

ML 300

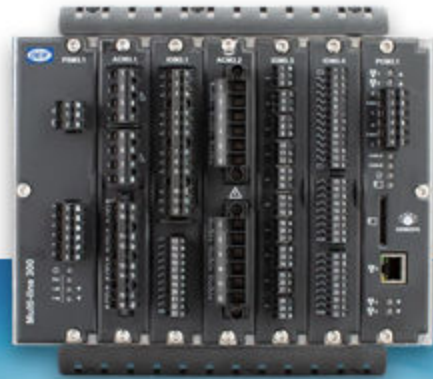
Multi-line 300 platform

CODESYS guidelines

4189341175C



Improve
Tomorrow



1. Introduction

1.1 About this document	4
1.1.1 Document overview.....	4
1.1.2 Software versions.....	4
1.1.3 Technical support.....	4
1.2 Warnings and safety	5
1.2.1 CustomLogic not available.....	5
1.2.2 Recommendations for data security.....	5
1.3 Legal information	5
1.3.1 Disclaimer.....	5
1.3.2 Trademark.....	5
1.3.3 Copyright.....	5

2. Get started with CODESYS

2.1 Multi-line 300 CODESYS functions	6
2.1.1 Multi-line 300 CODESYS functions.....	6
2.2 Software requirements	6
2.2.1 Software requirements.....	6
2.3 Download	7
2.3.1 Downloading the DEIF CODESYS software package.....	7
2.3.2 DEIF CODESYS software package contents.....	7
2.3.3 Download the DEIF CODESYS library package.....	8
2.3.4 DEIF CODESYS library package contents.....	8
2.4 Install	8
2.4.1 Install CODESYS Runtime on the controller.....	8
2.4.2 Install the device description in CODESYS.....	9
2.4.3 Install the ML 300 controller libraries in CODESYS.....	11

3. ML 300 CODESYS projects

3.1 Introduction	13
3.1.1 Introduction.....	13
3.2 Create a new project	13
3.2.1 Create a project file.....	13
3.2.2 CODESYS layout.....	14
3.2.3 Add the Multi-line 300 libraries to your application.....	16
3.3 Add the ML 300 function block	17
3.3.1 Introduction.....	17
3.3.2 Create a continuous function chart program.....	18
3.3.3 Add the ML 300 Read-Write function block.....	20
3.3.4 Add the ML 300 Read and ML 300 Write function blocks.....	22
3.3.5 ML300 function blocks' execution position.....	25
3.3.6 ML 300 function blocks' inputs and outputs.....	27
3.4 Communication with the controller	29
3.4.1 Introduction.....	29
3.4.2 Create a local gateway.....	30
3.4.3 Connect to the controller.....	32
3.5 Download the application to the controller	34
3.5.1 Pre-compile the application.....	34
3.5.2 Generate and download the application.....	34

3.5.3 Start and stop the application.....	35
3.6 Monitor the application.....	36
3.6.1 Introduction.....	36
3.6.2 Monitor in the working area.....	36
3.6.3 Monitor in watch windows.....	38
3.6.4 Write and force variables.....	41
4. Function blocks	
4.1 Version function block.....	43
4.1.1 Introduction.....	43
4.1.2 Add a version function block.....	43
4.1.3 Card_info function block overview.....	45
4.1.4 Software_info function block overview.....	46
4.1.5 Versions function block overview.....	47
4.2 I/O function block.....	48
4.2.1 Introduction.....	48
4.2.2 Add an I/O function block.....	49
4.2.3 Assign a CODESYS I/O function in the controller.....	52
4.3 Standard ML 300 functions.....	55
4.3.1 Introduction.....	55
4.3.2 Add standard ML 300 functions function blocks.....	55
4.3.3 Function conflicts.....	57
4.3.4 Alarm function block overview.....	59
4.3.5 Parameter function block overview.....	60
5. Extended ML 300 controller functionality	
5.1 Create a multiple ring network.....	65
5.1.1 Introduction.....	65
5.1.2 Requirements.....	65
5.1.3 Configure a Top Unit controller.....	65
5.2 Inter-controller communication.....	68
5.2.1 Introduction.....	68
5.2.2 Add an ICC output function block.....	69
5.2.3 Add an ICC input function block.....	71
6. Additional libraries	
6.1 Introduction.....	75
6.2 Custom parameters.....	75
6.2.1 Setup BOOL function block.....	75
6.2.2 Setup float/integer function block.....	78
6.2.3 Setup read function block.....	82
6.2.4 Setup write function block.....	86
6.3 Assign a CODESYS I/O function in the controller.....	90

1. Introduction

1.1 About this document

1.1.1 Document overview

This document explains how to:

- Download and install the Multi-line 300 (ML 300) CODESYS package on the DEIF controller and in the CODESYS Development system (CODESYS).
- Create a new CODESYS project for the ML 300 controller.
- Add DEIF-specific function blocks to the program to customise your DEIF controller.
- Download and run the program on the controller.
- Monitor the program through CODESYS.

This document assumes the reader is familiar with CODESYS, and only explains concepts about CODESYS to allow the reader to start programming a CODESYS project for a DEIF ML 300 controller.

All references to CODESYS in this manual were made using CODESYS V3.5 SP10.

To avoid compatibility problems, create and save your CODESYS projects as *Project files (CODEYS V3.5 SP10)(*project)*.

1.1.2 Software versions

The information in this document corresponds to the following software versions.

Table 1.1 Software versions

Product	Software	Details	Version
CODESYS IDE	External	CODESYS project creator	V3.5 SP10
Generator paralleling controller 300 (GPC 300)	PCM APPL	Controller application	1.0.0.x and higher
Paralleling and protection unit 300 (PPU 300)	PCM APPL	Controller application	1.0.3.x and higher

1.1.3 Technical support

You have the following options if you need technical support:

Table 1.2 Support for CODESYS, general

Type of support	Notes
CODESYS Store	General CODESYS support: <ul style="list-style-type: none">• My question: Ask questions about CODESYS.• FAQ: Frequently asked questions.• Forum: Discuss different CODESYS topics with other users.
CODESYS Development System online help	Online help is available from the Help menu.

Table 1.3 Support for CODESYS, DEIF-specific

Type of support	Notes
Technical documentation	Download all the product technical documentation from the DEIF website: www.deif.com/documentation . Refer to the Designer's handbook of your controller for more information about the product functions and parameters.
Support	DEIF offers 24-hour support. See www.deif.com for contact details and to find a DEIF subsidiary located near you. You can also e-mail support@deif.com .
Training	DEIF regularly offers training courses at the DEIF offices worldwide.
Service	DEIF engineers can help with design, commissioning, operating and optimisation.

1.2 Warnings and safety

1.2.1 CustomLogic not available

When the CODESYS package is installed on a controller, CustomLogic is not available on that controller.

In PICUS, the CustomLogic icon in the right side panel under the **Configuration** menu is unavailable after you installed CODESYS on the controller and you logged out of the controller.

1.2.2 Recommendations for data security

To minimise the risk of data security breaches DEIF recommends to:

- As far as possible, avoid exposing controllers and controller networks to public networks and the Internet.
- Use additional security layers like a VPN for remote access, and install firewall mechanisms.
- Restrict access to authorised persons.

1.3 Legal information

1.3.1 Disclaimer

DEIF A/S reserves the right to change any of the contents of this document without prior notice.

The English version of this document always contains the most recent and up-to-date information about the product. DEIF does not take responsibility for the accuracy of translations, and translations might not be updated at the same time as the English document. If there is a discrepancy, the English version prevails.

1.3.2 Trademark

DEIF is a trademark of DEIF A/S.

CODESYS[®] is a registered trademark of 3S-Smart Software Solutions GmbH.

Windows[®] is a registered trademark of Microsoft Corporation in the United States and other countries.

All trademarks are the properties of their respective owners.

1.3.3 Copyright

© Copyright DEIF A/S. All rights reserved.

2. Get started with CODESYS

2.1 Multi-line 300 CODESYS functions

2.1.1 Multi-line 300 CODESYS functions

	Functions
Runtime	<ul style="list-style-type: none"> • CODESYS Runtime runs with real-time behaviour.
Task configuration	<ul style="list-style-type: none"> • Configure custom routines: <ul style="list-style-type: none"> ◦ Function block diagram ◦ Structured text ◦ Ladder logic diagram ◦ Continuous function chart
Function blocks	<ul style="list-style-type: none"> • Controller overview: <ul style="list-style-type: none"> ◦ CODESYS application status check ◦ CODESYS communication error overview ◦ Controller application version check • Live data • Standard ML 300 controller functions: <ul style="list-style-type: none"> ◦ Protections (based on controller type) ◦ Controller inputs and outputs (based on controller type) • Controller status: <ul style="list-style-type: none"> ◦ Controller status text ◦ Controller pop-up text
Communication	<ul style="list-style-type: none"> • Configure the controller as a Modbus server or Modbus client. • Configure the controller as a CANopen master or a CANopen slave. • Raw CAN communication possible. • Inter-controller communication that integrates with Custom Logic: <ul style="list-style-type: none"> ◦ 16 outputs per controller ◦ 16 inputs from each controller in the system
Other	<ul style="list-style-type: none"> • 40 customisable inputs for each input type: <ul style="list-style-type: none"> ◦ Digital input ◦ Analogue input • 40 customisable outputs for each output type: <ul style="list-style-type: none"> ◦ Digital output ◦ Analogue output

2.2 Software requirements

2.2.1 Software requirements

Table 2.1 CODESYS requirements

Component	Requirement(s)	Notes
Operating system	Windows XP	Or higher.
Processor	Dual-core	
Memory	4 GB RAM	

Component	Requirement(s)	Notes
Free disk space	2 GB	
Network interface	Network adaptor with 1 free Ethernet port	To connect your computer to the controller.

Table 2.2 Additional requirements


Component	Requirement(s)	Notes
Additional software	PICUS	To install or update CODESYS on the controller. Download from www.deif.com .
Code size	Less than 20 MB.	The CODESYS application developer must ensure that the CODESYS code is within this limit.

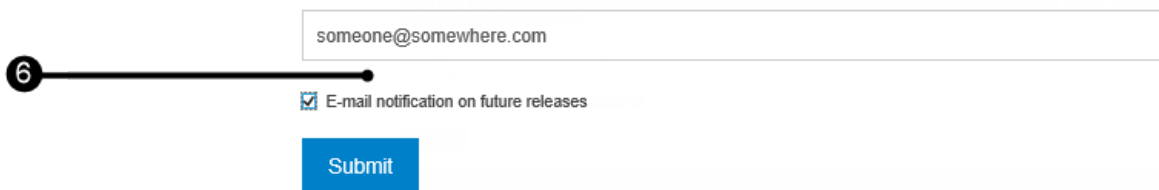
2.3 Download

2.3.1 Downloading the DEIF CODESYS software package

To use a CODESYS program on a DEIF controller you have to install the DEIF CODESYS application software (.packet-file) on the controller. You also need to add the Multi-line 300 to the CODESYS device repository. The DEIF CODESYS application software and the Multi-line 300 device description are bundled together in the DEIF CODESYS software package. You can download the DEIF CODESYS software package by submitting your email address via the DEIF website. A link is sent to you to download the CODESYS software package.

To download the DEIF CODESYS software package follow these steps:

1. Visit the DEIF website at: www.deif.com.
2. Open the search bar , and start to type the controller name to open a list of product options.
3. Select the controller from the list displayed.
4. Scroll down to the product Description, and select the Software tab.
5. Open the **CODESYS** list, and select **Multi-line 300 CODESYS add-on v 1.x.x**.
6. Submit your email address to receive a download link to the software.



7. Follow the link in the email to download the DEIF CODESYS software package to your computer.

2.3.2 DEIF CODESYS software package contents

The DEIF CODESYS software package includes:

- The ".packet"-file for the installation of CODESYS Runtime on an ML 300 controller.
- The CODESYS device description file to add the Multi-line 300 to the CODESYS device repository.



CAUTION


You are not allowed to modify the device description file provided by DEIF A/S in any way. Modification of the file can lead to unexpected behaviour of the software and the protections provided by the Multi-line 300 platform.

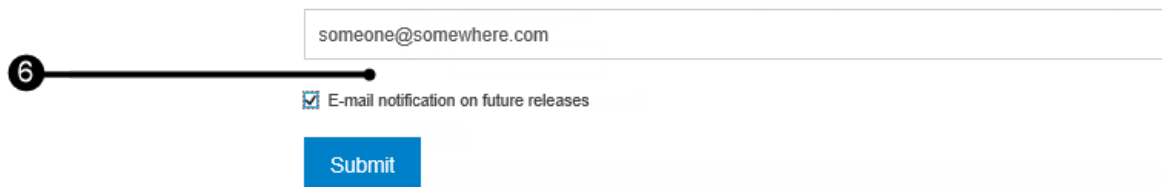
2.3.3 Download the DEIF CODESYS library package

To use the DEIF controller functions in CODESYS you need to install the DEIF CODESYS library for your controller in CODESYS. You can download the DEIF CODESYS library package by submitting your email address via the DEIF website. A link is sent to you to download the DEIF CODESYS library package.

To download the DEIF CODESYS library package follow these steps:

Added

1. Visit the DEIF website at: www.deif.com.
2. Open the search bar , and start to type the controller name to open a list of product options.
3. Select the controller from the list displayed.
4. Scroll down to the product Description, and select the Software tab.
5. Open the **CODESYS** list, and select **[Product] CODESYS libraries**, where [Product] is the product abbreviation (for example, GPC 300).
6. Submit your email address to receive a download link to the software.



6

someone@somewhere.com

E-mail notification on future releases

Submit

7. Follow the link in the email to download the DEIF CODESYS library package to your computer.

2.3.4 DEIF CODESYS library package contents

The DEIF CODESYS library package includes:

- The libraries for each controller type of the specific ML 300 product.

NOTE An example of an ML 300 product is the GPC 300. An example of a controller type for the product is a GPC 300 GENSET controller.

2.4 Install

2.4.1 Install CODESYS Runtime on the controller

To install CODESYS Runtime on your controller, use the *Update firmware* feature in PICUS. Update the controller firmware with the ".packet"-file that you received from the ML 300 CODESYS software package download. The ".packet"-file must be installed on every controller in your system that you need to use CODESYS on.

NOTE It is only possible to install CODESYS on an ML 300 controller, if the controller has a CODESYS license installed on the controller. If your controller does not have a CODESYS license, you can purchase an ML 300 controller from DEIF with a CODESYS license.



More information

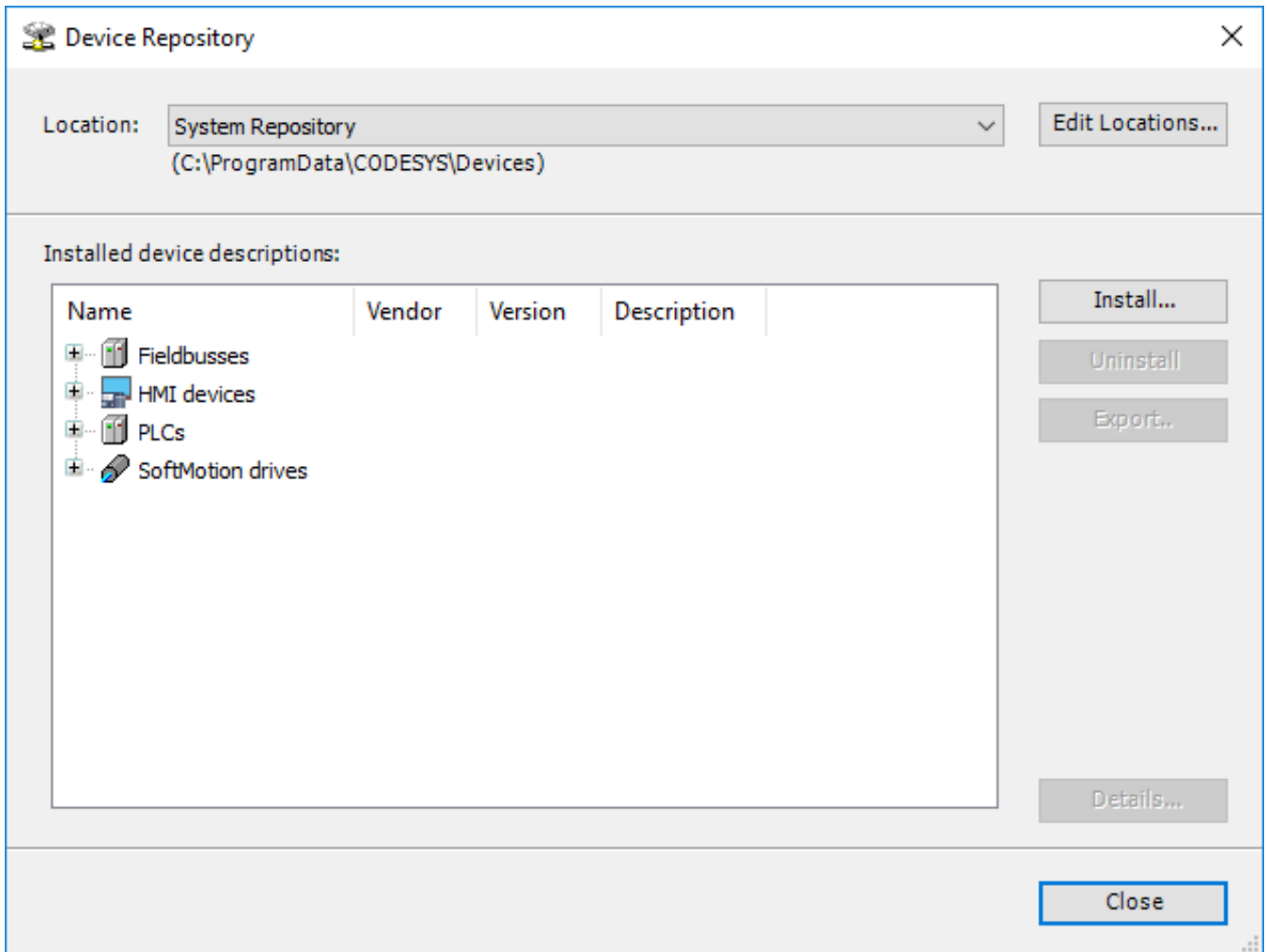
See **Firmware, Tasks, Install firmware** in the **PICUS manual** for more information about installing firmware on the controller.

2.4.2 Install the device description in CODESYS

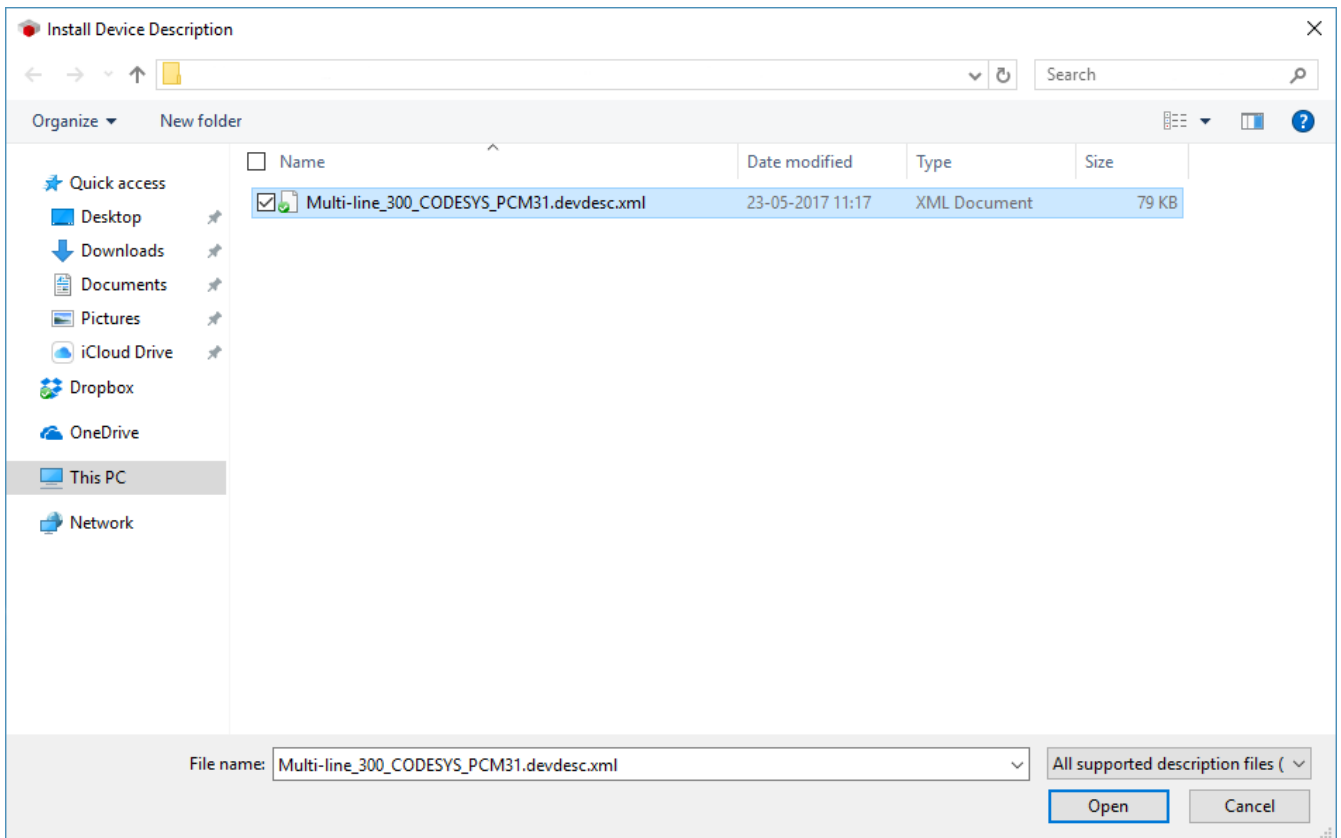
For CODESYS to recognise the DEIF Multi-line 300 controller, you must install the *Multi-line_300_CODESYS_PCM31.devdesc.xml* description file in the CODESYS Runtime Environment.

Follow these steps to install the Multi-line 300 device description file in CODESYS:

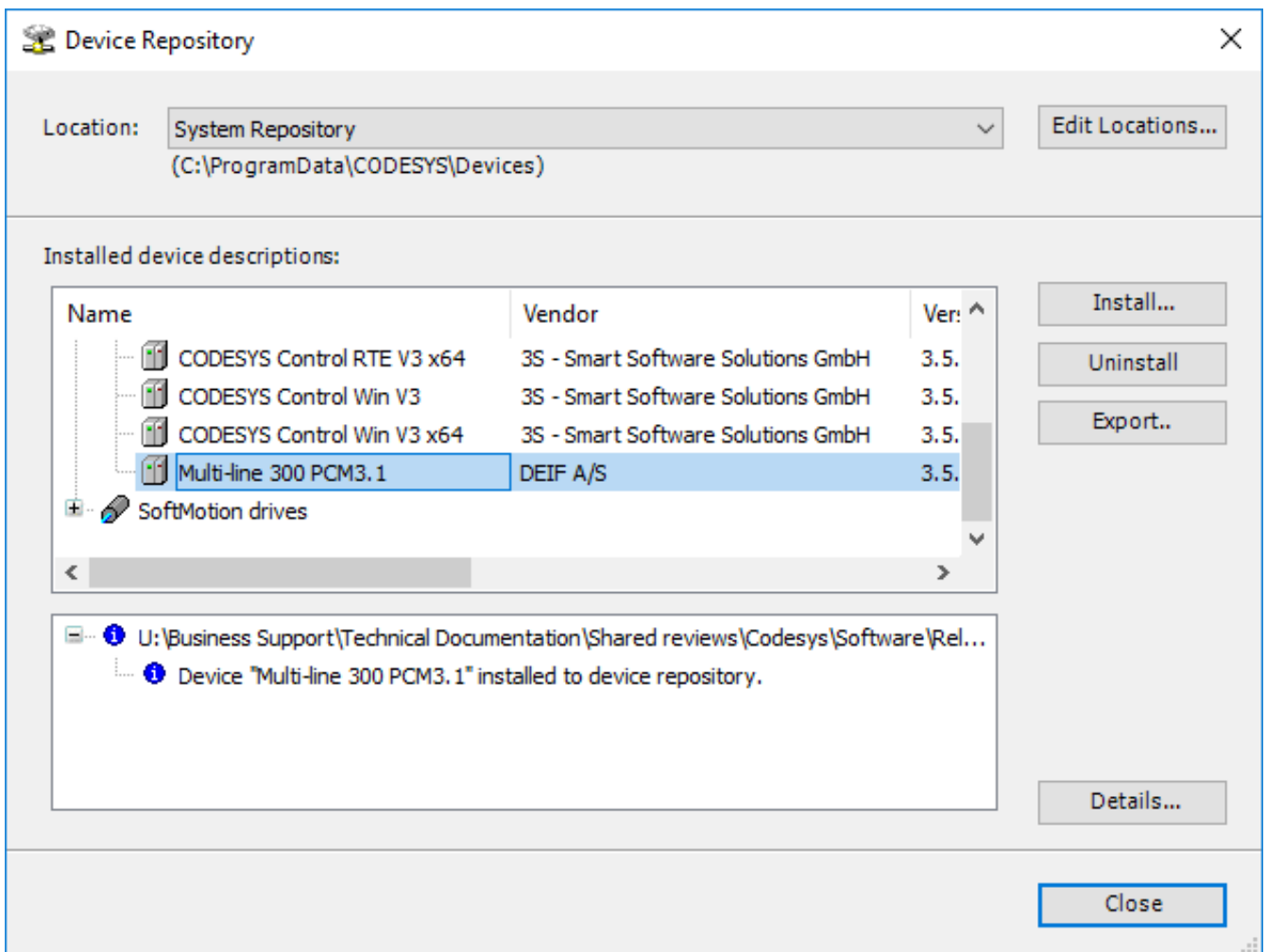
1. Go to **Tools > Device Repository...**
2. Select **Install...** from the *Device Repository* window:



3. Select and open the *Multi-line_300_CODESYS_PCM31.devdesc.xml* file:



4. The DEIF controller is now available in the CODESYS device repository:



2.4.3 Install the ML 300 controller libraries in CODESYS

To add the DEIF libraries to a CODESYS project, you must install the DEIF libraries in the CODESYS library repository. As a minimum you must install:

- The *Multiline_300_pcm31.compiled-library*
- The *Multiline_300_io.compiled-library*
- The controller type library for your controller type

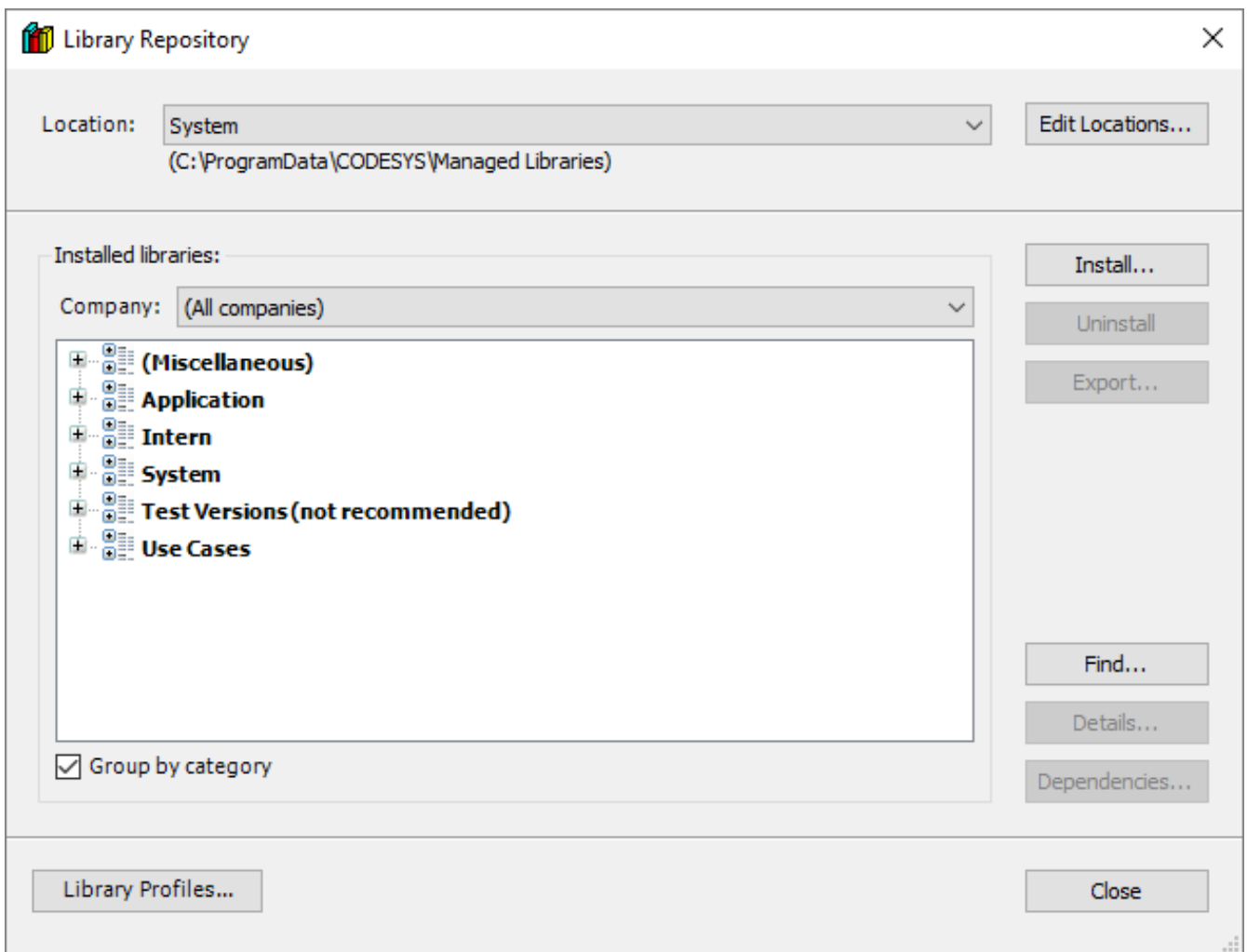


More information

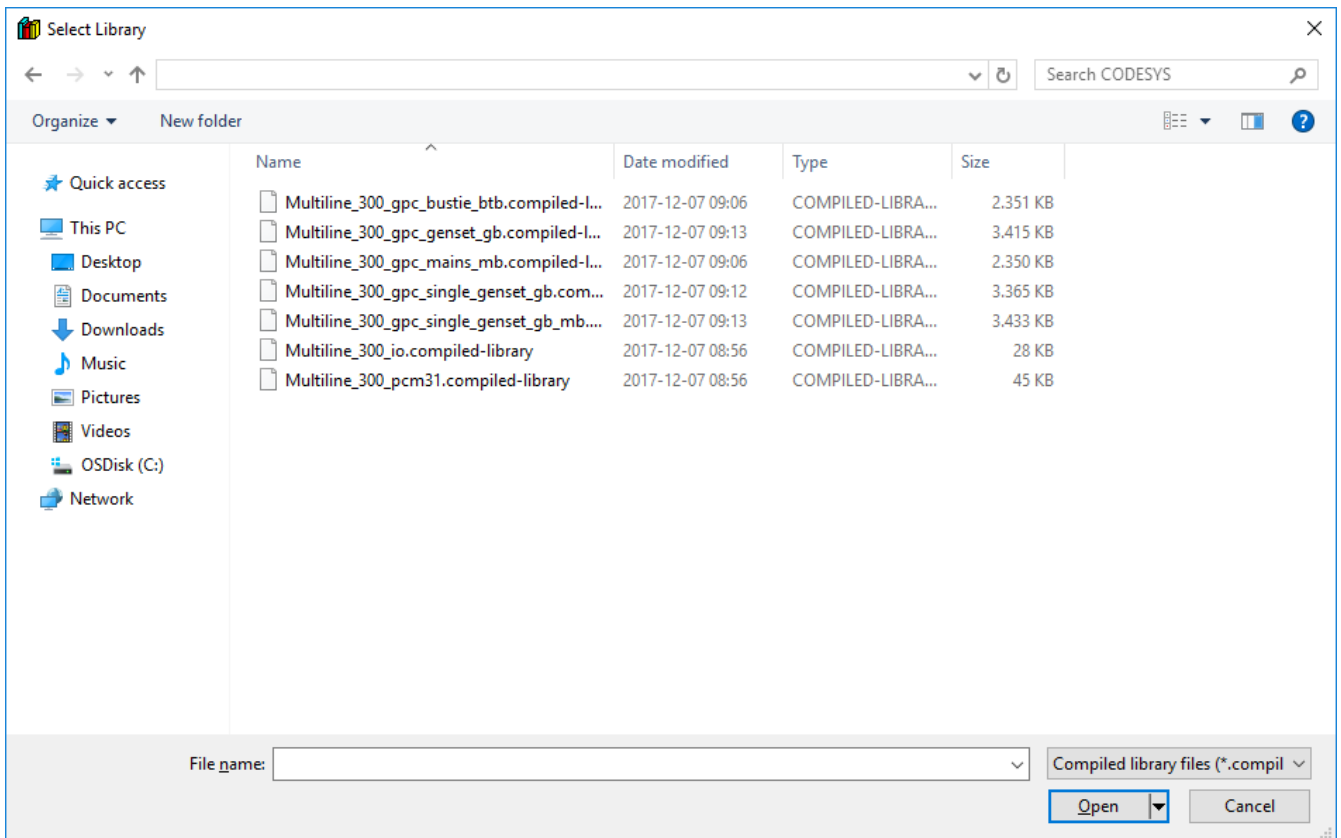
See **Get started with CODESYS, Download** for more information about downloading the CODESYS library package for DEIF controllers.

Follow these steps to install the controller libraries in CODESYS:

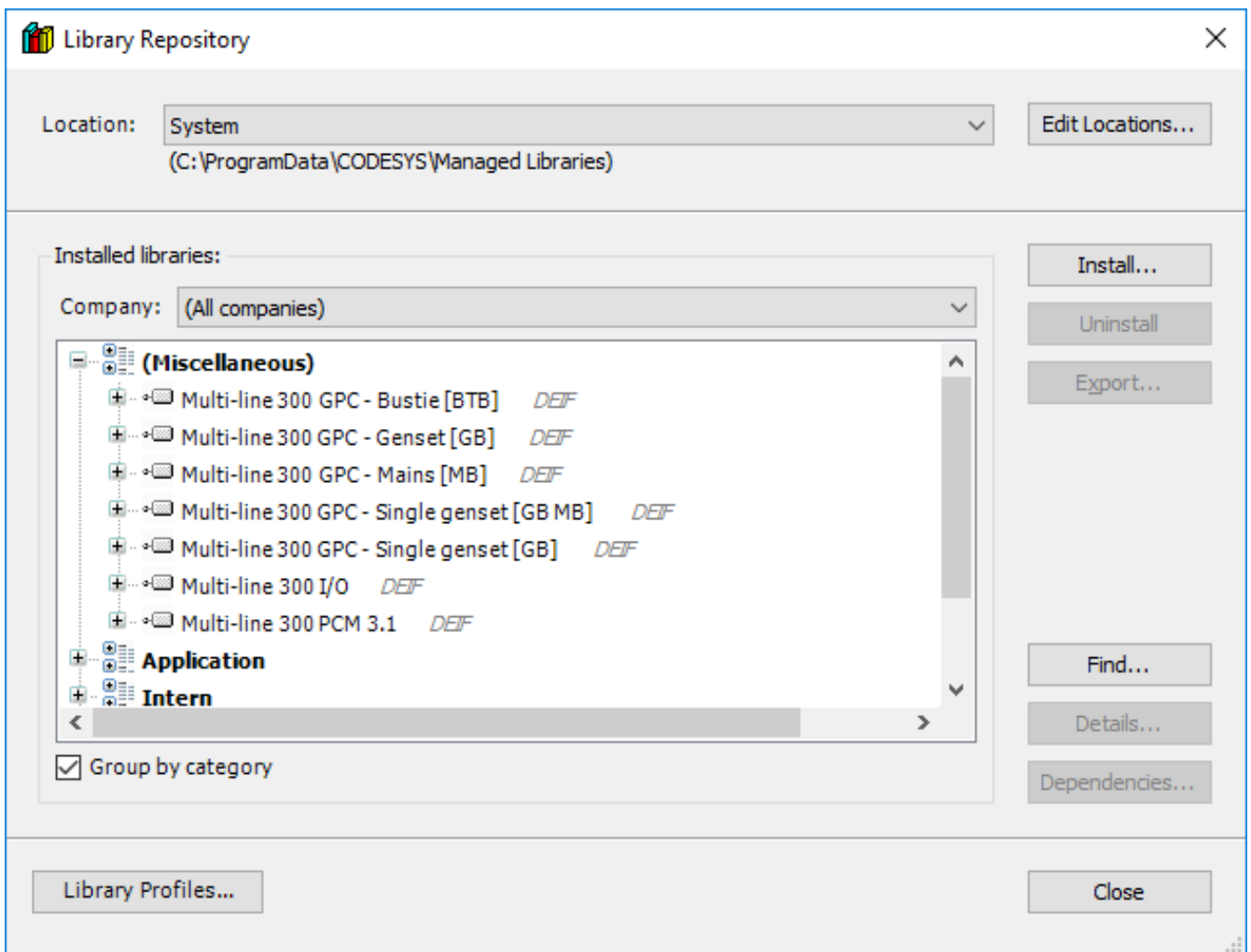
1. Go to **Tools > Library Repository...**
2. Select **Install...** from the *Library Repository* window:



3. Select all the controller library files, then select **Open**:



4. The ML 300 controller libraries are now available in the CODESYS library repository:



3. ML 300 CODESYS projects

3.1 Introduction

3.1.1 Introduction

This chapter describes how to prepare a new CODESYS project for an ML 300 controller. It describes how to:

- Create a new project.
- Add the ML 300 function block.
- Establish communication between CODESYS and the controller.
- Download your program to the controller.
- Monitor the running program on the controller.

