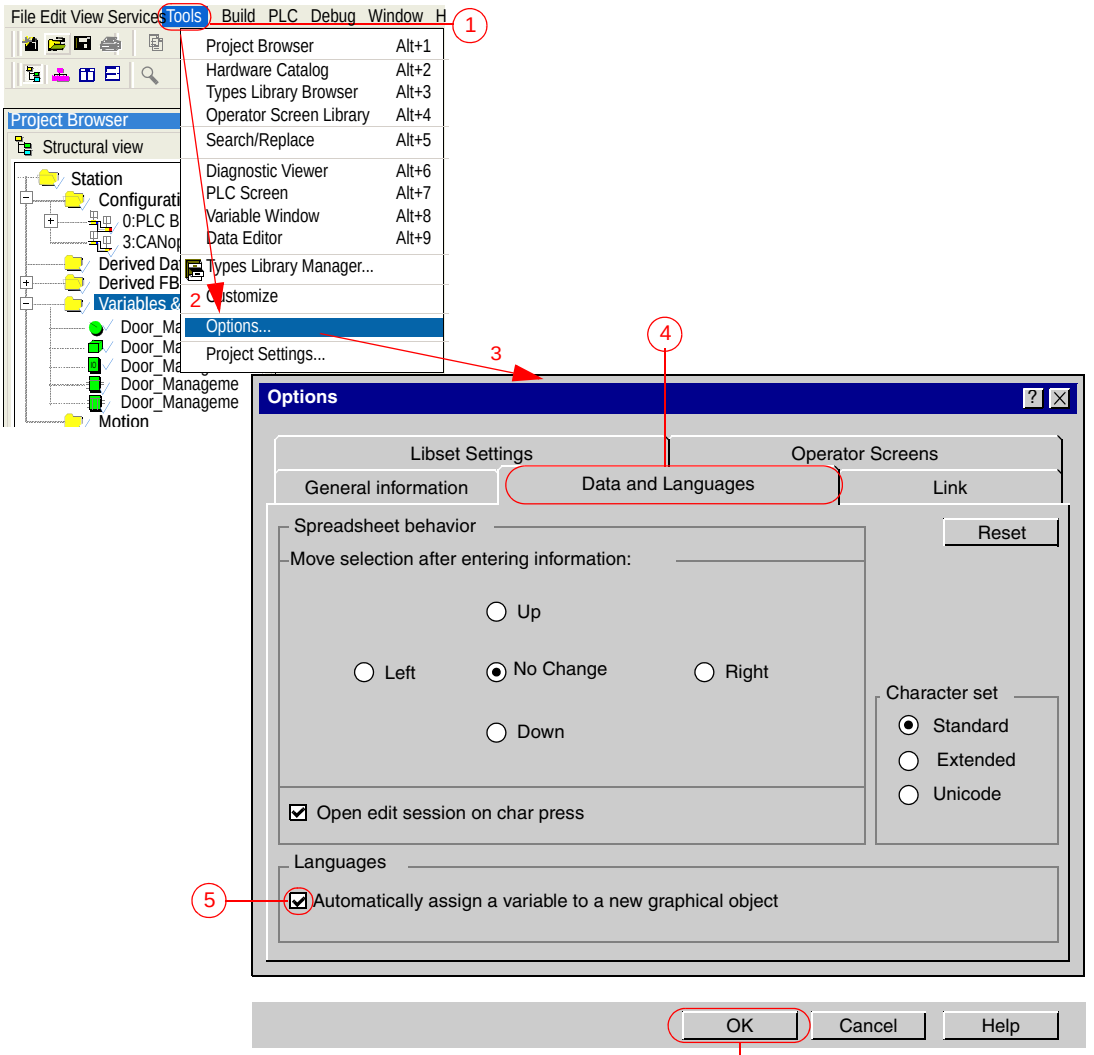
Every step is partially described in the following pages as a programming method. For each step, you can choose to program the entire project element (section or DFB type) by using the same method or to import the project element.
At any moment while programming the project, it is also possible to import the entire discovery kit application (see *Importing the Entire Project, p. 120*).

Step 1: Activating the Options and Project Settings

To make the programming easier and faster, activate the following options in Unity Pro software:

- Option "Automatically assign a variable to a new graphical object": When you place a graphical object, the properties dialog box (e.g. for contacts, coils, steps, transitions) or the FFB Input assistant (e.g. for functions, function blocks) is automatically opened for assigning the formal parameter of the object.
- Setting "Right-justify coils": Coils are automatically placed on the last column of the right power rail.

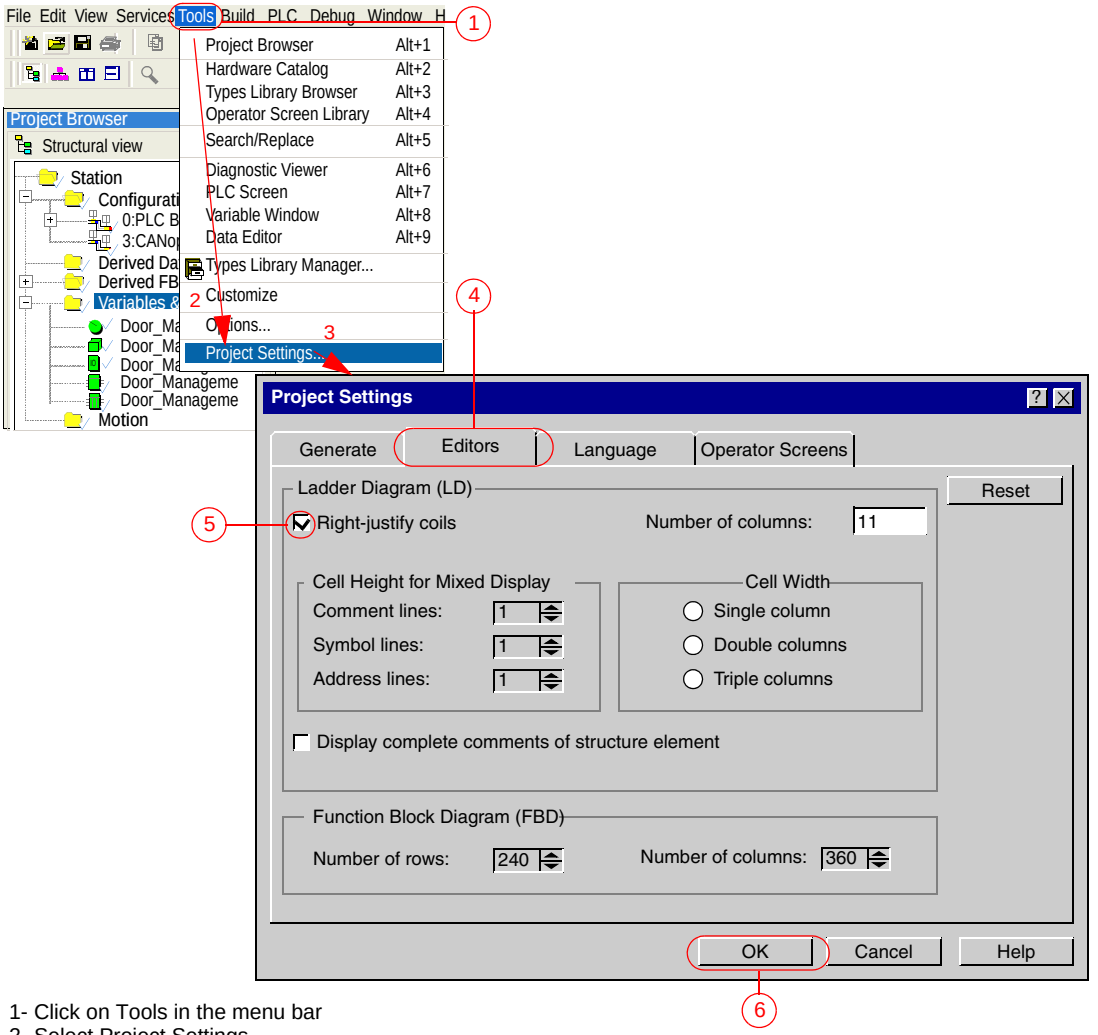
Activate the option "Automatically assign a variable to a new graphical object":



1- Click on Tools in the menu bar
2- Select Options
3- The Options screen appears
4- Click on the Data and Languages tab
5- Activate the checkbox Automatically assign a variable to a new graphical object
6- Click on the button OK to confirm

Note: Unity Pro options enable to customize the programming workshop.

Activate the setting "Right-justify coils":



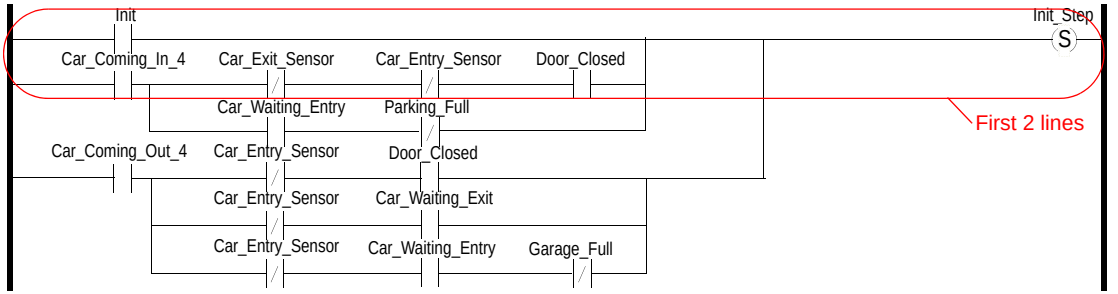
1- Click on Tools in the menu bar
2- Select Project Settings
3- The Project Settings screen appears
4- Click on the Editors tab
5- Activate the checkbox Right-justify coils
6- Click on the button OK to confirm

Note: The project settings enable to customize features in the project and belong to the project.

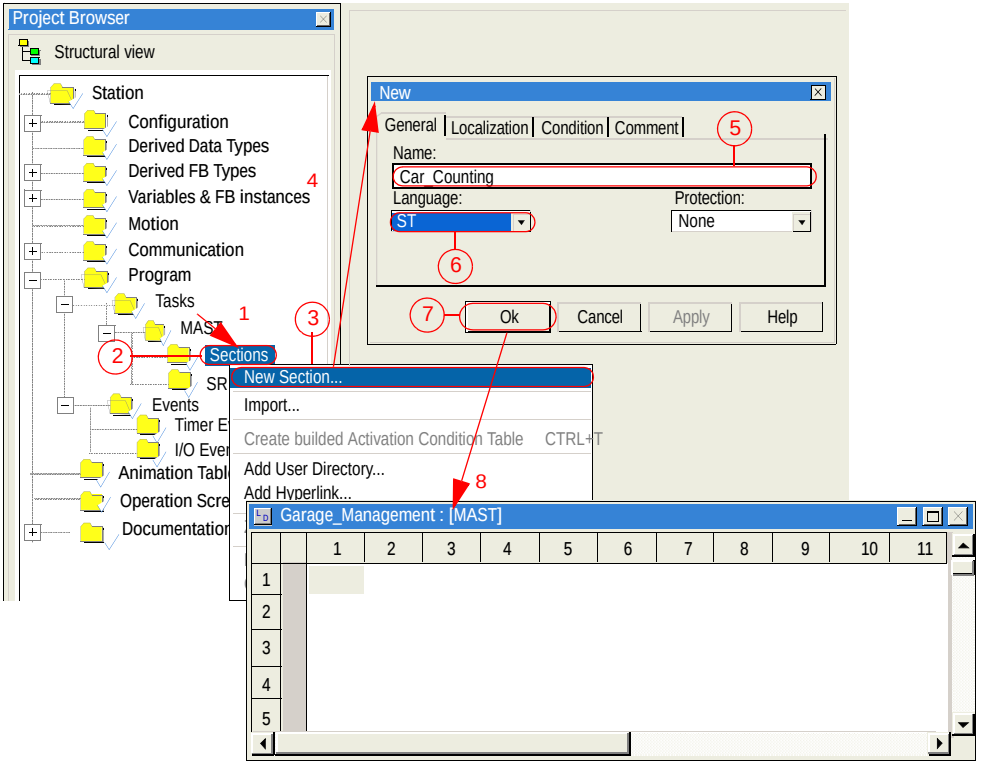
For further information about the project settings, see *Unity Pro Software/Operating Modes/Programming/Unity Pro software options/Project Settings/Editors*, in Unity Pro documentation.
For further information about Unity Pro options, see *Unity Pro Software/Operating Modes/Programming/Unity Pro software options/Options/Data and Languages*, in Unity Pro documentation.

Step 2: Programming the Section in LD Language

You are going to program the section `Garage_Management` in LD language. In the following pages, we are going to show you how to program the first 2 lines of `Init_Step` network. The programming method can be reused to program other contact networks for the section. The `Init_Step` network is as follows:

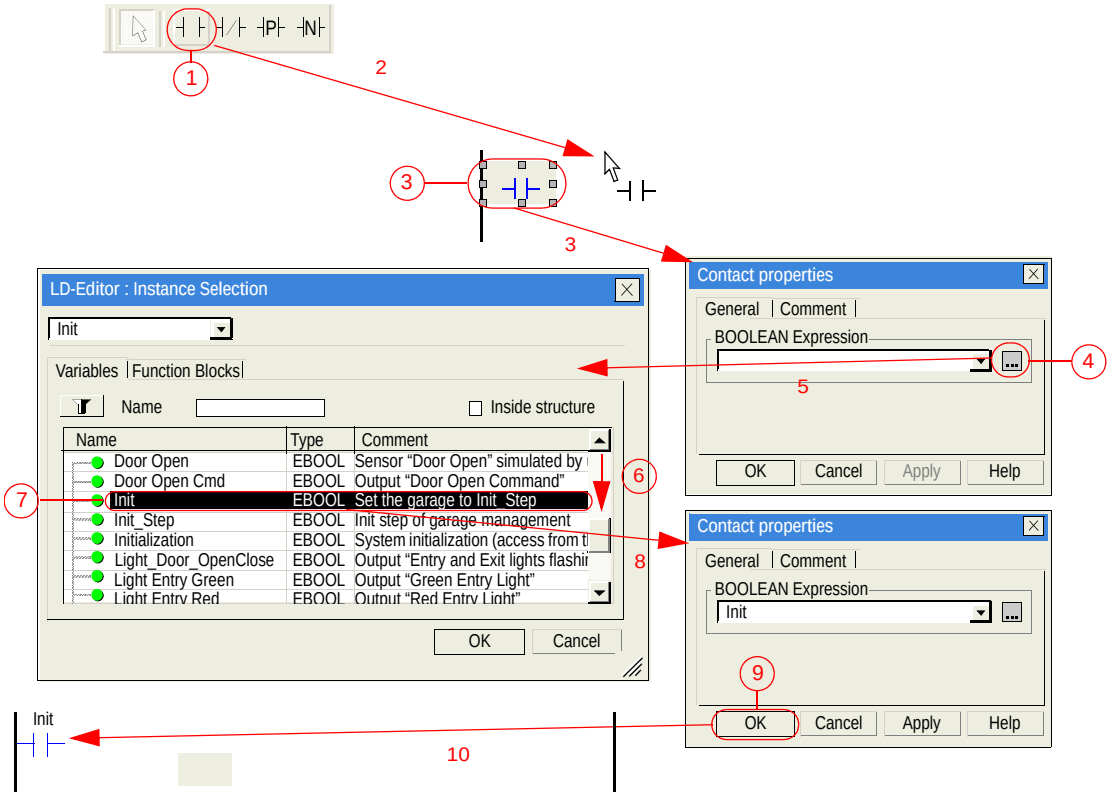


To program the section `Garage_Management`, first of all, create the section in LD language:



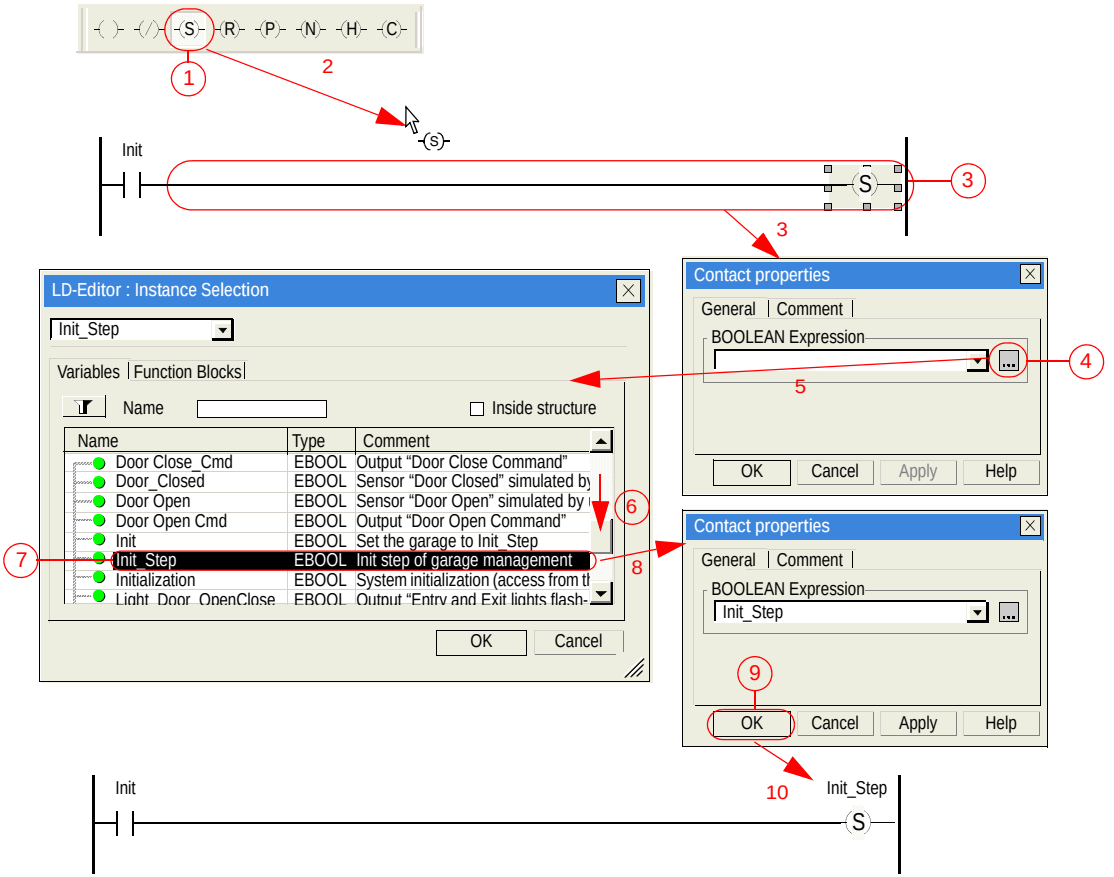
1- Open the Station/Program/Tasks/MAST/Sections directory
2- Right click on the Sections subdirectory
3- Click on New Section
4- The New screen appears
5- Enter “Garage_Management” in the Name field
6- Select “LD” in the Language field
7- Confirm by clicking on the button OK
8- The Garage_Management section editor is displayed

Implement the normally open contact assigned by the variable `Init`:



1- Click on the Normally open contact icon in the toolbar or press F3
2- The contact placement is indicated by a cursor symbol
3- Click on the target cell or press F3. The contact is placed and the Contact properties screen is displayed
4- Click on the Browse icon
5- The Instance Selection screen is displayed
6- Browse the variable Init
7- Double click the variable Init
8- The variable Init is assigned to the contact
9- Click on the button OK
10- The variable Init is assigned to the contact

Implement the set coil assigned by the variable `Init_Step`:

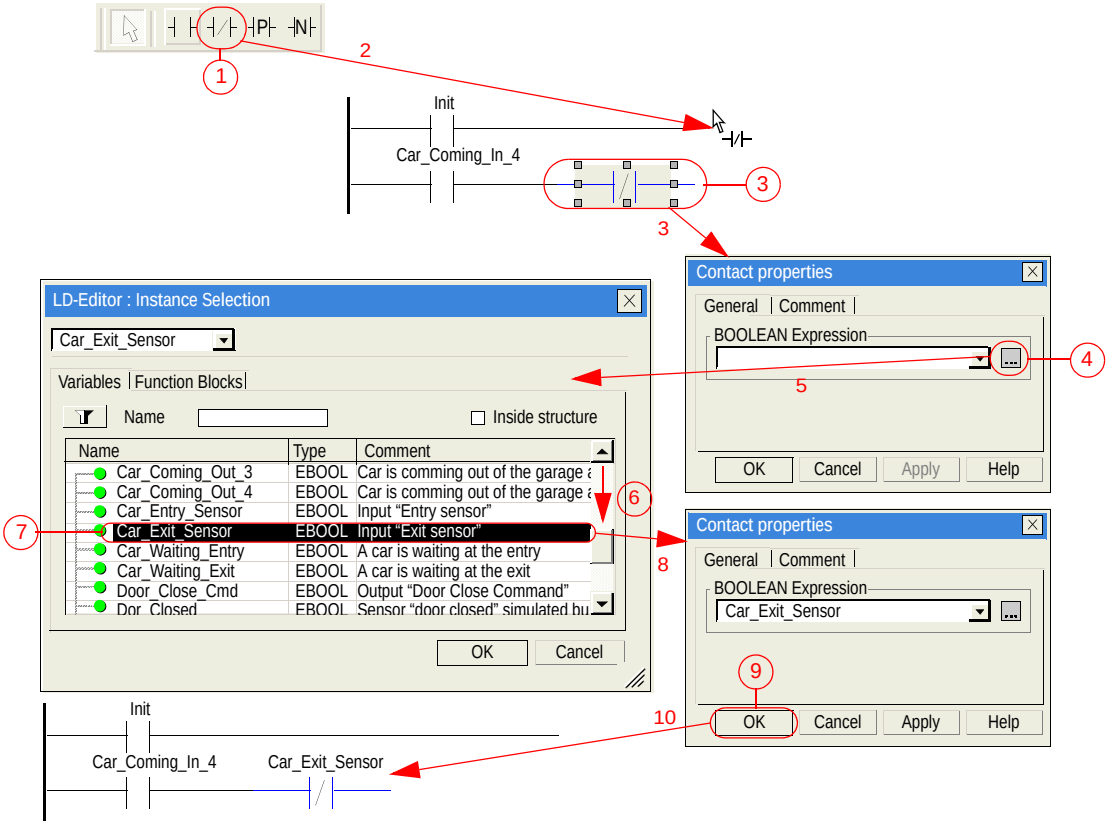


1- Click on the Set coil icon in the toolbar or press ALT+F5
2- The coil placement is indicated by a cursor symbol
3- Click on the target cell or press ALT+F5 to insert coil and the Coil properties screen is displayed
4- Click on the Browse icon
5- The Instance Selection screen is displayed
6- Browse the variable Init_Step
7- Double click the variable Init_Step
8- The variable Init_Step is assigned to the coil
9- Click on the button OK
10- The variable Init_Step is assigned to the set coil in the LD section

Note: The set coil is automatically linked to the contact assigned by the variable `Init`.