NOTE The descriptions in this chapter refer to the default configuration of the user interface provided with CODESYS V3.5 SP10.

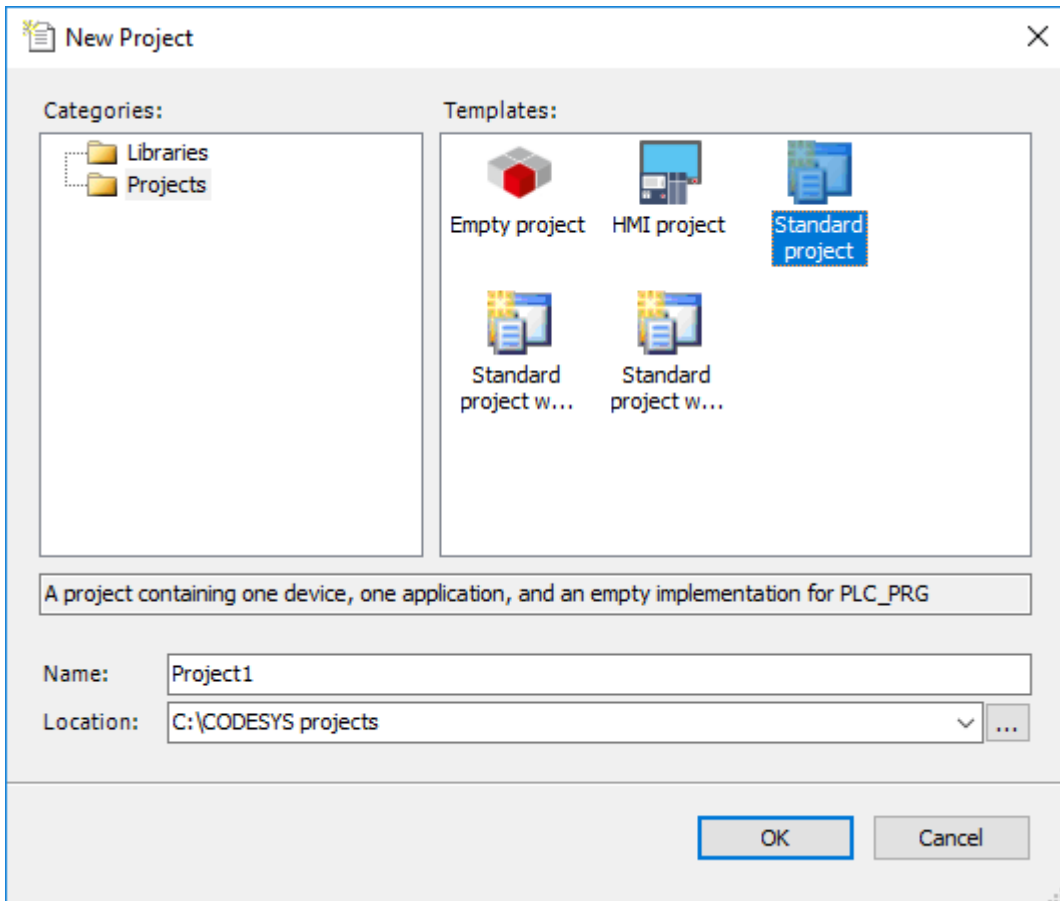
3.2 Create a new project

3.2.1 Create a project file

The following steps describe how to start a new project in the CODESYS user interface.

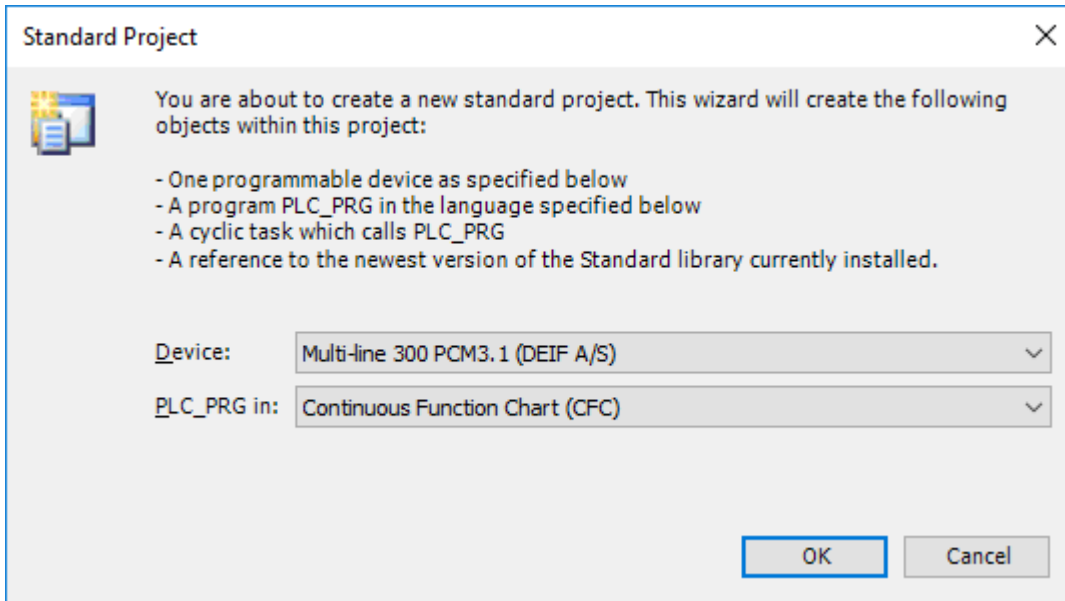
Follow these steps to create a new CODESYS project:

1. Go to **File > New Project...**
2. Select **Standard project** in the *Templates* field and enter a **Name** and a **Location** path for the project file:

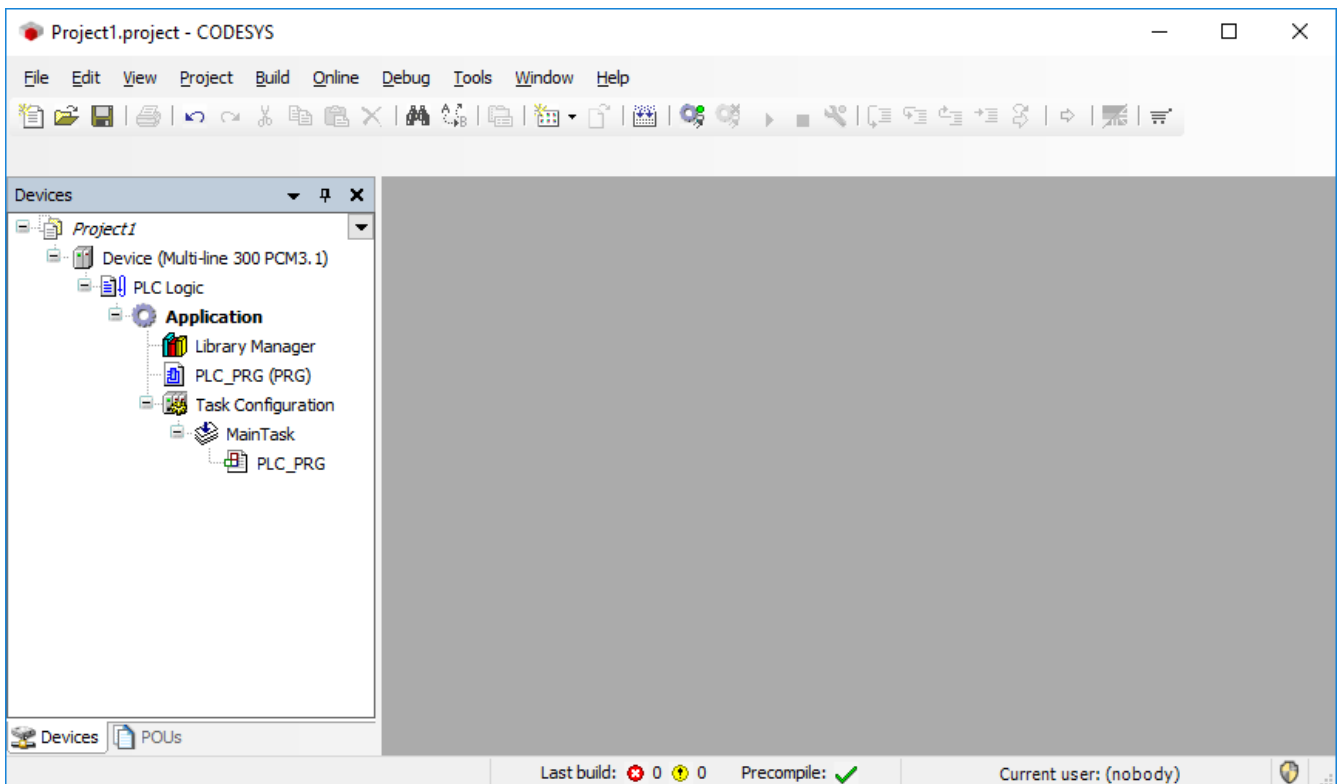


- Select **OK** to continue.

3. In the *Standard Project* window select **Multi-line300 PCM 3.1 (DEIF A/S)** as the device and select a programming language:



- The selected programming language is the programming language for the starting Program Organization Unit (POU).
 - New POU's can use a different programming language.
 - Select **OK** to continue.
4. The CODESYS project is ready to be programmed:

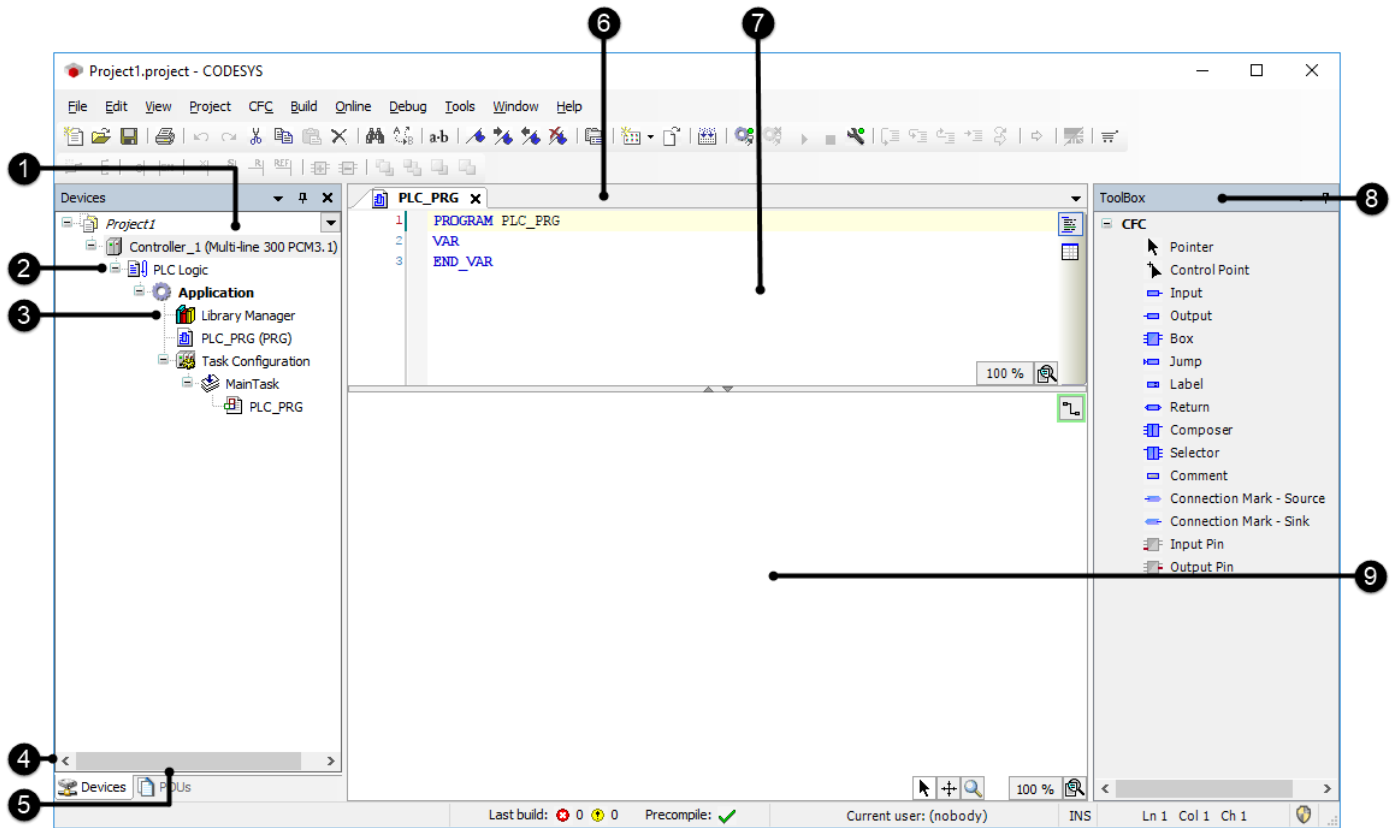


Save your CODESYS projects as *Project files (CODESYS V3.5 SP10)(*.project)*.

3.2.2 CODESYS layout

The image below shows the terminology used for the project view throughout the CODESYS manual.

Figure 3.1 CODESYS project overview



No.	Item	Notes
1.	Project tree	The project tree is an overview of your project. A typical project will consist of: <ul style="list-style-type: none"> • One or more devices (ML 300 controllers). • One or more applications containing the Libraries, POU's and tasks.
2.	PLC Logic node	A <i>PLC Logic</i> node shows that the device is a programmable device and has no other functions associated to it.
3.	Library manager	The <i>Library manager</i> contains the libraries for the project. Each library consists of functions and function blocks that can be used in your programs. Add the <i>Multi-line 300 PCM 3.1</i> , <i>Multi-line 300 I/O</i> , and the library for your controller type to the Library manager to be able to use the controller functions and function blocks in your program.
4.	Devices tab	The <i>Devices</i> tab gives you quick access to the project tree.
5.	POUs tab	The <i>POUs</i> tab gives you quick access to the project settings menu.
6.	Working area	The working area consists of tabs representing different parts of the project. Each tab contains different parts of the project. These project parts can be opened from the project tree. The picture above shows the working area for the POU, PLC_PRG . The working area for this POU consists of a declaration workspace, an implementation workspace and a toolbox menu.
7.	Declaration workspace	The declaration workspace consists of the variables for the POU functions.
8.	Additional toolboxes	Some POU's have additional toolboxes that help you to build your program.
9.	Implementation workspace	The implementation workspace is used to program your POU.

3.2.3 Add the Multi-line 300 libraries to your application

The ML 300 controller libraries must be installed in CODESYS before you can add them to your application.



More information

See **Get started with CODESYS, Install, Install the ML 300 controller libraries in CODESYS** for more information about how to install the ML 300 controller libraries.

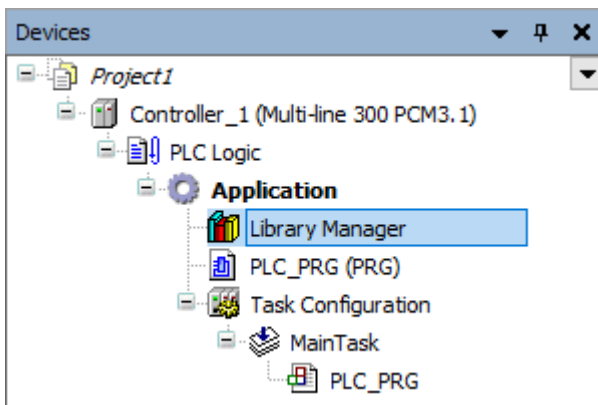
The following libraries have to be added to your application:

- Multi-line 300 PCM 3.1
- Multi-line 300 I/O
- A controller-specific library

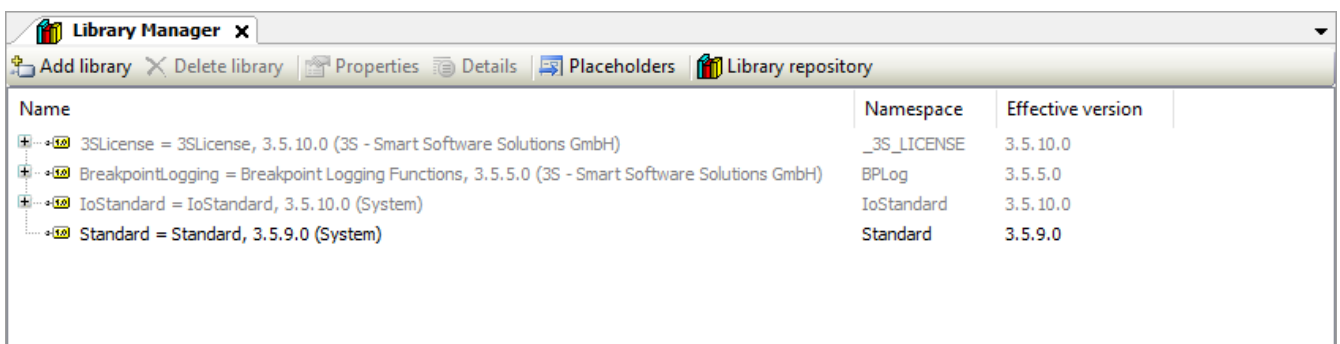
NOTE Only add the controller-specific library to the application for the specific controller you are connected to. For example, select the *Multi-line 300 GPC - Genset [DG]* for a GPC 300 GENSET controller. More than one controller-specific library can be added, but this is not recommended.

Follow these steps to add the ML 300 libraries to your application:

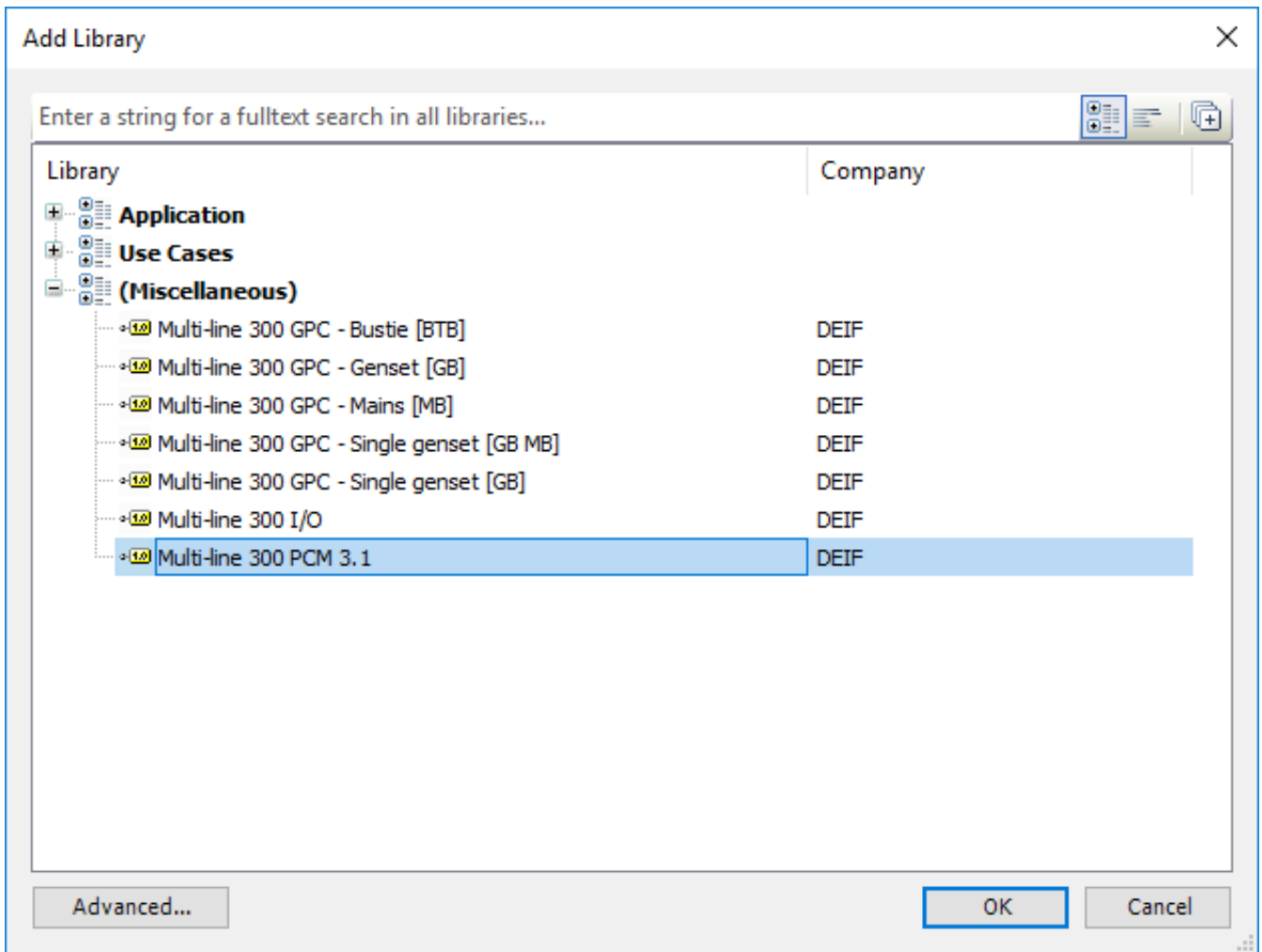
1. Double-click on **Library Manager** in the project tree to open the **Library Manager** in the working area:



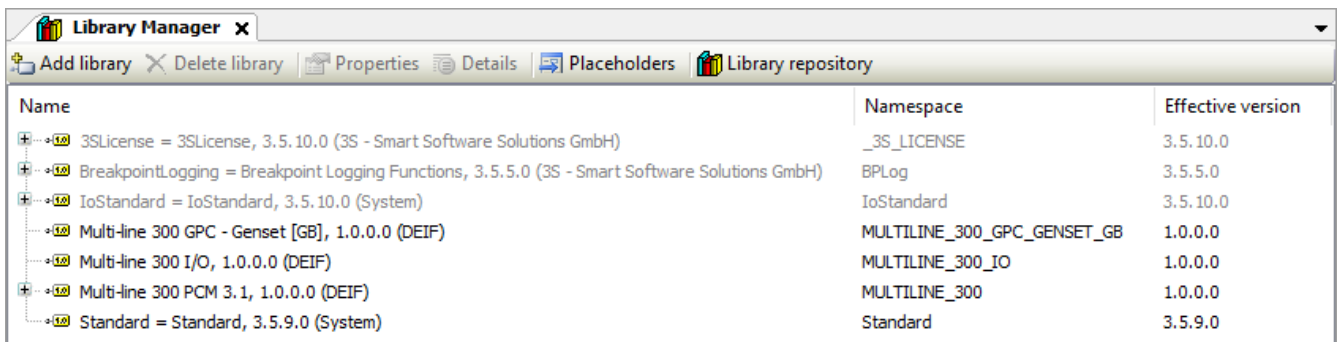
2. Select **Add Library**:



3. Select the *Multi-line 300 PCM 3.1* library under **(Miscellaneous)**:



- Select **OK**.
- Repeat Step 3 for the *Multi-line 300 I/O* library and the controller-specific library (for example, *Multi-line 300 GPC - Genset [GB]*).
 - The ML 300 controller libraries for your product are now available in the **Library Manager**:



3.3 Add the ML 300 function block

3.3.1 Introduction

To run your program on an ML 300 controller with CODESYS installed, your program must contain the ML 300 function block in a *Program* POU. This function block tells the controller that there are no errors with CODESYS, and starts and checks the link between the program and the controller.

If you have multiple POU in your program, then you can include the ML 300 function block in only one of the POU. Remember to link the POU containing the ML 300 function block to the *MainTask*.

The ML 300 function block must be the first item in the execution order, because it checks if the status of the CODESYS program on the controller.

3.3.2 Create a continuous function chart program

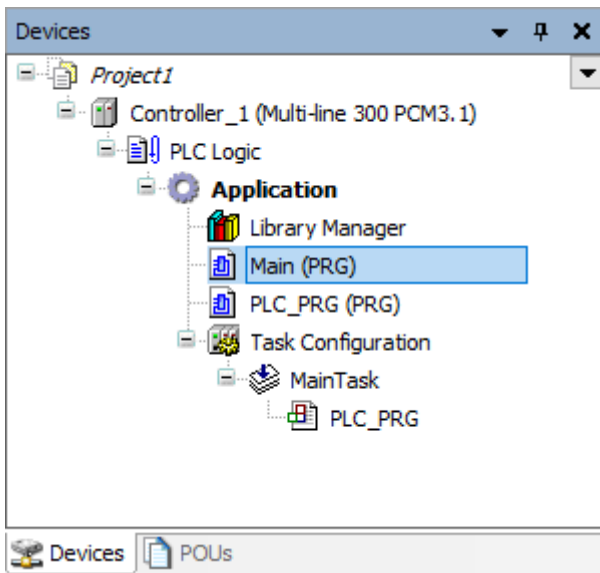
To add the ML 300 function block to your application, you must have a *Program* POU that uses the continuous function chart programming language.

Follow these steps to add the required POU to your **Application**:

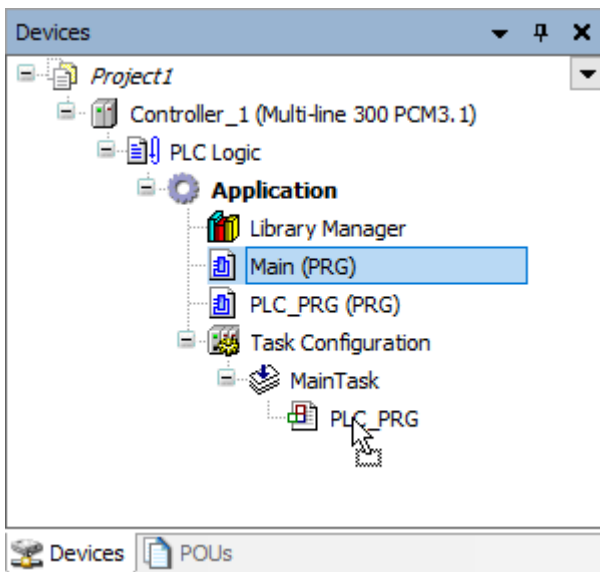
1. Right-click on **Application** in the project tree and select **Add Object > POU...**
2. Enter a name for the POU in the *Name:* field, set the *Type* to **Program** and select **Continuous Function Chart (CFC)** as the *Implementation language*. Select **Add** to continue.

The screenshot shows the 'Add POU' dialog box. The title bar says 'Add POU'. Below the title bar is a sub-dialog titled 'Create a new POU (Program Organization Unit)'. The 'Name:' field contains 'Main'. Under the 'Type' section, 'Program' is selected with a radio button. Below 'Program' are two radio buttons for 'Function Block'. Under 'Function Block', there are two checkboxes: 'Extends:' and 'Implements:'. Each checkbox has a text input field and a three-dot menu button. Below these is the 'Access specifier:' dropdown menu. Under 'Function Block', there is a 'Method implementation language:' dropdown menu set to 'Continuous Function Chart (CFC)'. Below 'Function Block' is a radio button for 'Function'. Under 'Function', there is a 'Return type:' text input field and a three-dot menu button. At the bottom of the dialog is the 'Implementation language:' dropdown menu, also set to 'Continuous Function Chart (CFC)'. At the very bottom are two buttons: 'Add' and 'Cancel'.

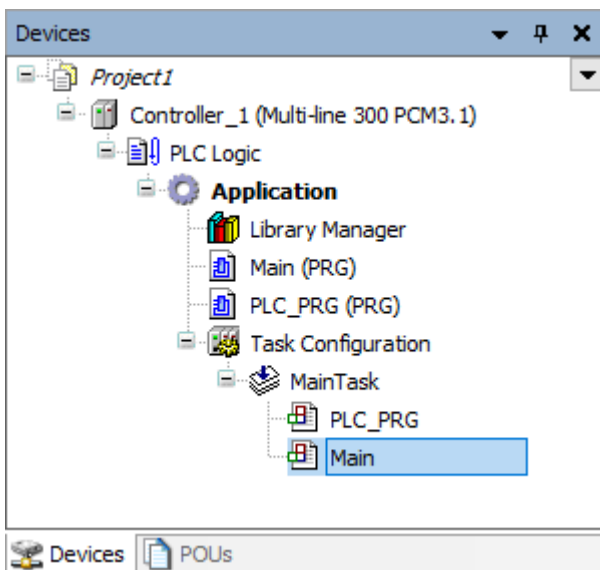
3. The POU has been added to your application:



- 4. Drag the POU to MainTask in the function tree, to add the POU to the application's Task configuration.



- 5. The POU is now ready to be used in the application for the controller.



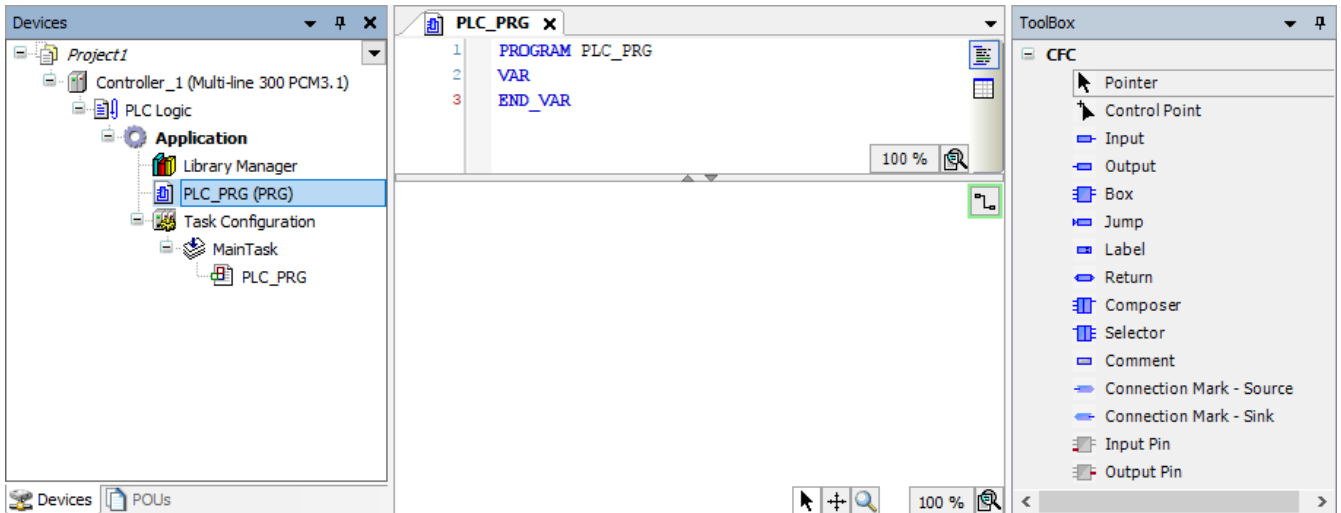
3.3.3 Add the ML 300 Read-Write function block

The ML 300 Read-Write function block which reads and writes data at the same time during a scan. This can be useful when you don't have heavy data processing that needs to be written to the controller at the end of the scan that the data was processed.

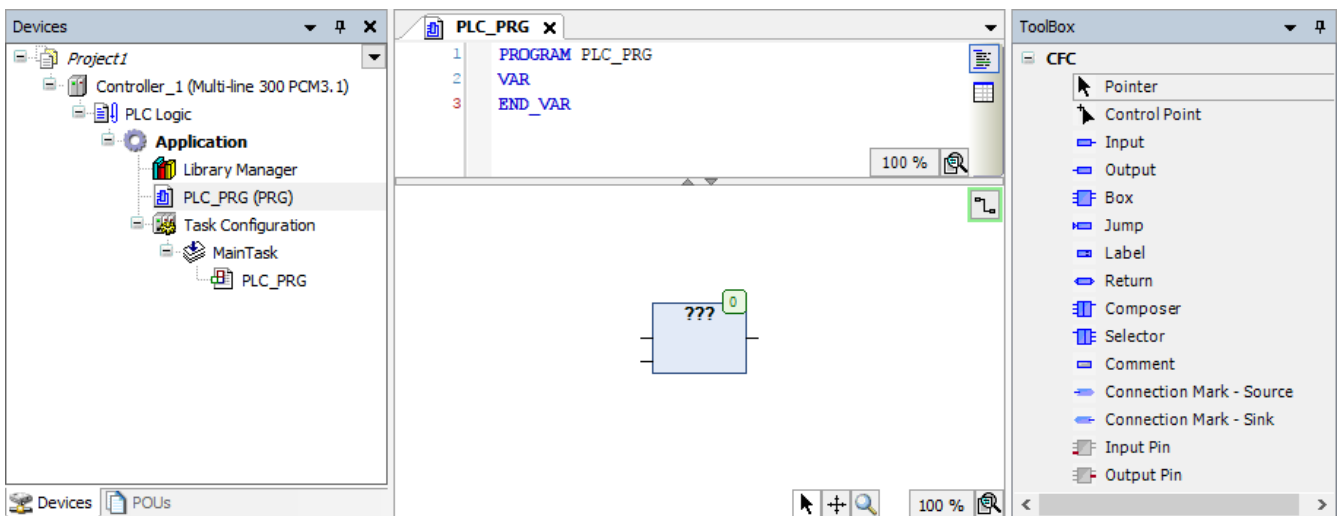
NOTE You can only add one ML 300 Read-Write function block to a Device. If you add an ML 300 Read-Write function block, you cannot add an ML 300 Read and/or ML 300 Write function block to the same Device.