Implement the normally open contact assigned by the variable `Car_Coming_In_4` as you implemented the contact assigned by the variable `Init`:

Implement the normally closed contact assigned by the variable `Car_Exit_Sensor`:

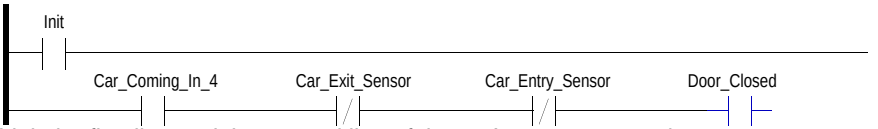


1- Click on the Normally closed contact icon in the toolbar or press Shift+F3
2- The contact placement is indicated by a cursor symbol
3- Click on the target cell or press Shift+F3 to insert the contact and the Contact properties screen is displayed
4- Click on the Browse icon
5- The Instance Selection screen is displayed
6- Browse the variable Car_Exit_Sensor
7- Double click the variable Car_Exit_Sensor
8- The variable Car_Exit_Sensor is assigned to the contact
9- Click on the button OK
10- The variable Car_Exit_Sensor is assigned to the contact

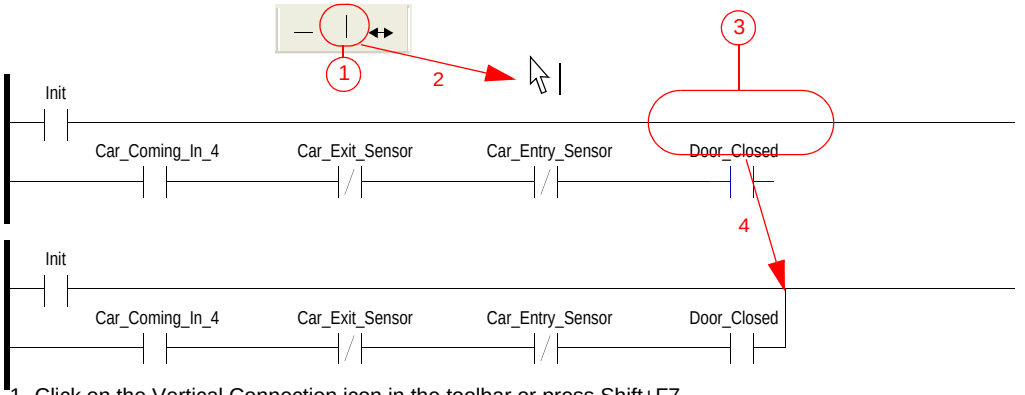
As you inserted the contacts before, perform the following steps:

- Implement the normally closed contact assigned by the variable `Car_Entry_Sensor`.
- Implement the normally open contact assigned by the variable `Door_Closed`.

You obtain the following contact network:



Link the first line and the second line of the `Init_Step` network:



1- Click on the Vertical Connection icon in the toolbar or press Shift+F7
2- The connection placement is indicated by a cursor symbol
3- Click on the target cell or press Shift+F7 to insert the vertical connection
4- The first line and the second line of Init_Step network are connected

The programming of the first 2 lines of the `Init_Step` network is finished.
You can choose to perform one of the following actions:

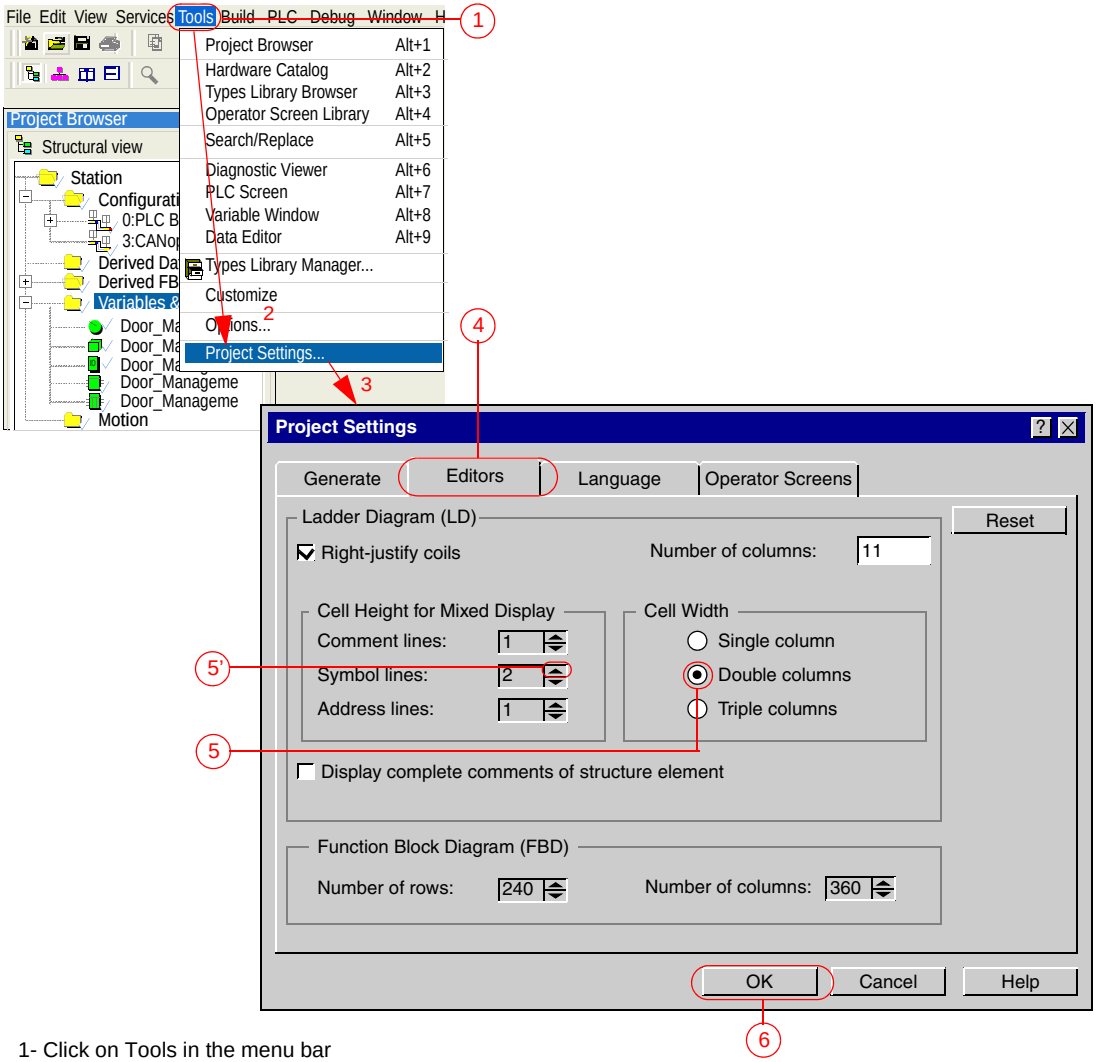
- Program all the contact networks of the section `Garage_Management` (see *Garage_Management Section, p. 122*) (do not program the DFB instance `DOOR`) by repeating the procedures related to the `Init_Step` network programming. For further information about contacts, coils and links, see *Unity Pro Software/ Operating Modes/Programming/LD Editor/Editing Contacts*, *Editing Coils* and *Editing Links*, in the Unity Pro documentation.
- Import the `Garage_Management` section without the DFB instance. To do this, perform the section Garage_Management importation procedure (see *Importing the Section Garage_Management, p. 117*) and import the file `Garage_Management_Without_DFB.xld`.
- Import the global project. To do this, perform the global project importation procedure (see *Importing the Entire Project, p. 120*).

Step 3: Activating the Options and Project Settings

Several options enable to improve display of sections in LD language. Among these options, we are going to show you three options:

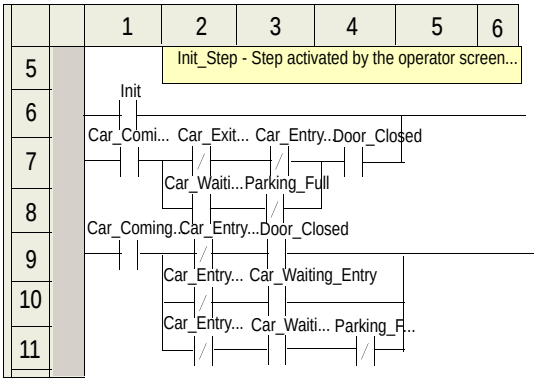
- Double columns: Sets the width of the cells to 2 standard cell widths to display longer names of variables.
- Symbol lines set to 2: Sets to 2 the number of symbol lines shown in mixed display mode (this value can be set up to 20 lines).
- Mixed display mode: Enables to display the comment, symbolic name and the address for every variable.

Activate the setting "Double columns" and set to 2 the "Symbol lines" value:

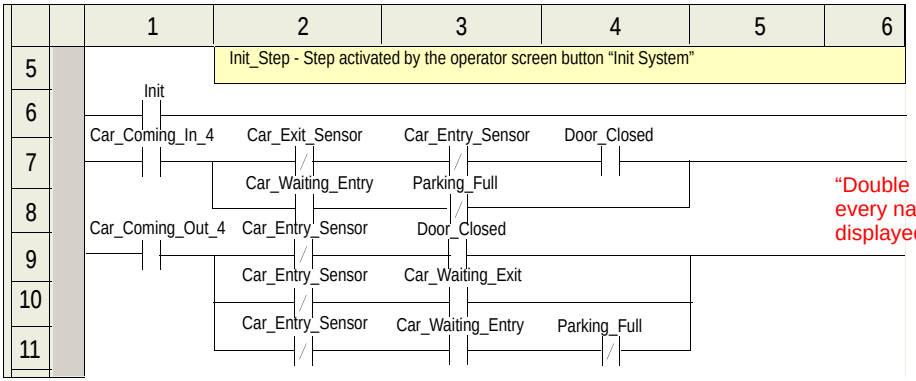


1- Click on Tools in the menu bar
2- Select Project Settings
3- The Project Settings screen appears
4- Click on the Editors tab
5- Activate the checkbox Double columns in the Cell Width zone
5'- Click on the arrow to set the Symbol lines to 2
6- Click on the button OK to confirm

The following diagram shows the section `Garage_Management` with setting "Single column" activated and the same section with setting "Double columns" activated:

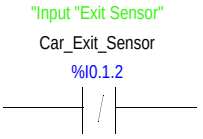


"Single column" activated: some names are not entirely displayed

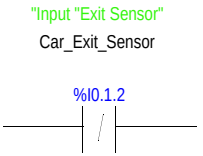


"Double columns" activated: every name is entirely displayed

The following diagram shows a contact with the setting "Symbol lines" set to 1 and the same contact with the setting "Symbol lines" set to 2:



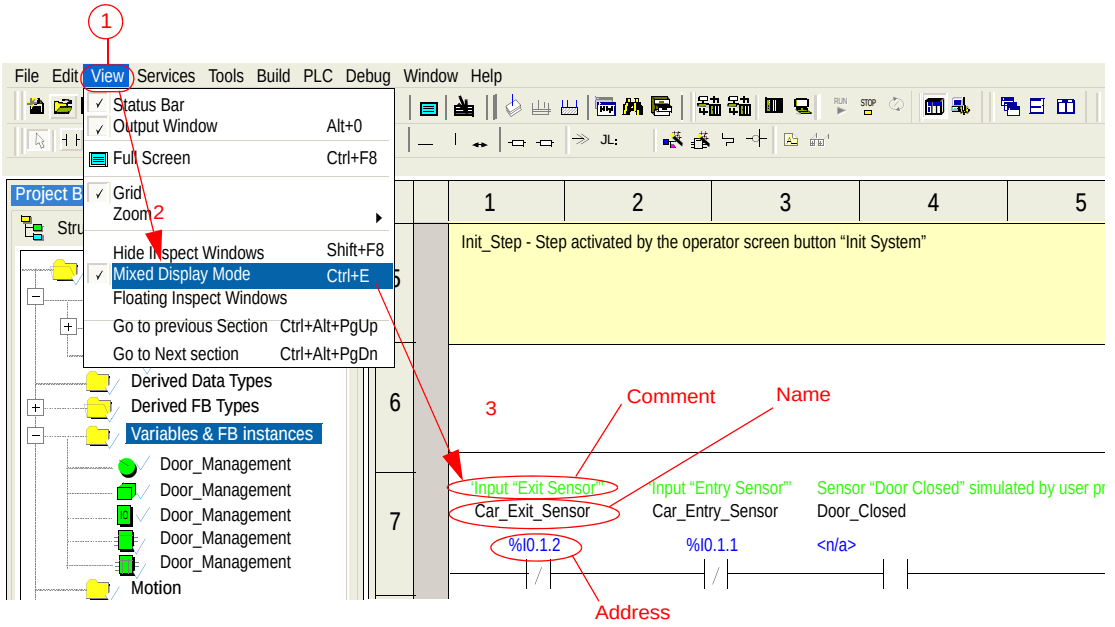
Setting "Symbol lines" set to 1: the variable symbol is displayed on 1 line



Setting "Symbol lines" set to 2: the variable symbol is displayed on 2 lines

Note: The display above is valid only if the mixed display mode is activated.

To display the comment, symbol (name) and address for each variable, you must activate the mixed display mode:



1- Click on View in the menu bar
2- Select Mixed Display Mode
3- Open the section Garage_Management if it is not open. The section is displayed as above.
For each variable, comment, name and address are displayed according to the project settings.
The name of each variable is displayed on 2 lines because the setting "Symbol lines" is set to 2.

To disable this display, click on **View/Mixed Display Mode** in the menu bar. You can see that the mixed display mode is active by way of a check-symbol in front of the menu command.

Presentation of DFBs

Unity Pro software enables you to create DFB user function blocks. A DFB is a program block that you write to meet the specific requirements of your application. It includes:

- One or more sections written in LD, IL, ST or FBD language
- Input/output parameters
- Public or private internal variables

A DFB type can be stored in the User-Defined library with version management.
Using a DFB function block in an application enables you to:

- Simplify the design and the program
- Increase the legibility of the program
- Facilitate the debugging of the application
- Reduce the volume of code generated.

A DFB function block can be used whenever a program sequence is repeated several times in your application, or to set a standard programming operation (for instance, an algorithm that controls a motor). To use a DFB type several times, you can create multiple instances of a same DFB type.
Moreover, by exporting then importing these blocks, DFB function blocks can be used by a group of programmers working on a single application or in different applications.
For further information about DFBs, see *Unity Pro Software/Operating Modes/Programming/User's Function Blocks/DFB Type/DFB Type*, U*nity Pro Software/Operating Modes/Programming/User's Function Blocks/DFB Instance/DFB Instance* and *Unity Pro Software/Languages Reference/User function blocs (DFB)/Overview of User Function Blocks (DFB)/Introduction to user function blocks*, in the Unity Pro documentation.
For this application example, you will implement the DFB type `Door_Management` by performing one of the following actions:

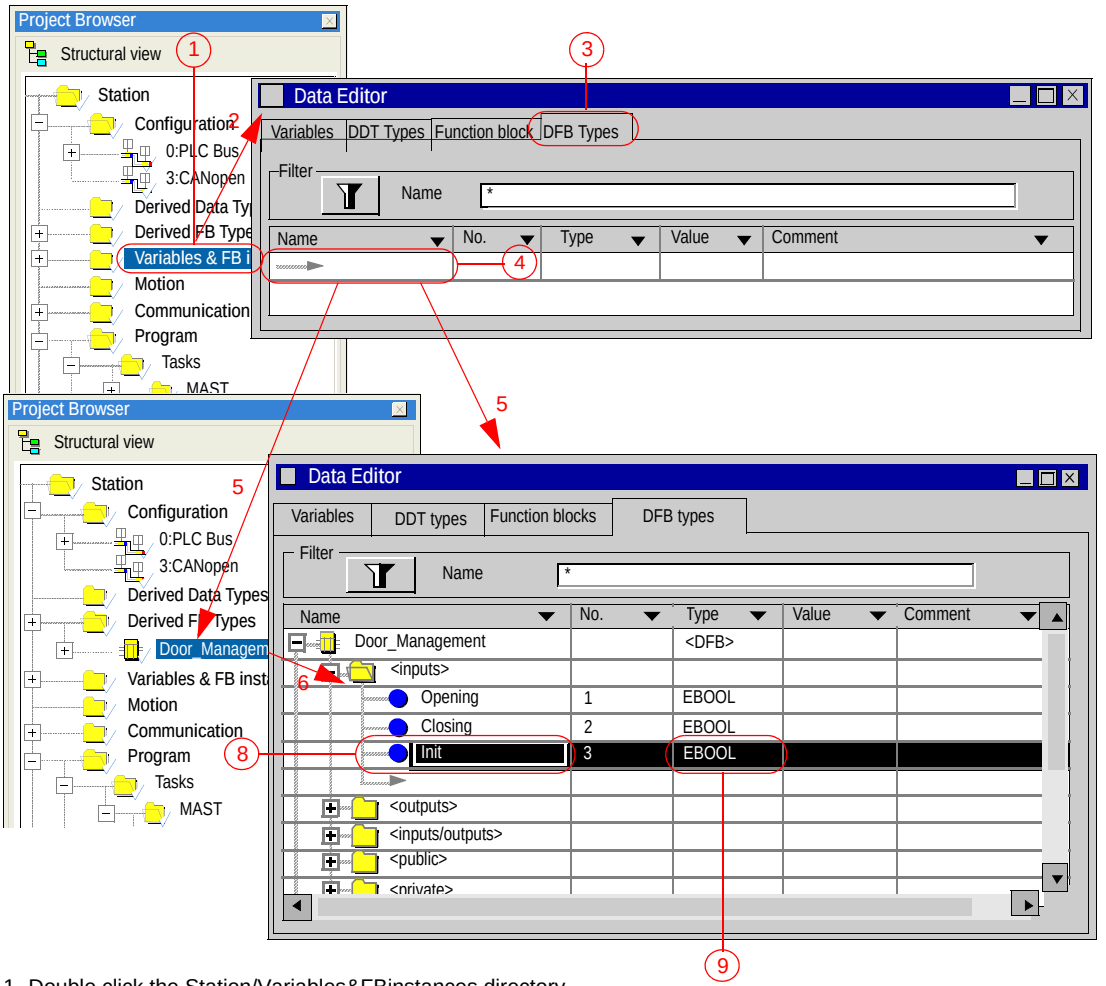
- Design of the entire `Door_Management` DFB type.
- Import the `Door_Management` DFB type by importing the XDB file.

Additionally, you can import the global application by importing the XEF file.
To program the DFB type `Door_Management`, you are going to perform the following steps:

- Create the DFB type `Door_Management`.
- Configure the parameters for this DFB type.
- Program the section associated to the DFB type.
- Instantiate the DFB type in the section `Garage_Management`.