Follow these steps to add the ML 300 Read-Write function block to a Program POU using the continuous function chart programming language:

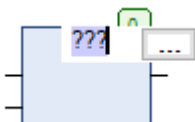
1. Double-click on the continuous function chart POU in the project tree to open the program in the working area:



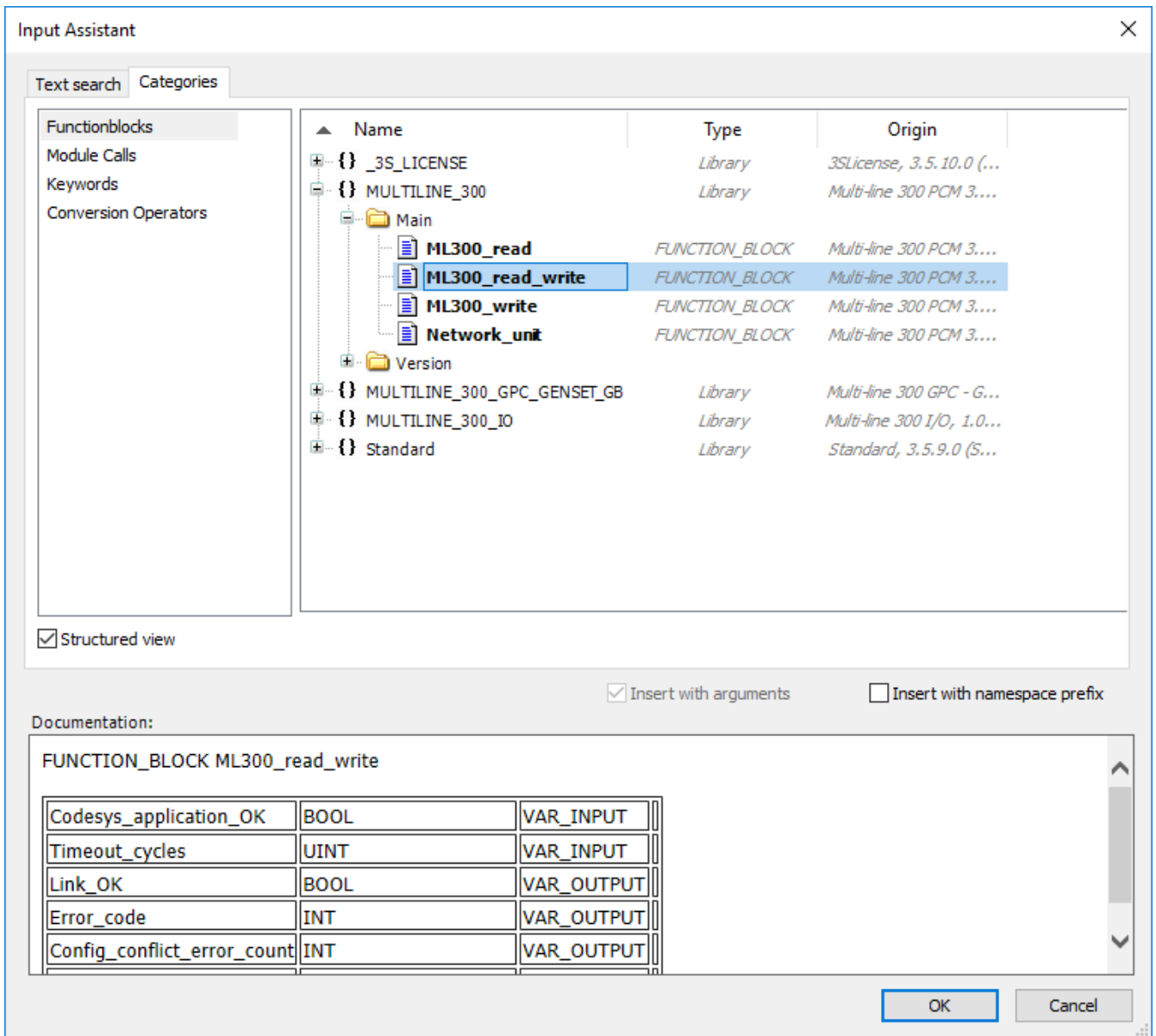
2. Drag a **Box** from the *ToolBox* to the implementation part of the working area:



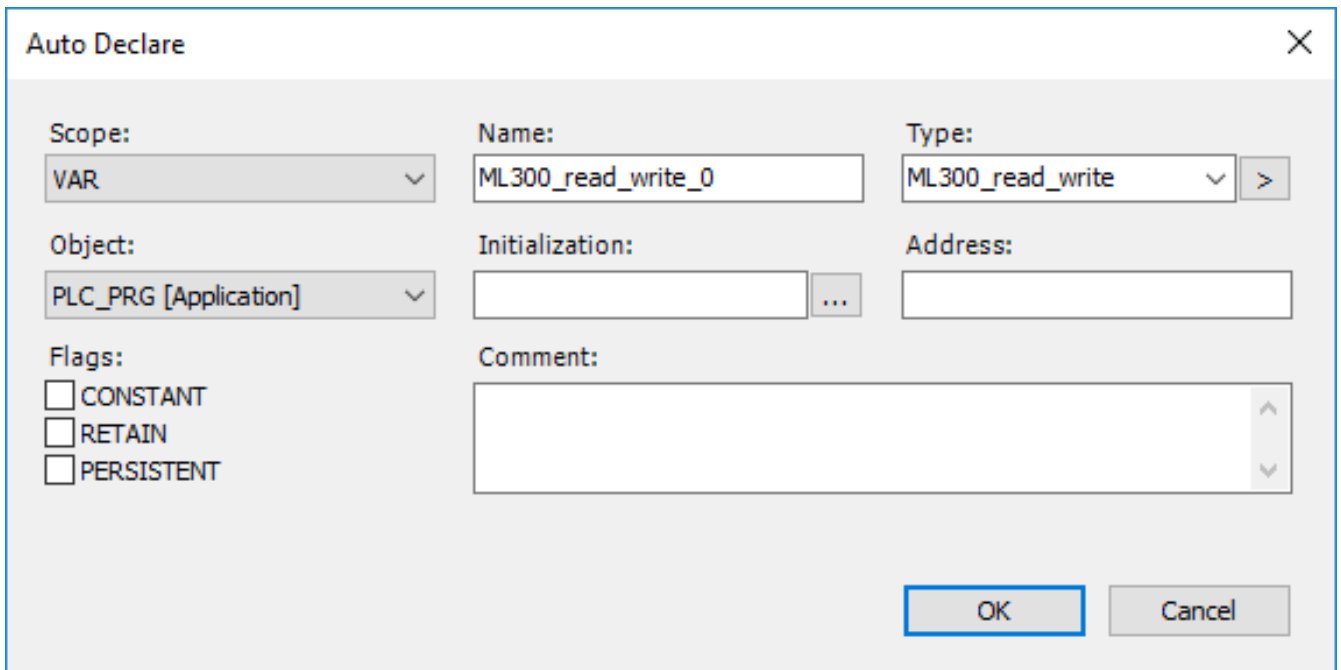
3. Select **???** in the **Box** and then select  to open the *Input Assistant* window:



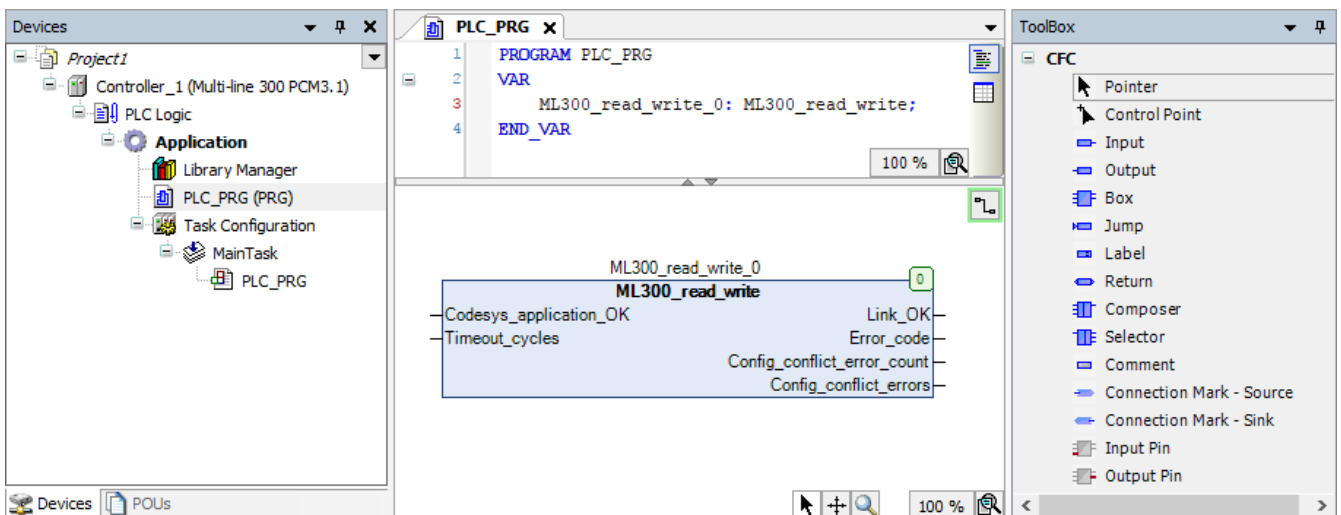
4. Go to **Categories > Functionblocks > MULTILINE_300 > Main** and select the **ML300_read_write** function block:



- Select **OK**.
5. Press *Return* on your keyboard two times to open the *Auto Declare* window for the function block. Select **OK** to declare the variables as they are shown:



6. The Multi-line 300 Read-Write function block has been added to the project:



- Remember to assign an *Input* to the **CODESYS_application_OK** input on the ML 300 Read-Write function block.

3.3.4 Add the ML 300 Read and ML 300 Write function blocks

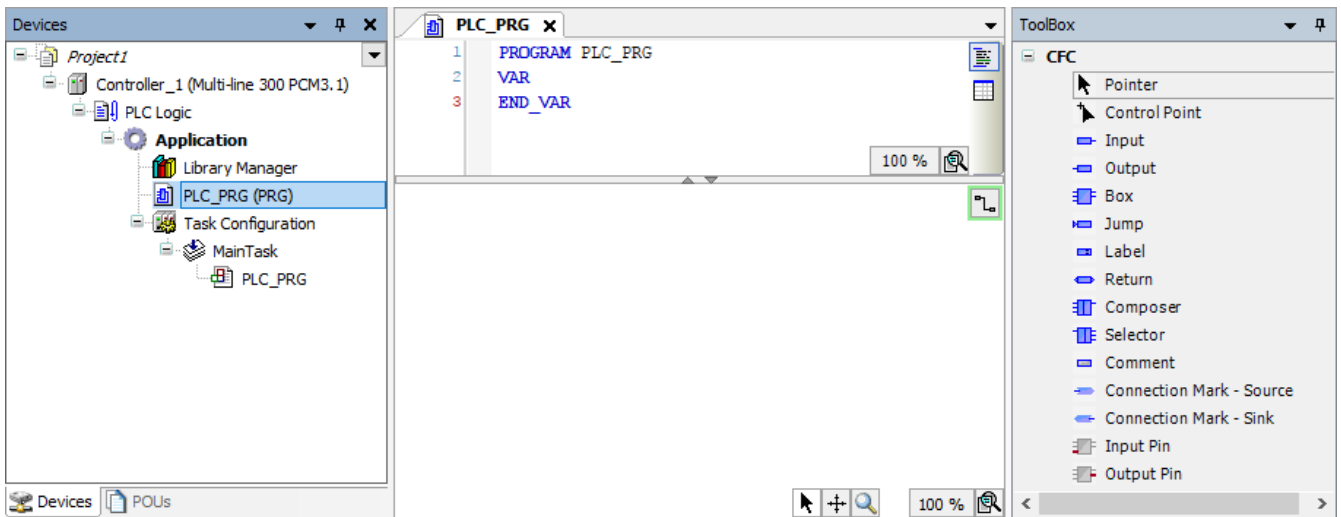
You can add the ML 300 Read and ML 300 Write functions blocks to your program to give you more flexibility in the PLC sequence. In contrast to the ML 300 Read-Write function block which reads and writes data at the same time during a scan, using separate read and write function blocks allows the PLC to read data at the beginning of a scan, process the data, and then write the data at the end of the scan.

NOTE You can only add one ML 300 Read and one ML 300 Write function block to a Device. If you add the ML 300 Read and ML 300 Write function blocks, you cannot add an ML 300 Read-Write function block to the same Device.

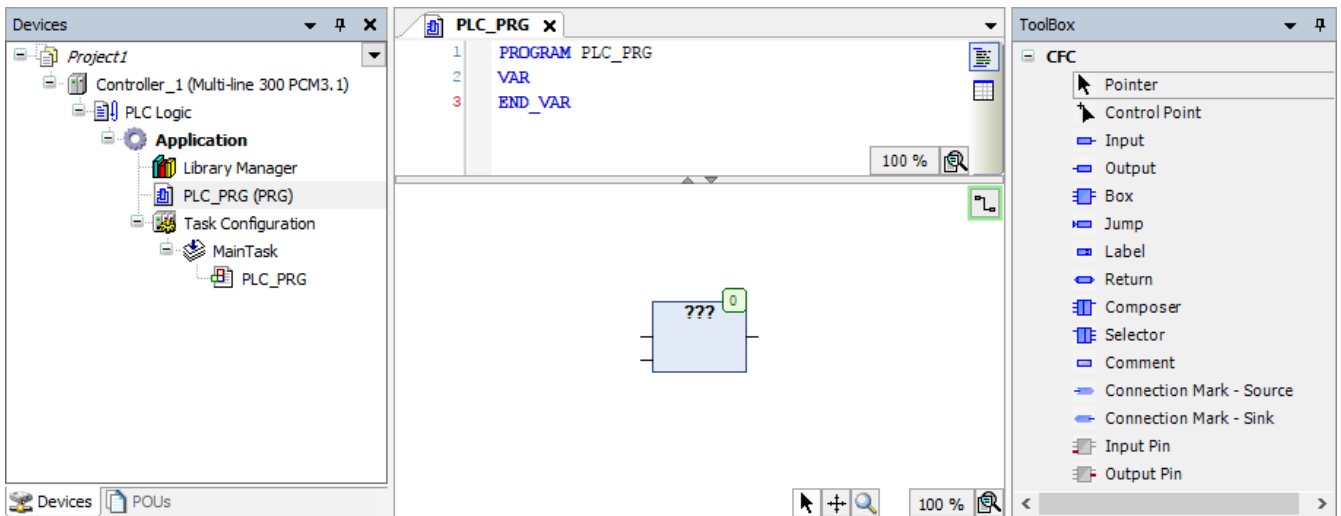
NOTE When you add the ML 300 Read function block, you must also add the ML 300 Write function block. When you add an ML 300 Write function block, you must also add an ML 300 Read function block.

Follow these steps to add the ML 300 Read and ML 300 Write function blocks to a Program POU using the continuous function chart programming language:

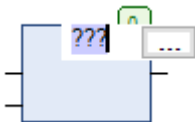
1. Double-click on the continuous function chart POU in the project tree to open the program in the working area:



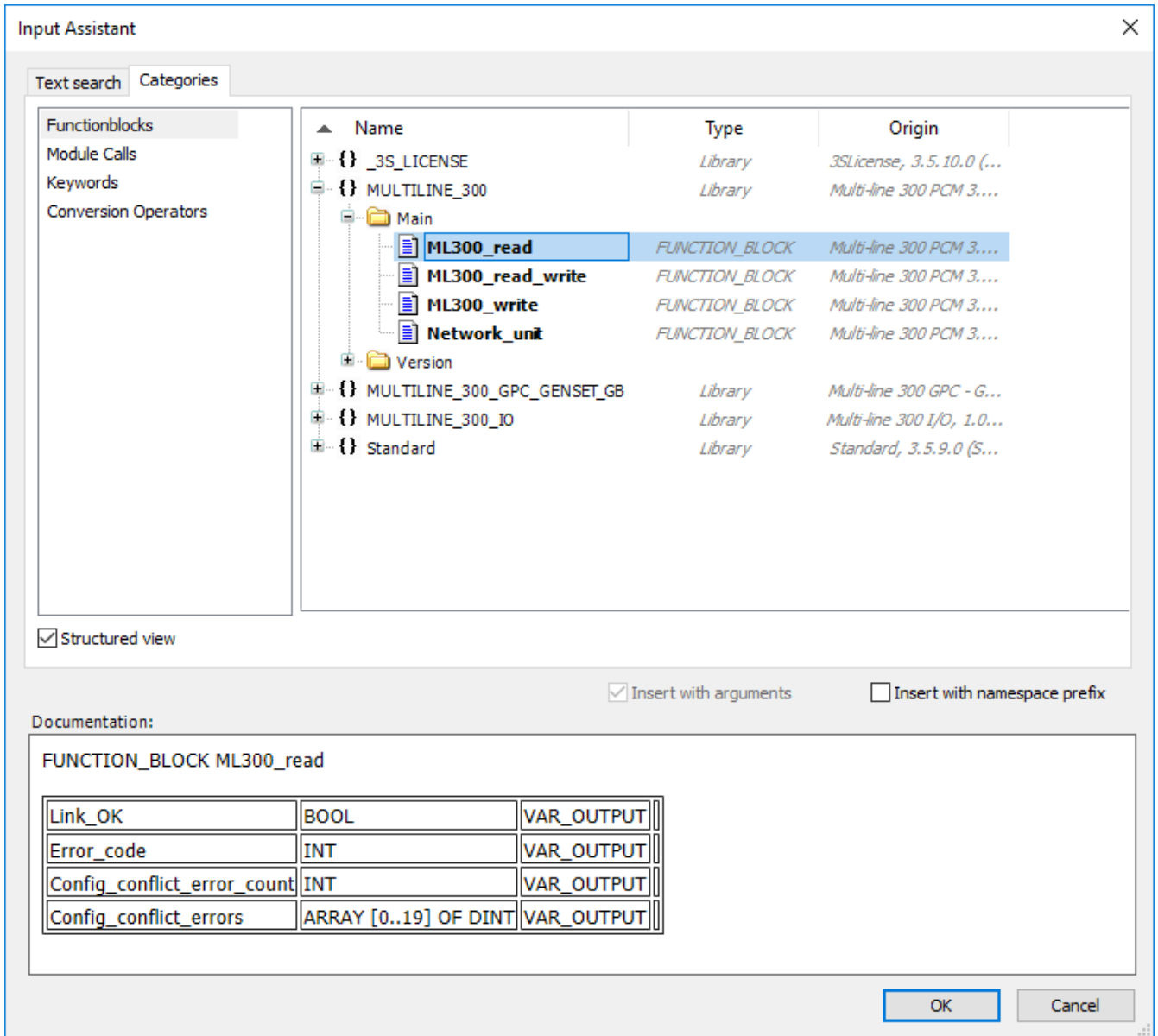
2. Drag a **Box** from the *ToolBox* to the implementation part of the working area:



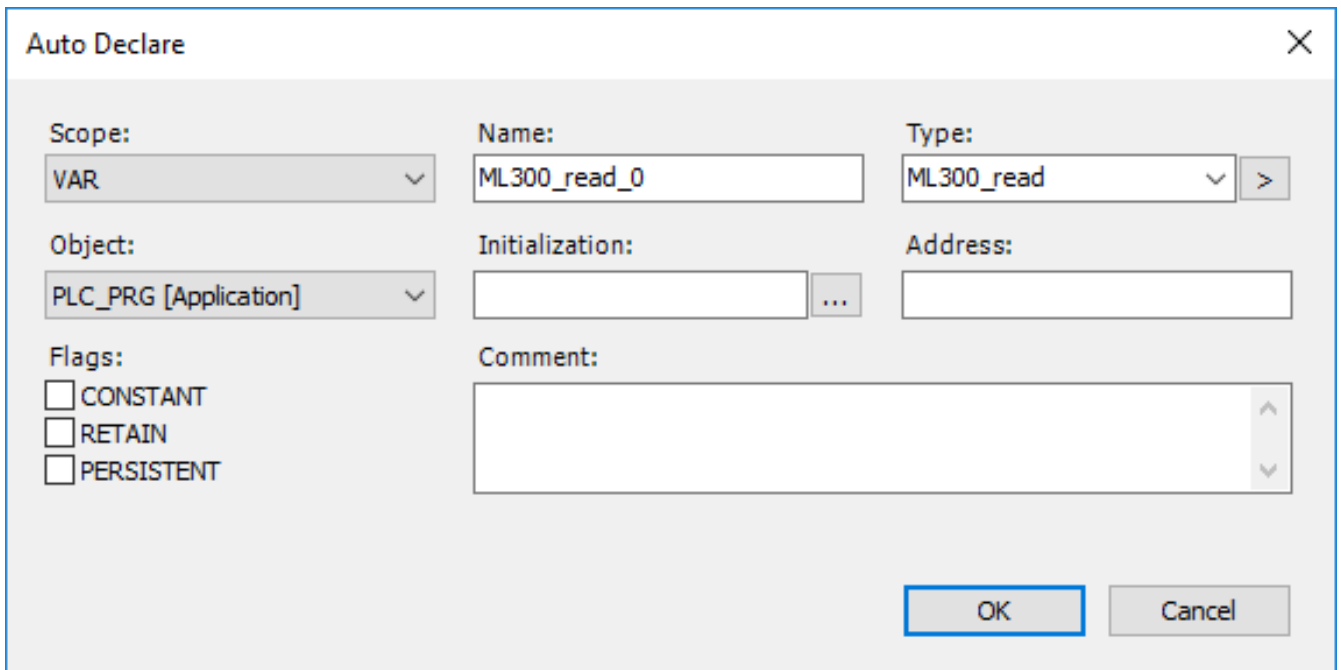
3. Select **???** in the **Box** and then select  to open the *Input Assistant* window:



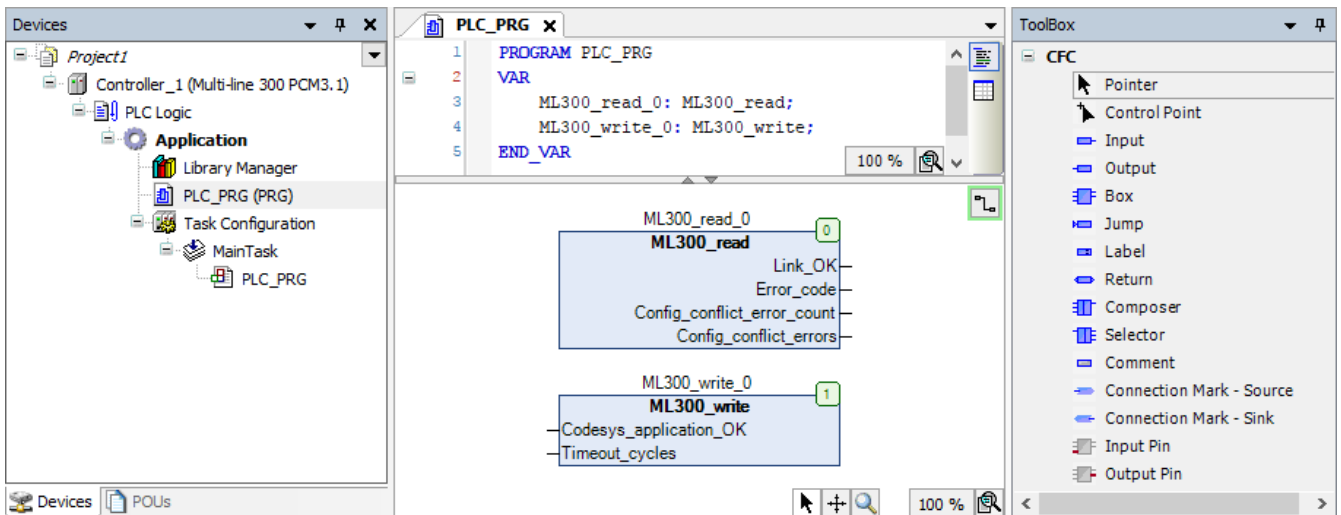
4. Go to **Categories > Functionblocks > MULTILINE_300 > Main** and select the **ML300_read** function block:



- - Select **OK**.
5. Press *Return* on your keyboard two times to open the *Auto Declare* window for the function block. Select **OK** to declare the variables as they are shown:



- 6. Repeat steps 2 to 5 to add the **ML300_write** function block.
- 7. The Multi-line 300 Read and Multi-line 300 Write function blocks have been added to the project:



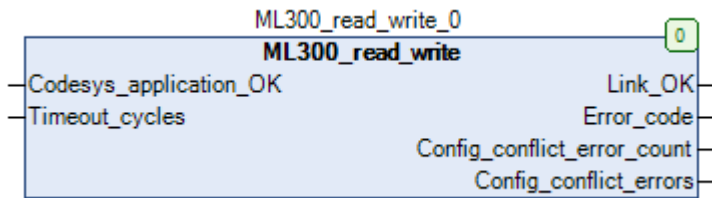
- Remember to assign an *Input* to the **CODESYS_application_OK** input on the ML 300 Write function block.
- The ML 300 Read function block must always be first in the execution order list (execution order number zero).
- The ML 300 Write function block must always be last in the execution order list (highest execution order number).

3.3.5 ML300 function blocks' execution position

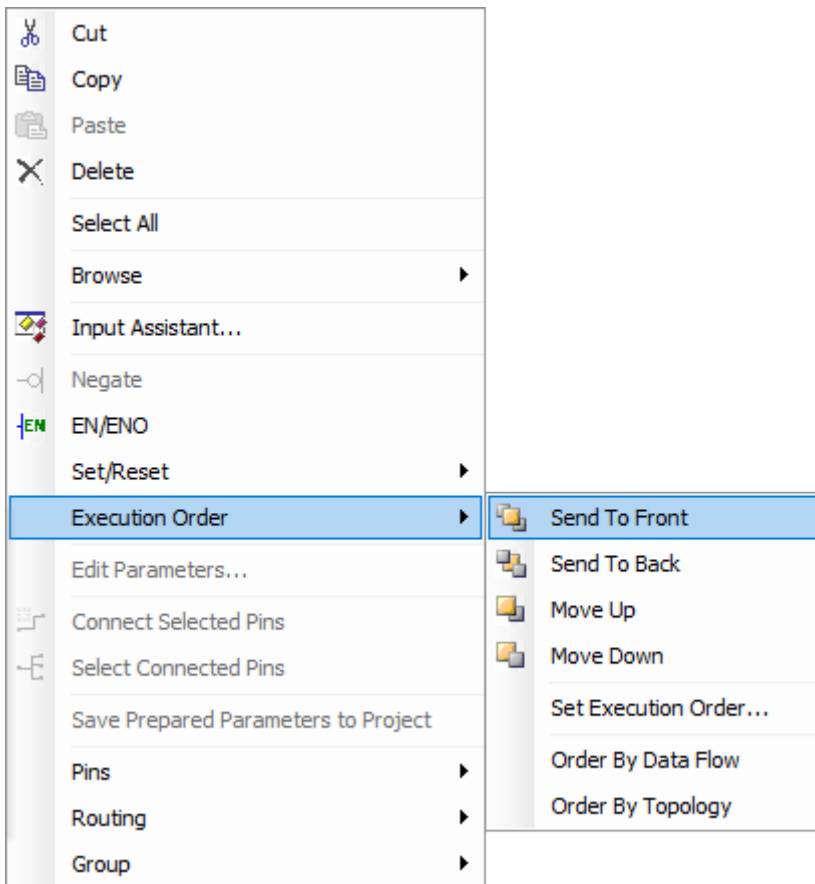
If the *ML300 Read-Write* function block is used in the application, then it must be the first item to execute in the application. If the *ML300 Read* and *ML300 Write* function blocks are used in the application, then the *ML300 Read* function block must be the first item to execute in the application. The *ML300 Write* function block must always be the last item to execute in the application.

The position of the function block in the execution order is shown in the upper-right corner of the function block in a green box. The execution order list's numbering starts at zero.

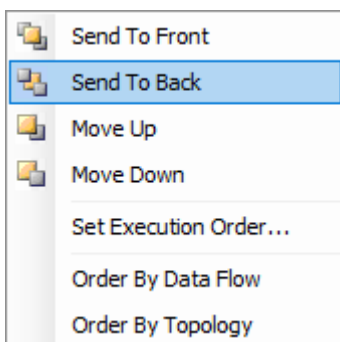
Figure 3.2 ML300 Read-Write function block with execution order zero



To set the function block to the first item in the execution order, right-click on the function block, and select **Execution Order > Send To Front**.



To set the function block to the last item in the execution order, right-click on the function block, and select **Execution Order > Send To Back**.



3.3.6 ML 300 function blocks' inputs and outputs

The ML300 function blocks give a status overview of the controller. The function blocks consists of following inputs and outputs:

Input or output	ML300_read_write	ML300_read	ML300_write
Codesys_application_OK	●		●
Timeout_cycles	●		●
Link_OK	●	●	
Error_code	●	●	
Config_conflict_error_count	●	●	
Config_conflict_errors	●	●	

Codesys_application_OK

This input shows whether the CODESYS application is running on the ML 300 controller. The CODESYS application can, for example, not be running because the CPU is overloaded, or because there is a failure in the program.

This input is connected to the *CODESYS application not OK* alarm in the controller. If the *CODESYS application not OK* alarm is enabled on the controller and the **Codesys_application_OK** input is **FALSE**, then the alarm activates on the controller.

If the input is not connected, then the default value for this input is **FALSE**.



More information

See **CODESYS** in the **Designer's handbook** for more information about configuring the *CODESYS application not OK* alarm.

Timeout_cycles

To ensure that the CODESYS program's scan period is synchronised with the CODESYS scan, the **Timeout_cycles** for the program must be configured. This input shows the maximum number of cycles that are allowed to pass without writing or reading data from the controller.

To see the scan time of the CODESYS program, open the **MainTask** in the workspace by double-clicking on **MainTask** in the project tree. The scan time value is shown under **Type > Interval (e.g. t#200ms)**. The scan time for an ML 300 controller is 40 ms.

Use the following formula to calculate the timeout cycle:

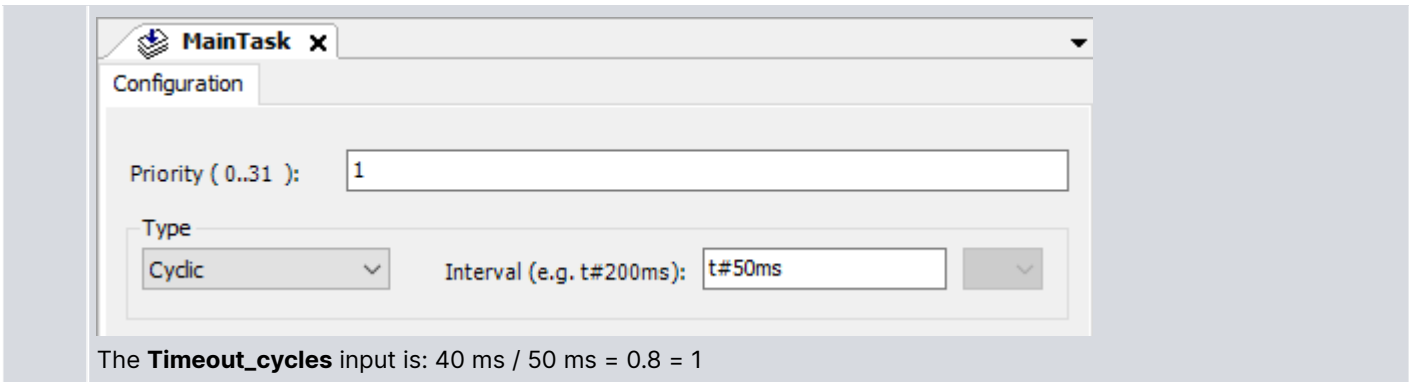
$$\text{timeout cycles} = \text{ML 300 application scan time} / \text{CODESYS scan interval} = 40 \text{ ms} / \text{CODESYS scan interval}$$

Always round the calculated timeout cycle up to determine the correct value for the **Timeout_cycles** input. For example, for a calculated timeout cycle of 1.2, use 2.



Required Timeout_cycles calculation example

The CODESYS cyclic interval is set to 50 ms, as shown in the image below.



Link_OK

The **Link_OK** output shows if there are errors present in the controller. The value of this output is determined by the value of the **Error_code** output. If the **Error_code** output is greater than 0, then **Link_OK** is *FALSE* and there is an error present in the CODESYS program on the controller.

This output is shown as a boolean value.

Error_code

The table below lists the error codes that can occur and their descriptions. The error code shown is the sum of all the errors present in the application. For example, if you have the same function configured on the controller and in CODESYS (error code 2) and you have more than one ML 300 function block present in your application (error code 64), then the error code shown is 66.

Error code	Description
1	The shared memory size has been exceeded.
2	The same function is configured in CODESYS and on the controller.
4	The controller did not read or write data on the shared memory yet.
8	The controller did not read or write data on the shared memory inside the Timeout_cycles limit.
16	The same LDO UID is used more than once.
32	The versions of the controller process, CODESYS process and library do not match. The application consists of the wrong number of main function blocks.
64	You are not allowed to have an ML300_read_write function block and the ML300_read function block and/or ML300_write function block in your application at the same time.
128	An unidentified error. Please contact DEIF service and support for assistance.

The error code is shown as an integer.

Config_conflict_error_count

The **Config_conflict_error_count** output shows how many I/O functions are configured in both the controller and in CODESYS. You can see which I/O functions are causing the conflict by referring to the values shown under the **Config_conflict_errors** output.

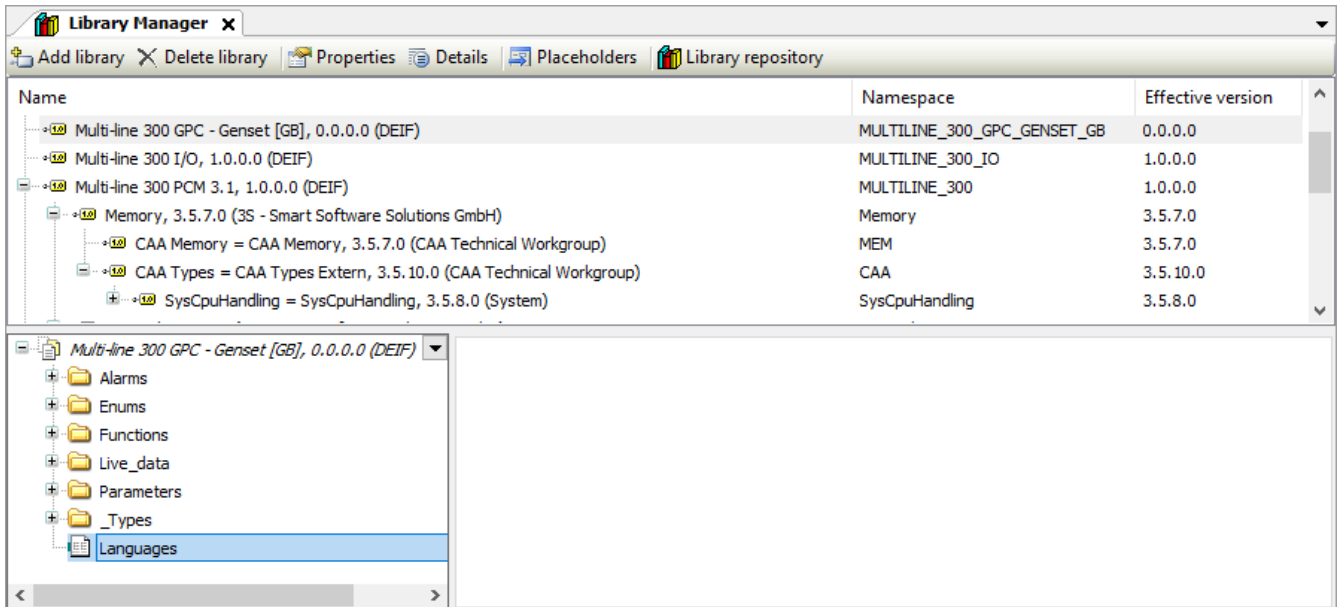
This output is shown as a integer value.

Config_conflict_errors

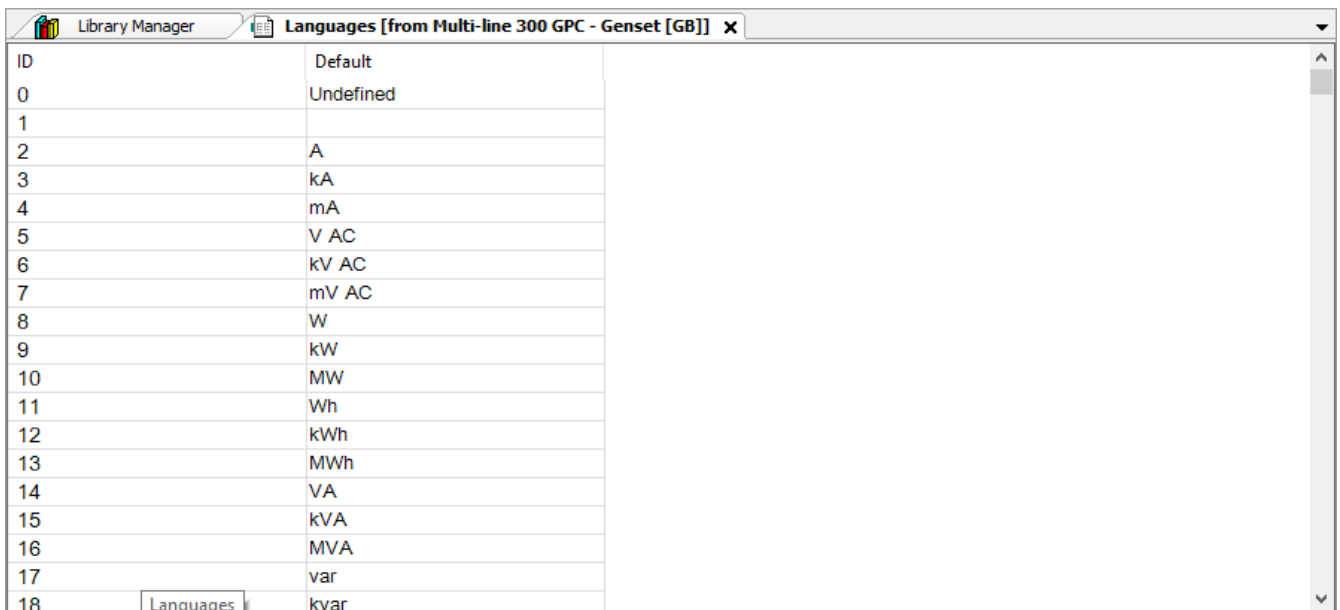
If the same controller function is configured in CODESYS and on the controller, then this output shows the *Text ID* of the IO that is causing the conflict.

This output is an array of 20 values that are shown as double integers. Follow these steps to relate the output double integer value to text:

1. Open the *Library Manager* tab for the *Application*.
2. Select the controller specific library from the list of available libraries for the application.
 - For example, *Multi-line 300 GPC - Genset [GB]* for a GPC 300 GENSET controller.
3. Double-click on "Languages" in the bottom part of the library workspace to open the *Languages* tab for the controller specific library.



4. Look up the value in the table to find the text description for the controller function.

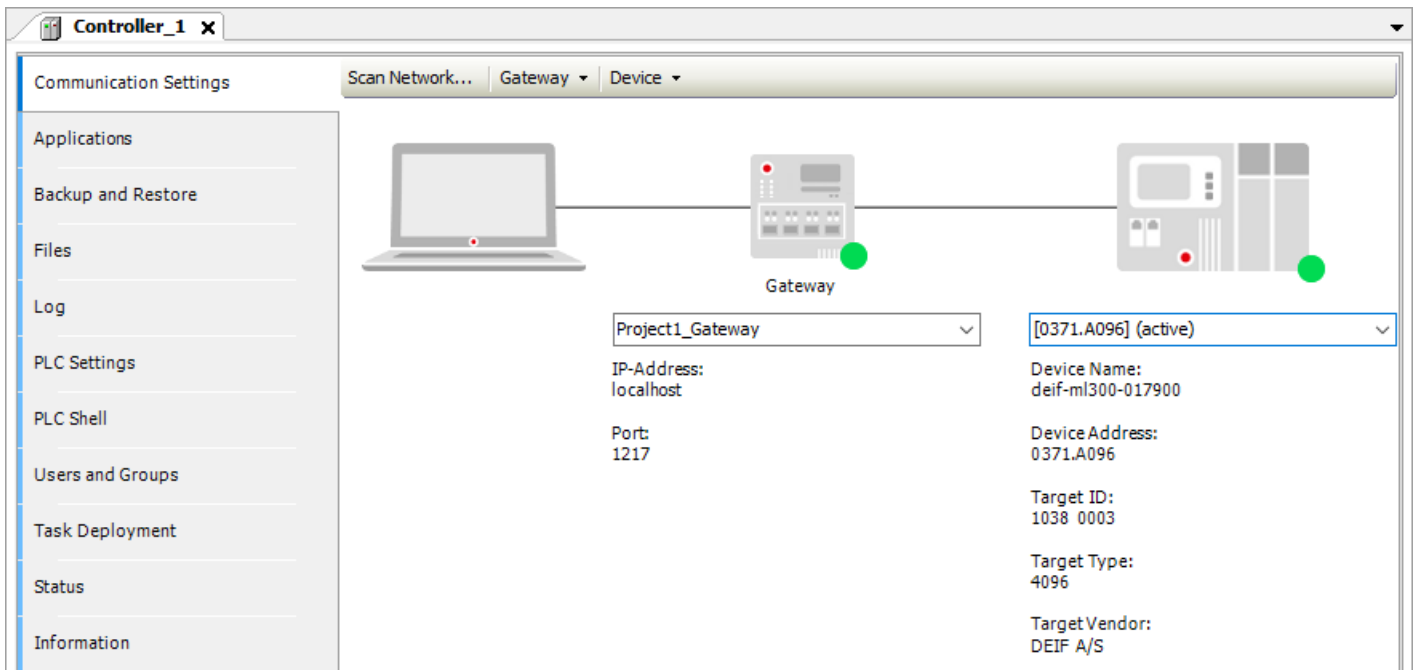


3.4 Communication with the controller

3.4.1 Introduction

To download your program to the controller, you have to establish a connection between CODESYS and the ML 300 controller. You can establish the connection through the *Device* tab in the working area.

Figure 3.3 Example of an established connection between CODESYS and the ML 300 controller



To open the *Device* tab in the working area, double-click on the device (for example, *Device (Multi-line300 PCM 3.1)*) in the project tree.

To connect to the controller:

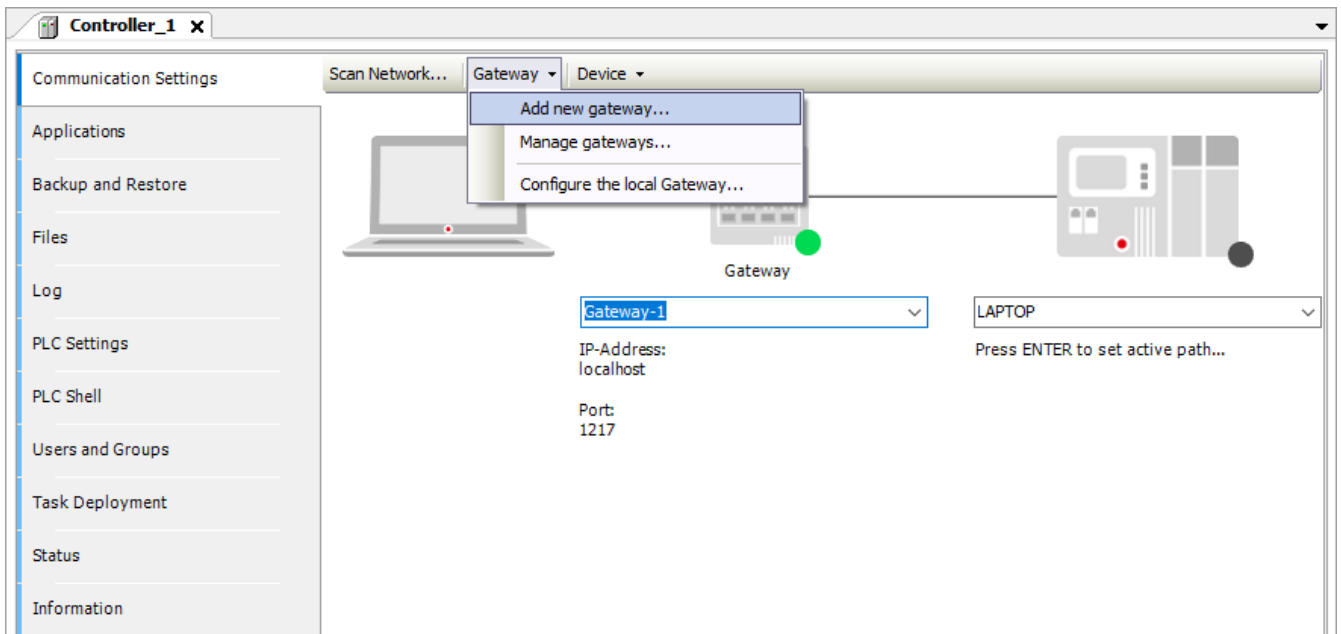
1. Create a local gateway.
 - This is only required if this is the first time you run CODESYS.
2. Use the gateway to search for ML 300 controllers that have CODESYS installed.

3.4.2 Create a local gateway

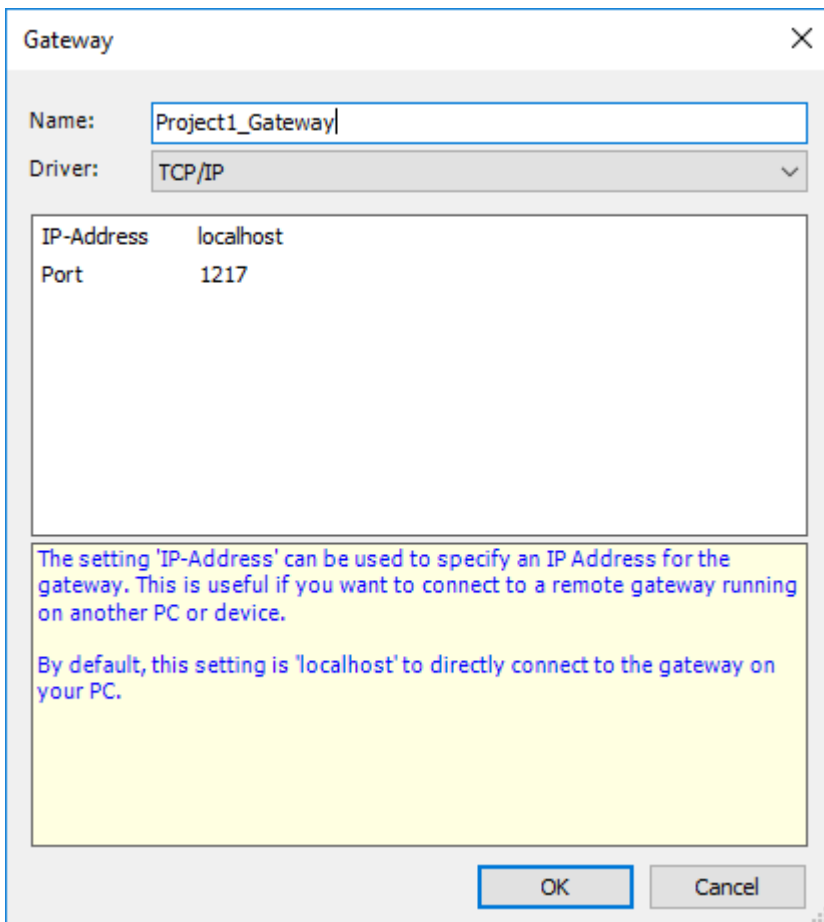
You only need to create a local gateway the first time you create a CODESYS project. This gateway will be used by default when you create new projects and it is the only defined gateway.

Follow these steps to create a local gateway:

1. Open the *Device* tab in the working area.
 - Double-click on the device (for example, *Device (Multi-line300 PCM 3.1)*) in the project tree to open the *Device* tab.
2. Under **Communication settings** select **Gateway > Add new gateway...**:

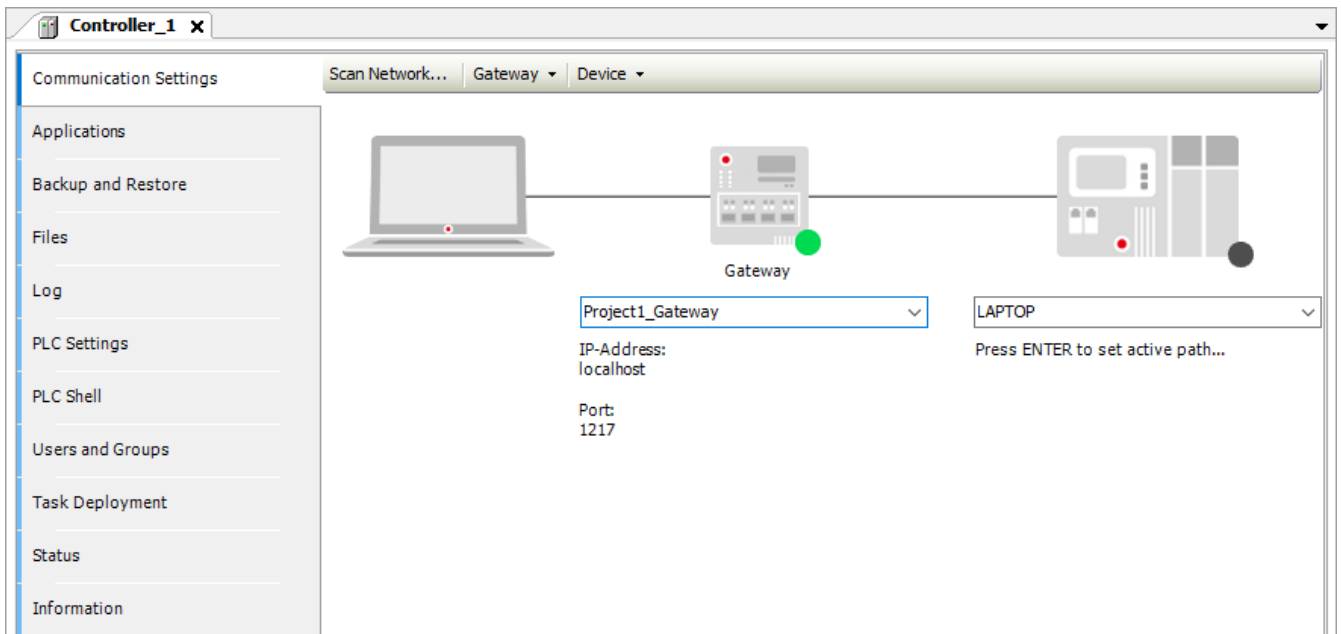


3. Complete the information in the **Gateway** window:



- Enter a name for the gateway.
- Set **Driver** to *TCP/IP*.
- Keep **IP-Address** set to *localhost*.
- Keep **Port** set to the default value.
- Select **OK**.

4. A green dot next to the gateway icon shows that the selected gateway is running without any errors:



- A red dot next to the gateway icon shows that there is a gateway error.

3.4.3 Connect to the controller

To detect and connect to a controller on the network:

- Your computer must be directly connected to the controller network.
- A local gateway must be defined and selected in CODESYS.
- The local gateway must be running without any errors.
- The controller must have an IPv4 address configured.

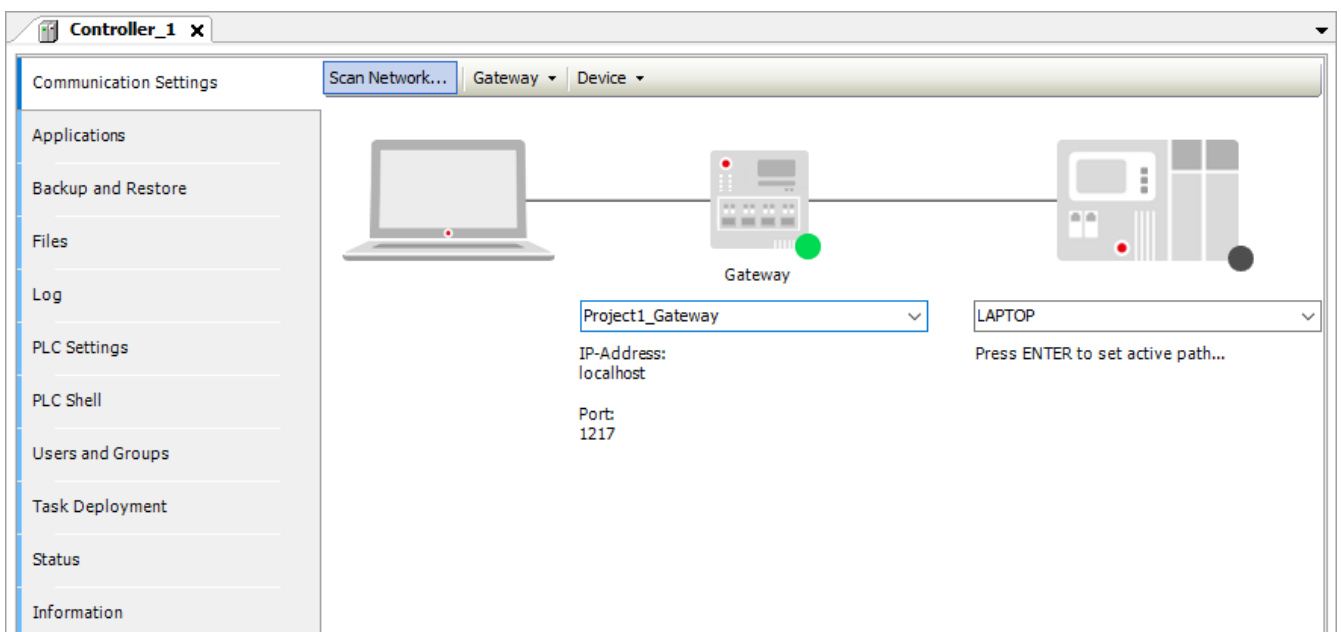


More information

See **Tools, Communication** in the **Operators manual** for more information about setting up an IPv4 address for the controller.

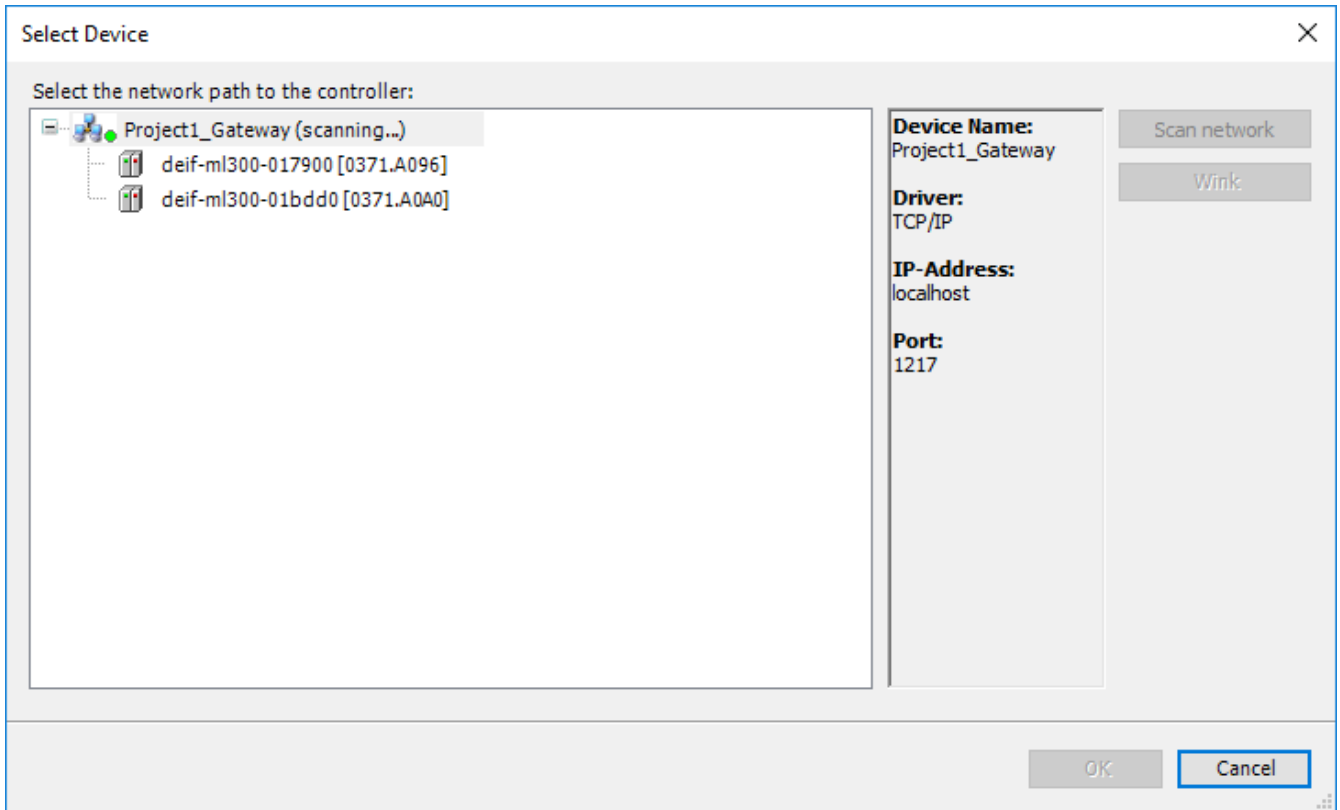
Follow these steps to connect to the controller:

1. Open the *Device* tab in the working area.
 - Double-click on the device (for example, *Device (Multi-line300 PCM 3.1)*) in the project tree to open the *Device* tab.
2. Under **Communication settings** select **Scan Network...**:



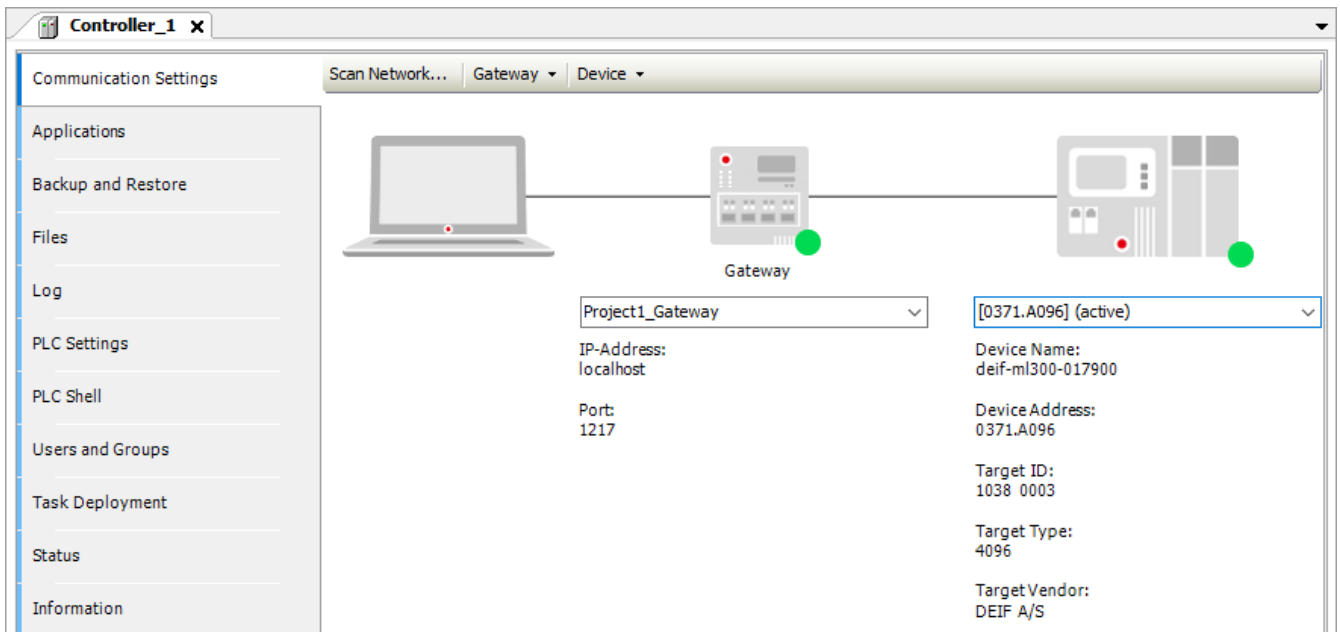
3. CODESYS automatically starts scanning the network through the selected gateway. A list of controllers with CODESYS installed is displayed.

- Select **Scan network** to start scanning the network if the scan does not start automatically.



4. Select the controller from the list that you wish to communicate with and select **OK**.

5. A green dot next to the controller icon shows that the communication with the selected controller is running without any errors:




- A grey dot next to the controller icon shows that there is no communication between the selected controller and CODESYS.
- A red dot next to the controller icon shows that there is an error communicating with the selected controller.

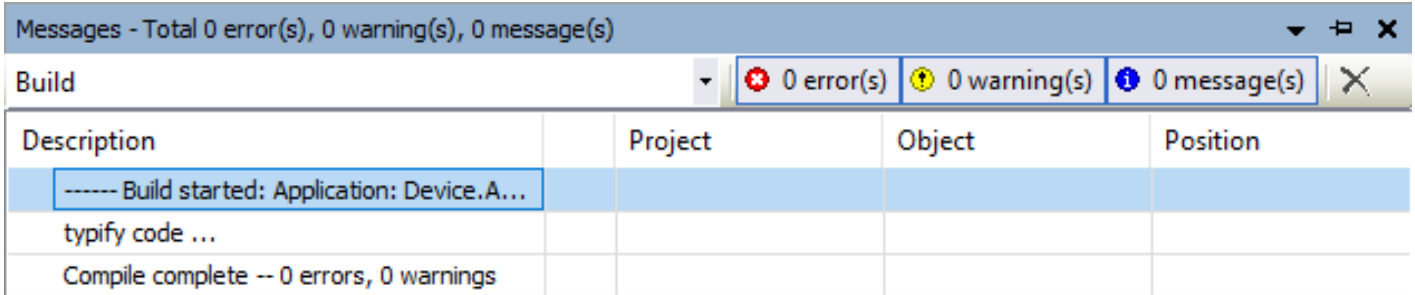
If you are not able to communicate with the Multi-line 300 controller, ensure that the controller is configured with an IPv4 address, and that your PC is within the same IP range as the controller.

3.5 Download the application to the controller

3.5.1 Pre-compile the application

To pre-compile your application, select **Build**  from the toolbar, or press *F11*. Pre-compiling can be used to check for syntax errors in your application without generating any code.

After the CODESYS finished pre-compiling, the result will be displayed in the **Messages** window. The **Messages** window is placed at the lower part of the user interface by default.



NOTE You can log into the controller without pre-compiling your application.

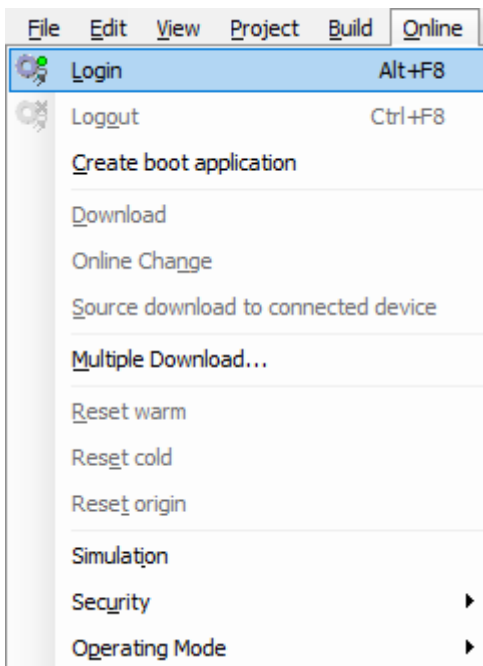
3.5.2 Generate and download the application

The application is automatically generated when you log on to the controller. To generate your code before logging onto the controller, open the **Build** menu and select *Generate code*.

Before logging on to the controller, check that the communication with the selected controller is running without any errors. That is, there is a green dot next to the controller icon in the *Communication settings* screen in the **Device** working area.

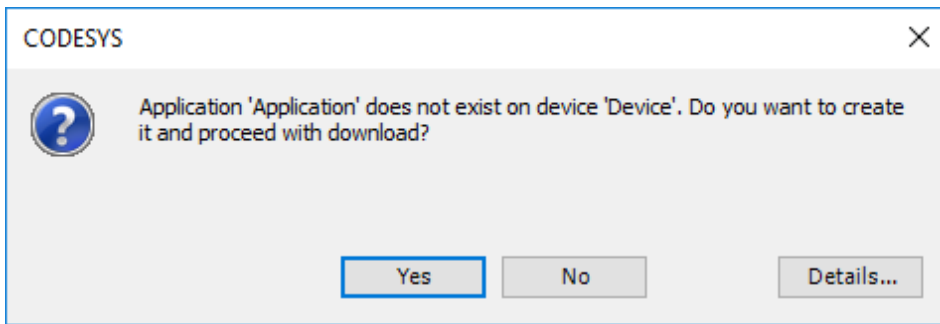
Your application is automatically downloaded to the controller when you log on to the controller. Follow these steps to log on to the controller:

1. Select **Online > Login**:



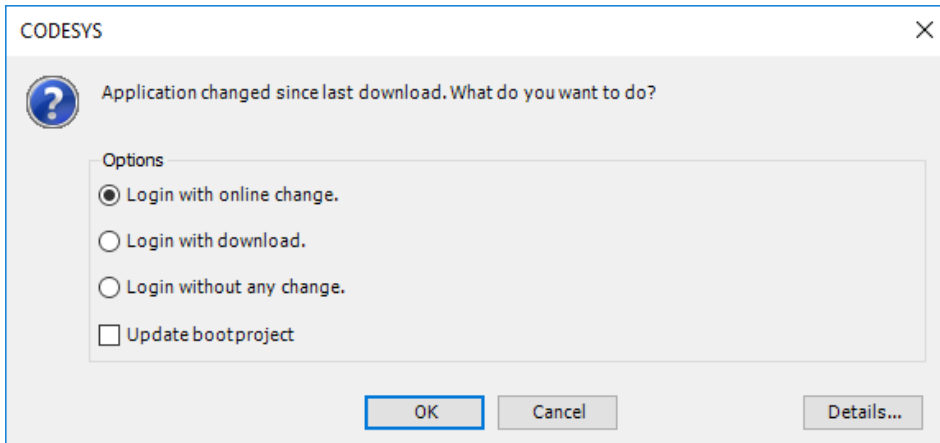
2. If the communication settings have been configured correctly, the following message boxes can appear:

- First connection:



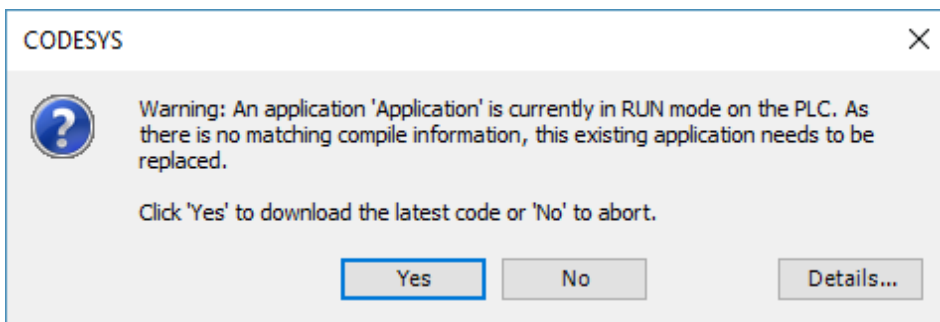
-
- Select **Yes** to download the program to the controller.

- If the application has been updated since the last download:



-
- Select *Login with download* and select **OK** to download the latest version of the application to the controller.
- Ensure *Update bootproject* is selected if you want your project to be available on the controller after you power cycle the controller.

- If the application is running on the controller, and CODESYS cannot determine if it is the same application:



-
- Only if you are sure it is safe to replace the application on the controller, select **Yes** to download the new application to the controller.

3. Once the application is successfully downloaded to the controller, CODESYS displays the result in the **Messages** window.

- If the project has been created correctly, you will not receive any compilation errors and you can run the application.
- If CODESYS detected any compilation errors during the download, they are also displayed in the **Messages** window.

NOTE The CODESYS manual also refers to being logged on to the controller as *online mode*, and the POU's displayed in the working area while logged on as *online views*.

3.5.3 Start and stop the application

When you are logged on to the controller you can start or stop the CODESYS application:


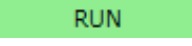


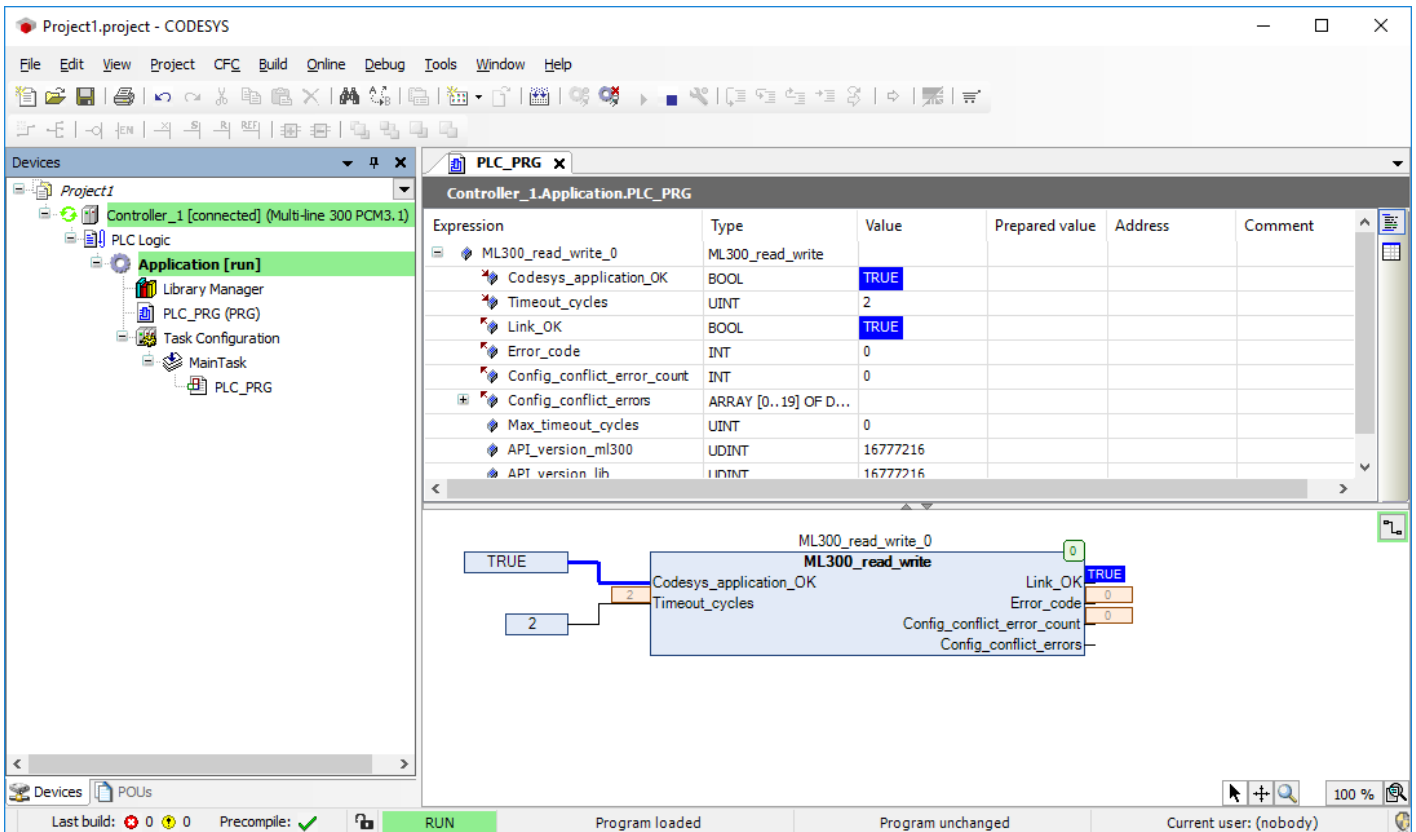
Action	Toolbar shortcut	Keyboard shortcut	Status bar
Start your application		F5	RUN  is visible when the program is running.
Stop your application		Shift + F8	STOP  is visible when the program is running.

Figure 3.4 Example of a running application



3.6 Monitor the application

3.6.1 Introduction

When the program is running on the controller and you are logged on to the controller through CODESYS, you can monitor the values of the variables. It is also possible to change the value of some variables while you are monitoring the variable values.

You can monitor the variable values of a specific POU by opening the POU in the working area.

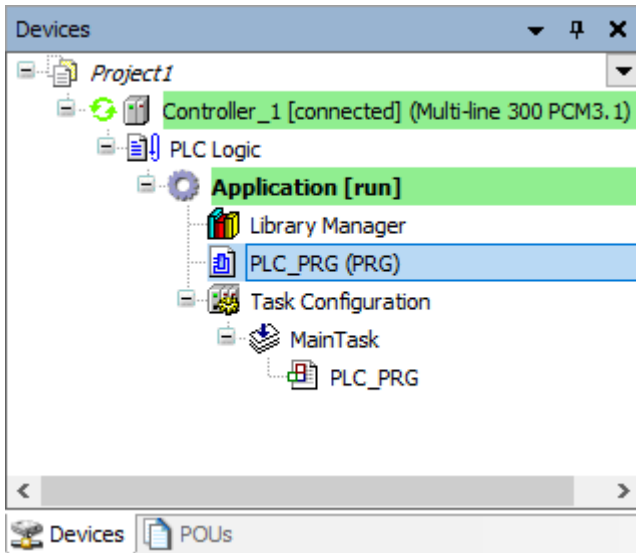
If you want to monitor the variable values of more than one POU at the same time or a specific set of variables, you can create a *Watch window*.

It is possible to change the variable values in the working area or the watch window by writing or forcing a new variable value to the controller.