Step 4: Creating the DFB Type and Configuring its Parameters

First of all, create the DFB type `Door_Management` and configure the inputs and the outputs for this DFB type:



1- Double click the Station/Variables&FBinstances directory
2- The data editor is displayed
3- Click on the DFB Types tab of the data editor
4- Select the first empty Name cell, then enter 'Door_Management' and confirm with Enter
5- The DFB type Door_Management is displayed in the data editor and the project browser
6- In the data editor, expand the directories Inputs and Outputs of the DFB type
7- Select the first empty Name cell
8- Enter the name of the parameter and confirm with Enter
9- Select the type of the parameter in the Type cell (parameters are described in the next page)

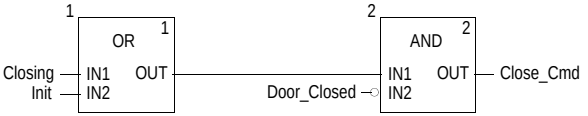
Repeat steps 7 to 10, for every parameter to be entered:

Inputs	
Name	**Type**
Opening	EBOOL
Closing	EBOOL
Init	EBOOL

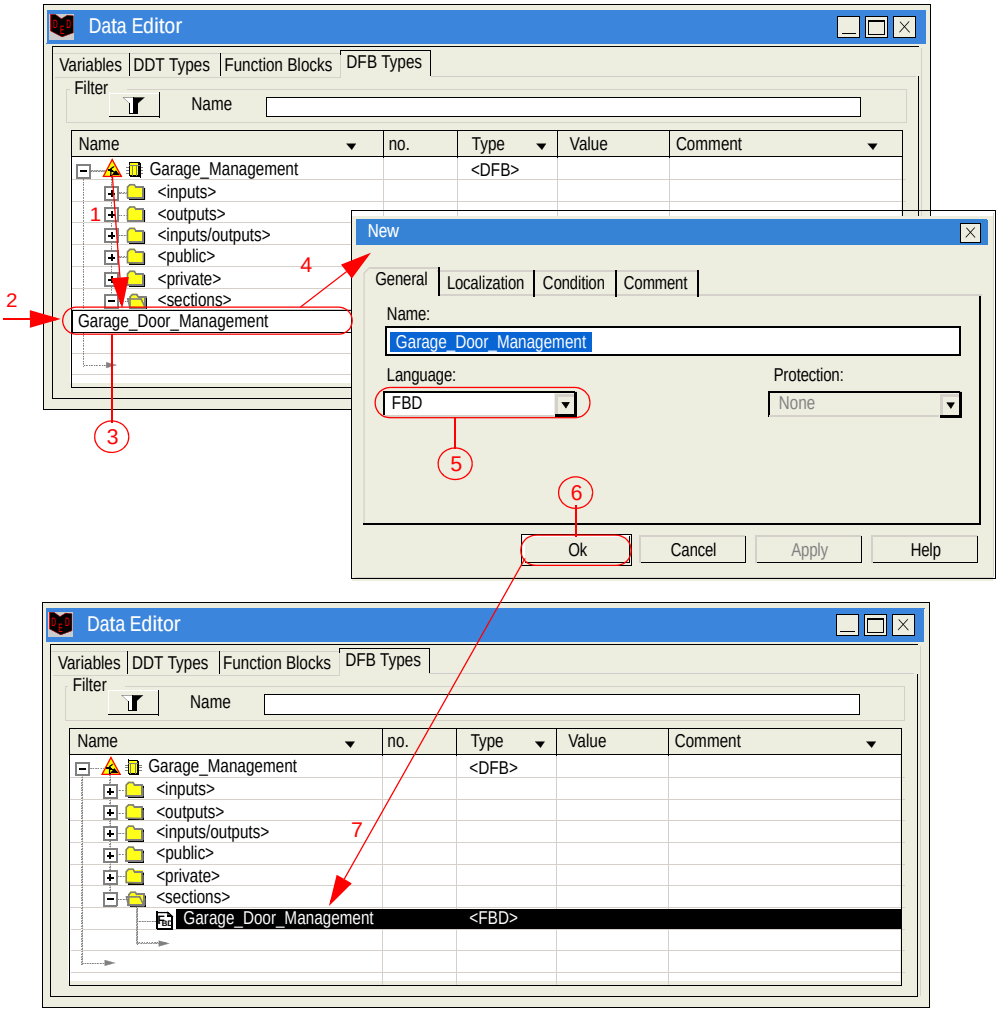
Outputs	
Name	**Type**
Open_Cmd	EBOOL
Close_Cmd	EBOOL
Door_Open	EBOOL
Door_Closed	EBOOL

Programming the Section Associated with the DFB Type

You are going to program the section `Garage_Door_Management` in FBD language. In the following pages, we are going to show you how to program the door close command. The programming method can be reused to program other functions for the section. The programming of the door close command is as follows:

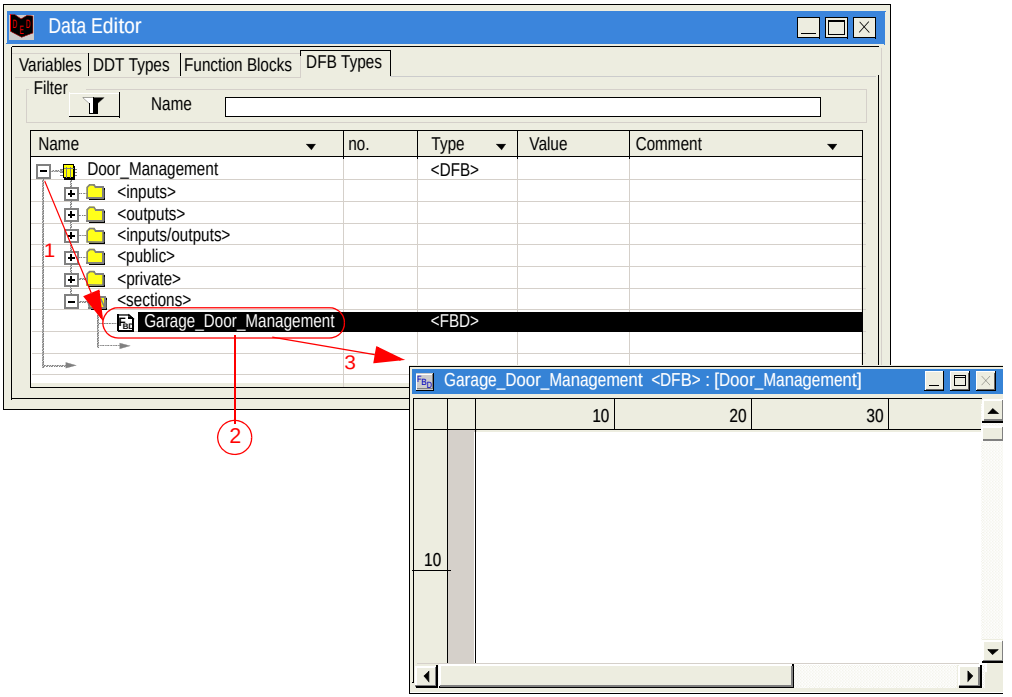


Create the section `Garage_Door_Management` associated with the DFB type in FBD language:



1- Expand the Sections directory
2- Select the first empty Name cell
3- Enter the section name Garage_Door_Management and confirm with Enter
4- The New screen appears
5- Select FBD in the Language field
6- Confirm by clicking on the button OK
7- The section Garage_Door_Management is created and is displayed in the data editor

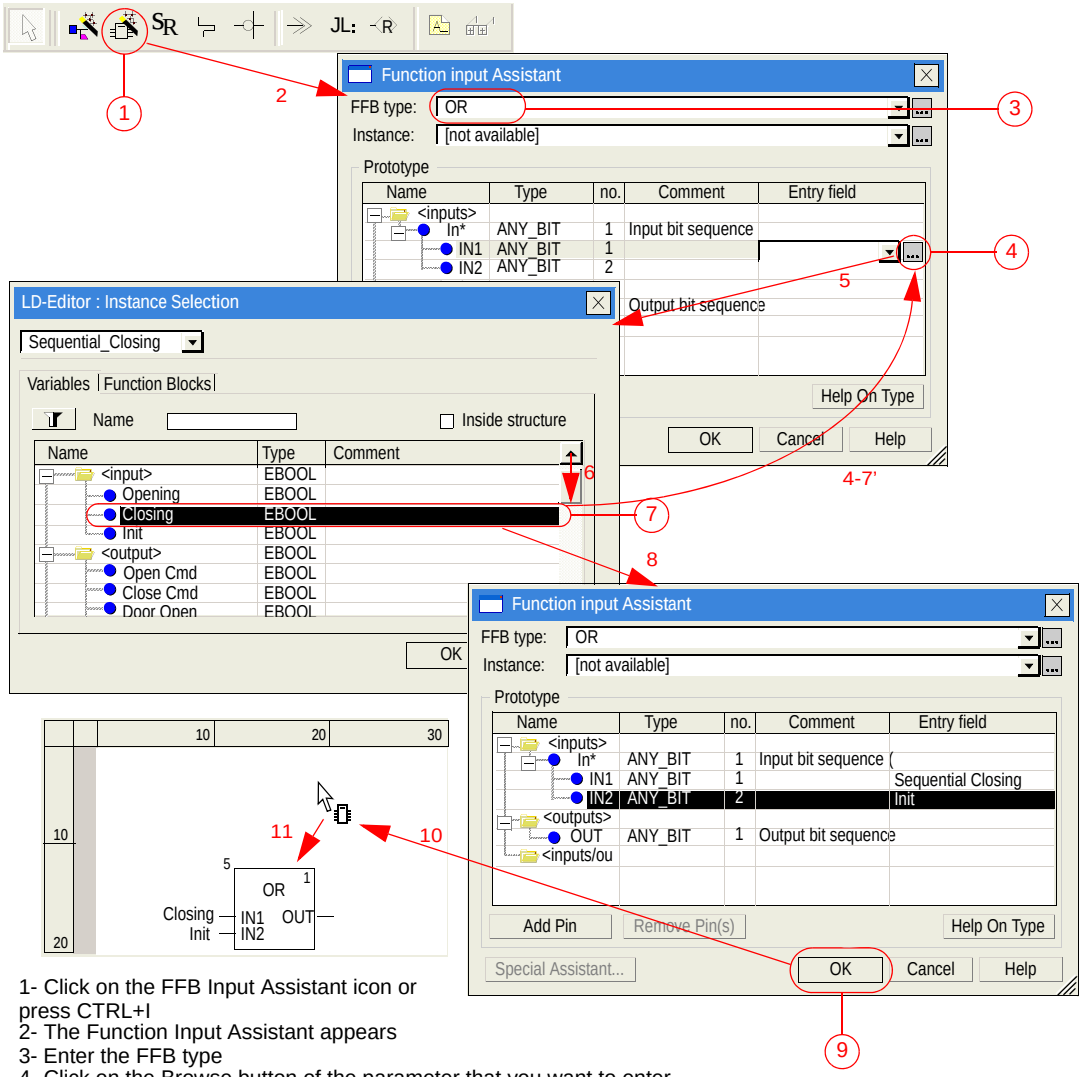
Open the FBD section editor:



1- Expand the Sections directory
2- Double click the Garage_Door_Management section
3- The FBD section editor is displayed

Insert the OR function and the AND function in the section `Garage_Door_Management`. There are several ways to program this section.

In this example, we are going to use the FFB input assistant:



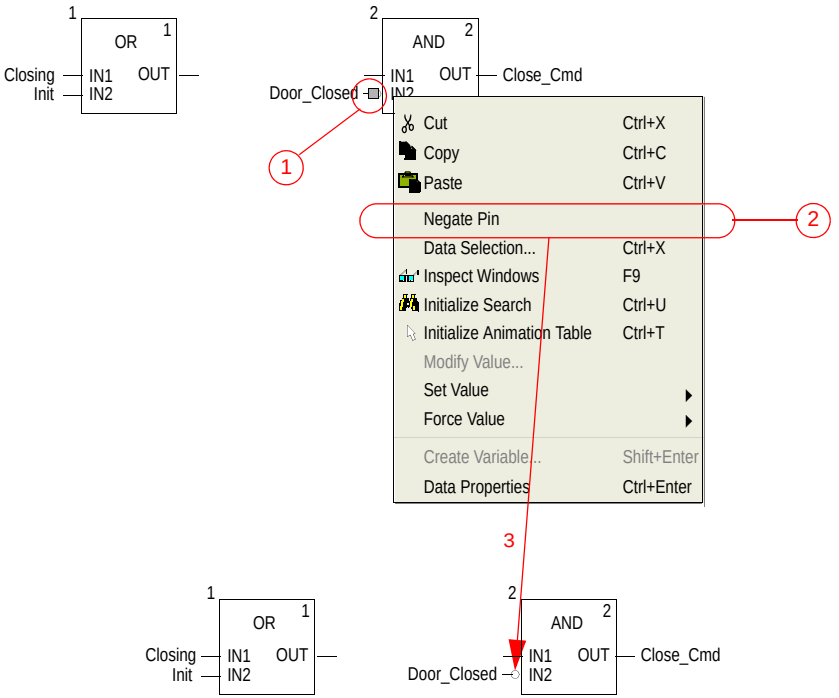
1- Click on the FFB Input Assistant icon or press CTRL+I
2- The Function Input Assistant appears
3- Enter the FFB type
4- Click on the Browse button of the parameter that you want to enter
5- The Instance Selection screen appears
6- Click on the Variables tab and browse the variable
7- Double click the variable
4-7'- Repeat the steps 4 to 7 to enter each variable in the Function Input Assistant
8- The Function Input Assistant with all the variables that you have entered is displayed
9- Click on the button OK
10- The function placement is indicated by a cursor symbol
11- Insert the function by clicking on an empty target cell

Repeat steps 1 to 11 to insert the 2 functions OR and AND:

FFB type	Parameter
OR	IN1: Closing
	IN2: Opening
	OUT: No variable (let an empty field)

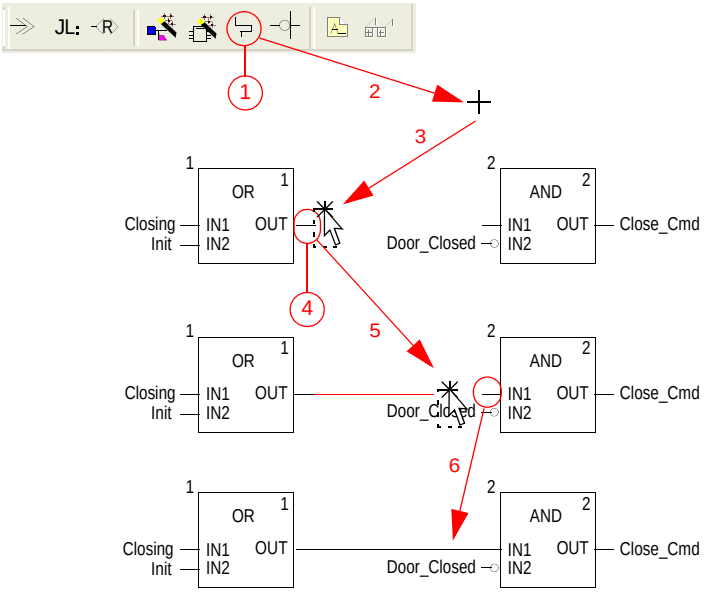
FFB type	Parameter
AND	IN1: No variable (let an empty field)
	IN2: Door_Closed
	OUT: Close_Cmd

Then, put a negative pin on the `Door_Closed` input of the AND function:



1- Right click on the pin Door_Closed of the AND function
2- Click on Negative Pin
3- The positive pin is replaced with a negative pin

Then, link the OR function to the AND function:



1- Click on the link icon in the toolbar or press F6
2- The link placement is indicated by a cursor symbol
3- Approach the OUT pin of the OR function until the link placement is indicated by this cursor symbol
4- Connect the OUT pin of the OR function by clicking on the OUT pin
5- Approach the IN1 pin of the AND function until the link placement is indicated by this cursor symbol
6- Connect the IN1 pin of the AND function by clicking on the IN1 pin

The programming of the door close command is finished.

You can choose to perform one of the following actions:

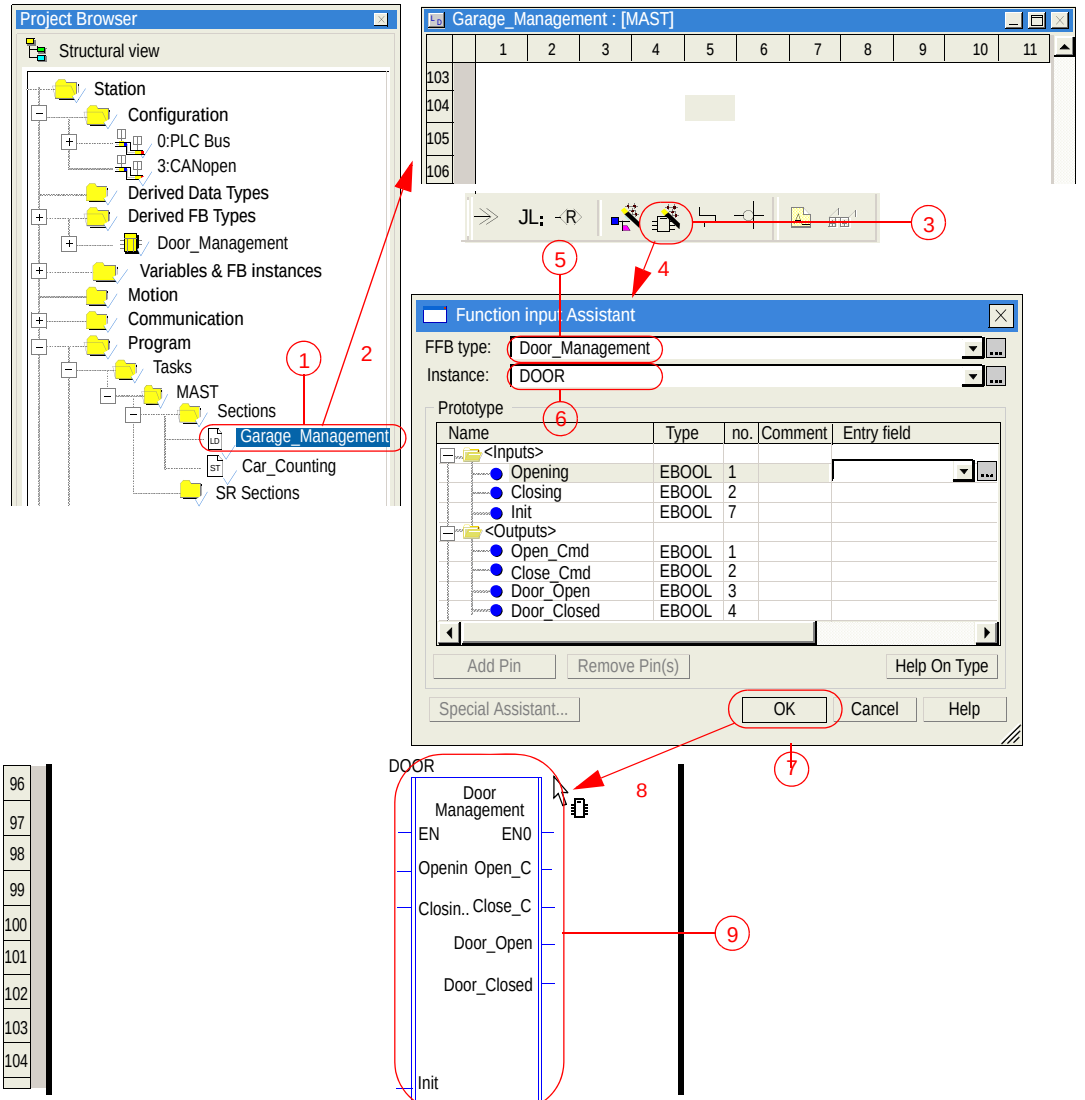
- Program all the functions of the section `Garage_Door_Management` (see *Garage_Door_Management Section, p. 126*) by repeating the procedures related to the programming of the door close command.
- Import the DFB type `Door_Management` and its section `Garage_Door_Management`. To do this, perform the DFB type importation procedure (see *Importing the DFB Type, p. 118*).
- Import the global project. To do this, perform the global project importation procedure (see *Importing the Entire Project, p. 120*).

Step 5: Instantiating the DFB type

You can choose to perform one of the following actions:

- Instantiate the DFB type by following the procedure described below.
- Import the complete `Garage_Management` section (including the DFB instance). To do this, perform the section `Garage_Management` importation procedure (see *Importing the Section Garage_Management, p. 117*) and import the file `Garage_Management_With_DFB.xld`.
- Import the global project. To do this, perform the global project importation procedure (see *Importing the Entire Project, p. 120*).

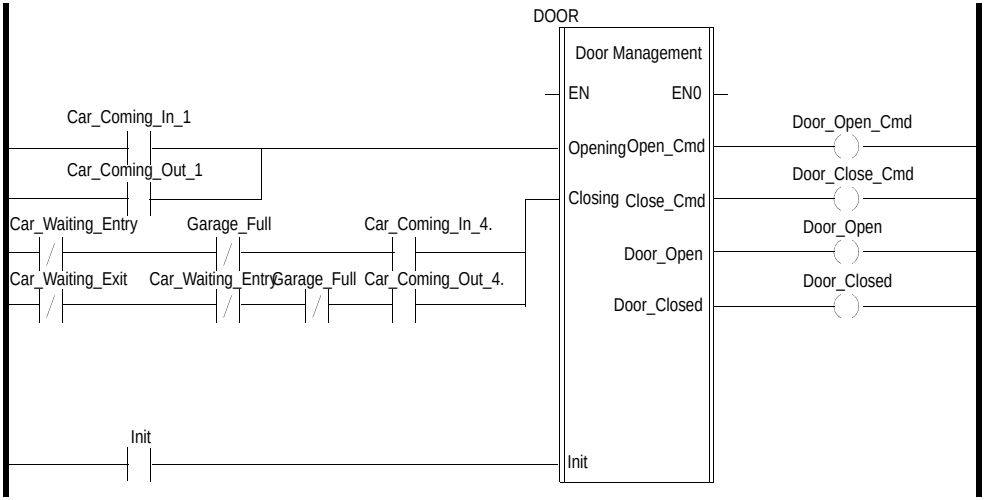
To instantiate the DFB type in the section `Garage_Management`, perform the following procedure:



1- Expand the Station/Program/Tasks/MAST/Sections directory and double click on the Garage_Management item
2- The Garage_Management section editor is displayed
3- Click on the FFB Input Assistant icon or press CTRL+I
4- The Function Input Assistant screen is displayed
5- Enter 'Door_Management' in the FFB type field
6- Enter DOOR in the Instance field

Do not fill in the parameters entry fields because it will be connected to contacts and coils
7- Confirm by clicking on the button OK
8- The function placement is indicated by a cursor
9- Insert the function by clicking on an empty target cell in a column 7, 8 or 9

To program the logic of the door management, connect contacts and coils to the DFB. The programming is as follows:



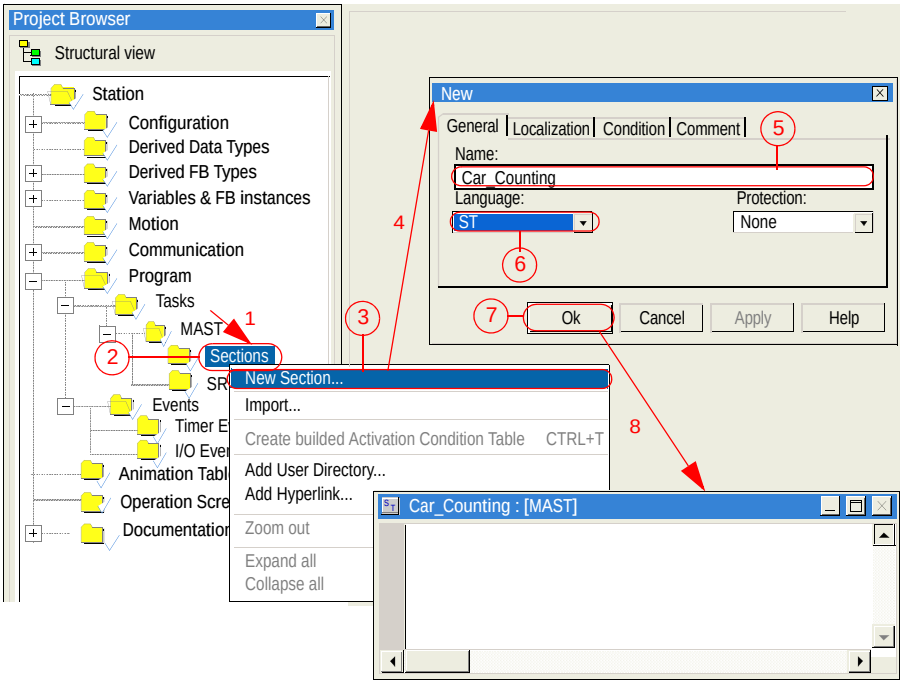
Step 6: Programming the Section in ST Language

This section consists of the variable `Number_Of_Cars_Max` set to 20 and the car counter.