3.6.2 Monitor in the working area

When you are logged on to the controller through CODESYS you can open a POU to monitor the variables in the working area. To open the POU in the working area, simply double-click on the POU in the project tree. Alternatively you can select the POU in the project tree and select **Edit object** from the right-click menu.

Figure 3.5 The POU, PLC_PRG, can be opened in the working area by double-clicking on it



In the *declaration part* of the open POU in the working area, the variable watch list is shown:

Expression	Type	Value	Prepared value	Address	Comment
ML300_read_write_0	ML300_read_write				
Codesys_application_OK	BOOL	TRUE			
Timeout_cycles	UINT	2			
Link_OK	BOOL	TRUE			
Error_code	INT	0			
Config_conflict_error_count	INT	0			
Config_conflict_errors	ARRAY [0..19] OF D...				
Max_timeout_cycles	UINT	0			
API_version_ml300	UDINT	16777216			
API_version_lib	UDINT	16777216			

All the variables relating to the open POU is shown in this list. You can change the values of some variables using this table.

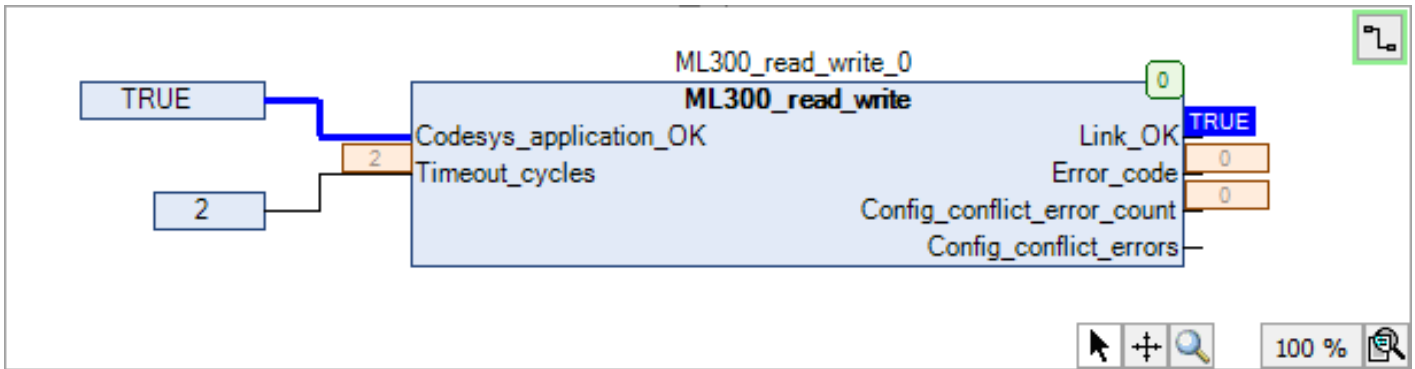


More information

See **Writing and forcing variables** for more information about how to change the variables values.

In the *implementation part* of the open POU in the working area, the program function diagram, ladder logic, or code is shown:

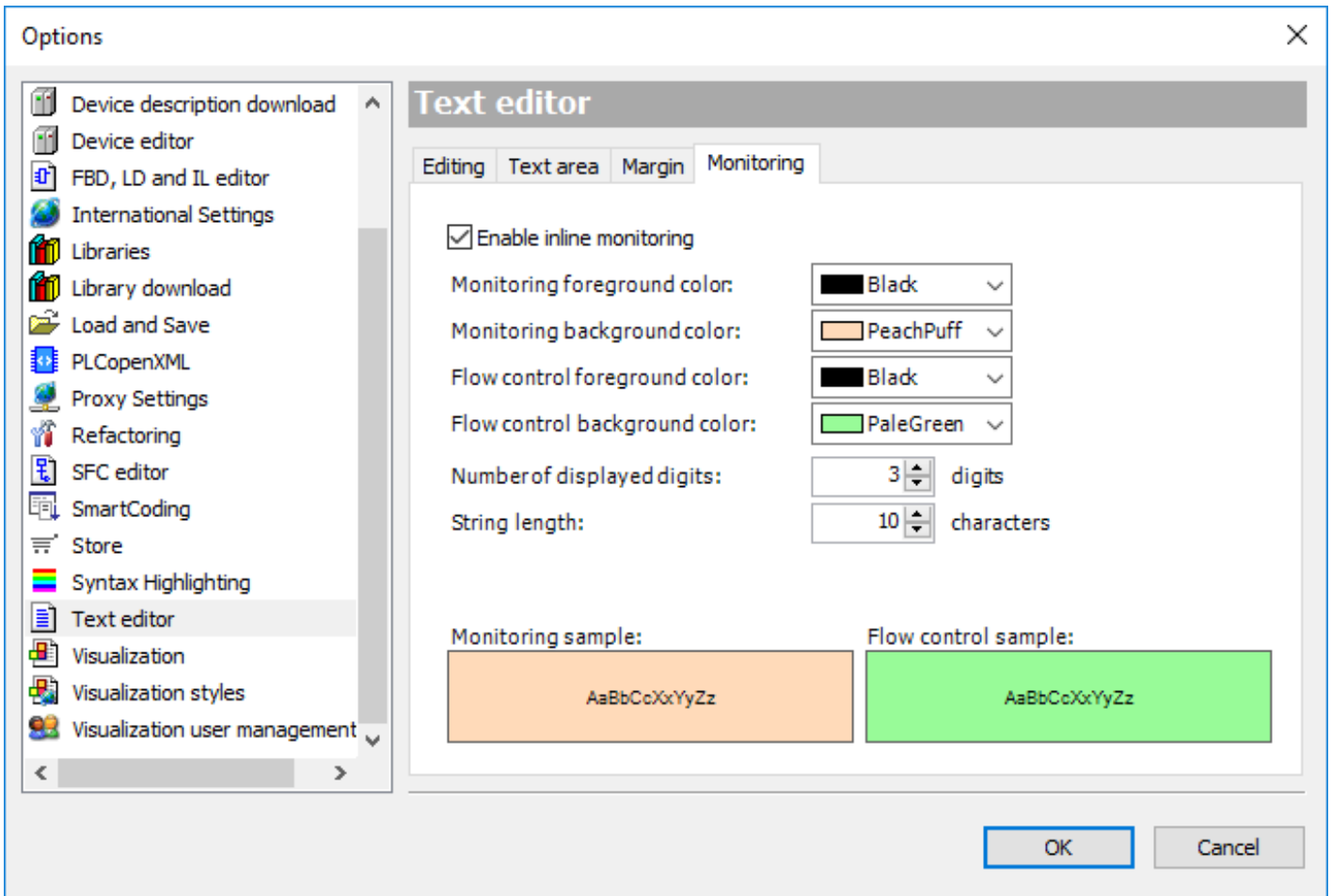
Figure 3.6 An ML 300 function block with inline monitoring activated



If *Inline monitoring* is activated, the inline monitoring boxes are placed behind each variable in the code, or next to the variable in the function block. The inline monitoring boxes shows the actual value of the variable in real time.

To activate or deactivate *Inline monitoring* go to **Tools > Options** to open the **Options** window. Activate or deactivate the function under **Text editor > Monitoring > Enable inline monitoring**.

Figure 3.7 Activate or deactivate Inline monitoring in the Options window



3.6.3 Monitor in watch windows

Watch windows are useful to monitor specific variables in a POU, or to monitor variables from different POUs in a single window.

You can also change the values of some variables in the watch window. This is useful, for example, to debug code.

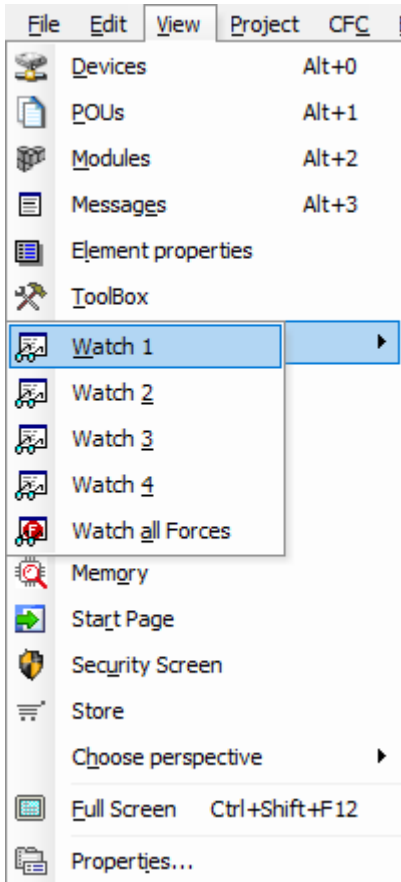


More information

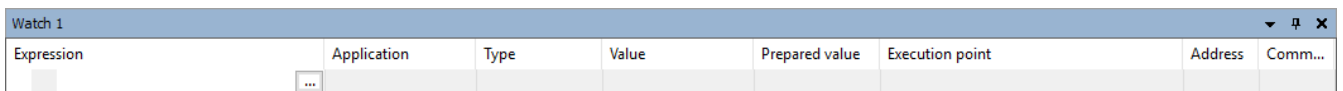
See **Writing and forcing variables** for more information about how to change the variables values.


Follow these steps to create a watch window:

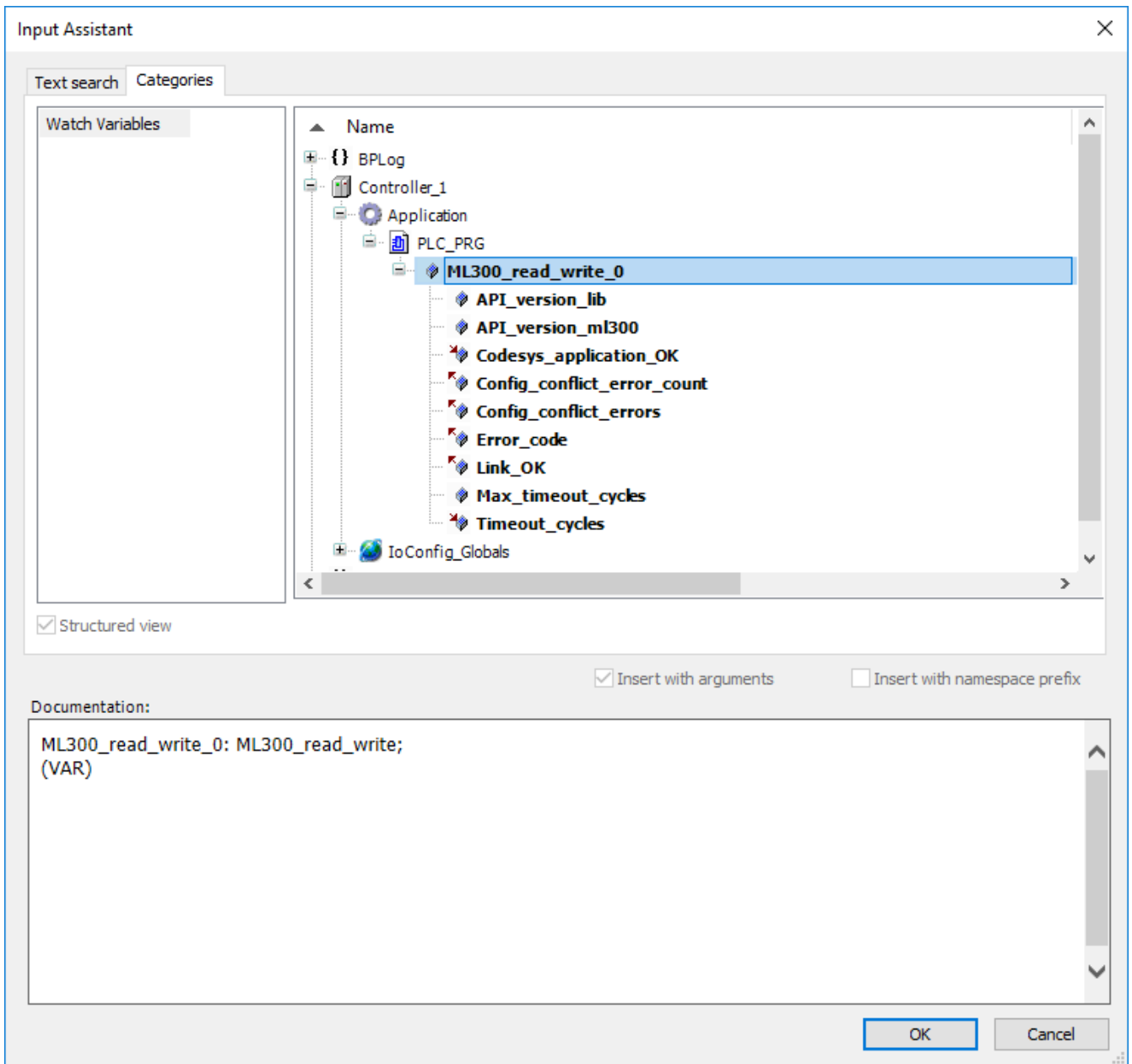
1. Select **Watch 1** from **View > Watch** to open a watch window.



2. Double-click in an empty cell in the **Expression** column.



3. Select the **Input assistant**  to open the **Input assistant** window.
4. Under **Categories > Watch Variables**, select a variable or group of variables (for example ML300_0) in a POU to watch:



- Select **OK** to confirm your selection and continue.
5. Press the *Return* key to add your selected variable(s) to the watch window.
- Details for the variable(s) (for example, the application and type) are automatically added to the watch window.
6. You can monitor the selected variable(s) in the watch window:

Expression	Application	Type	Value	Prepared value	Execution point	Address	Comment
PLC_PRG.ML300_read_write_0	Controller_1.Application	ML300_read_write			Cyclic Monitoring		
Codesys_application_OK		BOOL	TRUE		Cyclic Monitoring		
Timeout_cycles		UINT	2		Cyclic Monitoring		
Link_OK		BOOL	TRUE		Cyclic Monitoring		
Error_code		INT	0		Cyclic Monitoring		
Config_conflict_error_count		INT	0		Cyclic Monitoring		
Config_conflict_errors		ARRAY [0..19] OF DINT			Cyclic Monitoring		
Max_timeout_cycles		UINT	0		Cyclic Monitoring		
API_version_ml300		UDINT	16777216		Cyclic Monitoring		
API_version_lib		UDINT	16777216		Cyclic Monitoring		

- It is also possible to change the values of some variables using this watch window.

3.6.4 Write and force variables

Some variables in your program can be changed while the program is running. To change the value of the variables in your program, first prepare a new value for the variable. Then **Write** or **Force** the prepared value to the program.

Preparing variables

To change the value of a variable, you must first prepare a replacement value. The new value for the variable is stored in the *Prepared value* column. The new variable remains in the *Prepared value* column until the user chooses to **Write** or **Force** the prepared value to the variable. You can prepare multiple variables and **Write** or **Force** all of the prepared values at the same time.

To prepare a variable of the type *INT*, *DINT*, *UINT*, or *STRING*:

1. Double-click on the field in the *Prepared value* column.
2. Enter the new value.
3. Press the **Return** key or click outside the field.

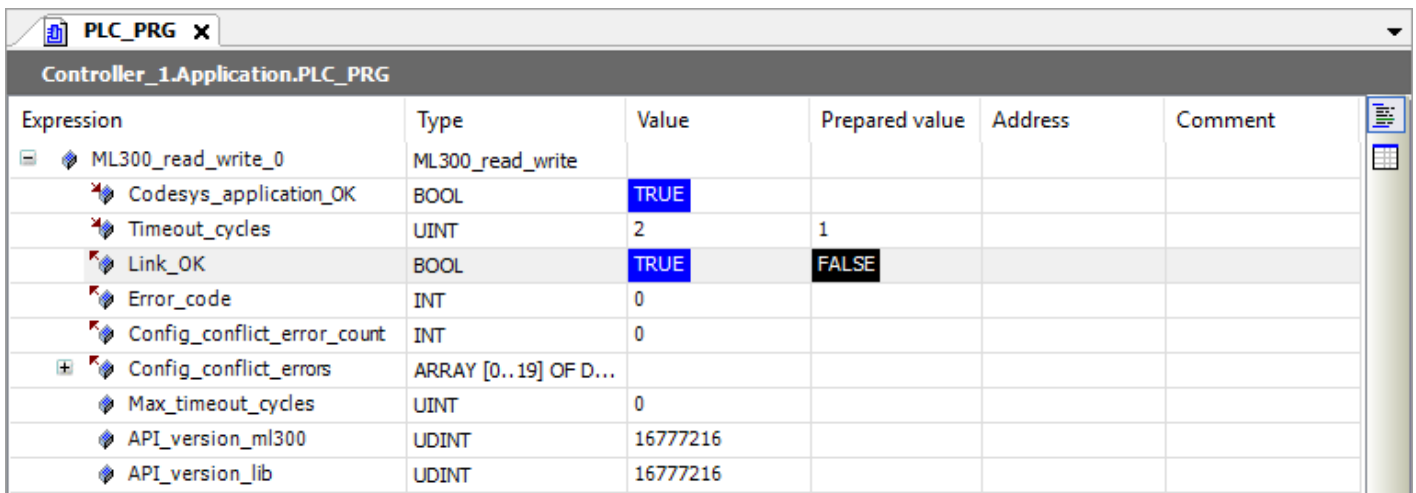
The *Prepared value* is ready to be written or forced to the program.

To prepare a variable of the type *BOOL*:

1. Click on the field in the *Prepared value* column until the desired value appears.

The *Prepared value* is ready to be written or forced to the program.

Figure 3.8 Example of a UINT and BOOL prepared value



Expression	Type	Value	Prepared value	Address	Comment
ML300_read_write_0	ML300_read_write				
Codesys_application_OK	BOOL	TRUE			
Timeout_cycles	UINT	2	1		
Link_OK	BOOL	TRUE	FALSE		
Error_code	INT	0			
Config_conflict_error_count	INT	0			
Config_conflict_errors	ARRAY [0..19] OF D...				
Max_timeout_cycles	UINT	0			
API_version_ml300	UDINT	16777216			
API_version_lib	UDINT	16777216			

Writing variables

When you **Write** a prepared value to the program, the variable updates during the next run cycle. The new value can be updated immediately by the program during the next run cycle.

Follow these steps to **Write** a new variable value to the controller:

1. Prepare the variable value(s).
2. Select **Debug > Write values**.
 - Alternatively press *Ctrl + F7*.
3. The *Value* column updates and shows the prepared value.

Expression	Type	Value	Prepared value	Address	Comment
ML300_read_write_0	ML300_read_write				
Codesys_application_OK	BOOL	TRUE			
Timeout_cycles	UINT	1			
Link_OK	BOOL	TRUE			
Error_code	INT	0			
Config_conflict_error_count	INT	0			
Config_conflict_errors	ARRAY [0..19] OF D...				
Max_timeout_cycles	UINT	0			
API_version_ml300	UDINT	16777216			
API_version_lib	UDINT	16777216			

Some variables cannot be updated. Typically such variables receive their input from a different source. If you attempt to **Write** a new value to such a variable, then the variable value remains unchanged.

Forcing variables

When you **Force** a prepared value to the program, the variable updates during the next run cycle. The forced value remains unchanged in the value column, until it is unforced.

Forced variables remain in the system until unforced by the user. The user always receives a warning, when there are forced variables remaining in the application when logging out.

Follow these steps to **Force** a new variable value to the controller:

1. Prepare the variable value(s).
2. Select **Debug > Force values**.
 - a. Alternatively press *F7*.
3. The *Value* column updates and shows the forced values.
 - The **F** icon to the left of the value indicates that the variable is a forced variable.

Expression	Type	Value	Prepared value	Address	Comment
ML300_read_write_0	ML300_read_write				
Codesys_application_OK	BOOL	F TRUE			
Timeout_cycles	UINT	F 1			
Link_OK	BOOL	F FALSE			
Error_code	INT	0			
Config_conflict_error_count	INT	0			
Config_conflict_errors	ARRAY [0..19] OF D...				
Max_timeout_cycles	UINT	0			
API_version_ml300	UDINT	16777216			
API_version_lib	UDINT	16777216			

Some variables cannot be updated. Typically such variables receive their input from a different source. If you attempt to **Force** a new value to such a variable, then the value in the *Value* column changes to appear as if it is forced, but any outputs connected to it will show the original value.

4. Function blocks

4.1 Version function block

4.1.1 Introduction

Version function blocks allow you to read the version numbers of the software on the controller and the software on the hardware modules that are installed on the controller. The version function blocks are only able to read this information from the controller that is running CODESYS.

There are three version function blocks:

- Versions
- Card_info
- Software_info

The **Versions** function block gives a basic overview of the controller and the controller software, and sends the additional software version information to the **Card_info** and **Software_info** function blocks.

The **Versions** function block must be included in your program when you use the **Card_info** and **Software_info** function blocks.

The **Card_info** function block gives information about the location and the software version that is installed on the selected hardware module.

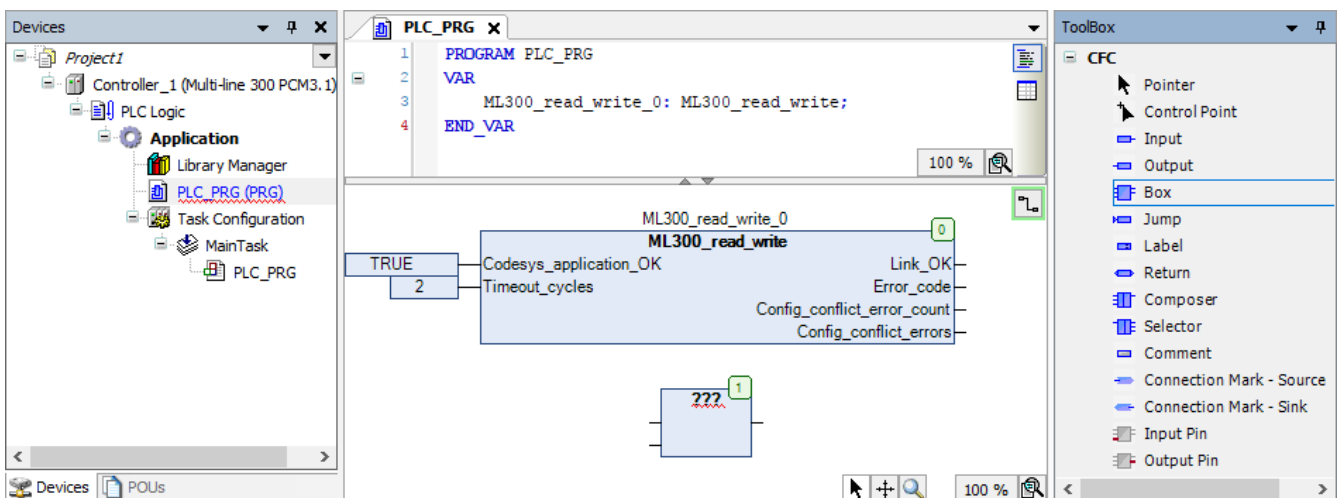
The **Software_info** function block gives information about the software version of the different software that is installed on the controller that is running CODESYS.

4.1.2 Add a version function block

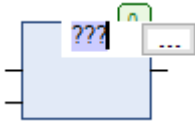
You must always add a **Versions** function block when you add a **Card_info** and/or **Software_info** function block.

Follow these steps to add a version function block (for example, **Card_info** function block) to your program:

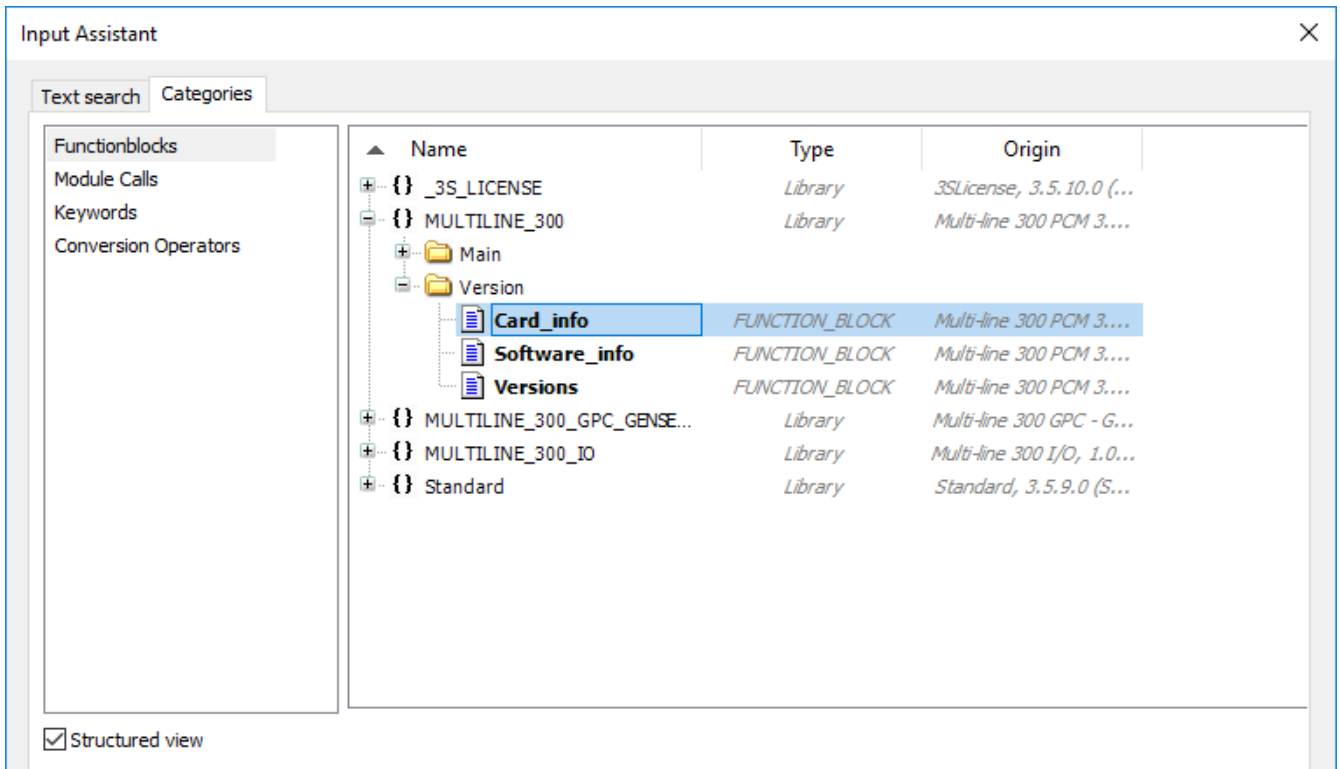
1. Drag a **Box** from the *ToolBox* to the implementation part of the working area:



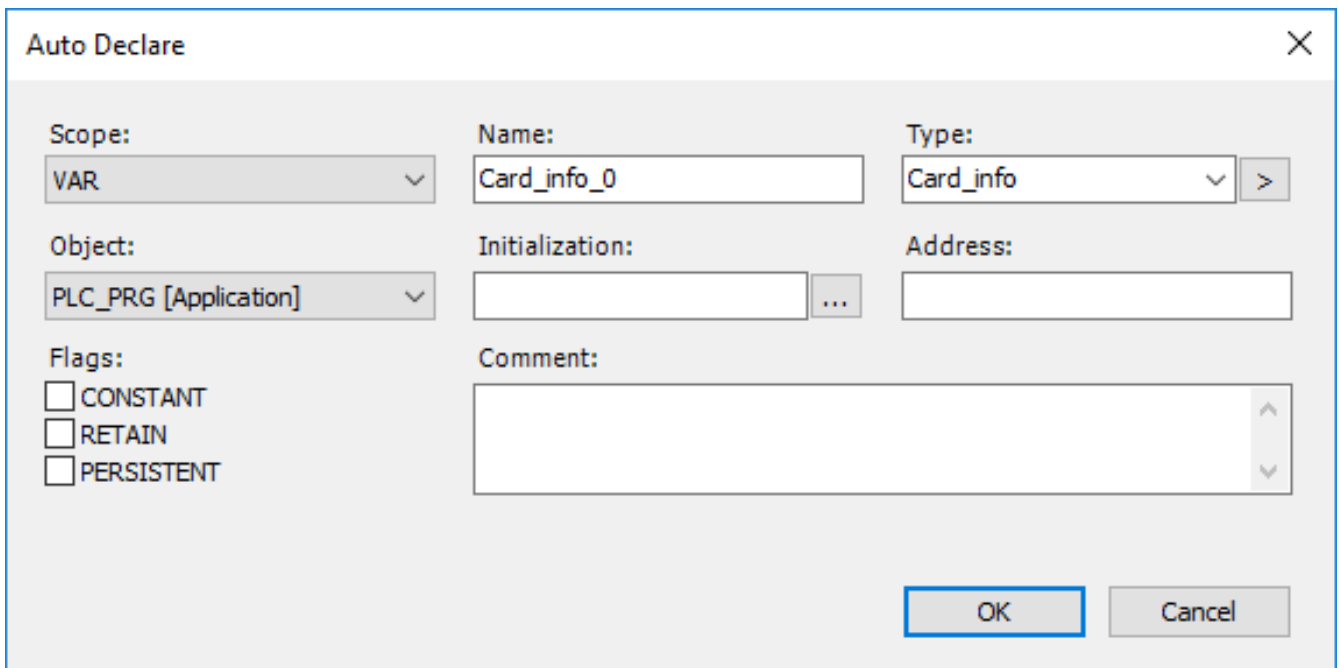
2. Select **???** in the **Box** and then select  to open the *Input Assistant* window:



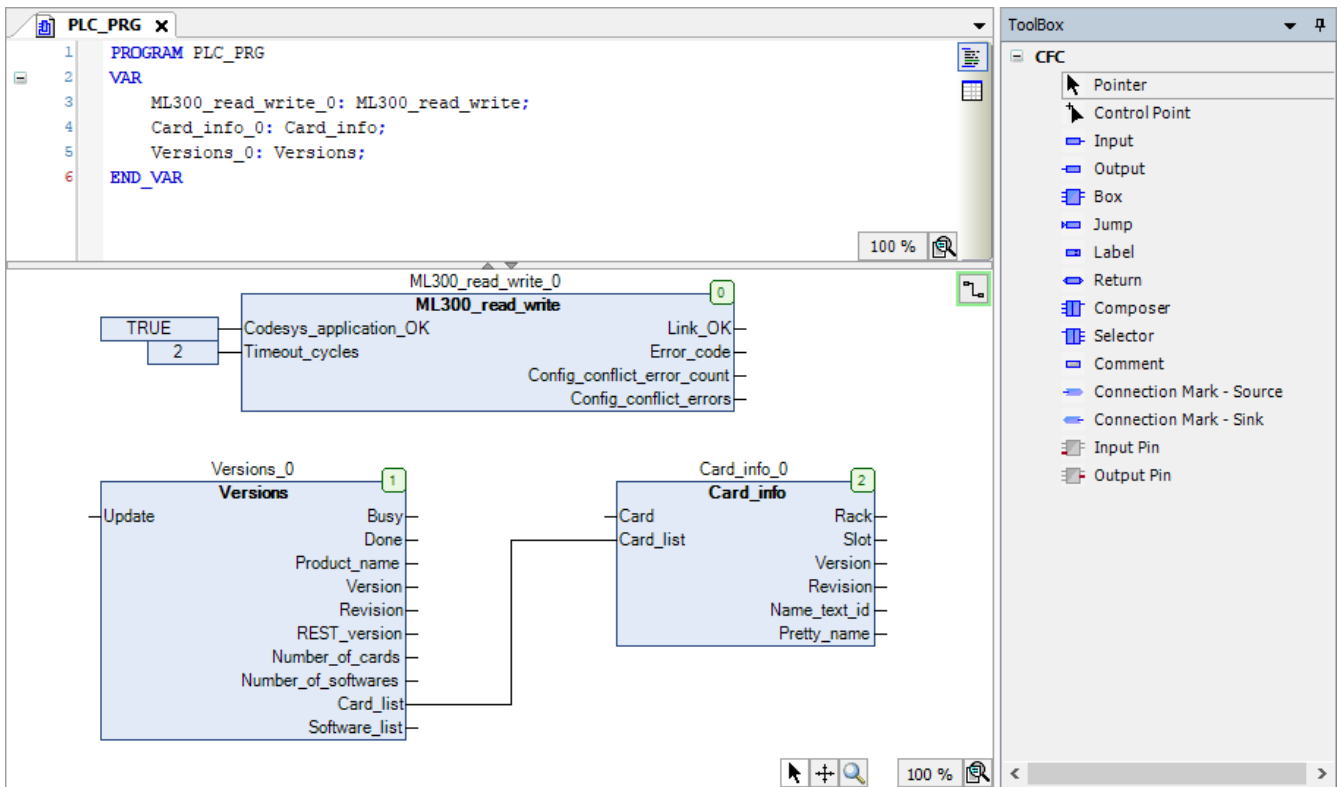
-
- 3. Go to **Categories > Functionblocks > MULTILINE_300 > Version** and select the version function block you want to add:
 - For example, **Versions** or **Card_info**.



-
- Select **OK**.
- 4. Press *Return* on your keyboard two times to open the *Auto Declare* window for the function block. Select **OK** to declare the variables as they are shown:



-
- 5. After connecting the inputs and/or outputs of the version function block in your program it is ready to be downloaded to the controller:



More information

See **ML 300 CODESYS projects**, **Download the application to the controller** for more information about how to download and run your program on an ML 300 controller.

4.1.3 Card_info function block overview

Card_info function blocks can be added to a program to read additional information about specific modules that are installed on the controller. Card_info function blocks can be used to:

- Read the location of the module.
- Read the version number of the module software.
- Read the name of the module.

Figure 4.1 Card_info function block example

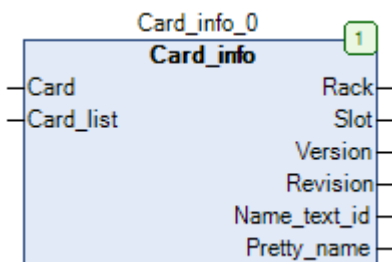


Table 4.1 Card_info function block input and output overview

Name	Input / Output	Type	Notes
Card	Input	UINT	Select the module number of the module that you want to read version information from.

Name	Input / Output	Type	Notes
			The input must be greater than zero, and must not be more than the number of cards in the rack.
			The number of modules installed in the rack can be read from the <i>Number_of_cards</i> output on the Versions function block.
Card_list	Input	Card_version_list	This input reads the card version information from the controller. This input must be connected to the <i>Card_list</i> output on the Versions function block in order for the Card_info function block outputs to output data.
Rack	Output	INT	This output displays the rack number where the module is installed.
Slot	Output	INT	This output displays the slot number where the module is installed in the rack.
Version	Output	STRING	This output displays the version number of the software installed on the module.
Revision	Output	STRING	This output displays the revision number of the version of the software installed on the module.
			This output displays the text id that the controller uses to display the module name.
Name_text_id	Output	DINT	You can also see the name of the module in the <i>Pretty name</i> output.
			Contact DEIF Service and Support if you require a document that you can use to look up the module name associated to the ID.
Pretty name	Output	STRING	This output displays the name of the module.

4.1.4 Software_info function block overview

Software_info function blocks can be used to read the version numbers of the controller software.

Figure 4.2 Software_info function block example

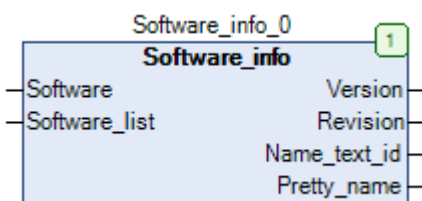


Table 4.2 Software_info function block input and output overview

Name	Input / Output	Type	Notes
			Select the software of the controller that you want to read version information from.
Software	Input	UINT	The input must be greater than zero, and must not be more than the number of software in the controller.

Name	Input / Output	Type	Notes
			The number of software can be read from the <i>Number_of_softwares</i> output on the Versions function block.
Software_list	Input	Software_version_list	This input reads the version information of the selected software from the controller. This input must be connected to the <i>Software_list</i> output on the Versions function block in order for the Software_info function block outputs to output data.
Version	Output	STRING	This output displays the version number of the selected software that is installed on the controller.
Revision	Output	STRING	This output displays the revision number of the version of the selected software that is installed on the controller.
			This output displays the text id that the controller uses to display the selected software name.
Name_text_ID	Output	DINT	You can also see the name of the selected software in the <i>Pretty name</i> output. Contact DEIF Service and Support if you require a document that you can use to look up the module name associated to the ID.
Pretty_name	Output	STRING	This output displays the name of the selected software.