You can perform one of the following actions:

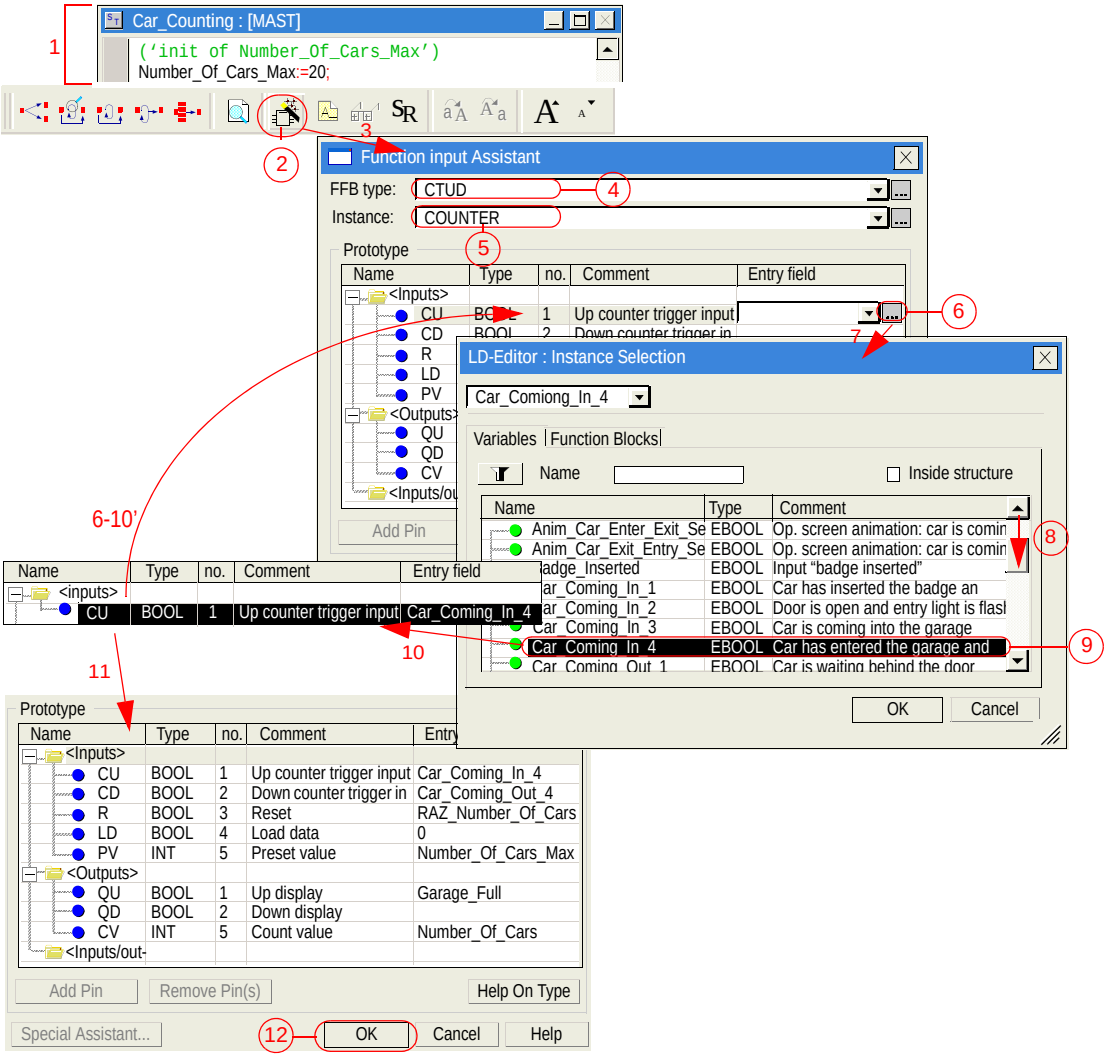
- Program the section Car_Counting (see *Car_Counting Section, p. 125*).
- Import the section `Car_Counting`. To do this, perform the section `Car_Counting` importation procedure (see *Importing the Section Car_Counting, p. 119*).
- Import the global project. To do this, perform the global project importation procedure (see *Importing the Entire Project, p. 120*).

To program the section `Car_Counting`, first of all, create the section in ST language:



1- Open the Station/Program/Tasks/MAST/Sections directory
2- Right click on the Sections subdirectory
3- Click on New Section
4- The New screen appears
5- Enter “Car_Counting” in the Name field
6- Select “ST” in the Language field
7- Confirm by clicking on the button OK
8- The section editor is displayed

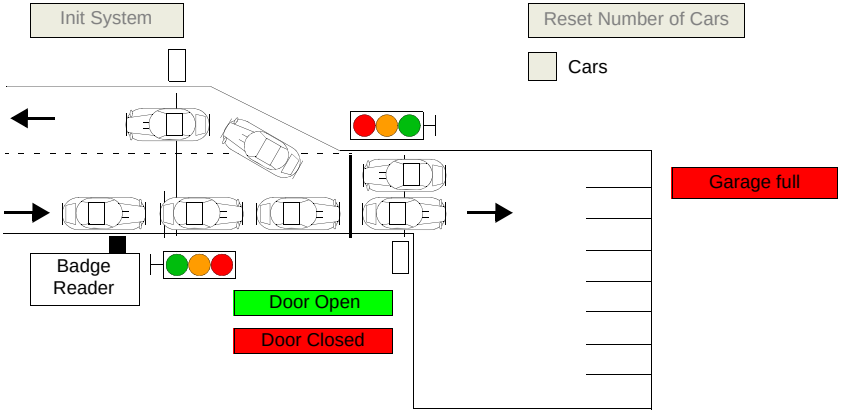
Program the section `Car_Counting`:



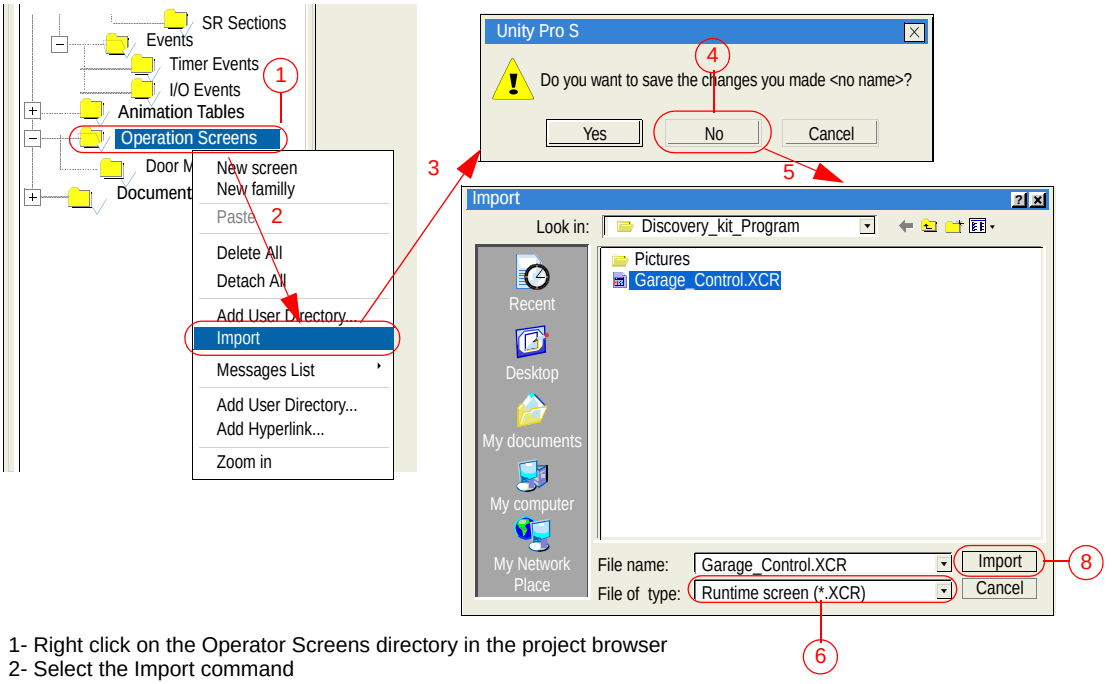
1- Enter the program for the first 2 lines
2- Click on the FFB Input Asistant icon in the toolbar or press CTRL+I
3- The Function Input Assistant screen is displayed
4- Enter CTUD in the FFB type field
5- Enter COUNTER in the Instance field
6- Click on the Browse icon of the first parameter
7- The Instance Selection screen is displayed
8- Browse the variable Car_Coming_In_4
9- Double click the variable Car_Coming_In_4
10- The variable Car_Coming_In_4 is assigned to the parameter CU
6-10'- Repeat the steps 6 to 10 to assign a variable to each parameter (see the last screenshot)
11- All the parameters of the COUNTER function are assigned
12- Click on the button OK. The COUNTER function is implemented in the Car_Counting section

Step 7: Importing the Operator Screen

A source file corresponding to a graphical display of the garage is stored on the Discovery Kit CD-ROM. In this example, you are not going to program the operator screen but you are going to import the file stored on the CD-ROM. It is the reason why the editing modes of the operator screen are not explained.
For further information about operator screens editing modes, see *Unity Pro Software/Operating Modes/Operator screen/Editing runtime screens*, in Unity Pro documentation.
The following diagram shows the operator screen in offline mode. Every animated object is displayed:



The operator screen of the application has a source file whose file extension is XCR. This file is available on the Discovery Kit CD-ROM in the directory `Discovery Kit Program` and is named `Garage_Control.XCR`. It enables to import the operator screen:

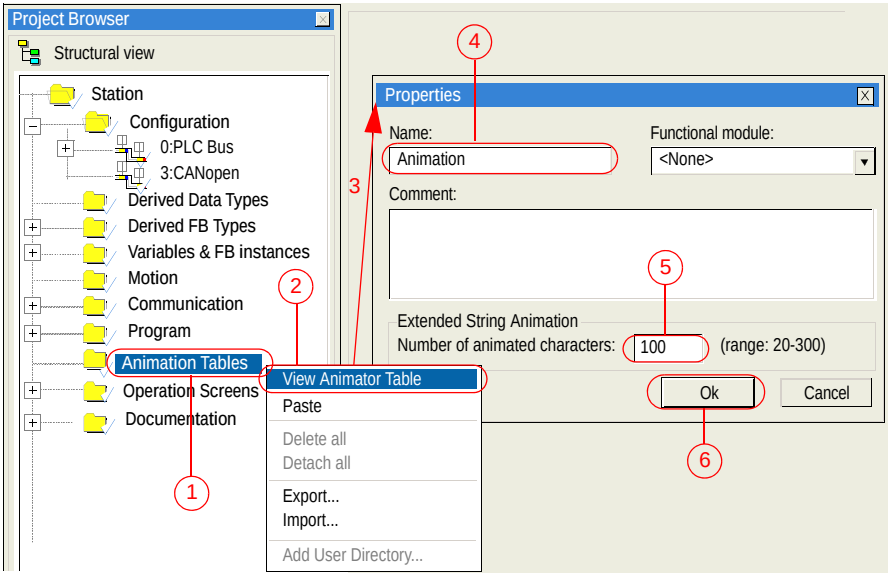


1- Right click on the Operator Screens directory in the project browser
2- Select the Import command
3- A screen appears
4- Click on the button NO
5- The Import screen appears
6- Select XCR in the File type field
7- Browse the XCR file to be imported
8- Click on the button Import

Result: The operator screen is imported. To access the operator screen, double click the **Operator Screens/Garage_Control** directory in the project browser.

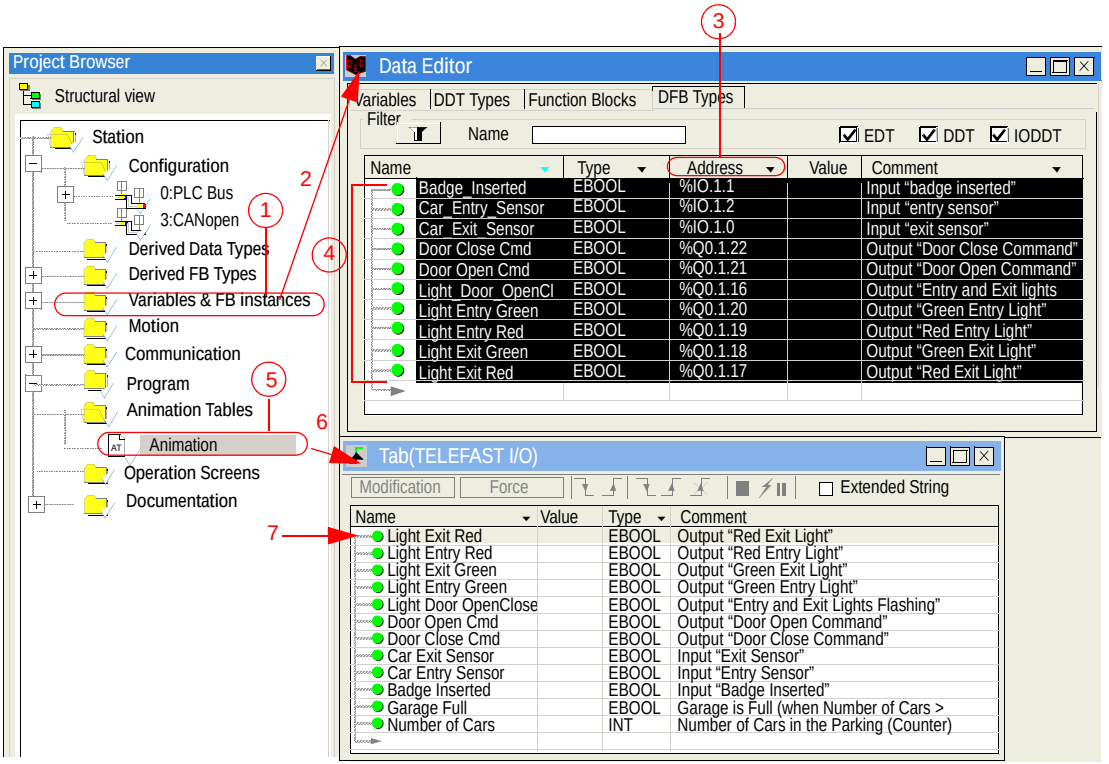
Step 8: Implementing the Animation Table

Create the animation table:



1- Right click on the Animation Tables directory in the project browser
2- Select New Animation Table
3- The properties screen appears
4- Enter “Animation” in the Name field
5- Keep the default value for the Number of animated characters field
6- Click on the button OK to confirm

Insert variables in the animation table:



1- Double click the Variables&FB instances directory in the project browser
2- The data editor is displayed
3- Sort the variables by clicking on the Address column to group the inputs and the outputs
4- Select all the inputs/outputs and press CTRL+C
5- Double click the animation table in the project browser
6- The animation table is displayed
7- Select the first empty row of the animation table and press CTRL+V

Result: All the inputs/outputs are displayed in the animation table.

Repeat the steps 4...7 to add the variables `Garage_Full` and `Number_Of_Cars` in the animation table.

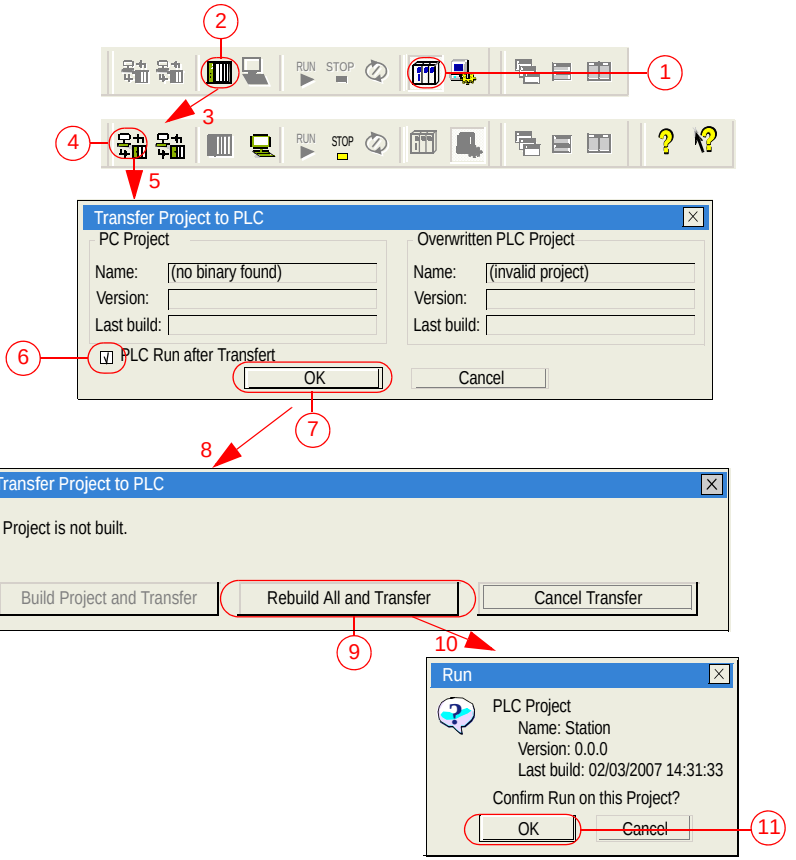
Transferring the Project from the Terminal to the PLC

At a Glance

The entire application is now implemented in Unity Pro.
The next step is to transfer the application from the terminal to the PLC.

Transfer from the Terminal to the PLC

Transfer the project to the PLC processor:



1- Click on the icon Standard mode in the menu bar if you are in Simulator mode
2- Click on the icon Connect in the menu bar
3- The Download project icon can be activated
4- Click on the icon Download project
5- The Tranfer Project to PLC screen appears
6- Activate the PLC run after Transfer checkbox
7- Click on the button Transfer
8- The Tranfer Project to PLC screen appears
9- Click on the button Rebuild All and Transfer
10- The Run screen appears
11- Click on the button OK to set the PLC processor to RUN mode

Executing the Application

The PLC is now set to RUN mode. Therefore, the application is executed by the PLC.

Simulating and Debugging the Application

At a Glance

The application implemented in Unity Pro is now executed by the PLC.
To simulate the application you will use the following elements:

- Unity Pro commands to control the execution of the application.
- TELEFAST module to activate the 3 inputs.
- The operator screen to view the garage status and car positions.
- The animation table to view the status of every input/output, the car counter and to view if the garage is full.

The following pages guide you through simulating of a car entering the garage.

Execution Management

The RUN mode enables to execute the application and the STOP mode enables to stop the execution of the application. Use the following procedure to set the processor to RUN mode or to STOP mode:

If the RUN icon is grayed out, it means the PLC is in RUN mode:

If you want to stop the execution of the application, click on the STOP icon
Thus, the application is no longer executed by the processor until the next click on the RUN icon.

If the STOP icon is grayed out, it means the PLC is in STOP mode:

If you want to stop the execution of the application, click on the RUN icon
Thus, the application is executed by the processor until the next click on the STOP icon.

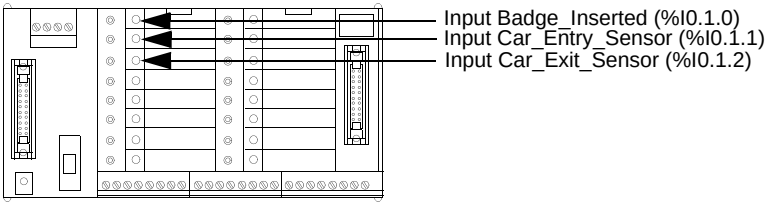
Note: A dialog box "Confirm Run of this project" or "Confirm Stop of this project" appears when you click on either the RUN icon or the STOP icon. Click on the button OK to confirm.

How to Use the TELEFAST Module

The following table presents all the discrete inputs that you can activate by using the TELEFAST module:

Address	Name	Comment
%I0.1.0	Badge_Inserted	Input "badge inserted"
%I0.1.1	Car_Entry_Sensor	Input "entry sensor"
%I0.1.2	Car_Exit_Sensor	Input "exit sensor"

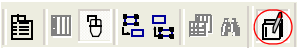
Each input has its switch on the TELEFAST module:



To switch on an input, flip the switch to the right. To switch off an input, flip the switch to the center.

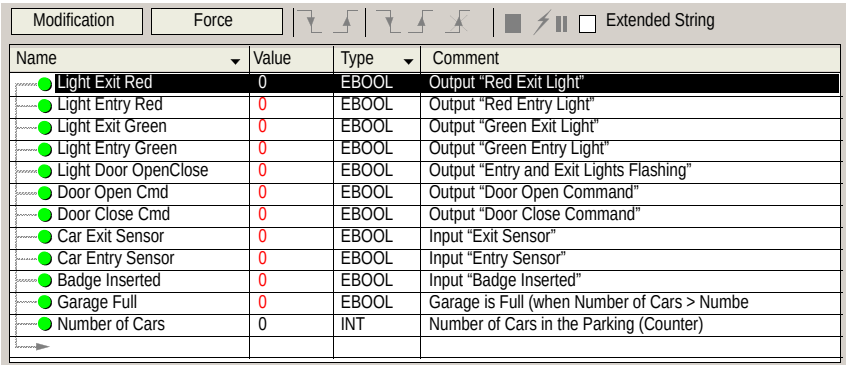
How to Use the Operator Screen

To access the operator screen, open the **Operator Screens** directory in the project browser and double click on `Garage_Control` item.
You will need to use the buttons `Reset Number Of Cars` and `Init System`. To access these buttons, click on the following icon in the toolbar:



How to Use the Animation Table

To access the animation table, open the **Animation Tables** directory in the project browser and double click on `Animation` item.
The animation table will appear:

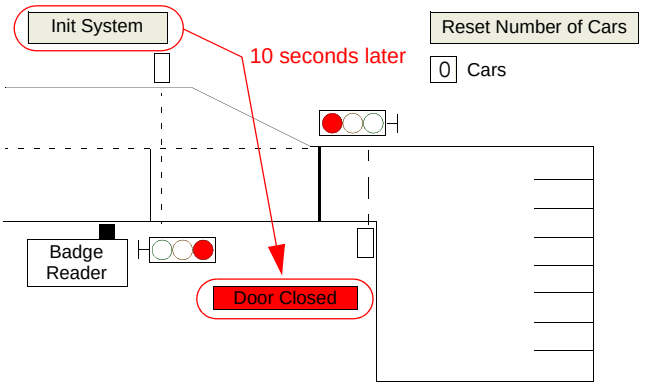


Name	Value	Type	Comment
Light Exit Red	0	EBOOL	Output “Red Exit Light”
Light Entry Red	0	EBOOL	Output “Red Entry Light”
Light Exit Green	0	EBOOL	Output “Green Exit Light”
Light Entry Green	0	EBOOL	Output “Green Entry Light”
Light Door OpenClose	0	EBOOL	Output “Entry and Exit Lights Flashing”
Door Open Cmd	0	EBOOL	Output “Door Open Command”
Door Close Cmd	0	EBOOL	Output “Door Close Command”
Car Exit Sensor	0	EBOOL	Input “Exit Sensor”
Car Entry Sensor	0	EBOOL	Input “Entry Sensor”
Badge Inserted	0	EBOOL	Input “Badge Inserted”
Garage Full	0	EBOOL	Garage is Full (when Number of Cars > Numbe
Number of Cars	0	INT	Number of Cars in the Parking (Counter)

The animation table displays the status of all the discrete inputs/outputs and the status of the variables `Garage_Full` and `Number_Of_Cars`.
Now, Unity Pro editor window displays the animation table and the operator screen.

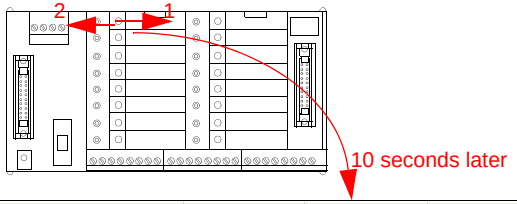
Initializing the System

First of all, initialize the application by clicking on the `Init System` button of the operator screen.
The door will be closed 10 seconds later:

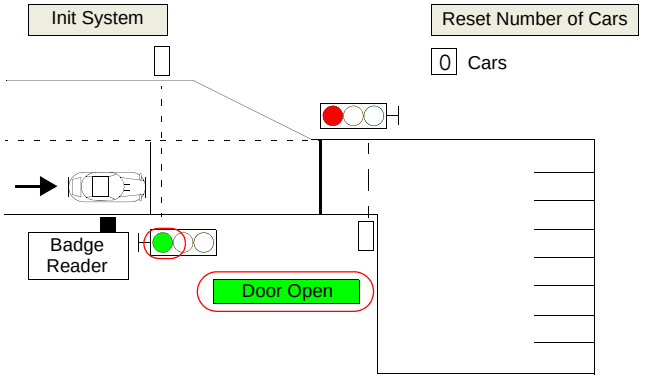


First Step: Car Waits at the Entry

Switch on the input `Badge_Inserted` (`%I0.1.0`) and switch off this input. You will see a car waiting at the entry and the door opening.
The door will be fully open 10 seconds later you switched on the input `Badge_Inserted` (`%I0.1.0`):

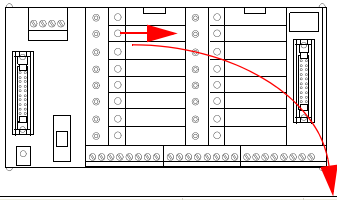


Name	Value	Type	Comment
Badge_Inserted	F1	EBOOL	Input “badge inserted”
Car_Entry_Sensor	0	EBOOL	Input “entry sensor”
Car_Exit_Sensor	0	EBOOL	Input “exit sensor”
Door Close Cmd	0	EBOOL	Output “Door Close Command”
Door Open Cmd	0	EBOOL	Output “Door Open Command”
Light_Door_OpenClose	0	EBOOL	Output “Entry and Exit lights flashing”
Light Entry Green	0	EBOOL	Output “Green Entry Light”
Light Entry Red	0	EBOOL	Output “Red Entry Light”
Light Exit Green	0	EBOOL	Output “Green Exit Light”
Light Exit Red	0	EBOOL	Output “Red Exit Light”
Number_Of_Cars	0	INT	Number of cars in the parking (counter)
Garage_Full	0	EBOOL	Garage is full (when Number_Of_Cars >

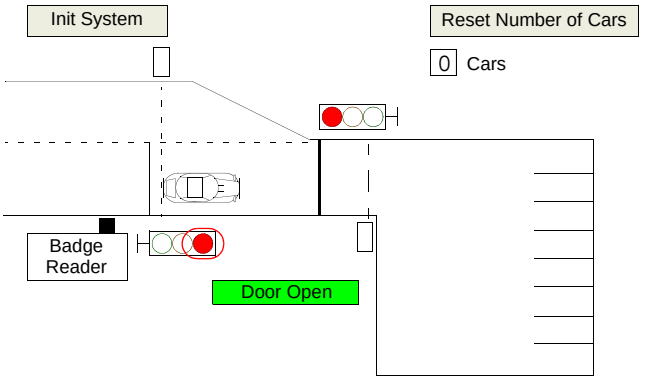


Second Step: Car Activates the Entry Sensor

Once the door is open, the entry light turns green and the entry barrier is open. It means the car can come into the garage. Switch on the input `Car_Entry_Sensor` (`%I0.1.1`) to make the car appear in front of the entry sensor.
Once you switch on the input `Car_Entry_Sensor` (`%I0.1.1`), the entry barrier closes and the entry light turns red:

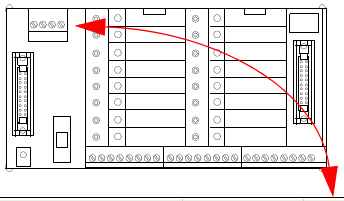


Name	Value	Type	Comment
Badge_Inserted	0	EBOOL	Input "badge inserted"
Car_Entry_Sensor	1	EBOOL	Input "entry sensor"
Car_Exit_Sensor	0	EBOOL	Input "exit sensor"
Door Close Cmd	0	EBOOL	Output "Door Close Command"
Door Open Cmd	0	EBOOL	Output "Door Open Command"
Light_Door_OpenClose	0	EBOOL	Output "Entry and Exit lights flashing"
Light Entry Green	0	EBOOL	Output "Green Entry Light"
Light Entry Red	1	EBOOL	Output "Red Entry Light"
Light Exit Green	0	EBOOL	Output "Green Exit Light"
Light Exit Red	1	EBOOL	Output "Red Exit Light"
Number_Of_Cars	0	INT	Number of cars in the parking (counter)
Garage_Full	0	EBOOL	Garage is full (when Number_Of_Cars >

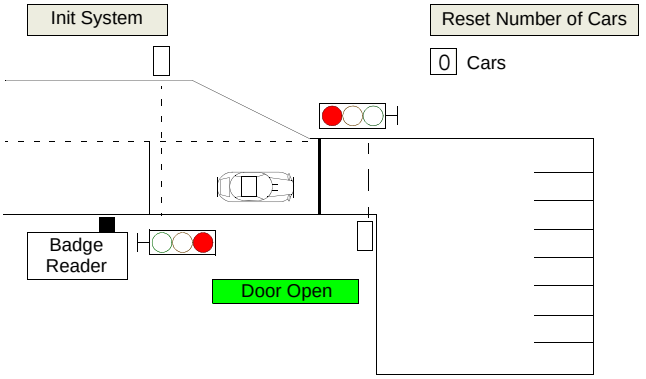


Third Step: Car is Between the Entry Sensor and the Exit Sensor

Switch off the input `Car_Entry_Sensor` (`%I0.1.1`). The car appears between the entry sensor and the exit sensor:

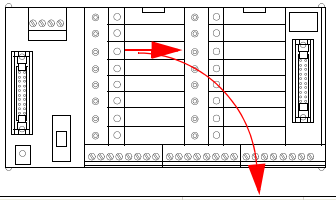


Name	Value	Type	Comment
Badge_Inserted	0	EBOOL	Input “badge inserted”
Car_Entry_Sensor	0	EBOOL	Input “entry sensor”
Car_Exit_Sensor	0	EBOOL	Input “exit sensor”
Door Close Cmd	0	EBOOL	Output “Door Close Command”
Door Open Cmd	0	EBOOL	Output “Door Open Command”
Light_Door_OpenClose	0	EBOOL	Output “Entry and Exit lights flashing”
Light Entry Green	0	EBOOL	Output “Green Entry Light”
Light Entry Red	1	EBOOL	Output “Red Entry Light”
Light Exit Green	0	EBOOL	Output “Green Exit Light”
Light Exit Red	1	EBOOL	Output “Red Exit Light”
Number_Of_Cars	0	INT	Number of cars in the parking (counter)
Garage_Full	0	EBOOL	Garage is full (when Number_Of_Cars >

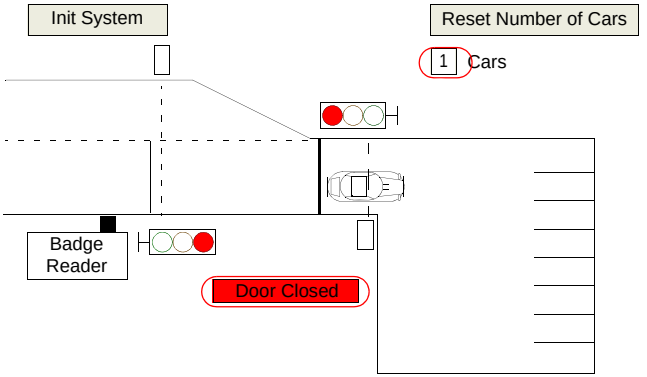


Forth Step: Car Activates the Exit Sensor

Switch on the input `Car_Exit_Sensor` (`%I0.1.2`) to make the car appear in front of the exit sensor.
Once you switch on the input `Car_Exit_Sensor` (`%I0.1.2`), the door closes and the counter counts:



Name	Value	Type	Comment
Badge_Inserted	0	EBOOL	Input "badge inserted"
Car_Entry_Sensor	0	EBOOL	Input "entry sensor"
Car_Exit_Sensor	1	EBOOL	Input "exit sensor"
Door Close Cmd	0	EBOOL	Output "Door Close Command"
Door Open Cmd	0	EBOOL	Output "Door Open Command"
Light_Door_OpenClose	0	EBOOL	Output "Entry and Exit lights flashing"
Light Entry Green	0	EBOOL	Output "Green Entry Light"
Light Entry Red	1	EBOOL	Output "Red Entry Light"
Light Exit Green	0	EBOOL	Output "Green Exit Light"
Light Exit Red	1	EBOOL	Output "Red Exit Light"
Number_Of_Cars	1	INT	Number of cars in the parking (counter)
Garage_Full	0	EBOOL	Garage is full (when Number_Of_Cars >



Fifth Step: Car Is Parked in the Garage

Switch off the input `Car_Exit_Sensor` (`%I0.1.2`) to make the car disappear. The car is parked in the garage.
You have entirely simulated a car parking in the garage.

Other Operating Modes

To simulate other operating modes, see the timing diagrams of the other operating modes (see *Application Operating Modes, p. 31*).

Other Functionalities

3

At a Glance

Subject of this Chapter

This chapter describes the other functionalities which could be easily performed with Unity Pro.

What's in this Chapter?

This chapter contains the following topics:

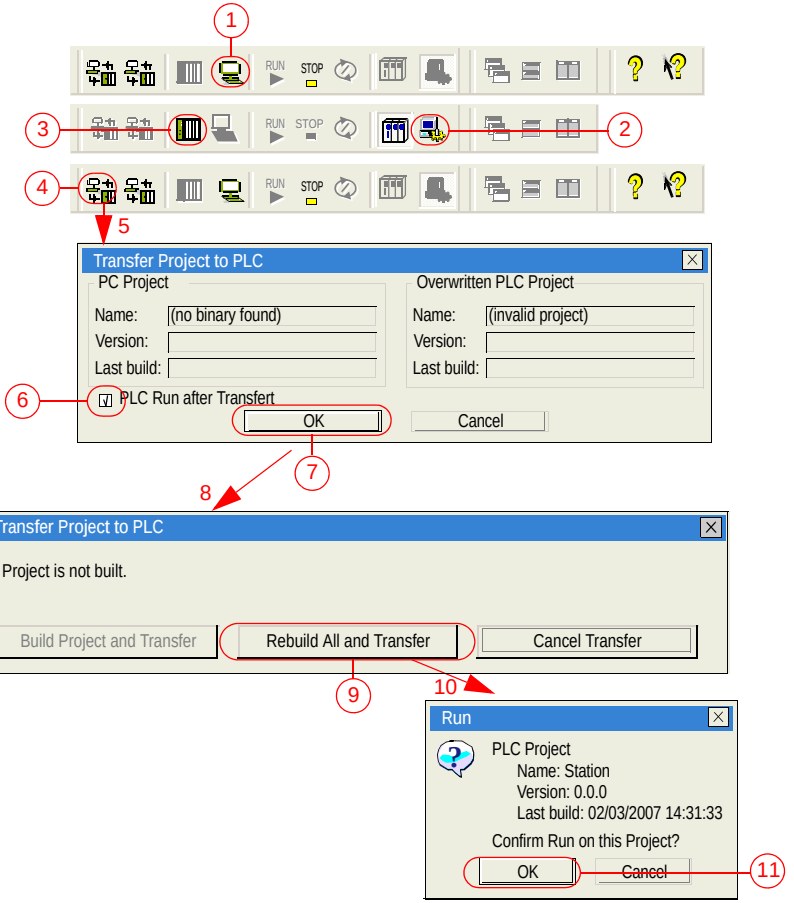
Using the PLC Simulator

PLC Simulator

The PLC simulator enables error searches to be carried out in the project without being connected to a real PLC.
As you simulated and debugged the application example with the PLC platform, it is possible to use the PLC simulator to simulate the application.

Setting Simulator Mode

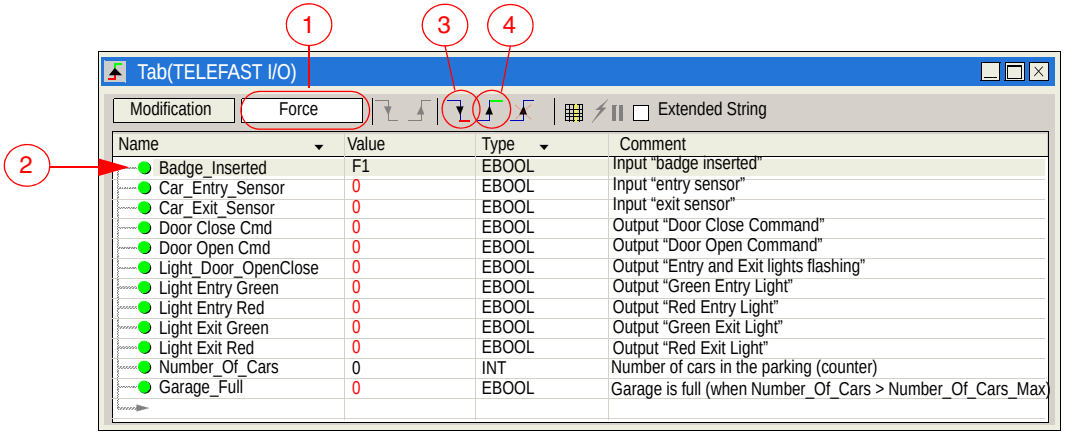
Switch standard mode to simulator mode and load the project in the PLC simulator:



1- Click on the icon Disconnect in the toolbar
2- Click on the icon Simulator mode in the toolbar if you are in Standard mode
3- Click on the icon Connect in the toolbar
4- Click on the icon Download project
5- The Tranfer Project to PLC screen appears
6- Activate the PLC run after Transfer checkbox
7- Click on the button Transfer
8- The Tranfer Project to PLC screen appears
9- Click on the button Rebuild All and Transfer
10- The Run screen appears
11- Click on the button OK to set the simulator to RUN mode

Simulating the Application by Using the PLC Simulator

The application is executed by the PLC simulator. For simulating the application you cannot use TELEFAST module because the application is no longer executed by the PLC platform. Therefore, you have to use the animation table to force the discrete inputs:



1- Click on the button Force to set the force mode
2- Select the input you want to force (among the three inputs)
3- Click on this icon to force to 0 the selected input
Or
4- Click on this icon to force to 1 the selected input

For simulating the application by using the PLC simulator, follow the same procedures you have performed by using the PLC platform (see *Simulating and Debugging the Application, p. 83*).

Switch to offline mode by clicking on the icon Disconnect in the toolbar. The purpose is to continue programming in offline mode. Indeed, the import function is available in offline mode only.

Programming a Functional Module Door1

Introduction

A functional module is a group of program elements intended to perform a PLC function. We propose to create and program a functional module `Door1` to group all program elements intended to perform the garage management.
The functional module `Door1` will group the following elements:

- The section in LD language
- The section in ST language
- The animation table
- The operator screen

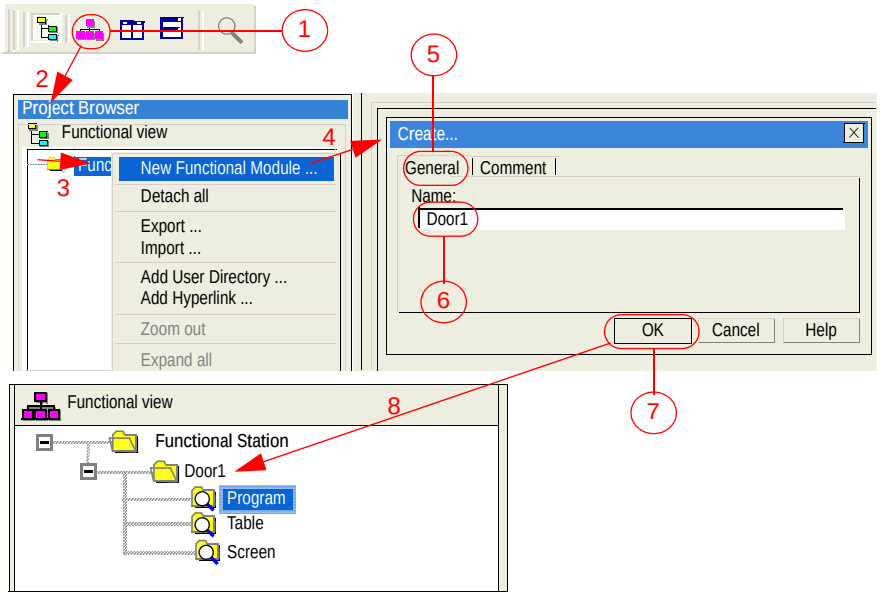
Programming Methodology

To implement the functional module `Door1`, you are going to perform the following steps:

- Create the functional module.
- Locate the various elements of the application to the functional module.