4.1.5 Versions function block overview

The **Versions** function block collects the version information of the controller hardware modules and controller software, and prepares it to be displayed by the **Card_info** function block and the **Software_info** function block. The **Versions** function block also collects and displays the following controller information:

- Product name.
- Application software version.
- Application software revision.
- REST interface version.
- Number of modules installed on the controller.
- Number of software installed on the controller.

Figure 4.3 Versions function block example

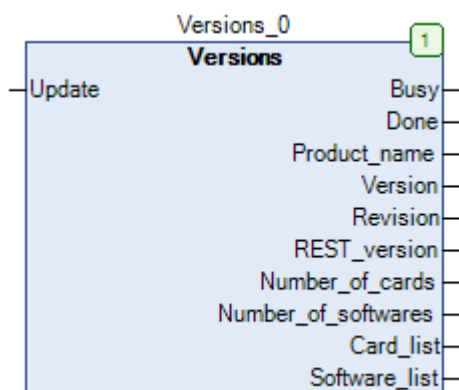


Table 4.3 Versions function block input and output overview

Name	Input / Output	Type	Notes
Update	Input	BOOL	When this input is set to TRUE , the output information is updated. The update only happens once for each time the input is toggled from FALSE to TRUE .
Busy	Output	BOOL	When this output is TRUE , the version information is being updated. When this output is FALSE , the version information is not being updated.
Done	Output	BOOL	When this output is TRUE , the version information update complete. When this output is FALSE , the version information the version information is being updated, or the version information has not been updated since the program was started on the controller.
Product_name	Output	STRING	This output displays the controller product type, for example GPC 300 or PPU 300.
Version	Output	STRING	This output displays the software version of the application software that is installed on the controller.
Revision	Output	STRING	This output displays the revision of the of the application software version that is installed on the controller.
REST_version	Output	STRING	This output displays the version of the REST interface on the controller.
Number_of_cards	Output	INT	This output displays how many modules are installed on the controller. A module can for example be a power supply module 3.1 (PSM 3.1).
Number_of_softwares	Output	INT	This output displays how many software types are installed on the controller. A software type can for example be the controller application software.
Card_list	Output	Card_version_software	This output is used to send the version information of the installed hardware modules to the Card_info function block. This output must be connected to the <i>Card_list</i> input on the Card_info function block in order for the Card_info function block outputs to output data.
Software_list	Output	Software_version_list	This output is used to send the version information of the installed software on the controller to the Software_info function block. This output must be connected to the <i>Software_list</i> input on the Software_info function block in order for the Software_info function block outputs to output data.

4.2 I/O function block

4.2.1 Introduction

I/O function blocks provide a useful interface in your program to the hardware terminals on the ML 300 controller.

The digital and analogue input values allow you to read the terminal value as measured by the controller, and use the value in your program.

Similarly your program can write the output values to digital and analogue output terminals on the controller, which can for example, be used to control equipment on the switchboard. An example can be a digital output on the controller that is connected to a warning light on the switchboard.

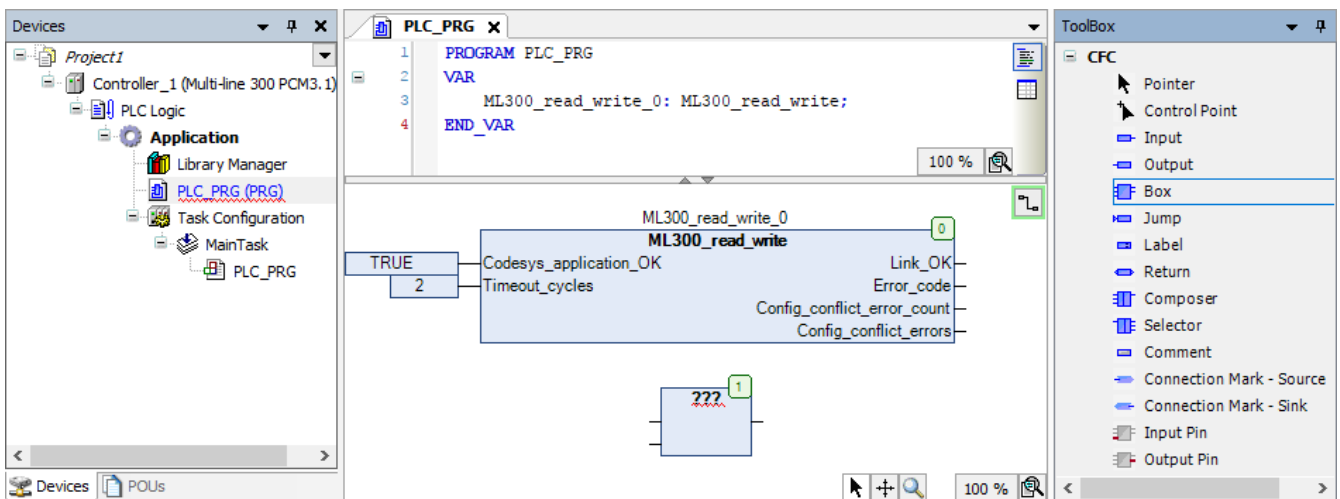
To use these I/O function blocks, you must:

1. Add the I/O function block to the CODESYS program.
2. Assign the required inputs to the function block.
3. Download the program on the controller.
4. Run the program.
5. Assign the CODESYS I/O to a free terminal on the controller.

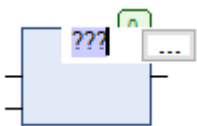
4.2.2 Add an I/O function block

Follow these steps to add an I/O function block to your application:

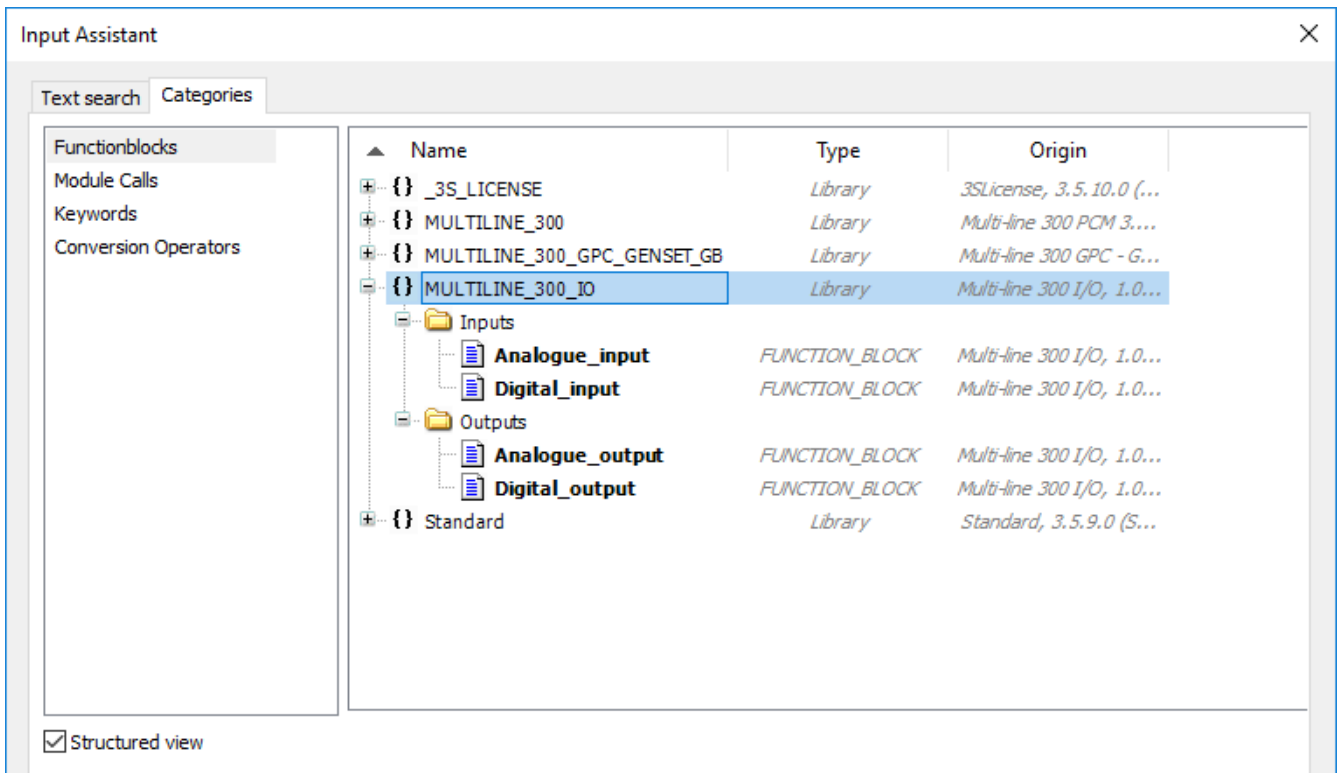
1. Drag a **Box** from the *ToolBox* to the implementation part of the working area:



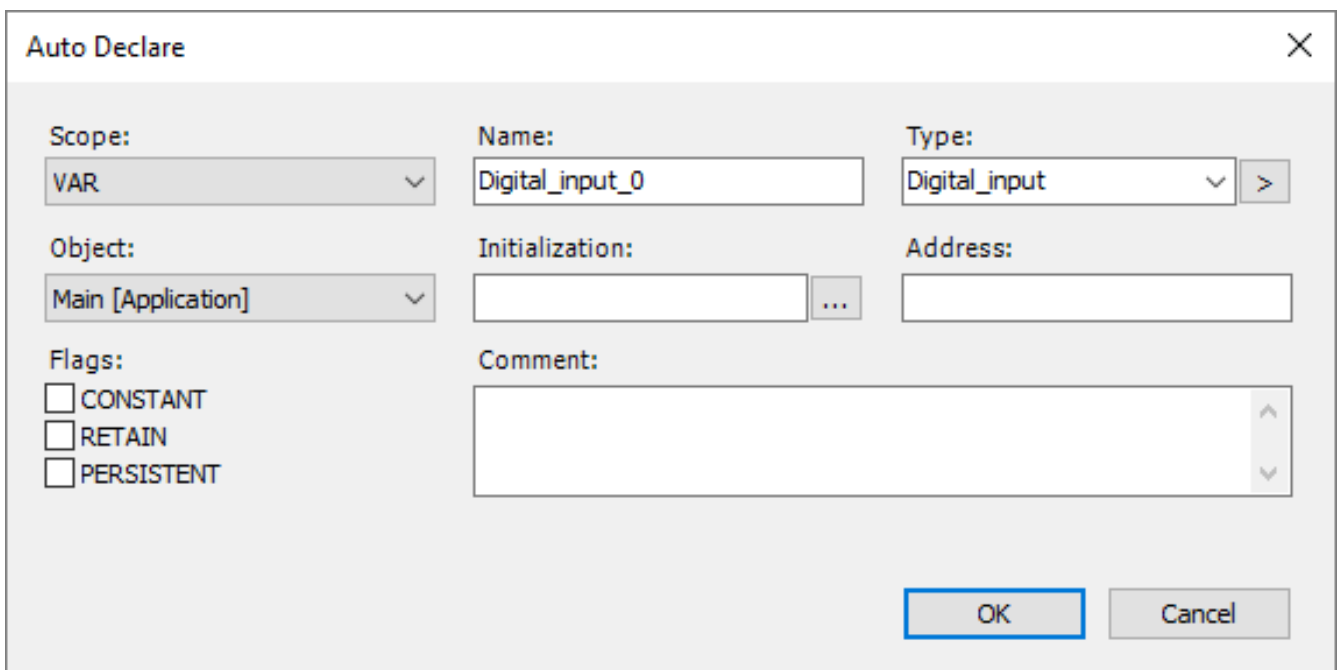
2. Select **???** in the **Box** and then select  to open the *Input Assistant* window:



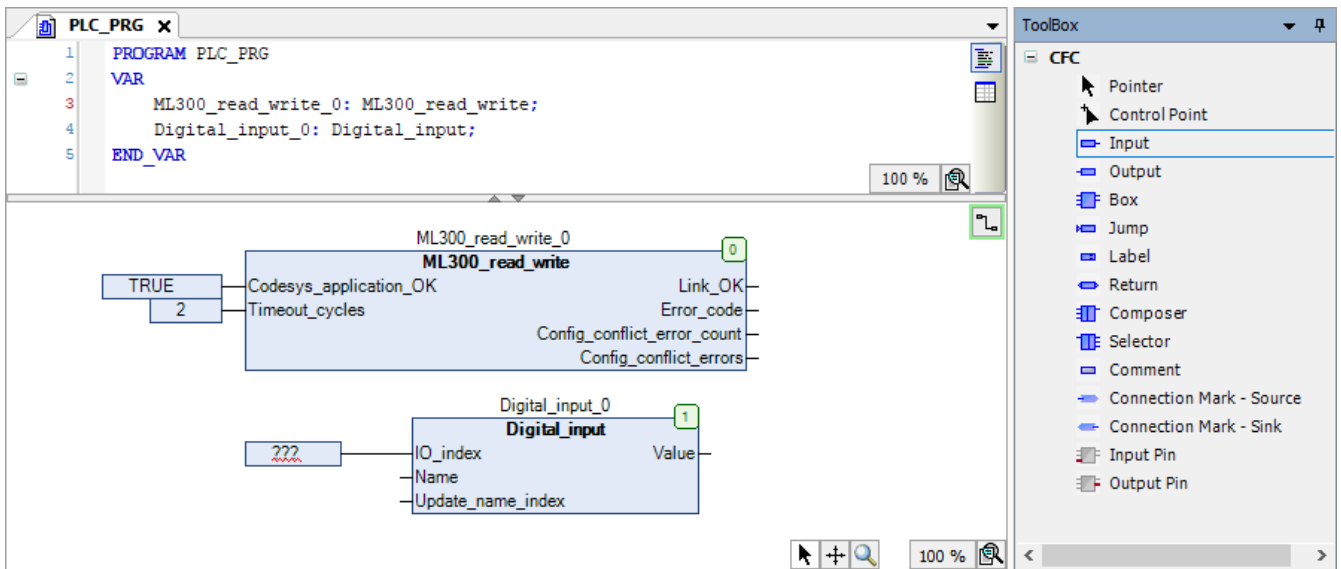
3. Go to **Categories > Functionblocks > MULTILINE_300_IO** and select the input or output you want to add:




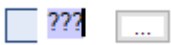
- Select **OK**.
4. Press *Return* on your keyboard two times to open the *Auto Declare* window for the function block. Select **OK** to declare the variables as they are shown:



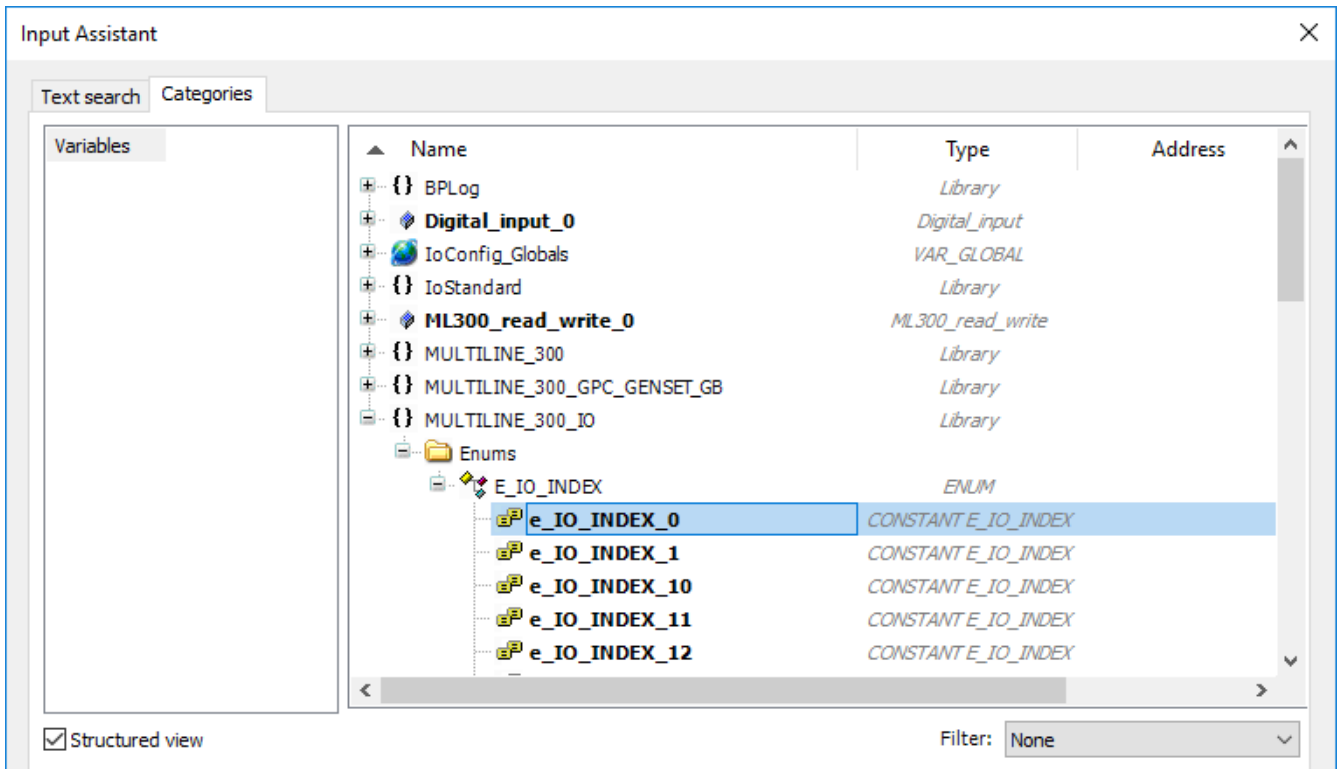
-
5. Drag an **Input** from the *ToolBox* to the implementation part of the working area and connect it to the *IO_index* input:



6. Select **???** in the **Input** and then select  to open the *Input Assistant* window for the *Input*:

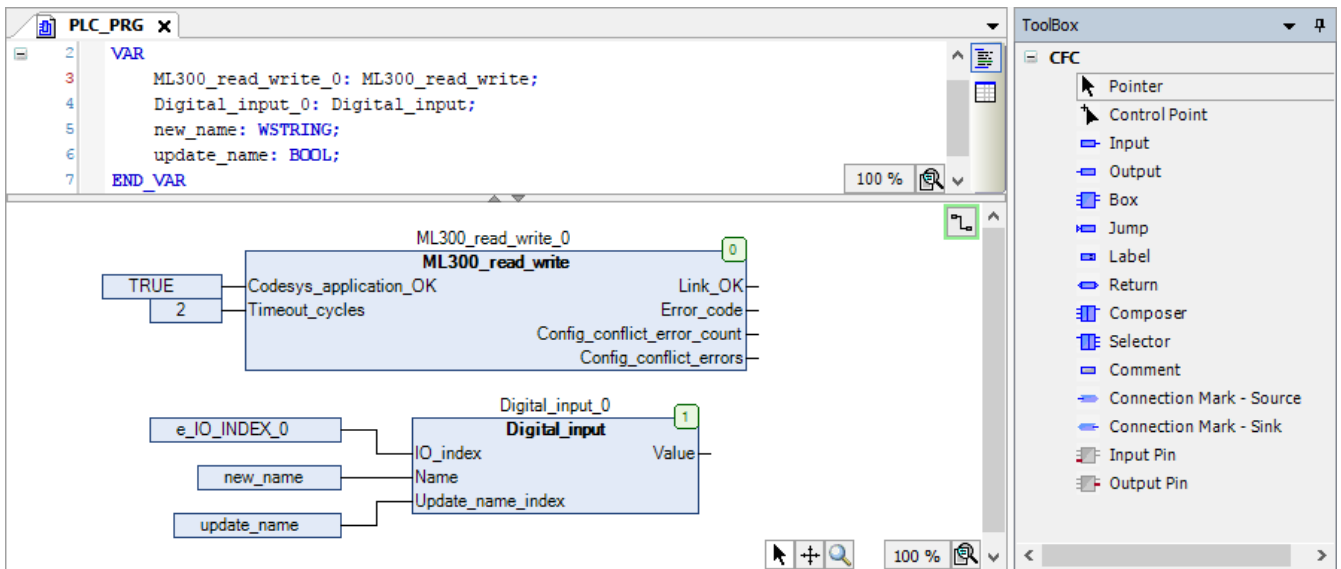


7. Go to **Categories > Variables > MULTILINE_300_IO > Enums > E_IO_INDEX** and select the index number of the input (0 to 39):



- The *IO_index* value must be unique for the I/O type in your application.
- Select **OK**.

8. Optional: Drag an **Input** from the *ToolBox* to the implementation part of the working area and connect it to the *Name* input, and one to the *Update_name_index* input:



- To assign a **fixed name** to the I/O, type the name **must be surrounded by double quotation marks**.
 - For example, to rename the I/O with **IO_INDEX 0** to *Procedure 1*, select **???** in the **Input**, type *"Procedure 1"* and press *Return* on your keyboard. The new name is only visible on the controller after it is written to the controller using the *Update_name_index* input.
- To assign a **variable** to the I/O *Name* input:
 - a. Select the input that is connected to the *Name* input.
 - b. Type the variable name (for example, *new_name*) and press *Return* on your keyboard.
 - c. Select **OK** to declare the variable as it is shown.

NOTE To rename an I/O that has a variable assigned to the *Name* input you must set the variable value to the name that you want to display, then write the new name to the controller using the *Update_name_index* input.

- If you connect an input to the *Name* input, you must also add a variable to the *Update_name_index* input. This is used to write the selected name to the controller.
9. The I/O function block has been added to the project and can be assigned in the ML 300 controller after the application is downloaded to the controller.

4.2.3 Assign a CODESYS I/O function in the controller

To see the custom CODESYS I/Os in the controller:

1. The POU containing the I/O function block must be added to **MainTask** in the project tree.
2. The program must be downloaded to the controller.
3. The program must have run at least once.



More information

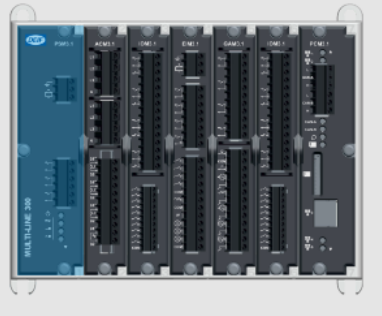
See **ML 300 CODESYS projects, Download the application to the controller** for more information about how to download and run your program on an ML 300 controller.

Follow these steps to assign the CODESYS I/O function block to a controller terminal:

1. Use PICUS to log on to the ML 300 controller to which the I/O is assigned, and go to **Configure > Input/output**:

Connect Live data Supervision Alarms Log Tools **Configure** DEIF - power by control

Controller rack

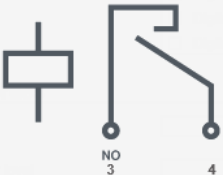


Slot 1, terminals 3, 4 | Digital output

Name: Rename


Relay setup Alarms Functions

Coil state:



NO 3 4

Function



Coil

NO Circuit

Terminals

PSM3.1, Slot 1

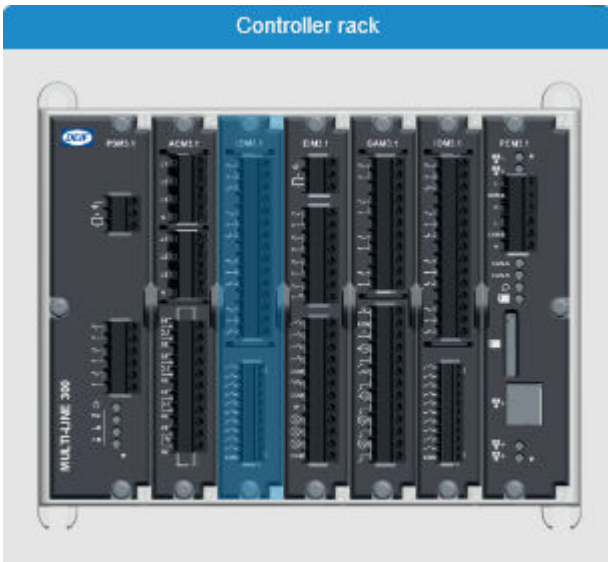
State/Value	Terminal(s)	Name	Type	Function
True	3, 4	Status OK	DO	yes(1)
True	5, 6	Any alarm	DO	yes(1)
True	7, 8	Horn 1	DO	yes(1)

Menu

- Single-line
- Input/output
- Parameters
- CustomLogic
- Actions
- Refresh
- Write

Save

2. Select a hardware module where you want to configure the I/O and select an unused terminal:



Terminals

IOM3.1, Slot 3

State/Value	Terminal(s)	Name	Type
False	1, 2, 3	GB close	^
False	4, 5, 6	GB open	
False	7, 8, 9	IOM out 3	
False	10, 11, 12	IOM out 4	
True	13, 23	GB open	
False	14, 23	GB closed	
False	15, 23	GB short circuit	
True	16, 23	Acknowledge all alarms	
False	17, 23	GB close	
False	18, 23	GB open	
False	19, 23	Activate inhibit 1	
True	20, 23	IOM in 8	v

- For example, if you want to configure a digital input, then select an IOM3.1 module.
3. Go to **Functions > Local > CODESYS** and select the I/O that you configured in the CODESYS program.

Slot 3, terminals 15, 23 | Digital input

Name: Rename

Alarms
Functions

- Engine
- Regulators
- Breakers
- Alarm system
- ▼ Local
 - Mode
 - ▼ CODESYS
 - The first input
 - CODESYS digital input
 - CODESYS digital input

- The CODESYS I/O is in the position you set under IO_index + 1. For example, if the IO_index is set to 0, then the I/O is located in position 1 of the CODESYS list for that I/O.
- If you cannot see the configured CODESYS parameters in the I/O page in PICUS:

- Refresh the page in PICUS.
 - Check that the CODESYS *Update_name_index* for the I/O has been toggled from **FALSE** to **TRUE**.
4. Select **Save**, and then select **Write** from the right side panel.
 5. The CODESYS I/O is configured to a terminal on the ML 300 controller. The controller can now send the measured value of an input terminal to the CODESYS program, or send the value from the CODESYS program to the output terminal of the controller.

4.3 Standard ML 300 functions

4.3.1 Introduction

You can assign ML 300 input and output functions to function blocks in CODESYS. This allows you to program additional logic to customise your controller application. By assigning some functions to the CODESYS program, you are also able to reduce the number of physical terminals that need to be connected on the controller.

The ML 300 function blocks are divided into four groups:

- Alarms
- Functions
- Live data
- Parameters (It is only possible to read parameter values on marine controllers, for example PPU 300.)

Alarm function blocks contain an overview of the alarm status, some alarm parameter settings, and the ability to acknowledge and unlatch alarms.

Functions function blocks contain a variety of functions. These functions include commands that can be sent to the controller, status of the controller, controller alarms and functions, and inter-controller communication.

Live data function blocks contain an overview of the immediate actual values that the controller receives from measurements.

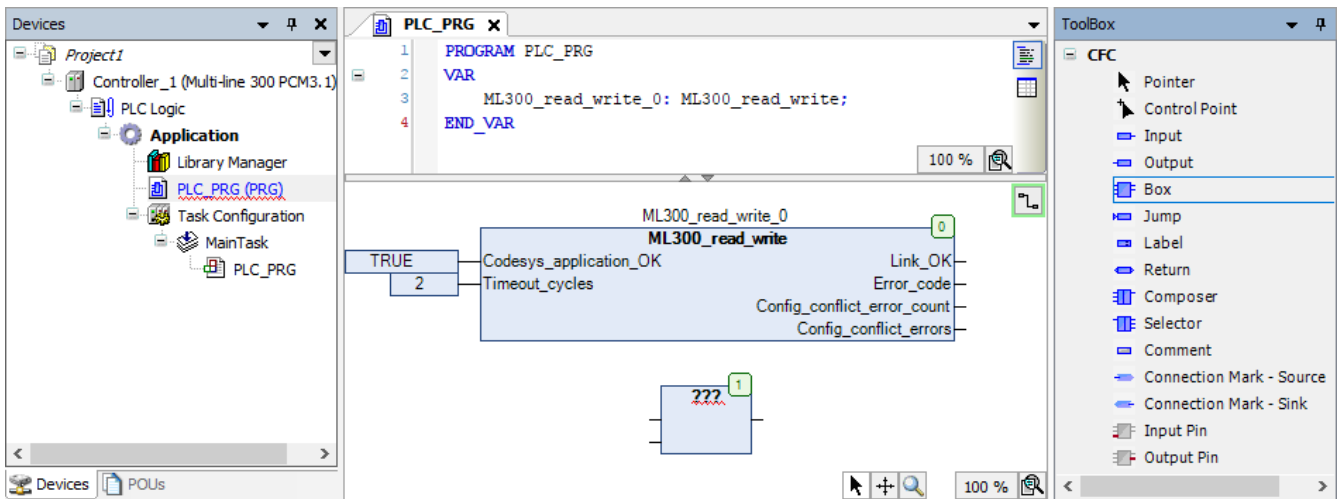
Parameter function blocks contain parameter overviews for alarm and controller functions. The parameter function blocks are divided into a read-only function block to read the state of the parameter, and a write function block to change individual settings for a specific parameter.

The function paths for the standard ML 300 functions are the same as used in other ML 300 controller interfaces, for example PICUS. The function name includes the path used to navigate to the function.

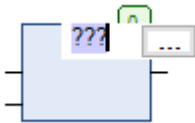
4.3.2 Add standard ML 300 functions function blocks

Follow these steps to add a standard ML 300 function function block to your program:

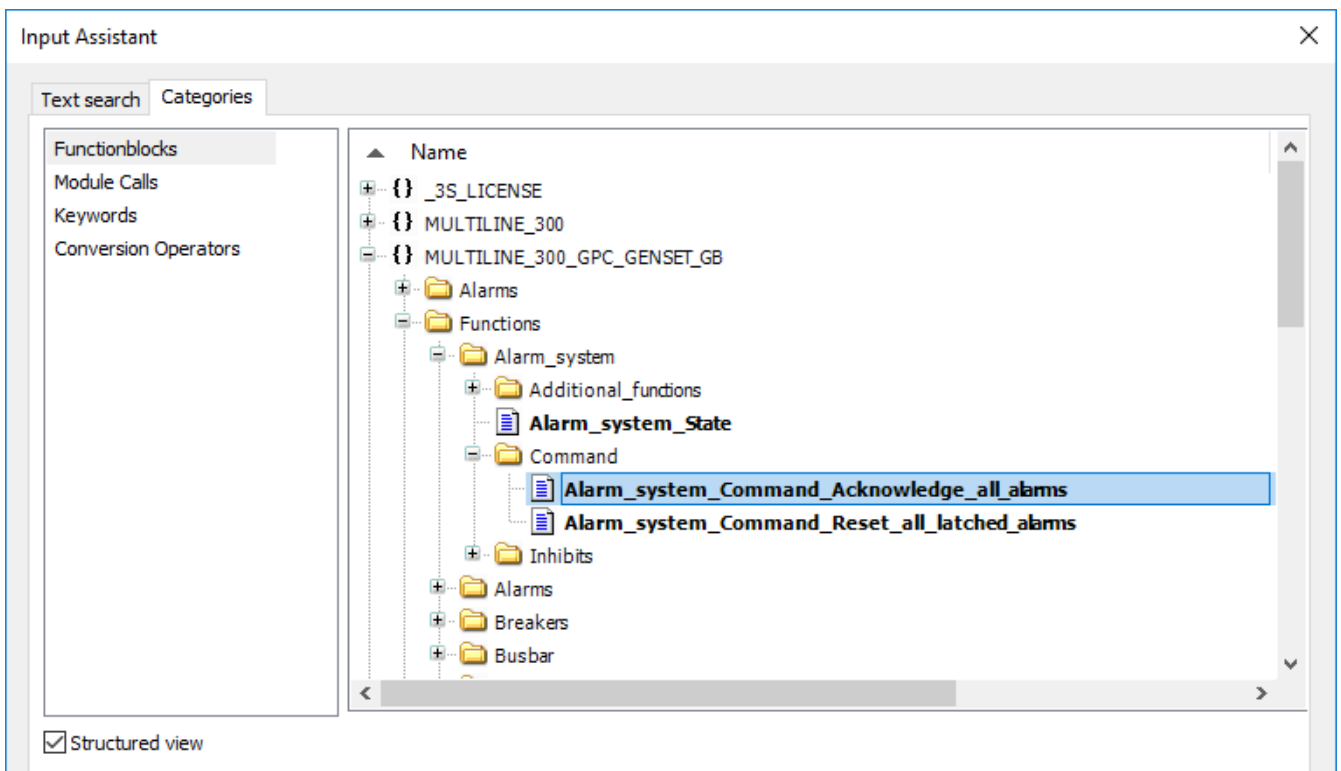
1. Drag a **Box** from the *ToolBox* to the implementation part of the working area:



2. Select **???** in the **Box** and then select **...** to open the *Input Assistant* window:



3. Go to **Categories > Functionblocks > [Controller library]** and select the function you want to add:
 - For example, **Functions > Alarm_system > Command > Alarm_system_Command_Acknowledge_all_alarms**.

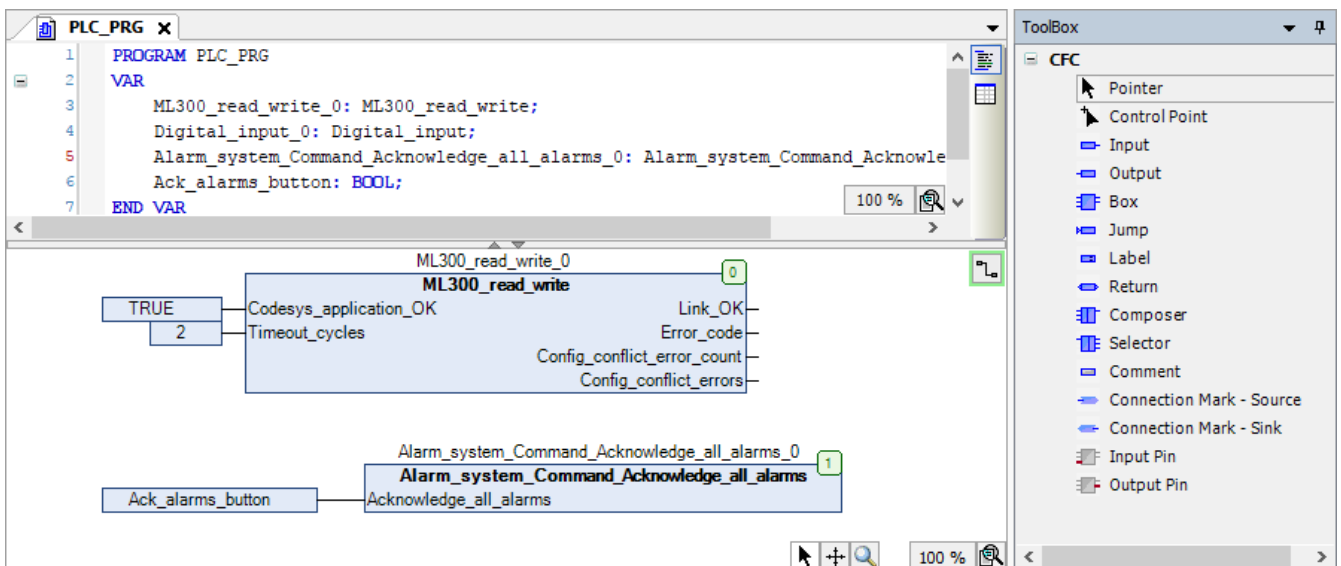


- Select **OK**.
4. Press *Return* on your keyboard two times to open the *Auto Declare* window for the function block. Select **OK** to declare the variables as they are shown:

Auto Declare ✕

Scope: VAR	Name: Alarm_system_Command_Acknow	Type: Alarm_system_Command
Object: PLC_PRG [Application]	Initialization: <input type="text"/>	Address: <input type="text"/>
Flags: <input type="checkbox"/> CONSTANT <input type="checkbox"/> RETAIN <input type="checkbox"/> PERSISTENT	Comment: <input style="height: 40px;" type="text"/>	

5. After connecting the inputs and/or outputs of the standard function in your program it is ready to be downloaded to the controller:



More information

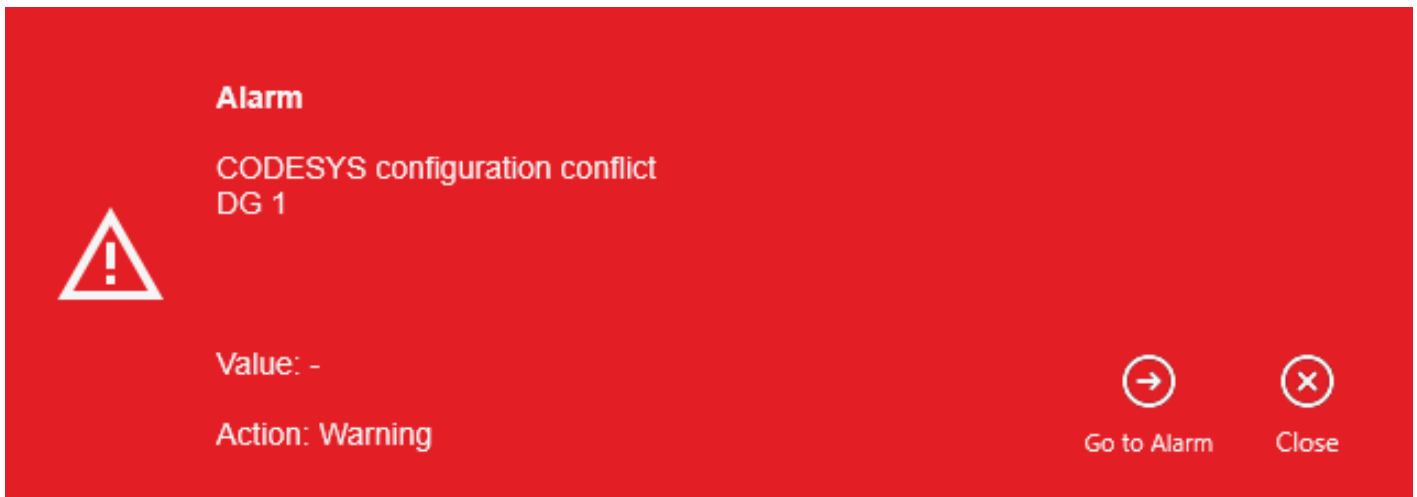
See **ML 300 CODESYS projects, Download the application to the controller** for more information about how to download and run your program on an ML 300 controller.