Creating the Functional Module Door1

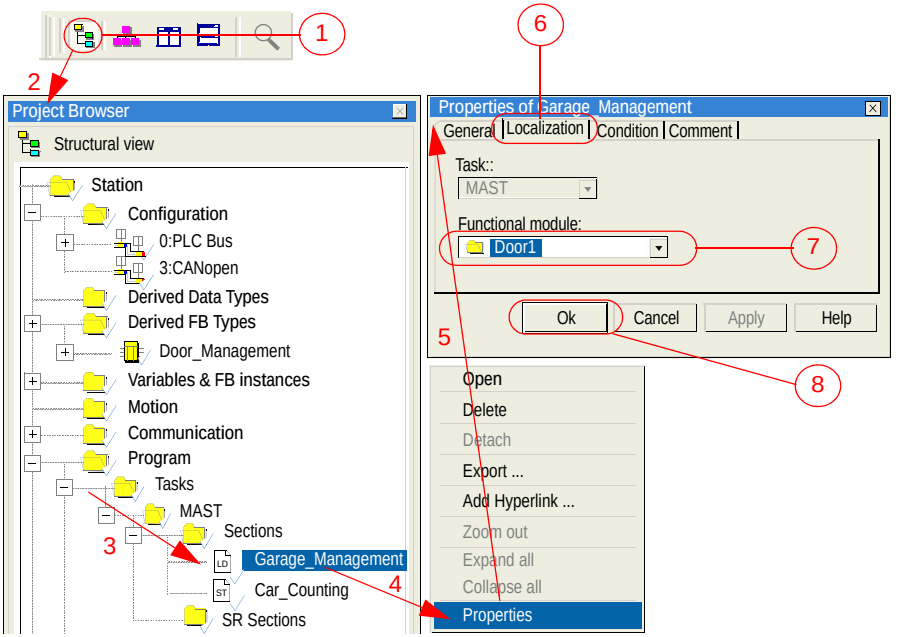
Create the functional module `Door1`:



1- Click on the Functional View icon
2- The functional view appears
3- Right click on the Functional Station directory and select New Functional Module
4- The Create screen appears
5- Click on the General tab
6- Enter Door1 in the Name field
7- Click on the OK button to confirm
8- The functional module Door1 is created and is displayed in the functional view

Locating the Sections

Locate the section `Garage_Management` to the functional module `Door1`:

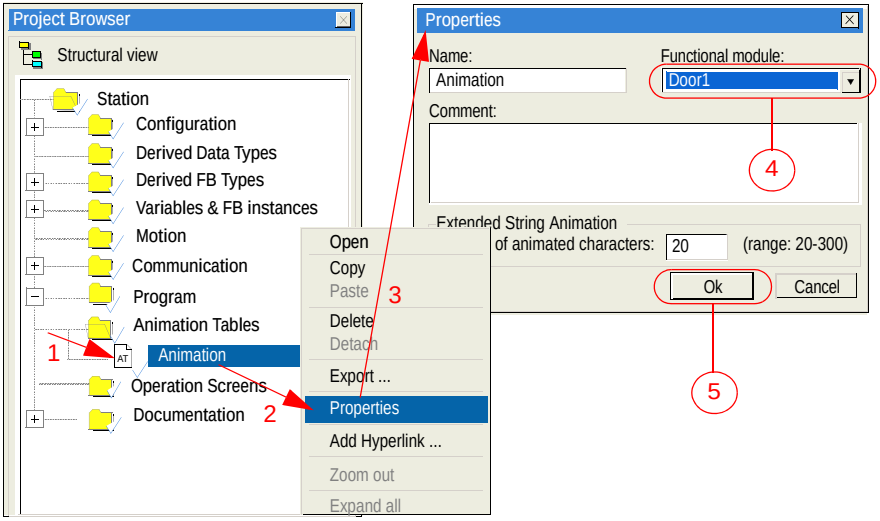


1- Click on the Structural View icon
2- The structural view appears
3- Expand the Station/Program/Tasks/MAST/Sections directory
4- Right click on the section Garage_Management and select Properties
5- The Properties screen appears
6- Click on the Localization tab
7- Select Door1 from the Functional Module zone
8- Click on the button OK to confirm

Note: Repeat this procedure to locate the section `Car_Counting`.

Locating the Animation Table

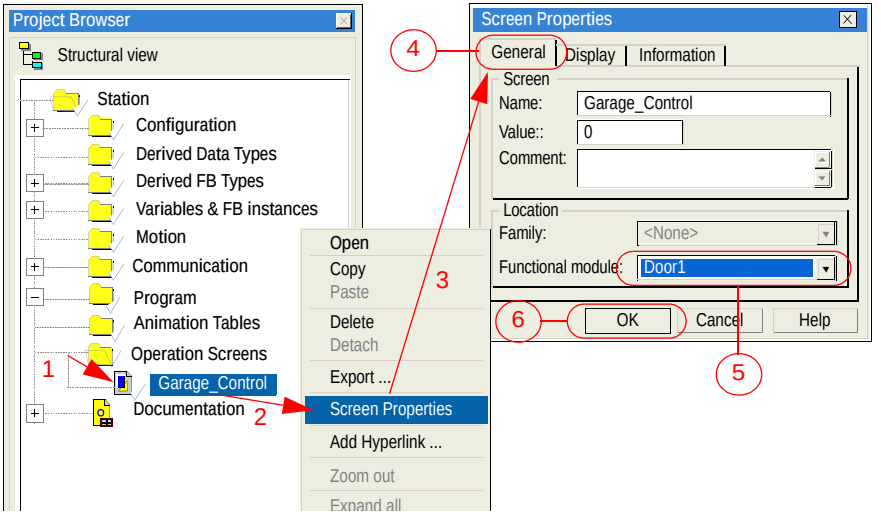
Locate the animation table to the functional module `Door1`:



1- Expand the Station/Animation Tables directory
2- Right click on the animation table and select Properties
3- The Properties screen appears
4- Select Door1 from the Functional Module zone
5- Click on the button OK to confirm

Locating the Operator Screen

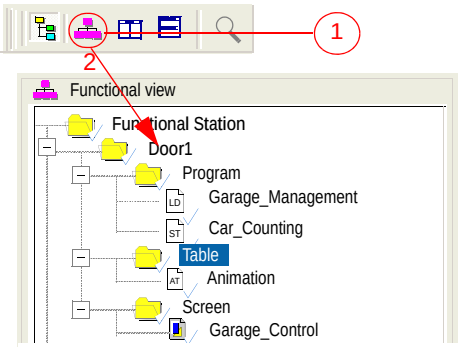
Locate the operator screen to the functional module `Door1`:



1- Expand the Station/Operator Screens directory
2- Right click on the operator screen and select Screen properties
3- The Screen Properties screen appears
4- Click on the General tab
5- Select Door1 from the Functional Module zone
6- Click on the button OK to confirm

Result: Functional Module Door1 Implemented

You can check the functional module `Door1` is implemented by opening the functional view:



1- Click on the Functional View icon
2- The functional view appears. The 2 sections, the animation table and the operator screen are attached to the functional module Door1.

Other Way to Program the Functional Module Door1

There is another easier and faster way to attach elements to a functional module. First, display both structural view and functional view by clicking on the icon in the toolbar:

Then drag and drop the various elements from the structural view to the functional module `Door1`.

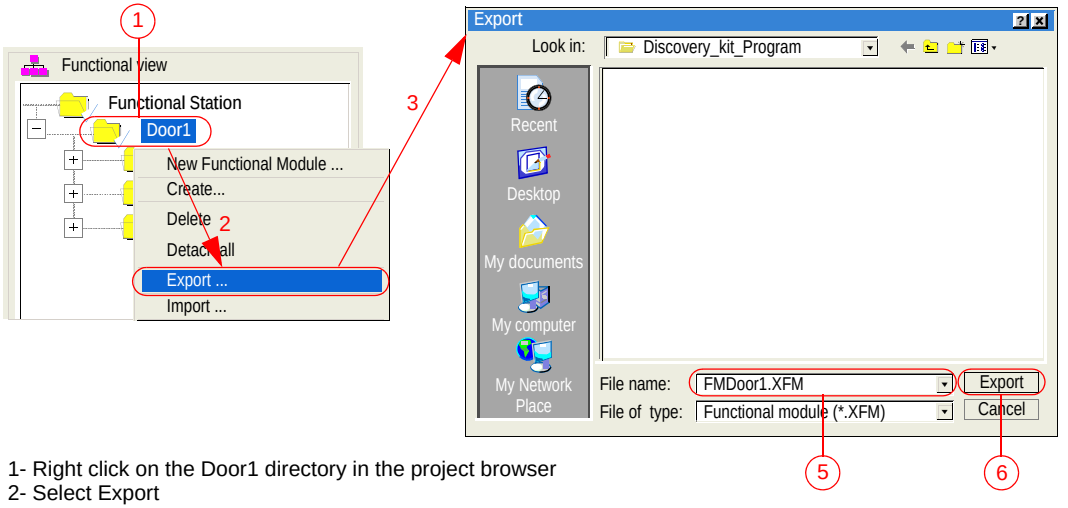
Programming a Second Door

Introduction

The purpose of a second door is to have a second access to the garage. The second door should have the same behavior as the door already implemented in the application.
To program a second door, many programming changes are necessary. For instance, every input/output, section `Garage_Management` etc. must be duplicated.
Unity Pro functionalities make these changes very easy. Indeed, we are going to create the functional module `Door2` by importing the functional module `Door1` and by keeping or duplicating `Door1` elements. The functional module `Door2` will contain all the functions related to the second door. The main changes will be automatically performed when importing the functional module `Door1`. Some changes will remain undone and should be made manually.

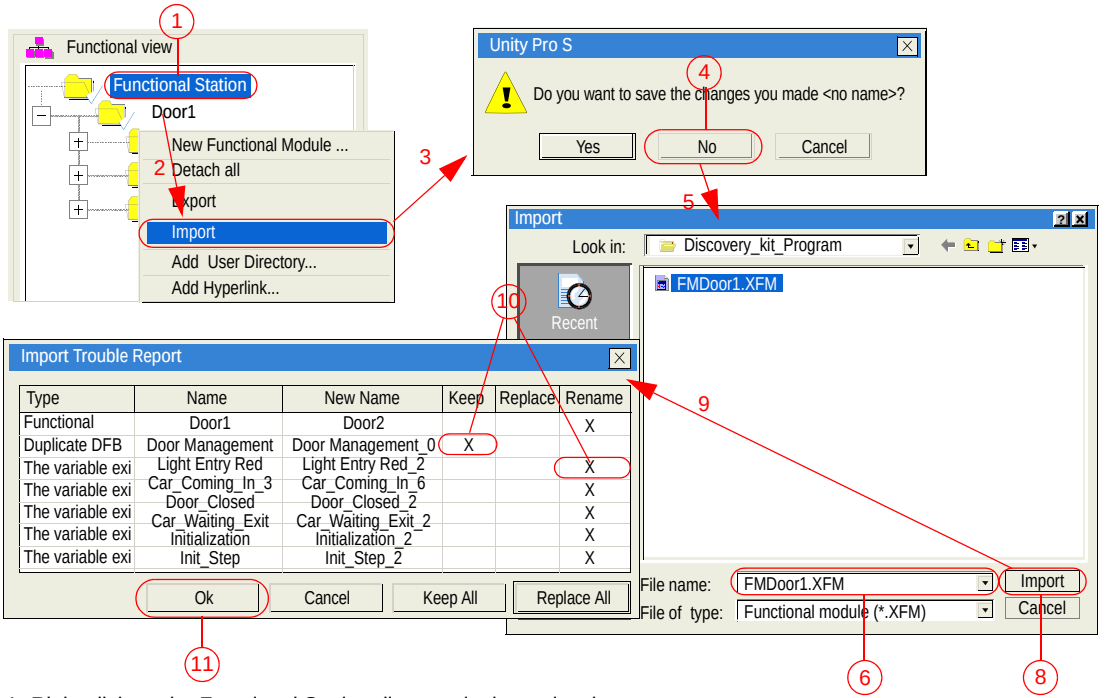
Duplicating the Functional Module Door1

First, export the functional module `Door1` to a target file whose you can choose the location:



1- Right click on the Door1 directory in the project browser
2- Select Export
3- The Export screen appears
4- Choose the location of the exported file (it can be any location available on your computer)
5- Enter the name of the exported file
6- Click on the button Export

To duplicate the functional module Door1, you have to import the XFM file you have previously exported. An Import Trouble Report dialog box will enable you to keep or to duplicate the elements of the functional module Door1:



1- Right click on the Functional Station directory in the project browser
2- Select Import
3- The following screen appears
4- Click on the button No
5- The Import screen appears
6- Select XFM in the Files of type field
7- Browse the XFM file you have previously exported (corresponding to the functional module Door1)
8- Click on the button Import
9- An Import Trouble Report dialog box appears and enables to duplicate or to keep Door1 elements
By default, the column Rename is checked for each element (except Door1 and Animation)
10- Double click the column Keep to keep the element or let the column Rename checked to duplicate the element. See the table below to know the elements to keep or to rename.
11- Click on OK to import the functional module

The following table describe the Door1 elements to keep or to duplicate (to rename):

Type	Name	Keep	Rename
Functional	Door1		X
Duplicate DFB	Door_Management	X	
The variable exists already	Number_Of_Cars	X	
The variable exists already	RAZ_Number_Of_Cars	X	
The variable exists already	Number_Of_Cars_Max	X	
The variable exists already	COUNTER	X	
Duplicate identifier	Garage_Management		X
Duplicate identifier	Car_Counting	X	
Animation Tables	Animation		X

For all the elements which are not present in the table above, let the column Rename checked. Do not change the default name of all the renamed elements.

Notes

To program the second door, some manual changes should be done after importing the functional module Door1:

- Assign new addresses to all the duplicated discrete inputs and outputs.
- Merge the buttons of the original operator screen and the duplicated one.
- Assign duplicated variables to the inputs CU and CD of COUNTER function for the section Car_Counting (common section to the functional modules Door1 and Door2).

Note: There is nothing to change in the section duplicated from Garage_Management and in the duplicated animation table because these elements are automatically programmed with the duplicated variables.

Discovery Kit Troubleshooting

4

Discovery Kit Troubleshooting

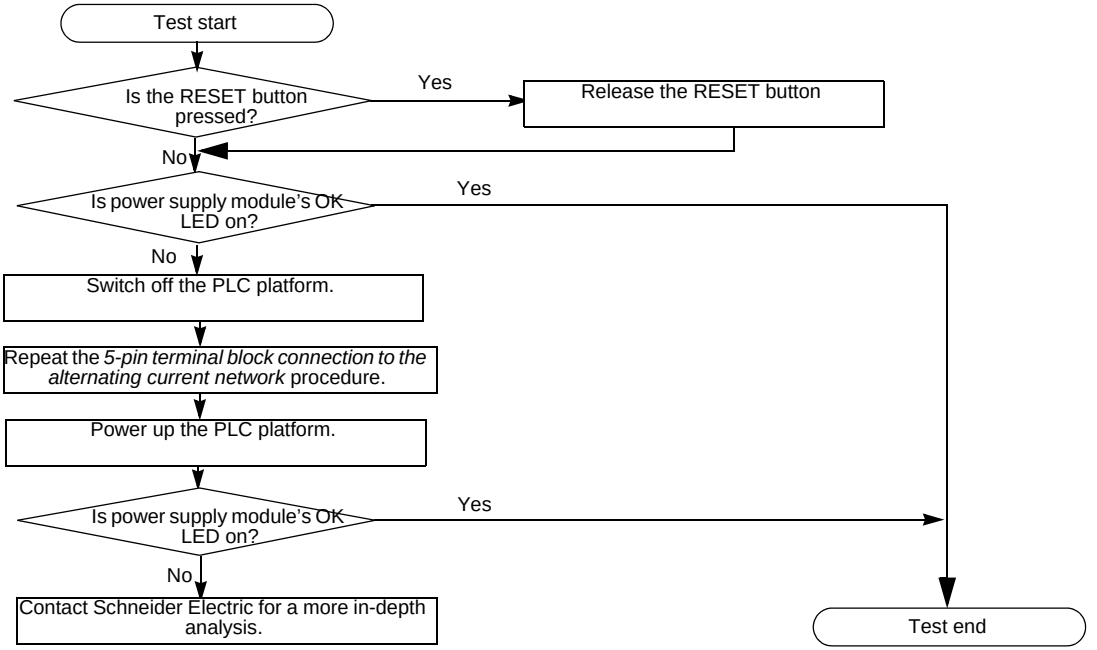
Introduction

The following pages present procedures for troubleshooting the discovery kit platform:

- Power supply module OK LED test
- Power supply module 24V LED test
- TELEFAST module 24V LED test
- Input activation test
- Terminal connection test
- Procedure for disabling the discrete module supply monitoring

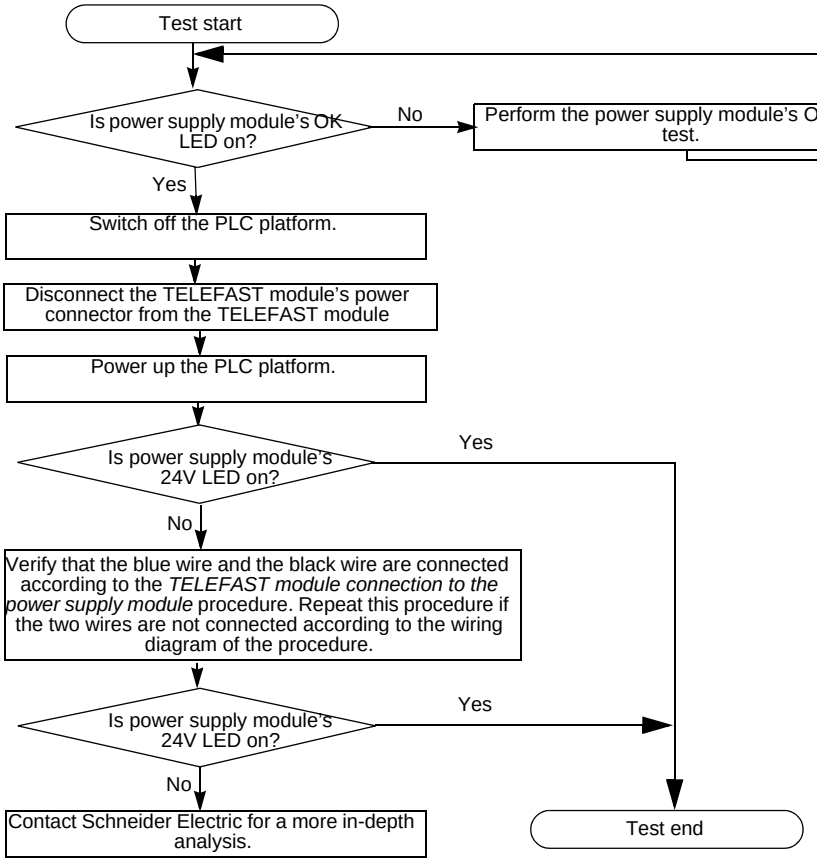
Power Supply Module OK LED Test

Perform the following test if the power supply module OK LED is off when the PLC platform is powered:



Power Supply Module 24V LED Test

Perform the following test if the power supply module's 24V LED is off when the PLC platform is powered:

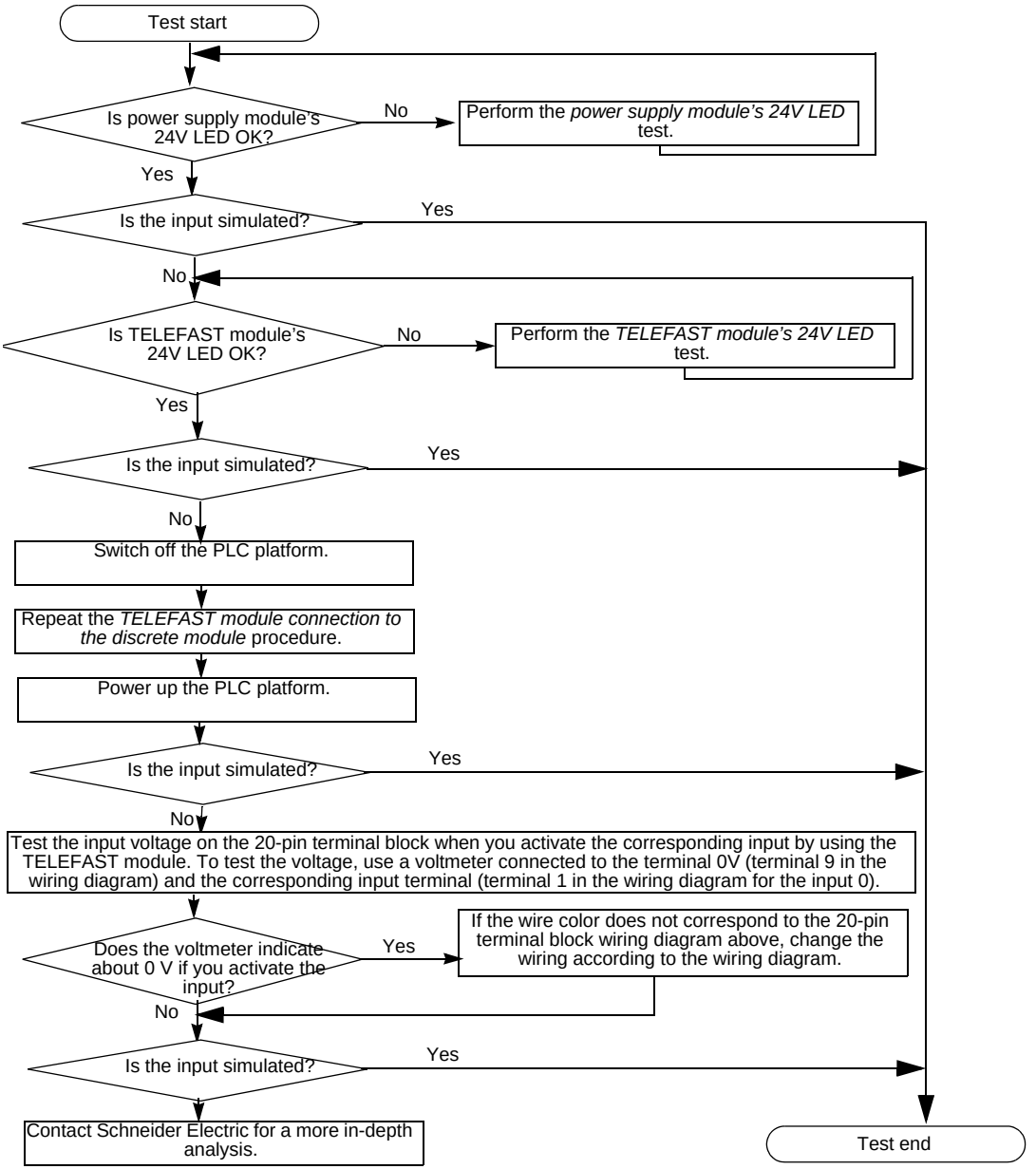


The TELEFAST module connection to the power supply module procedure is described in the part installing the discovery kit (see *Second Step: Connecting TELEFAST Module to the Power Supply Module, p. 16*).

The 20-pin terminal block wiring diagram is available in the next pages of this chapter.

Inputs Activation Test

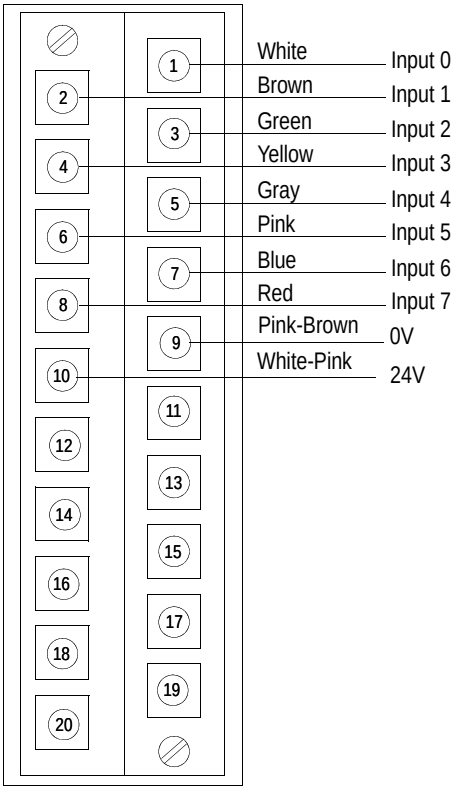
Perform the following test if one or more discrete inputs are not activated by the TELEFAST module when the PLC platform is powered.
The following test is valid for each input which is not activated:



The TELEFAST module connection to the power supply module procedure is described in the part installing the discovery kit (see *Second Step: Connecting TELEFAST Module to the Power Supply Module, p. 16*).
The 20-pin terminal block wiring diagram is available in the next pages of this chapter.

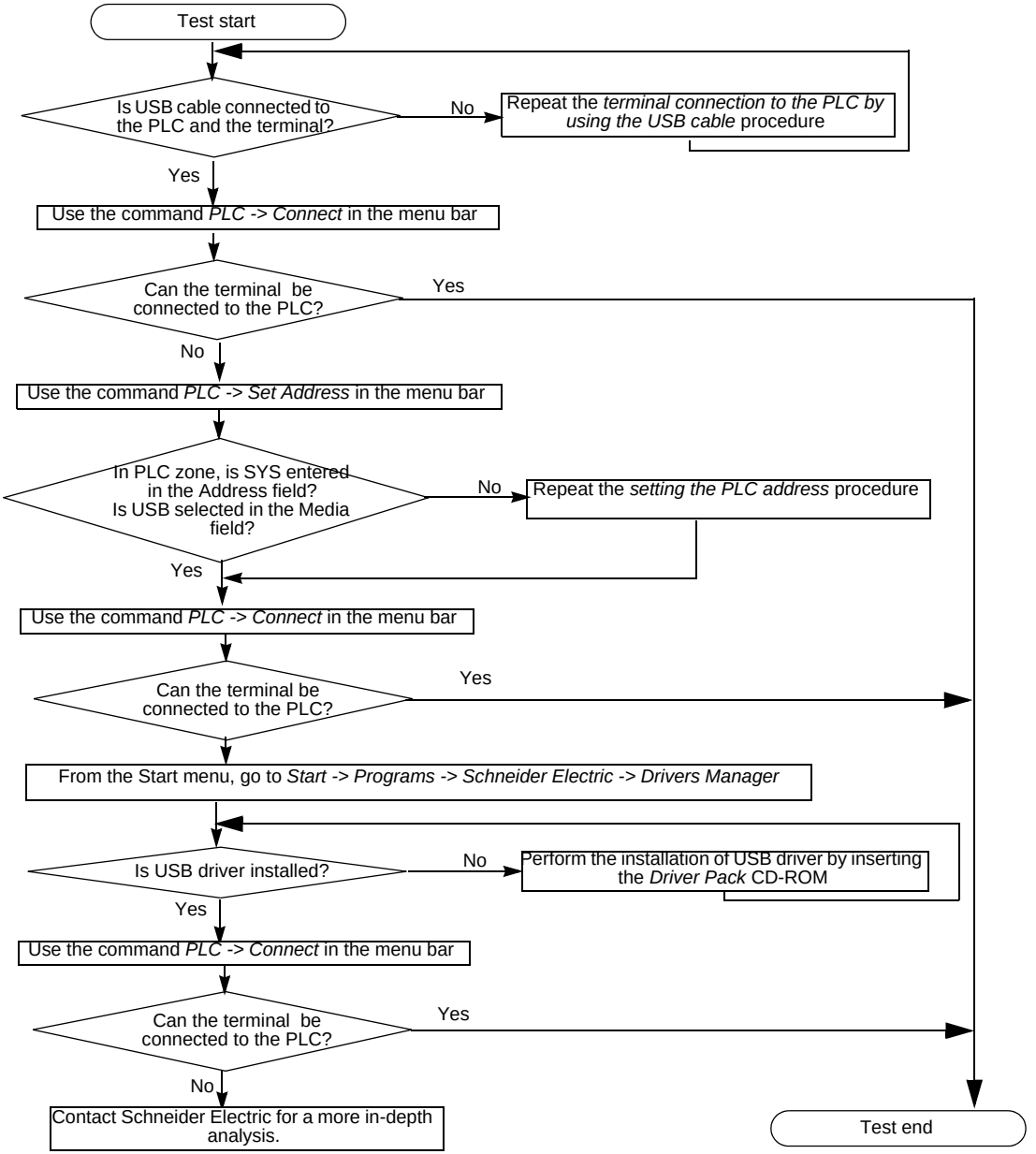
20-Pin Terminal Block Wiring Diagram

The figure below shows the 20-pin terminal block wiring. It represents the front view of the terminal block when it is connected to the discrete module:



Terminal Connection Test

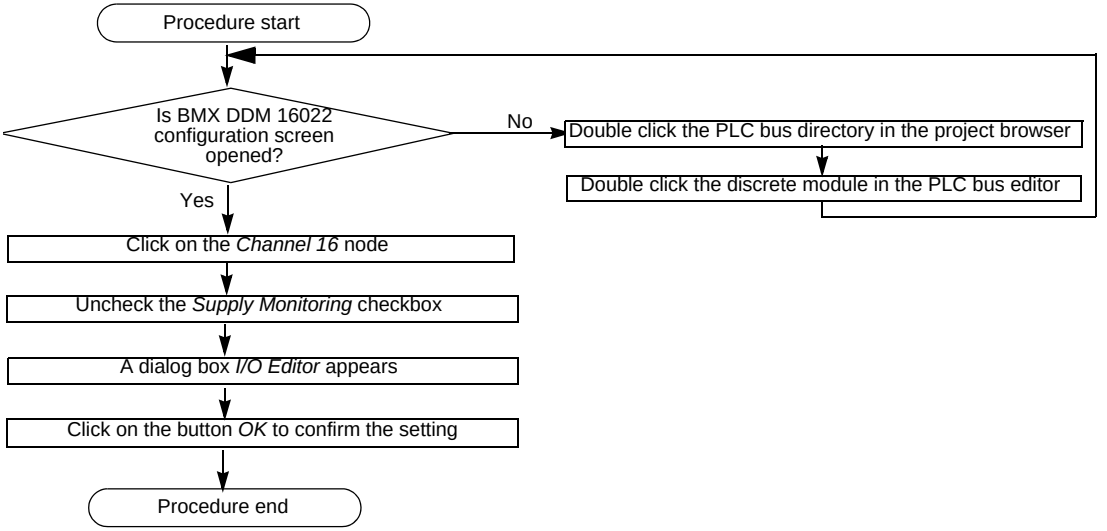
Perform the following test if the terminal cannot connect to the PLC when Unity Pro is launched and the PLC platform is powered:



The terminal connection to the PLC by using USB cable procedure is described in the part installing the discovery kit (see *Checking the Discovery Kit Configuration Normally Works, p. 23*).
The setting the PLC address procedure is described in the part installing the discovery kit (see *Checking the Discovery Kit Configuration Normally Works, p. 23*).
To check USB driver is installed, from the Start menu go to **Programs** → **Schneider Electric** → **Communication Drivers** → **Drivers Manager**. A screen appears. Click on the DRIVERS Manager tab. The installed drivers are displayed. If USB driver is absent in the list of installed drivers, it means the USB driver is not installed.
For further information about the installation of USB driver procedure, see Unity Pro Software/Communication Drivers/USB driver/Installation/How to install the driver in Unity Pro documentation.

How to Disable the Discrete Module Supply Monitoring

When the PLC platform is powered or when you will transfer the project from the terminal to the PLC, the processor and discrete module I/O LEDs are switched on. Indeed, the checkbox **Supply monitoring** is checked by default for the channel group 16 of the discrete module and there is no external power supply for the channel group 16. Therefore, to switch off the I/O LEDs, you have to disable the supply monitoring for the discrete channel group 16:



Appendices

At a Glance

Subject of this Appendix

This appendix presents how to import the various elements of the discovery kit application and the programming of the sections into Unity Pro.

What's in this Appendix?

The appendix contains the following chapters:

Importing the Elements of the Discovery Kit Application

A

Importing the Various Elements of the Discovery Kit Application

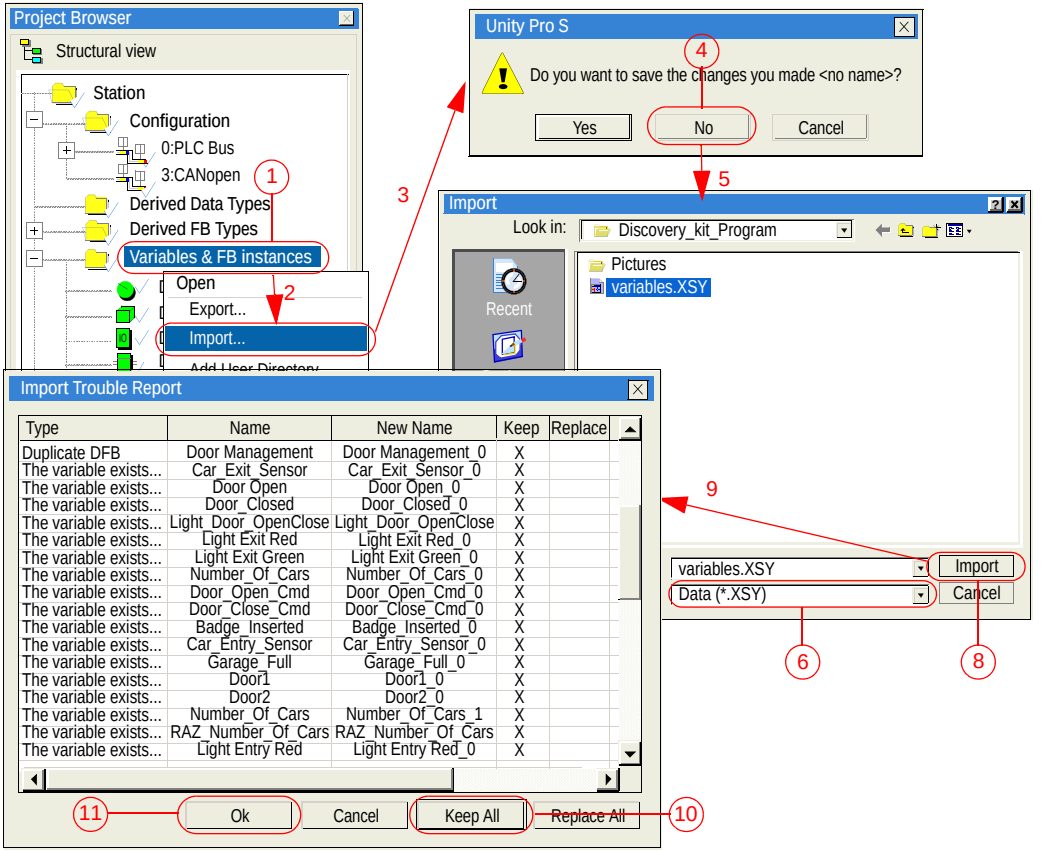
Overview

The following pages explain how to import the various elements of the discovery kit application:

- All the variables of the application.
- The section `Garage_Management` in LD language without the DFB instance. This file is to be imported before you instanciate the DFB.
- The section `Garage_Management` in LD language including the DFB instance. You can import this file once the DFB is programmed and instantiated.
- The DFB type `Door_Management`.
- The section `Car_Counting` in ST language.
- The entire application.

Importing the Variables

All the variables of the application have a source file whose file extension is XSY. This file is available on the Discovery Kit CD-ROM in the directory `Discovery Kit Program` and is named `Variables.XSY`. It enables to import all the variables and all the discrete inputs/outputs of the application:

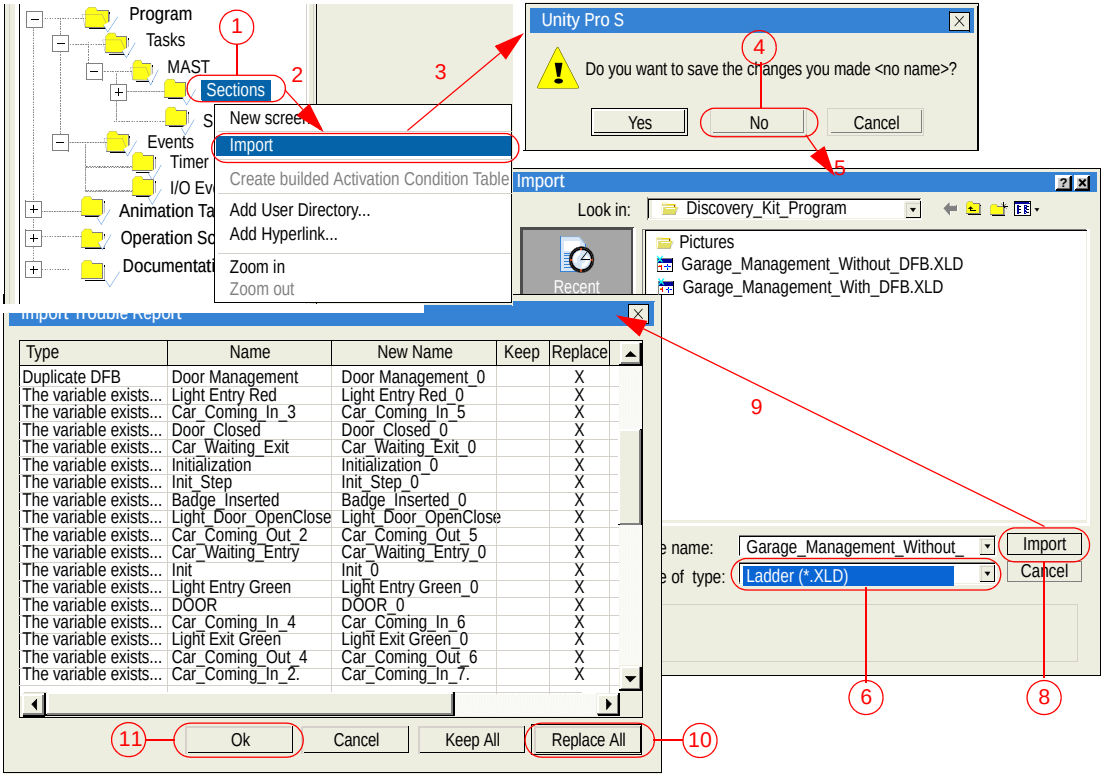


1- Right click on the Variables & FB instances directory in the project browser
2- Click on the Import command
3- The following screen appears
4- Click on the button No
5- The Import screen appears
6- Select XSY in the Files of type field
7- Browse the XSY file to be imported
8- Click on the button Import
9- If you have declared one or several variables, an Import Trouble Report dialog box appears
10- Click on the button Keep All
11- Click on the button OK to confirm

Result: The variables are imported. To view all the variables and all the inputs/outputs, double click the **Variables & FB instances** directory in the project browser.

Importing the Section Garage_Management

The section `Garage_Management` has two source files whose file extension is `XLD`. These files are available on the Discovery Kit CD-ROM in the directory `Discovery Kit Program` and are named `Garage_Management_With_DFB.XLD` and `Garage_Management_Without_DFB.XLD`. The files enable to import the section `Garage_Management`:

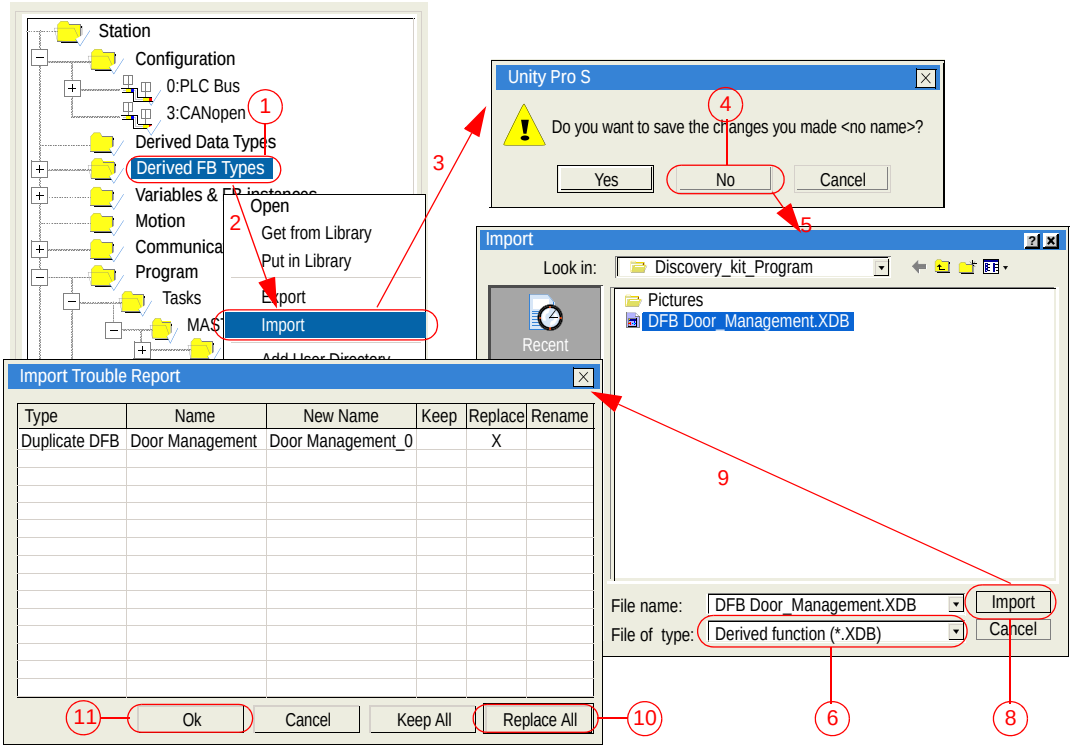


1- Right click on the Station/Program/Tasks/MAST/Sections directory in the project browser
2- Select the Import command
3- A screen appears
4- Click on the button NO
5- The Import screen appears
6- Select XLD in the File type field
7- Browse the XLD file to be imported
8- Click on the button Import
9- The Import Trouble Report screen appears. It displays all the conflicts between the imported file and the existing project.
10- Click on the button Replace All. The existing section will be replaced with the imported file.
11- Click on the button OK to confirm

Result: The `Garage_Management` section is imported. To open the `Garage_Management` section, double click the **Station/Program/Tasks/MAST/Sections/Garage_Management** directory in the project browser.