4.3.3 Function conflicts

When an ML 300 function that is already assigned to the controller is configured in the CODESYS program that is running on the controller, a conflict occurs. This conflict sets the Link_OK output on the ML 300 function block in the program to **FALSE**, which prevents the application from running on the controller.

If there is a configuration conflict, or if CODESYS tries to write a value to the controller that is out of range, then the controller also activates a warning alarm.

Figure 4.4 Example of the CODESYS configuration conflict alarm in PICUS



To resolve the conflict you can:

- Remove the conflicting ML 300 function from the CODESYS program, and update the application on the controller.
- Remove the conflicting I/O function from the controller, and perform a warm reset of the CODESYS application.

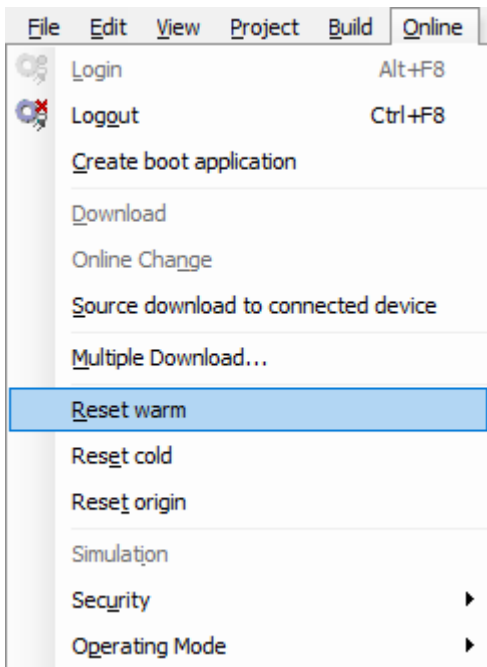


More information

See **ML 300 CODESYS projects, Add the ML 300 function block, ML 300 function blocks' inputs and outputs** for more information about how to use CODESYS to see which function is causing the conflict.

Warm reset

To perform a warm reset, select **Online > Reset warm**.



After performing a warm reset of the CODESYS program, you must start the CODESYS application again for the application to run on the controller.

4.3.4 Alarm function block overview

Alarm function blocks are added to the working area in the same way as other standard ML 300 functions function blocks. The parameters of an alarm cannot be changed with an alarm function block. The parameters of an alarm can only be updated using a parameter function block for the corresponding alarm. Alarm function blocks can be used to:

- Acknowledge alarms
- Unlatch acknowledged latched alarms
- Read the status of the alarm
- Read some alarm parameters

Figure 4.5 Alarm function block example

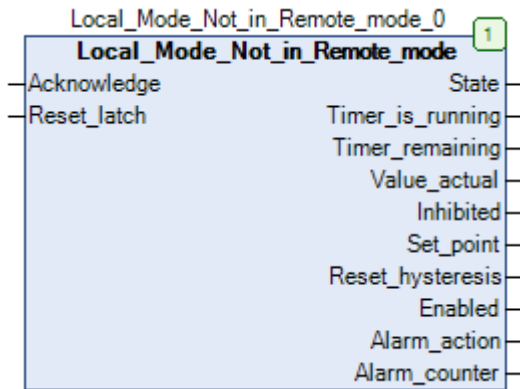


Table 4.4 Alarm function block input and output overview

Name	Input / Output	Type	Notes
Acknowledge	Input	BOOL	When this input receives a true signal, the unacknowledged alarm is acknowledged. If the alarm is not present in the controller, or if the alarm has already been acknowledged, then nothing happens.
Reset_latch	Input	BOOL	When this input receives a true signal, the acknowledged and latched alarm is unlatched. If the alarm is not present in the controller, or if the alarm has not yet been acknowledged, then nothing happens.
State	Output	E_ALARM_STATE	This output displays the current state of the alarm.
Timer_is_running	Output	BOOL	When this output is true, the alarm timer is running and the alarm has not been triggered on the controller.
Timer_remaining	Output	TIME	This output displays the time remaining before the alarm is activated on the controller.
Value_actual	Output	REAL	This output displays the measured value of the alarm. When the measured value is above the set point the alarm delay timer is activated. When the timer expires the alarm activates.
Inhibited	Output	BOOL	When this output is true, the alarm is inhibited and does not activate when the activation parameters are fulfilled.
Set_point	Output	REAL	This output displays the set point at which the alarm activates. If the alarm does not have a set point, the value is zero by default.
Reset_hysteresis	Output	REAL	This output displays the reset hysteresis value of the alarm.
Enabled	Output	BOOL	When this output is true, the alarm is enabled in the alarm parameters. When this output is false, the alarm is not enabled in the alarm parameters.

Name	Input / Output	Type	Notes
Alarm_action	Output	E_ALARM_ACTION	This output displays the alarm action that the controller performs when the alarm activates.
Alarm_counter	Output	UDINT	This output counts the number of times the alarm appeared in the alarm list of the controller. The alarm counter only increases if the alarm was not present in the alarm list when the alarm activates.

4.3.5 Parameter function block overview

NOTE To comply with the regulations from Marine Class societies, it is not possible to write parameters to the controller if the controller is designed for marine applications (for example, PPU 300).

Parameter function blocks are added to the working area in the same way as other standard ML 300 functions function blocks. Parameter function blocks are divided into two groups:

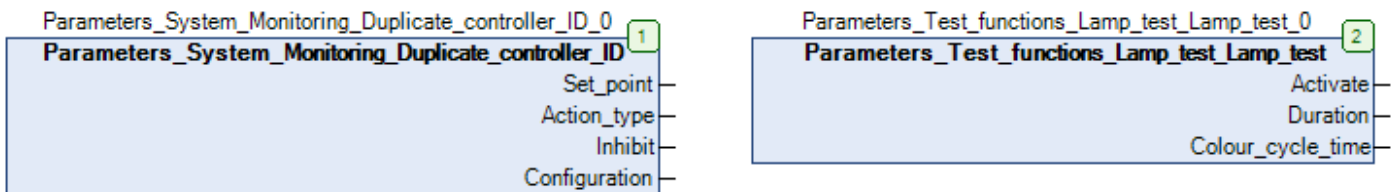
- Read configuration function blocks.
- Configure function blocks (not available on marine products).

Read configuration function blocks

Each controller function with configurable parameters (including alarms), has a function block to read the function's parameter configuration. The read configuration function block is located under **Categories > Functionblocks > [Controller library] > Parameters > [Function path]**.

[Controller library] is the ML 300 library for the controller that you are running CODESYS on, for example *MULTILINE_300_PPU_DG*. **[Function path]** is the location of the function's or alarm's parameters as described in the designer's handbook for the product.

Figure 4.6 Alarm and controller function read configuration function block examples



Configure function blocks

To comply with the regulations from Marine Class societies, parameter configure function blocks are not available if the controller is designed for marine applications (for example, PPU 300).

Each controller function with configurable parameters (including alarms), has a function block for each parameter to configure each configurable function of the parameter. For example, *Lamp test* has three configurable parameters:

- Activate
- Duration
- Colour cycle time

Each of these configurable parameters has a function block associated to it to change the parameter value. The configure function block always contains two inputs: *Write* and the configurable parameter.

The *Write* input is used to write the parameter value of the configurable parameter to the controller. This input is a pulse, and only writes the new value to the controller when the input changes to TRUE. If *Write* remains TRUE and the parameter value updates, the updated value is not written to the controller. To write the new value to the controller, *Write* must first be

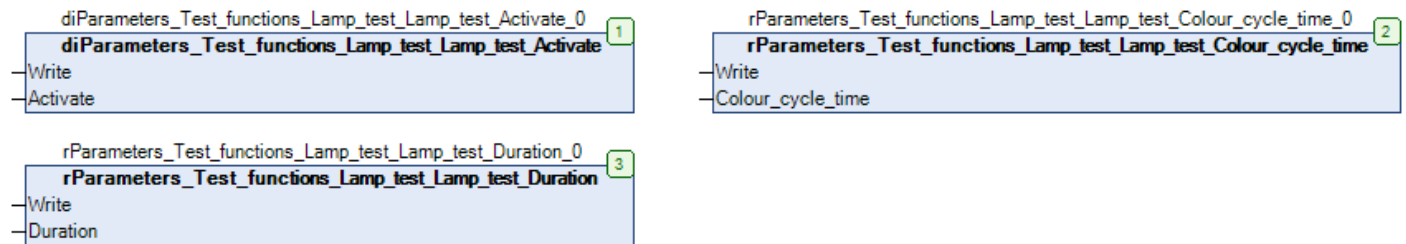
changed to FALSE, and then back to TRUE to write the value to the controller. If a value is written to the controller that is not acceptable, for example an out of range value or a non-configurable parameter, the controller activates the **CODESYS configuration conflict** alarm.

The configurable parameter depends on the selected parameter function block for the controller function. This input stores the new parameter value until *Write* changes from FALSE to TRUE.

The configure function blocks are located under **Categories > Functionblocks > [Controller library] > Parameters > [Function path] > Configure**.

[Controller library] is the ML 300 library for the controller that you are running CODESYS on. **[Function path]** is the location of the function's or alarm's parameters as described in the designer's handbook for the product.

Figure 4.7 Configure function blocks for Lamp test

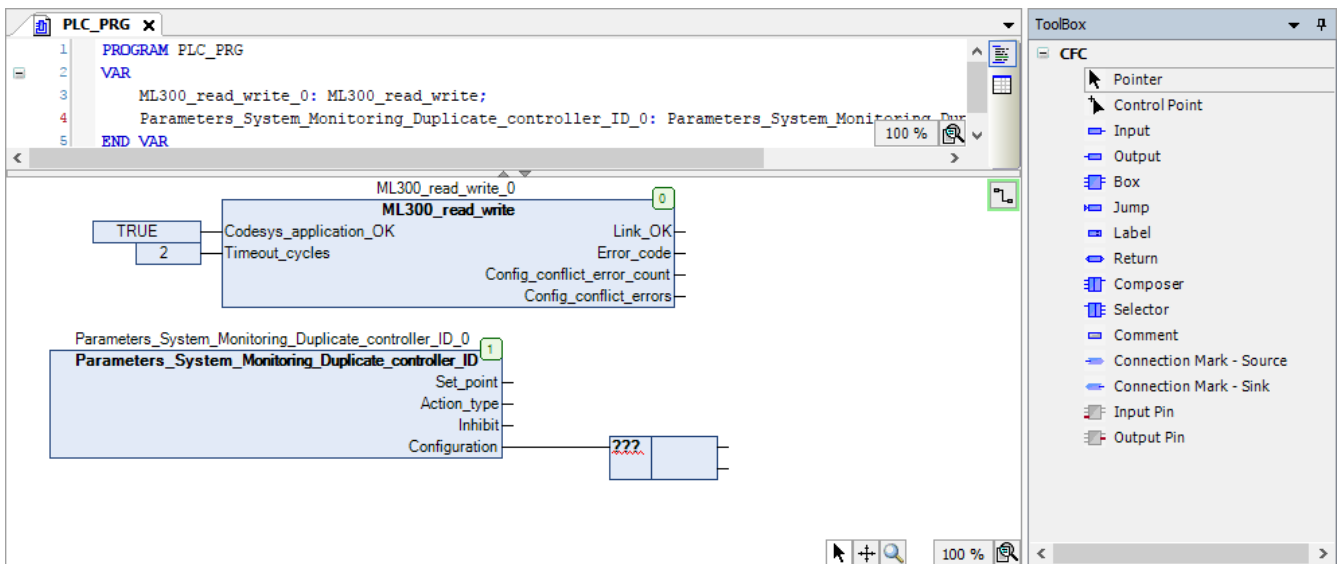


Array inputs and outputs

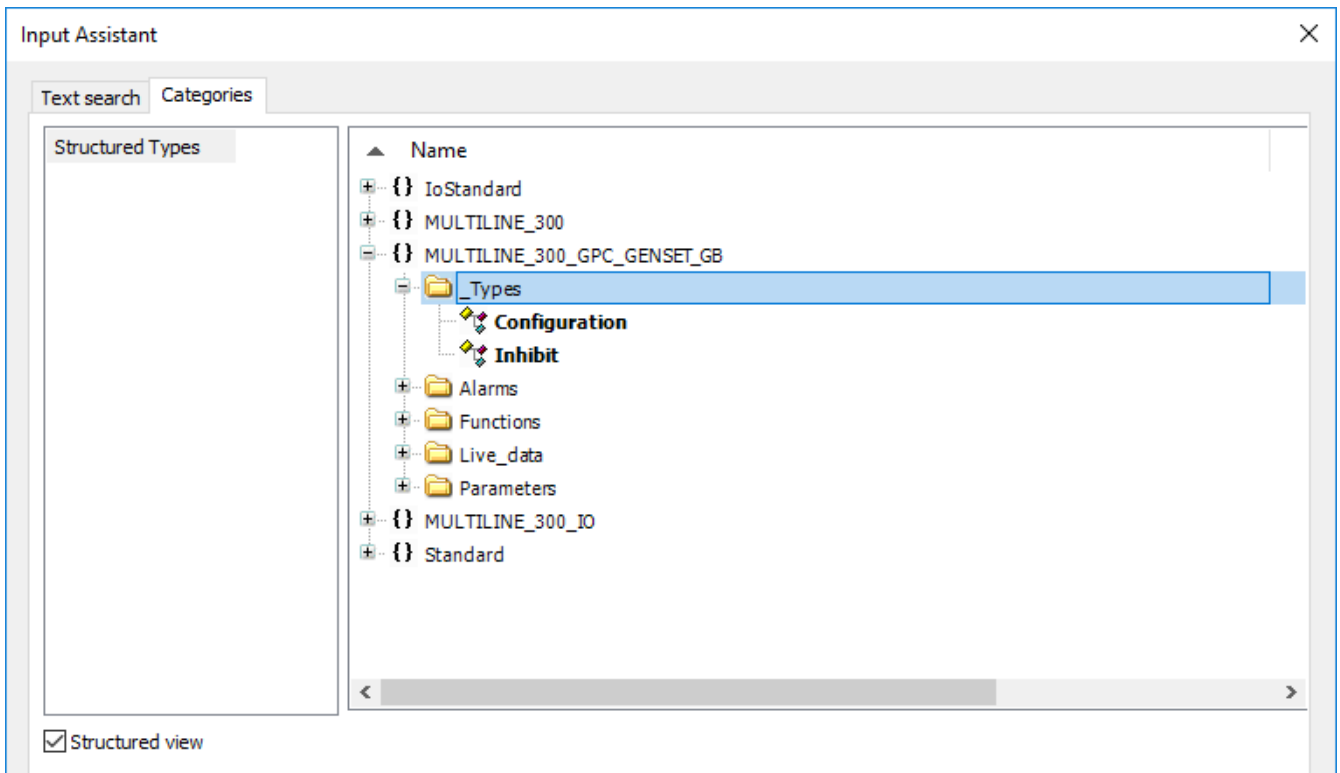
Alarm parameter function blocks can contain array inputs or outputs. Array inputs and outputs are used for the *Inhibit* and *Configuration* parameters.

To read the individual bit values of the array in a read configuration function block, a *Selector* element is required. Follow these steps to add and configure the *Selector* element:

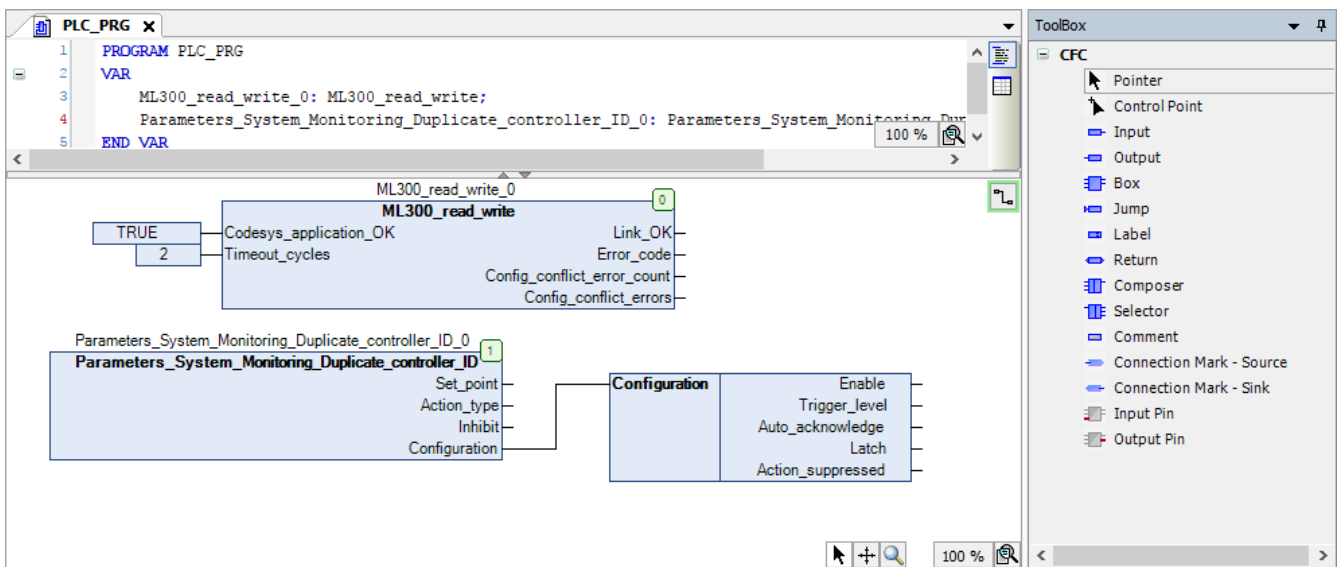
1. Drag the *Selector* element from the **Toolbox** to the working area.
2. Connect the input of the *Selector* element to the *Inhibit* or *Configuration* output of the read configuration function block.



3. Select **???** in the **Selector** and then select **...** to open the Input Assistant window for the *Selector*.
4. Go to **Categories > Structured types > [Controller library] > Parameters > _Types** and select the same parameter as the output that you connected the *Selector* element to.

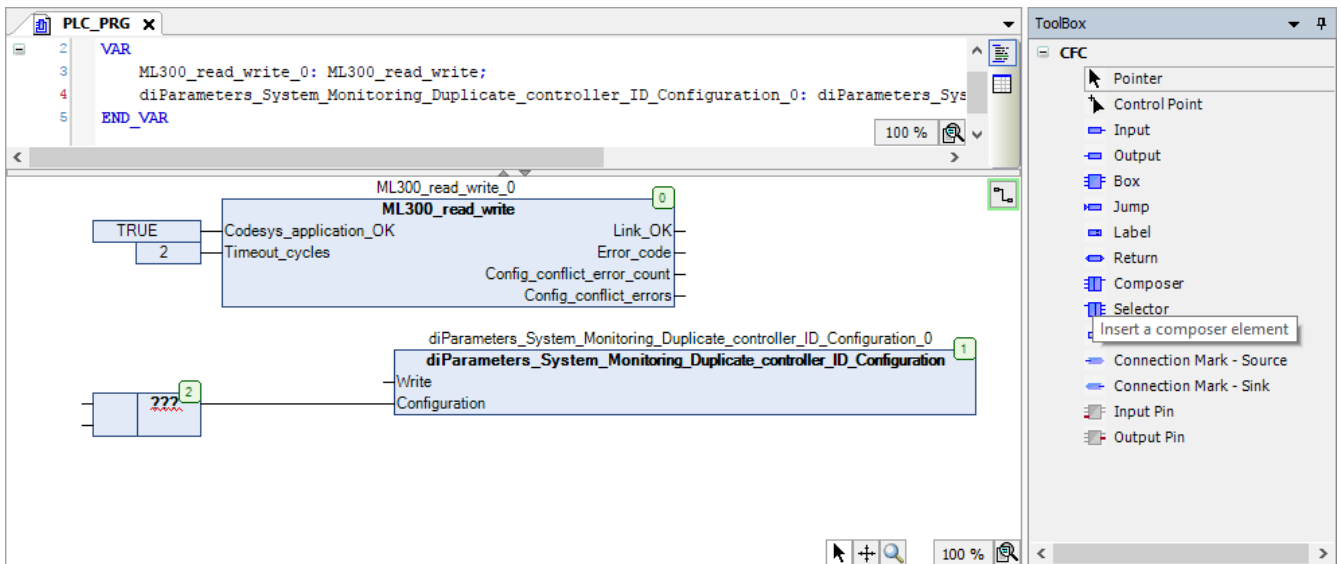



- Select **OK** and then press *Return*.
5. You are now ready to connect outputs to the *Selector* element.
- The output variables of the *Selector* element must be declared as **Boolean**.

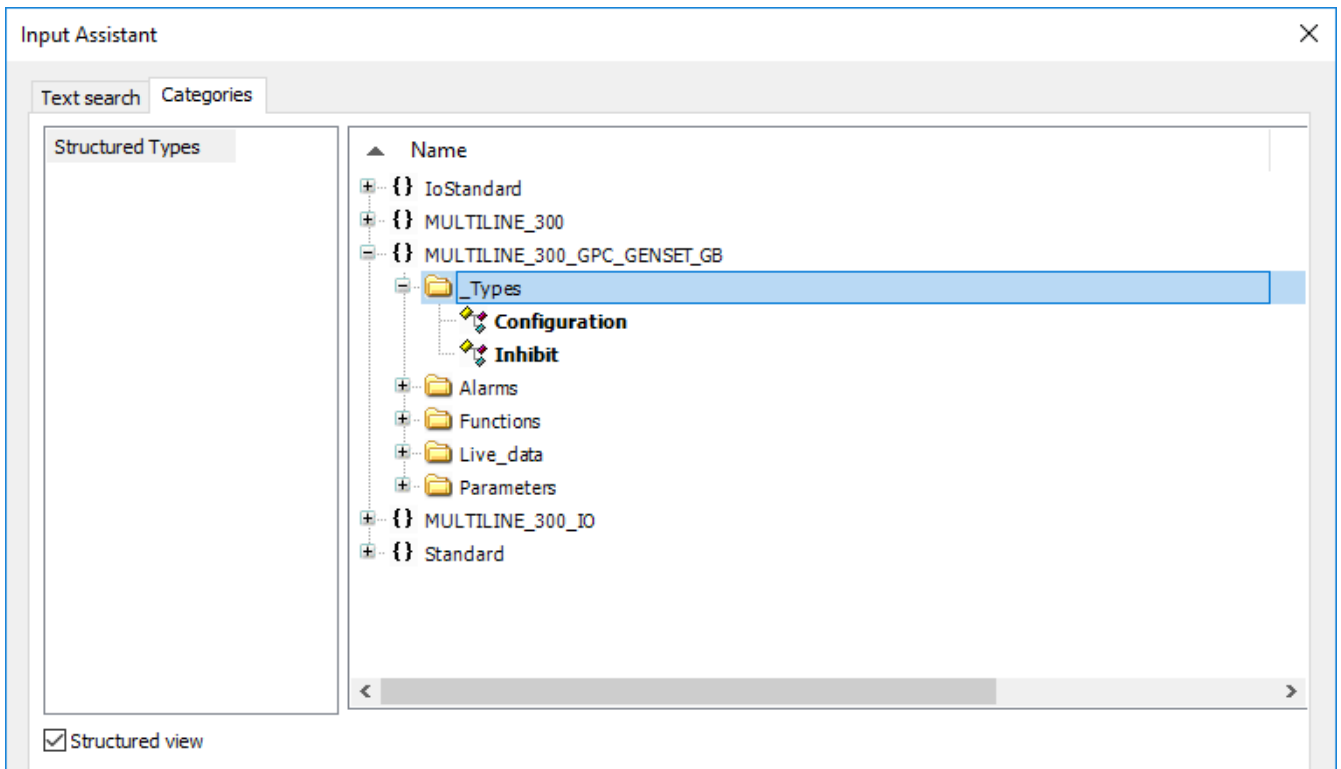


To write values to the individual bits of the array in a parameter configure function block, a *Composer* element is required. Follow these steps to add and configure the *Composer* element:

1. Drag the *Composer* element from the **Toolbox** to the working area.
2. Connect the output of the *Composer* element to the *Inhibit* or *Configuration* input of the parameter configure function block.



3. Select **???** in the **Composer** and then select  to open the Input Assistant window for the *Composer*.
4. Go to **Categories > Structured types > [Controller library] > Parameters > _Types** and select the same parameter as the input that you connected the *Composer* element to.



5. You are now ready to connect inputs to the *Composer* element.
 - The input variables of the *Composer* element must be declared as **Boolean**.

PLC_PRG x

```

2 VAR
3 ML300_read_write_0: ML300_read_write;
4 diParameters_System_Monitoring_Duplicate_controller_ID_Configuration_0: diParameters_Sys
5 END_VAR

```

100 %

The diagram shows three function blocks:

- ML300_read_write_0** (ID 0): Inputs include TRUE, Codesys_application_OK, and Timeout_cycles (value 2). Outputs include Link_OK, Error_code, Config_conflict_error_count, and Config_conflict_errors.
- Configuration** (ID 2): Inputs include Enable, Trigger_level, Auto_acknowledge, Latch, and Action_suppressed.
- diParameters_System_Monitoring_Duplicate_controller_ID_Configuration_0** (ID 1): Input is Write Configuration.

Connections: A line connects the 'Write Configuration' input of block 1 to the 'Configuration' input of block 2. Another line connects the 'Configuration' input of block 2 to the 'ML300_read_write_0' block.

ToolBox

CFC

- Pointer
- Control Point
- Input
- Output
- Box
- Jump
- Label
- Return
- Composer
- Selector
- Comment
- Connection Mark - Source
- Connection Mark - Sink
- Input Pin
- Output Pin

100 %

5. Extended ML 300 controller functionality

5.1 Create a multiple ring network

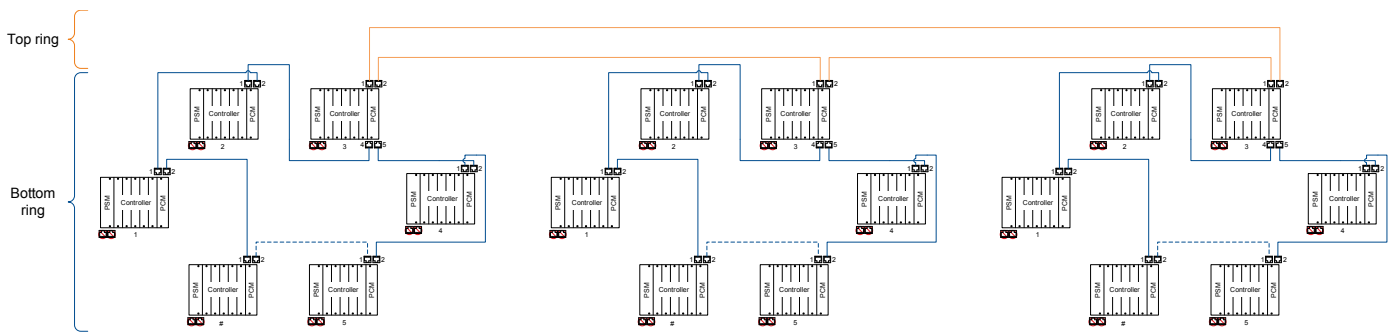
5.1.1 Introduction

The controllers communicate with each other to manage the system over the DEIF network (Ethernet network).

For communication redundancy, controllers can be connected in multiple rings if CODESYS is installed on the controllers. If there is a network disruption or failure, the DEIF proprietary ring protocol changes the communication path within 100 milliseconds.

If you want to run a system with a multiple ring network, then you must configure each controller in the top ring as a **Top unit**.

Figure 5.1 Example of a multiple ring network



5.1.2 Requirements

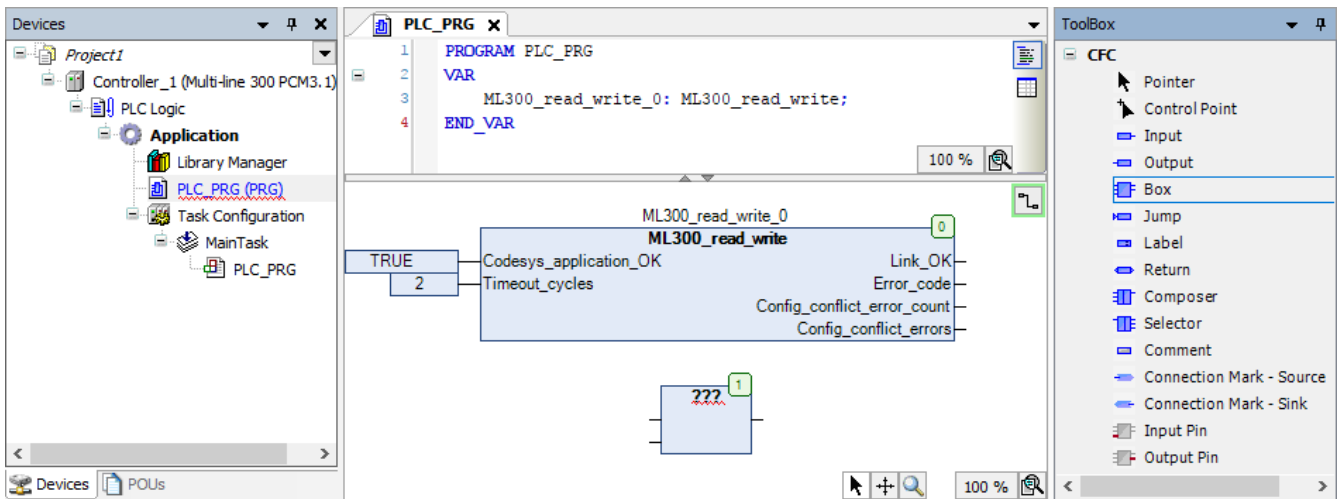
To use multiple rings, the following conditions must be met:

- No display units in the network.
- Each controller that forms the top ring must have CODESYS installed.
- Each controller that forms the top ring must be configured as a **Top unit**.