Importing the DFB Type

The DFB type `Door_Management` has a source file whose file extension is `XDB`. This file is available on the Discovery Kit CD-ROM in the directory `Discovery Kit Program` and is named `DFB Door_Management.XDB`. It enables to import the DFB type `Door_Management`:

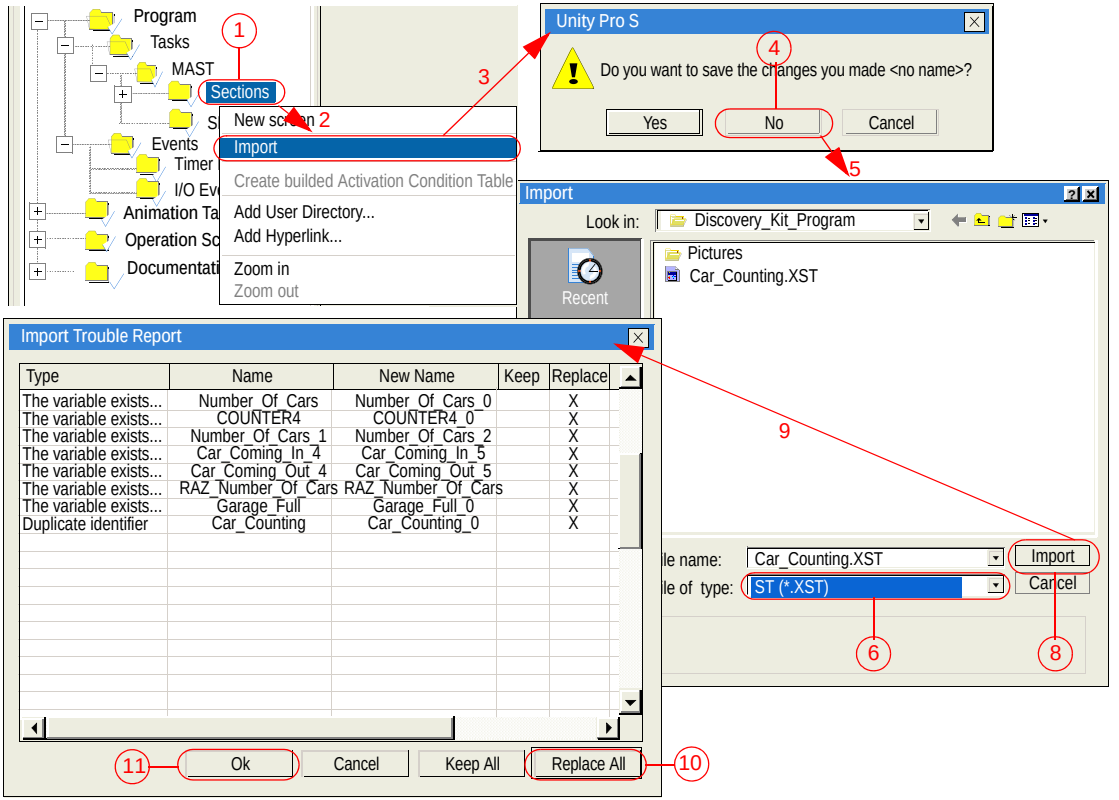


1- Right click on the Station/Derived FB Types directory in the project browser
2- Select the Import command
3- A screen appears
4- Click on the button NO
5- The Import screen appears
6- Select XDB in the File type field
7- Browse the XDB file to be imported
8- Click on the button Import
9- The Import Trouble Report screen appears. It displays all the conflicts between the imported file and the existing project
10- Click on the button Replace All. The existing DFB will be replaced with the imported DFB.
11- Click on the button OK to confirm

Result: The `Door_Management` DFB type is imported. To open the `Door_Management` DFB type, double click the **Station/Derived FB Types** directory in the project browser.

Importing the Section Car_Counting

The section `Car_Counting` has a source file whose file extension is `XST`. This file is available on the Discovery Kit CD-ROM in the directory `Discovery Kit Program` and is named `Car_Counting.XST`. It enables to import the section `Car_Counting`:

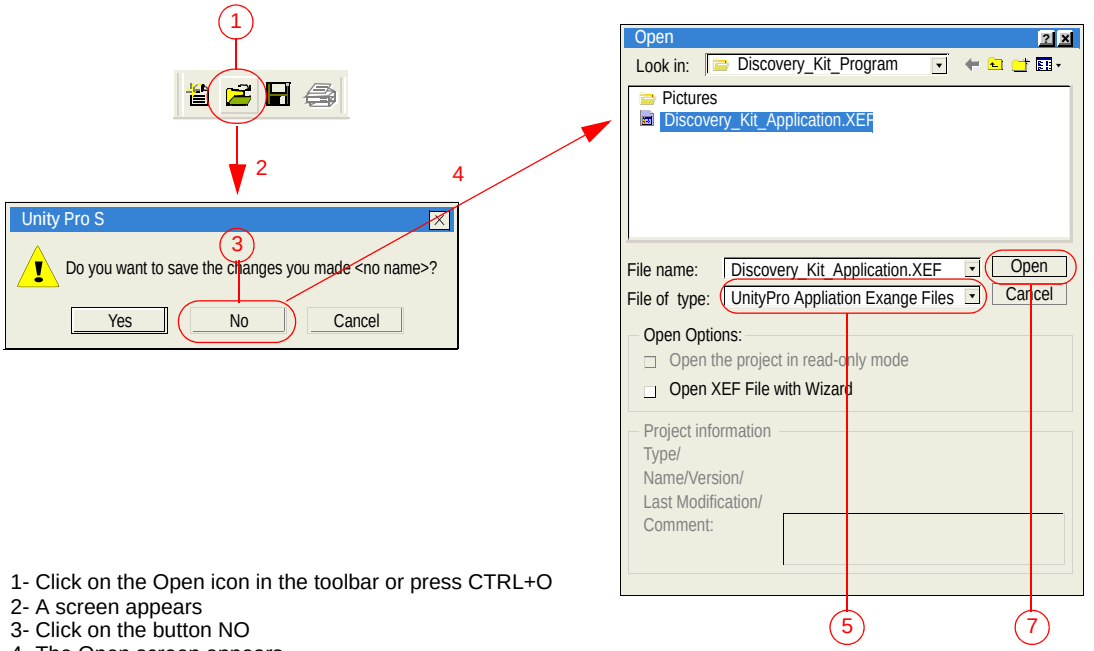


1- Right click on the Station/Program/Tasks/MAST/Sections directory in the project browser
2- Select the Import command
3- A screen appears
4- Click on the button NO
5- The Import screen appears
6- Select XST in the File type field
7- Browse the XST file to be imported
8- Click on the button Import
9- The Import Trouble Report screen appears. It displays all the conflicts between the imported file and the existing project
10- Click on the button Replace All. The existing section will be replaced with the imported file
11- Click on the button OK to confirm

Result: The `Car_Counting` section is imported. To open the `Car_Counting` section, double click the **Station/Program/Tasks/MAST/Sections/Car_Counting** directory in the project browser.

Importing the Entire Project

The project has a source file whose file extension is XEF. This file is available on the Discovery Kit CD-ROM in the directory `Discovery Kit Program` and is named `Discovery_Kit_Application.XEF`. It enables to import the entire application:



1- Click on the Open icon in the toolbar or press CTRL+O
2- A screen appears
3- Click on the button NO
4- The Open screen appears
5- Select XEF in the File type field
6- Browse the XEF file to be imported
7- Click on the button Open

Result: The entire application is imported. To view the various elements of the application, browse the elements in the project browser.

Application Sections

B

At a Glance

At a Glance

The following pages describe all the sections which are parts of the application.

What's in this Chapter?

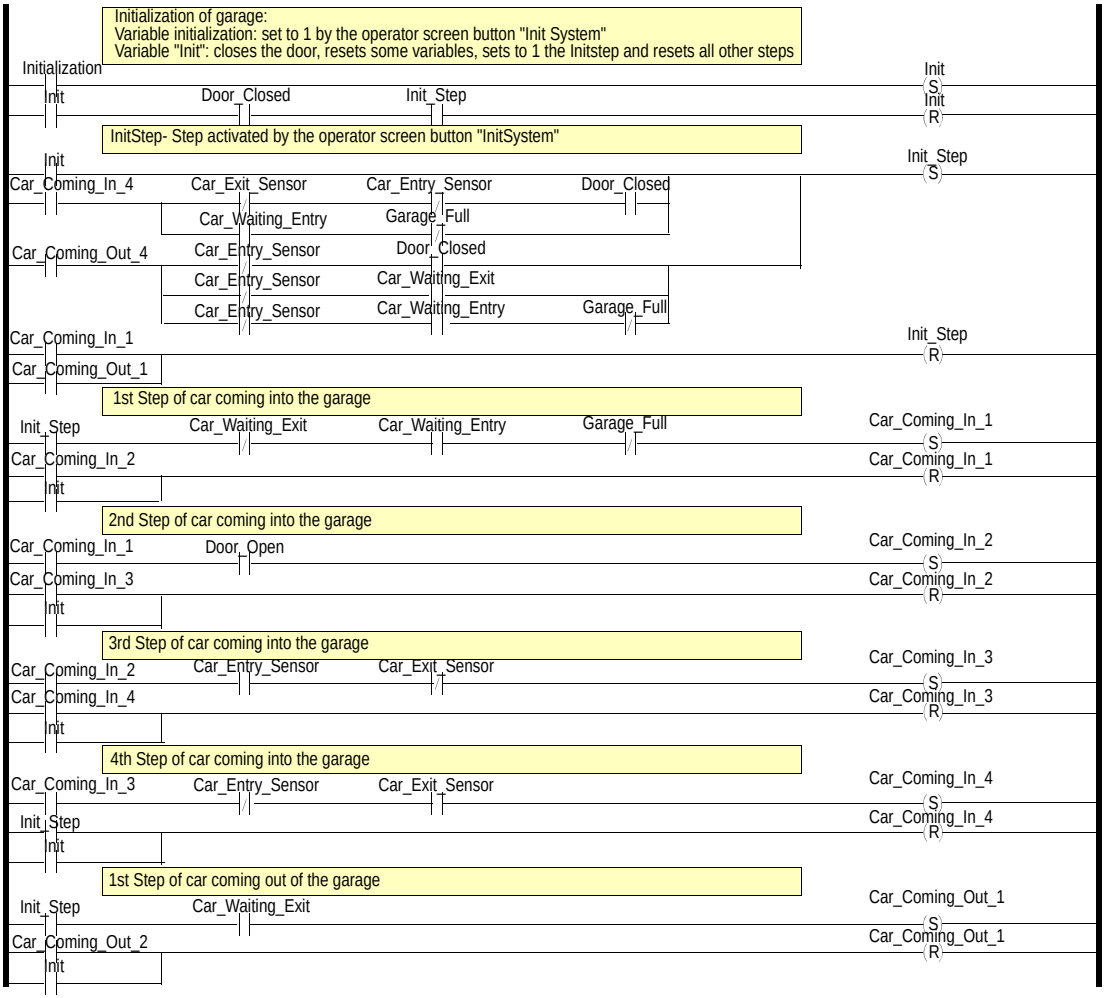
This chapter contains the following topics:

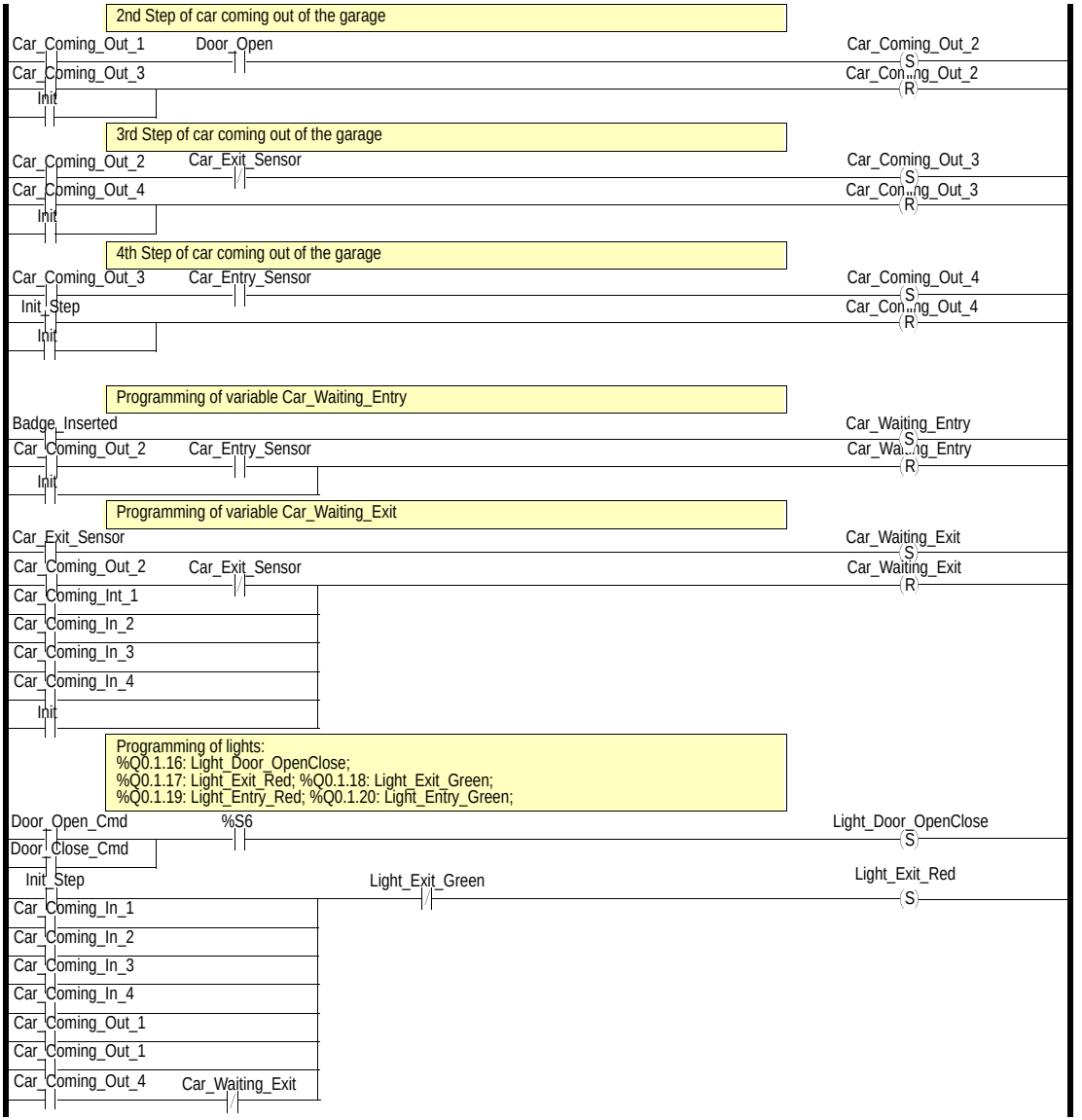
Garage_Management Section

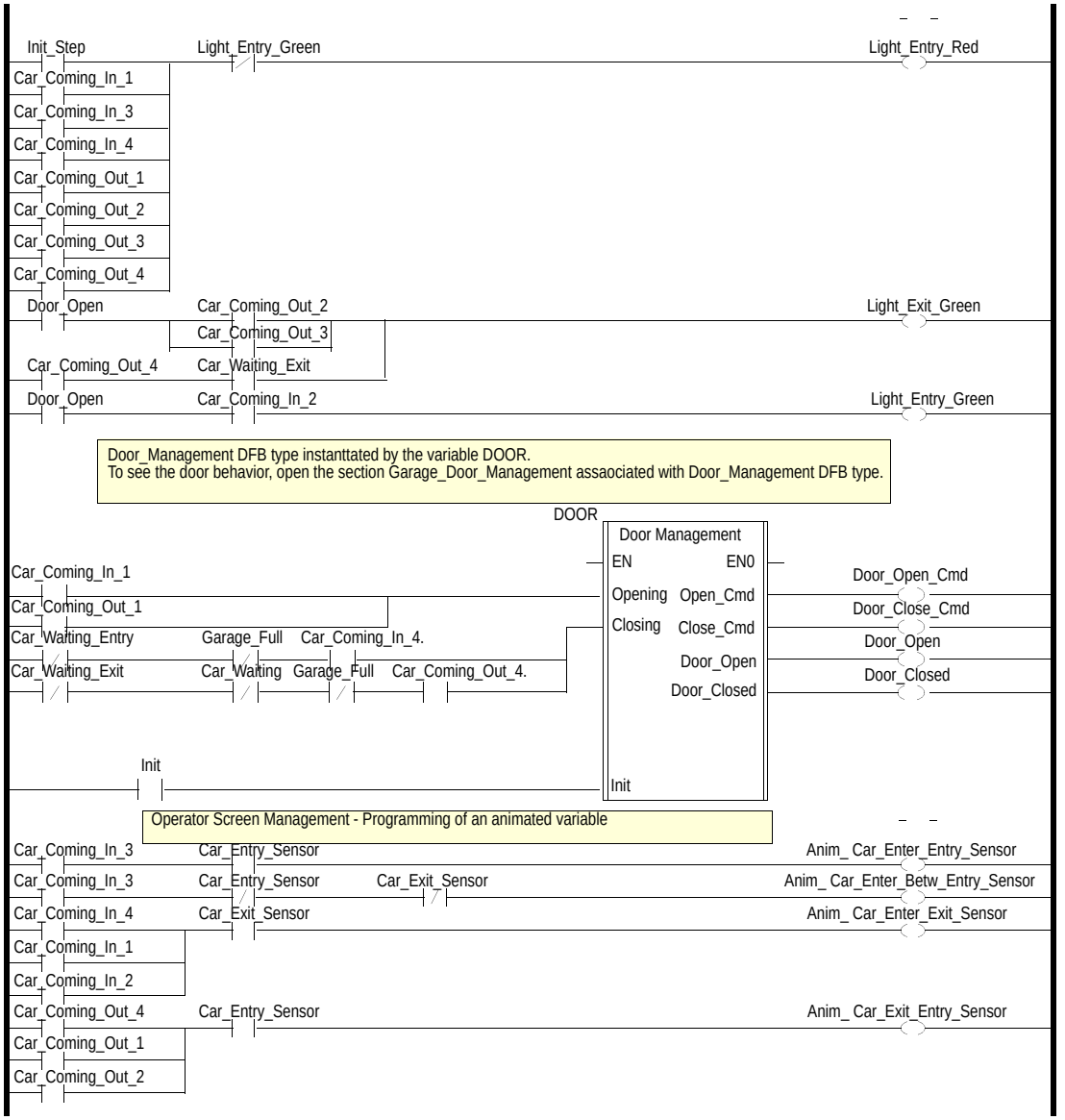
At a Glance

This section is implemented in LD language. It manages all the functions related to the garage.

Section Program




2nd Step of car coming out of the garage
Car_Coming_Out_1
Door_Open
Car_Coming_Out_2
S
Car_Coming_Out_3
Car_Coming_Out_2
R
Init
3rd Step of car coming out of the garage
Car_Coming_Out_2
Car_Exit_Sensor
Car_Coming_Out_3
S
Car_Coming_Out_4
Car_Coming_Out_3
R
Init
4th Step of car coming out of the garage
Car_Coming_Out_3
Car_Entry_Sensor
Car_Coming_Out_4
S
Init_Step
Car_Coming_Out_4
R
Init
Programming of variable Car_Waiting_Entry
Badge_Inserted
Car_Waiting_Entry
S
Car_Coming_Out_2
Car_Entry_Sensor
Car_Waiting_Entry
R
Init
Programming of variable Car_Waiting_Exit
Car_Exit_Sensor
Car_Waiting_Exit
S
Car_Coming_Out_2
Car_Exit_Sensor
Car_Waiting_Exit
R
Car_Coming_Int_1
Car_Coming_In_2
Car_Coming_In_3
Car_Coming_In_4
Init
Programming of lights:
%Q0.1.16: Light_Door_OpenClose;
%Q0.1.17: Light_Exit_Red; %Q0.1.18: Light_Exit_Green;
%Q0.1.19: Light_Entry_Red; %Q0.1.20: Light_Entry_Green;
Door_Open_Cmd
%S6
Light_Door_OpenClose
S
Door_Close_Cmd
Init_Step
Light_Exit_Green
Light_Exit_Red
S
Car_Coming_In_1
Car_Coming_In_2
Car_Coming_In_3
Car_Coming_In_4
Car_Coming_Out_1
Car_Coming_Out_1
Car_Coming_Out_4
Car_Waiting_Exit


Init_Step
Light_Entry_Green
Light_Entry_Red
Car_Coming_In_1
Car_Coming_In_3
Car_Coming_In_4
Car_Coming_Out_1
Car_Coming_Out_2
Car_Coming_Out_3
Car_Coming_Out_4
Door_Open
Car_Coming_Out_2
Car_Coming_Out_3
Light_Exit_Green
Car_Coming_Out_4
Car_Waiting_Exit
Door_Open
Car_Coming_In_2
Light_Entry_Green
Door_Management DFB type instanttated by the variable DOOR.
To see the door behavior, open the section Garage_Door_Management assaociated with Door_Management DFB type.
DOOR
Door Management
EN
EN0
Car_Coming_In_1
Opening
Open_Cmd
Door_Open_Cmd
Car_Coming_Out_1
Closing
Close_Cmd
Door_Close_Cmd
Car_Waiting_Entry
Garage_Full
Car_Coming_In_4.
Door_Open
Door_Open
Car_Waiting_Exit
Car_Waiting
Garage_Full
Car_Coming_Out_4.
Door_Closed
Door_Closed
Init
Init
Operator Screen Management - Programming of an animated variable
Car_Coming_In_3
Car_Entry_Sensor
Anim_ Car_Enter_Entry_Sensor
Car_Coming_In_3
Car_Entry_Sensor
Car_Exit_Sensor
Anim_ Car_Enter_Betw_Entry_Sensor
Car_Coming_In_4
Car_Exit_Sensor
Anim_ Car_Enter_Exit_Sensor
Car_Coming_In_1
Car_Coming_In_2
Car_Coming_Out_4
Car_Entry_Sensor
Anim_ Car_Exit_Entry_Sensor
Car_Coming_Out_1
Car_Coming_Out_2

Car_Counting Section

At a Glance

This section is implemented in ST language. It manages the car counter.

Section Program

```
('init of Number_Of_Cars_Max')
Number_Of_Cars_Max:=20;

COUNTER  (CU := Car_Coming_In_4,
    CD  :=  Car_Coming_Out_4,
    R :=  RAZ_Number_Of_Cars,
    LD  :=  0,
    PV  :=  Number_Of_Cars_Max,
    QU  => Garage_Full,
    CV  => Number_Of_Cars);
```

Garage_Door_Management Section

At a Glance

This section is implemented in FBD language. It is associated to the DFB type `Door_Management` which is instantiated by the variable `DOOR`, in the `Garage_Management` section.

Section Program