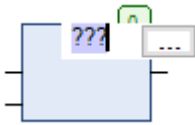
5.1.3 Configure a Top Unit controller

Follow these steps to configure a controller as a **Top unit**:

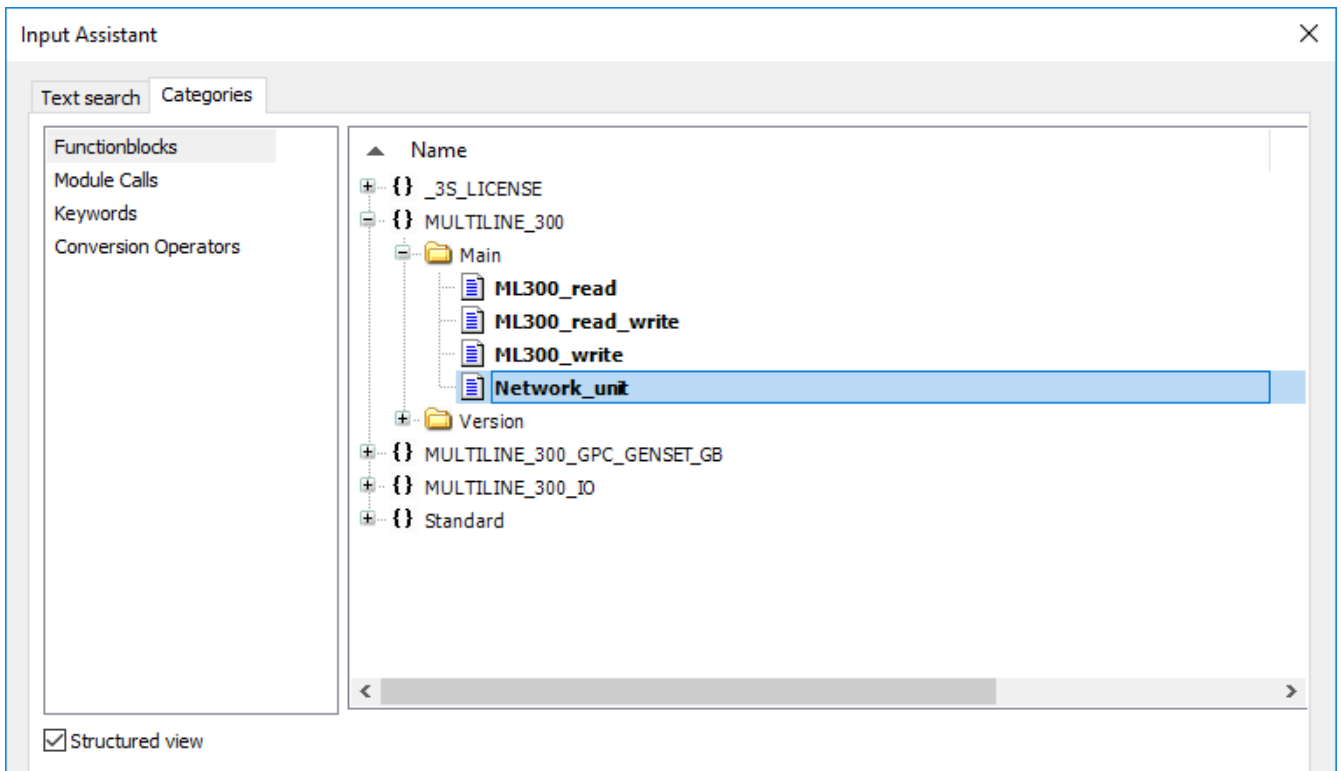
1. Drag a **Box** from the *ToolBox* to the implementation part of the working area:



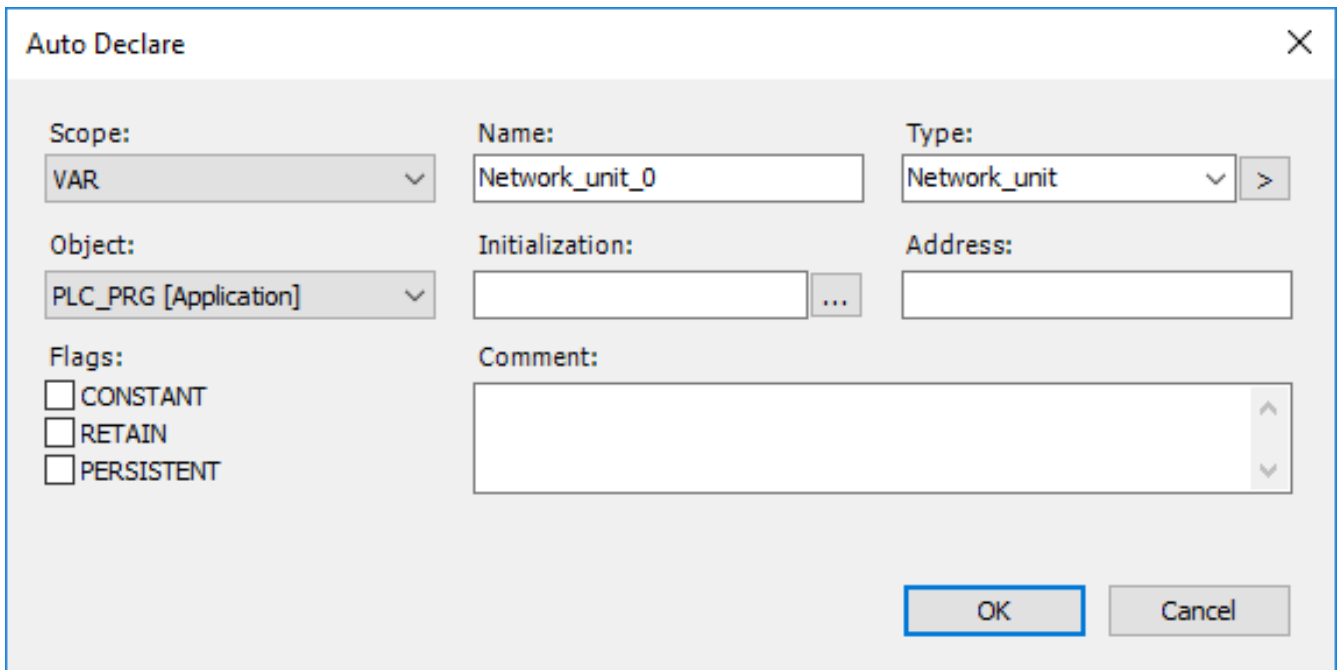
2. Select **???** in the **Box** and then select **...** to open the *Input Assistant* window:



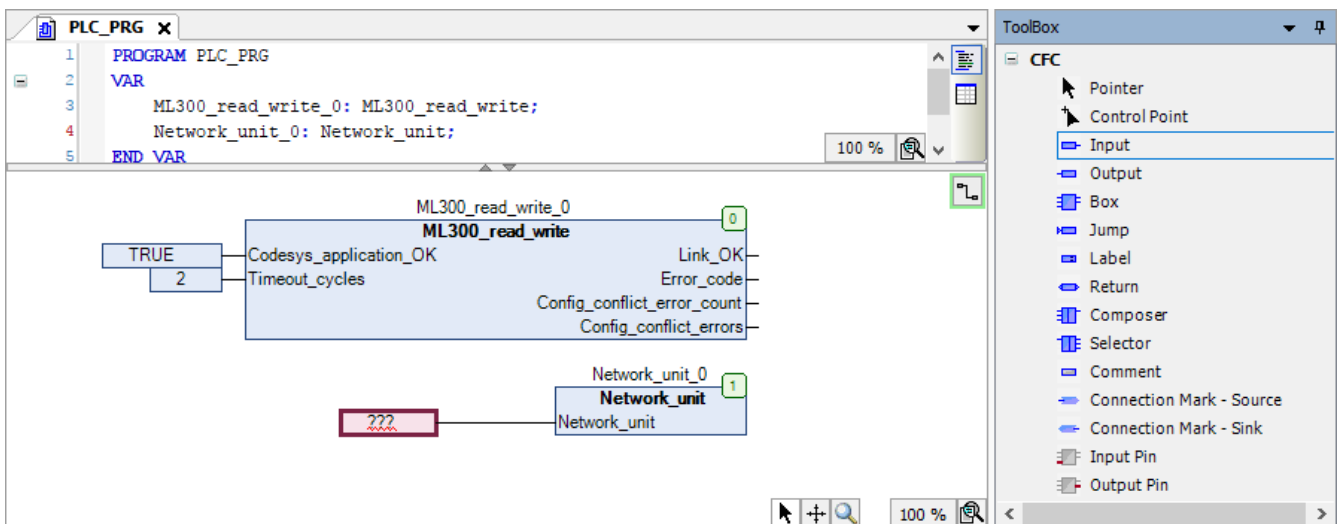
3. Go to **Categories > Functionblocks > [Controller library] > Main** and select the **Network_unit** function block:




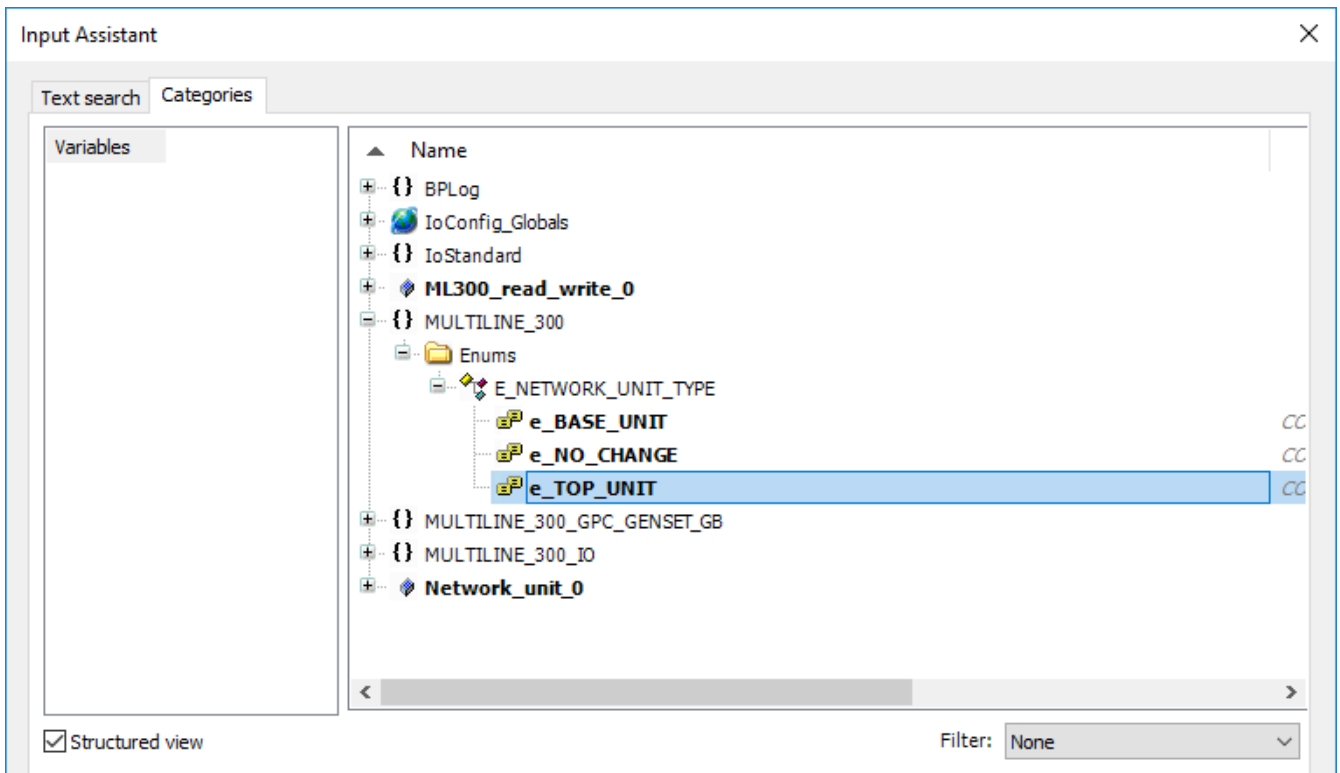
- **[Controller library]** is the library for the controller you are programming for. For example, MULTILINE_300_PPU_DG.
 - Select **OK**.
4. Press *Return* on your keyboard two times to open the *Auto Declare* window for the function block. Select **OK** to declare the variables as they are shown:



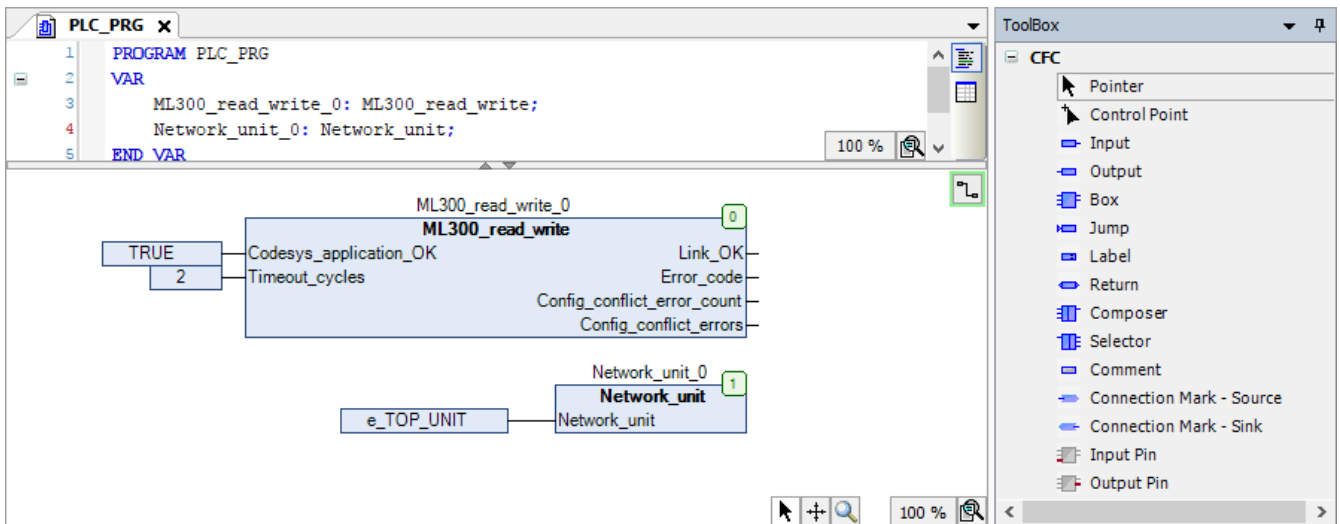
5. Drag an **Input** from the *ToolBox* to the implementation part of the working area and connect it to the *Network_unit* input:



6. Select **???** in the **Input** and then select  to open the *Input Assistant* window.
7. Go to **Categories > Variables > MULTILINE_300 > Enums > E_NETWORK_UNIT_TYPE** and select **e_TOP_UNIT**:



- Select **OK**.
8. Download the program to the controller and run the program.
 9. Power cycle the controller.
 10. The controller is configured as a top ring controller and will function after the program has successfully run on the controller and the controller has been power cycled.



5.2 Inter-controller communication

5.2.1 Introduction

Controllers using CODESYS and CustomLogic can use inter-controller communication (ICC) to send to other controllers and receive logic signals from other controllers. The logic signals can be used in the controller's CODESYS application or CustomLogic configuration. Communication is possible between:

- ML 300 controllers with CODESYS
- ML 300 controllers without CODESYS (using CustomLogic)
- ML 300 controllers without CODESYS and ML 300 controllers with CODESYS.

Inter-controller communication signals can only be sent between controllers in the same bottom ring of a multiple ring network.

This section will show you how to add the ICC function blocks in CODESYS.



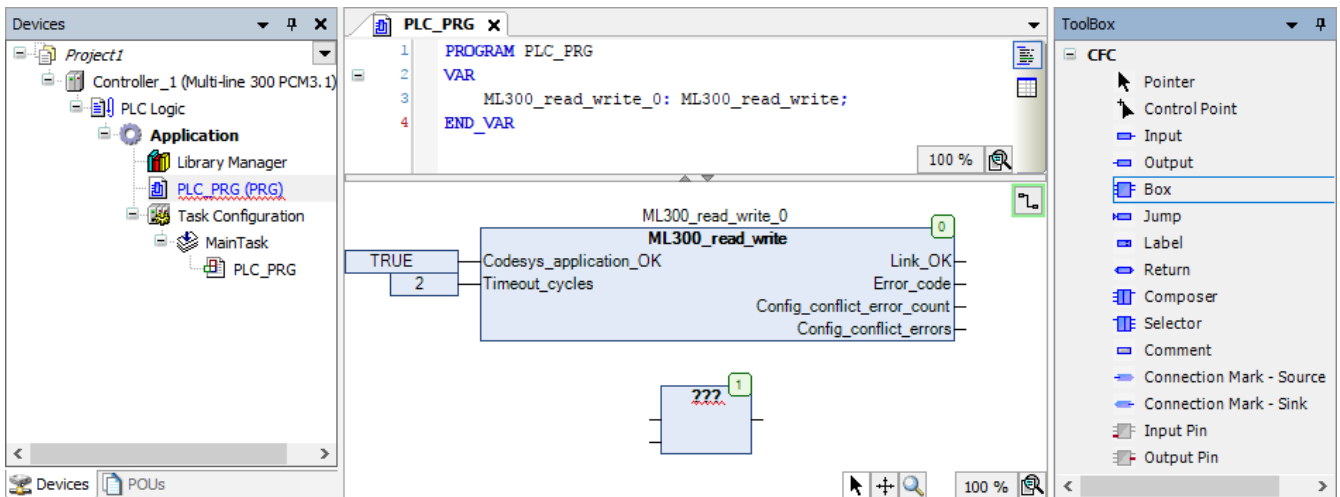
More information

See **CustomLogic, ICC (Inter-controller communication)** in the **PICUS manual** for more information about how to configure ICC in PICUS CustomLogic.

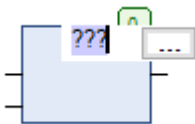
5.2.2 Add an ICC output function block

Follow these steps to add the **ICC_Output** function block to a Program POU using the continuous function chart programming language:

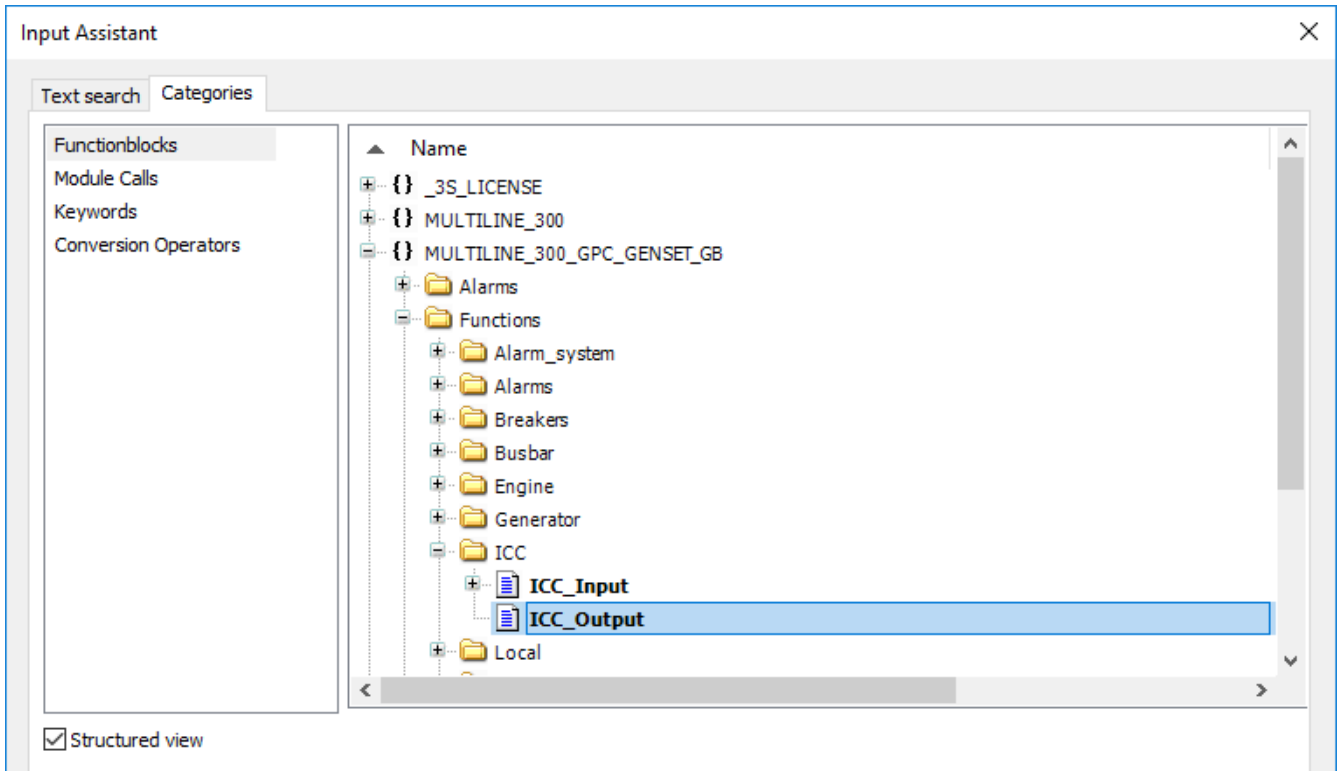
1. Drag a **Box** from the *ToolBox* to the implementation part of the working area:



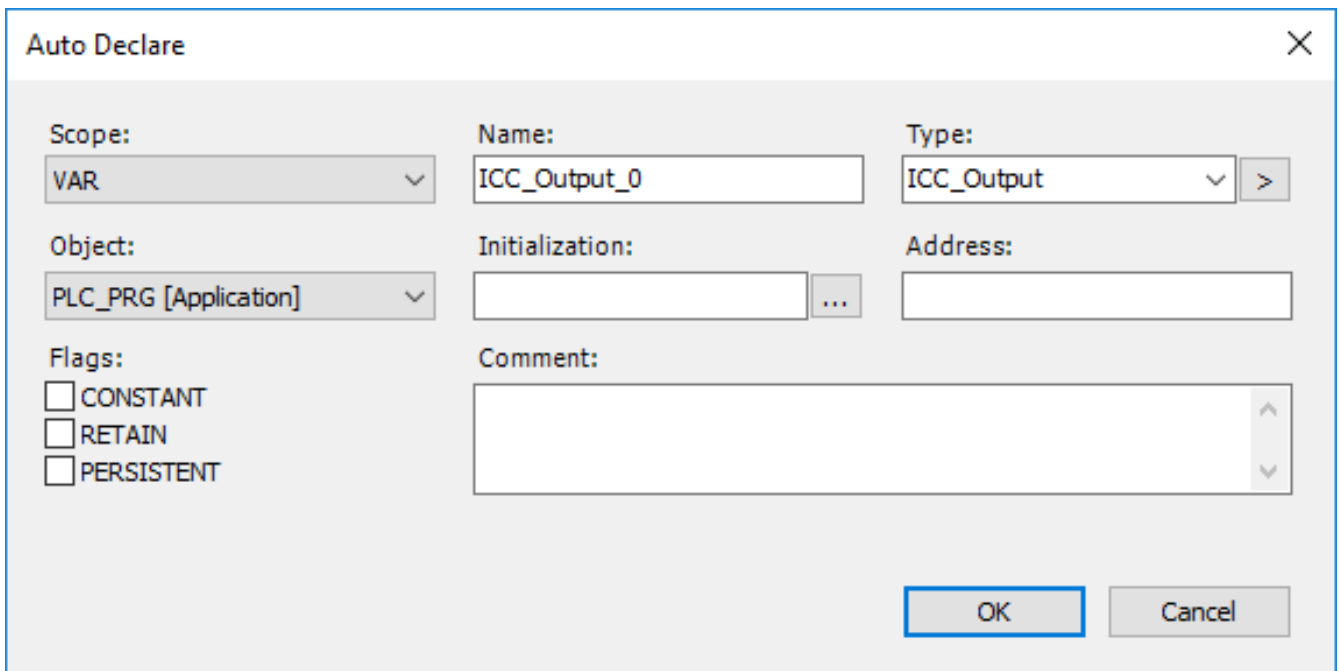
2. Select **???** in the **Box** and then select  to open the *Input Assistant* window:



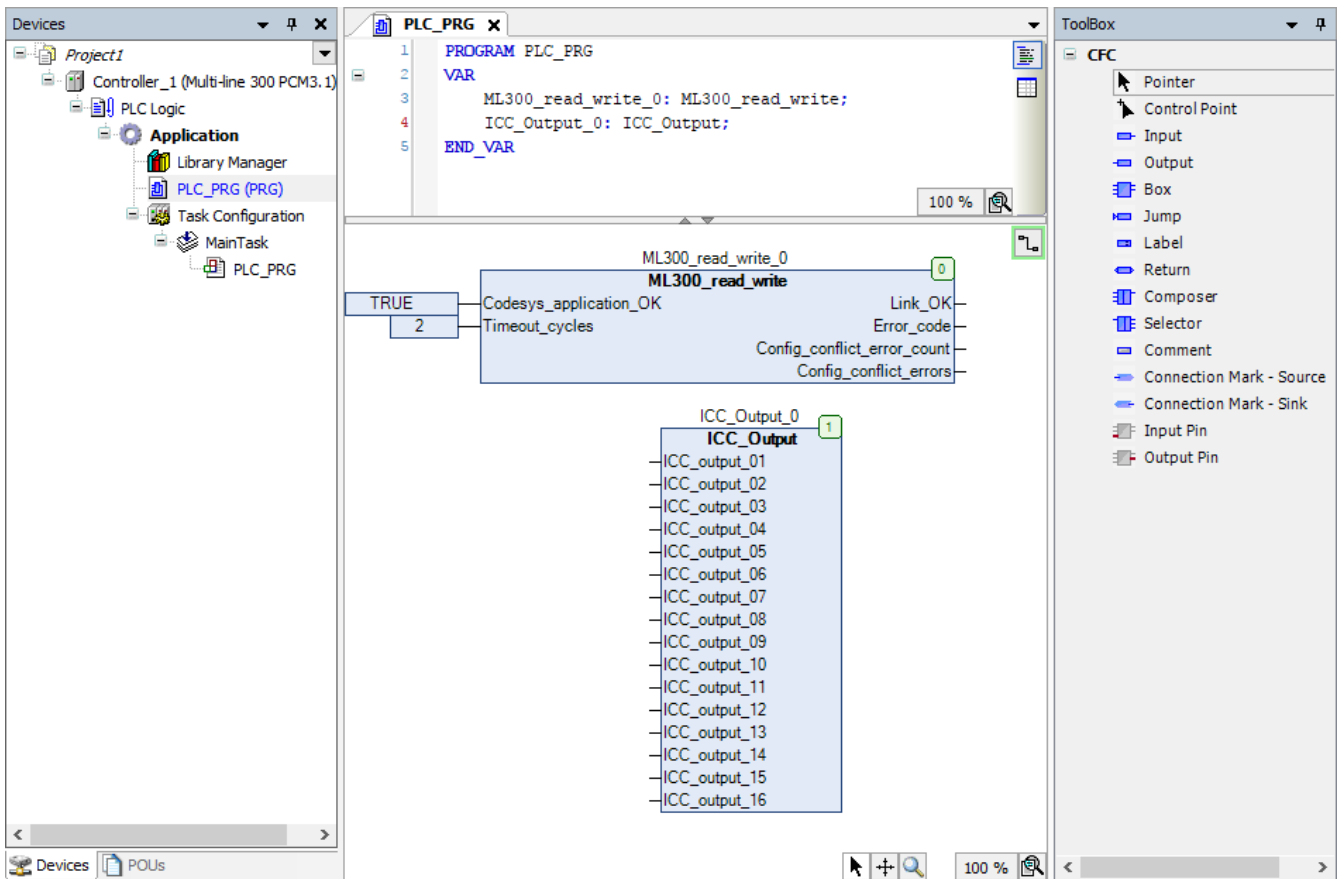
3. Go to **Categories > Functionblocks > [Controller library] > Functions > ICC** and select the **ICC_Output** function block:



- Where **[Controller library]** is the library for your specific controller type, for example *MULTILINE_300_GPC_GENSET_GB* for a GPC 300 genset controller.
 - Select **OK**.
4. Press *Return* on your keyboard two times to open the *Auto Declare* window for the function block. Select **OK** to declare the variables as they are shown:



5. The **ICC_Output** function block has been added to the project:

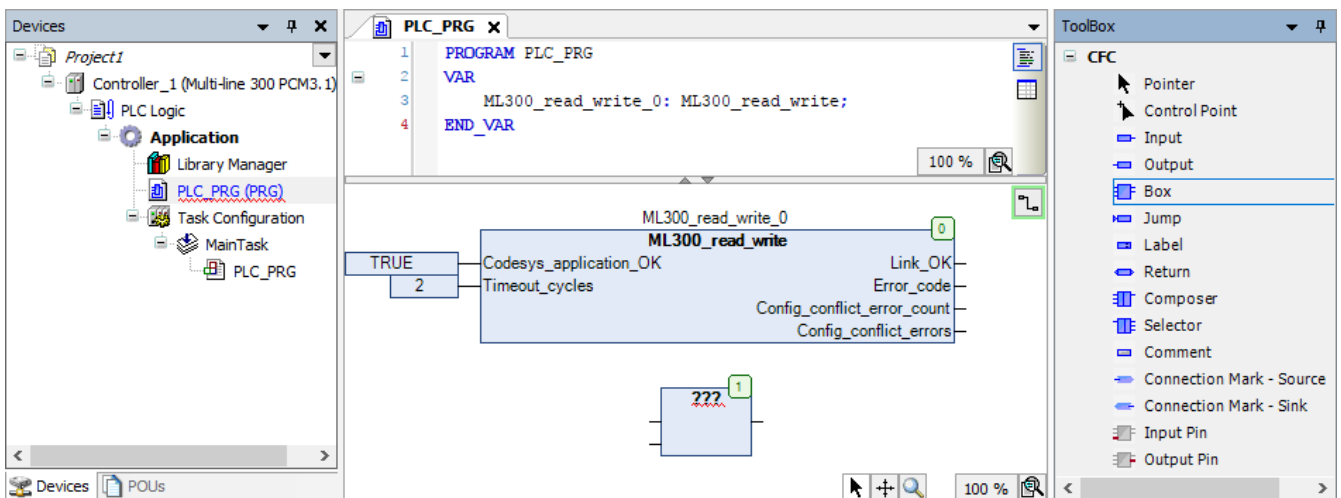


- You can now assign Boolean outputs to the **ICC_Output** function block inputs. The state of the outputs can be read by other controllers in the network.

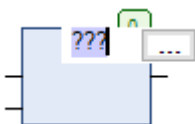
5.2.3 Add an ICC input function block

Follow these steps to add the **ICC_Input** function block to a Program POU using the continuous function chart programming language:

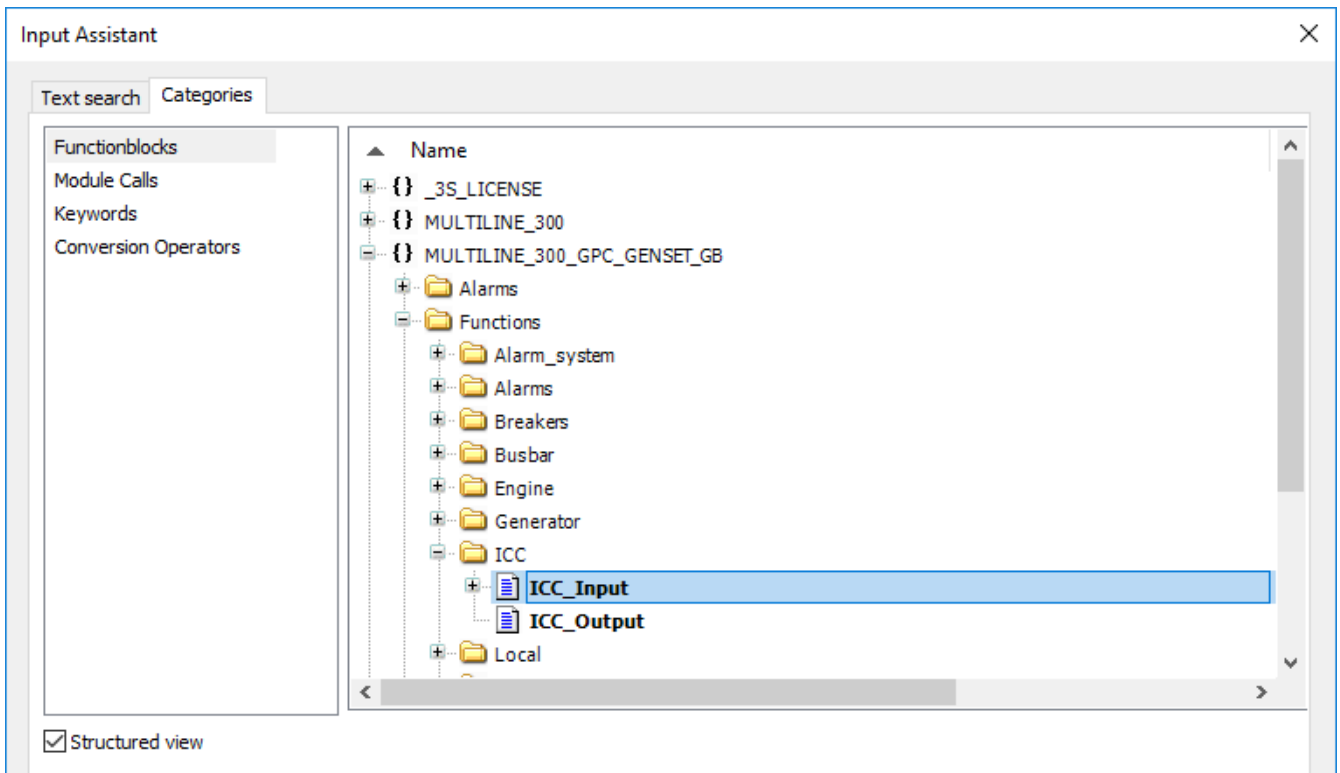
1. Drag a **Box** from the *ToolBox* to the implementation part of the working area:



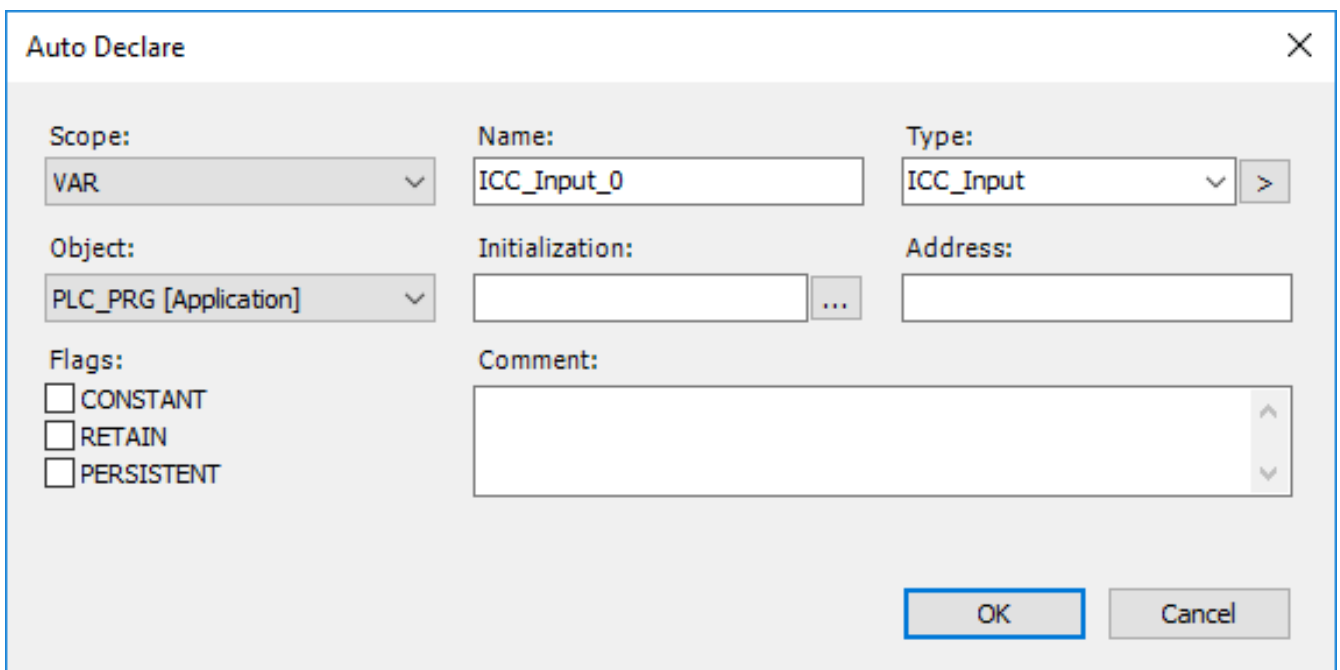
2. Select **???** in the **Box** and then select  to open the *Input Assistant* window:



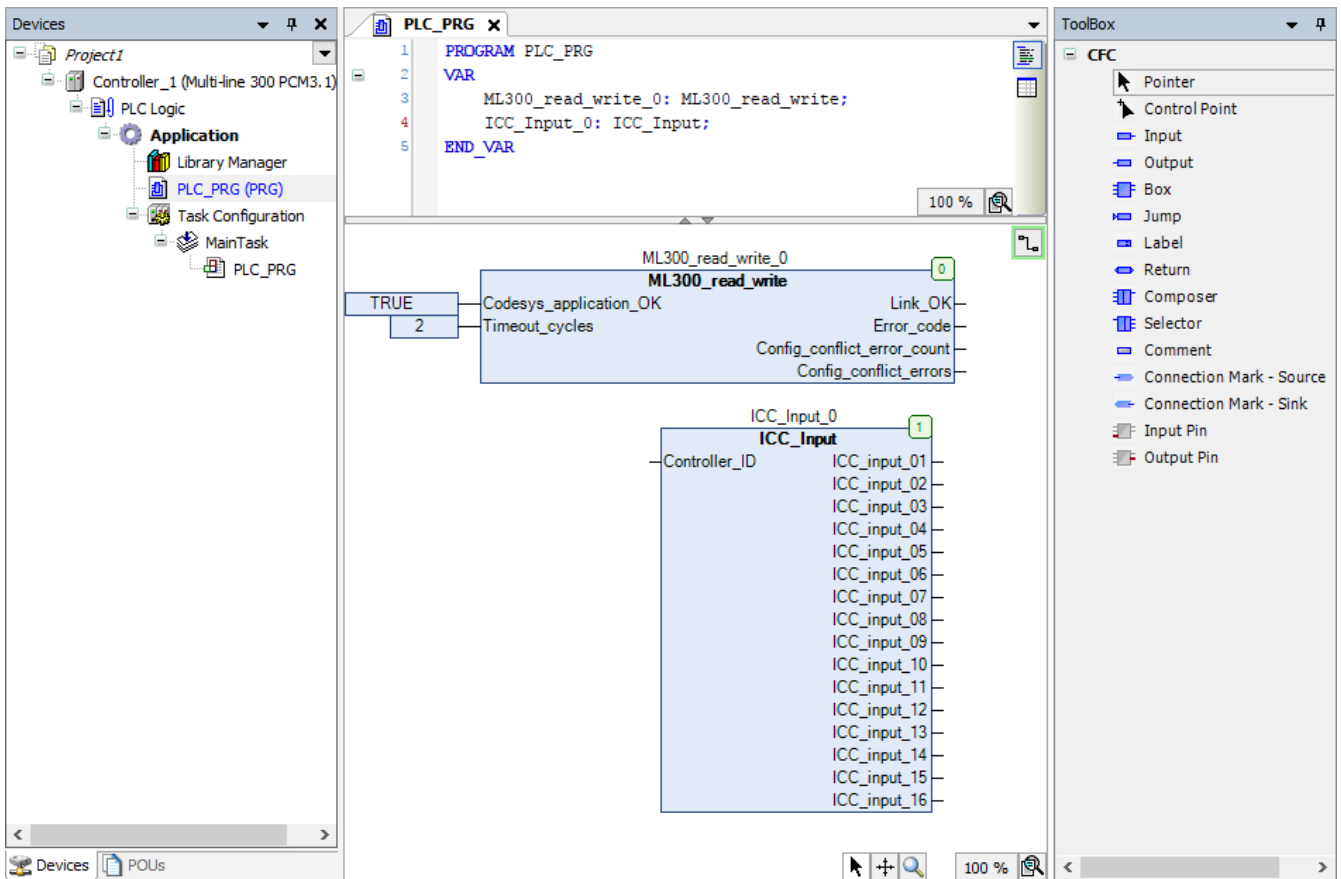
3. Go to **Categories > Functionblocks > [Controller library] > Functions > ICC** and select the **ICC_Input** function block:



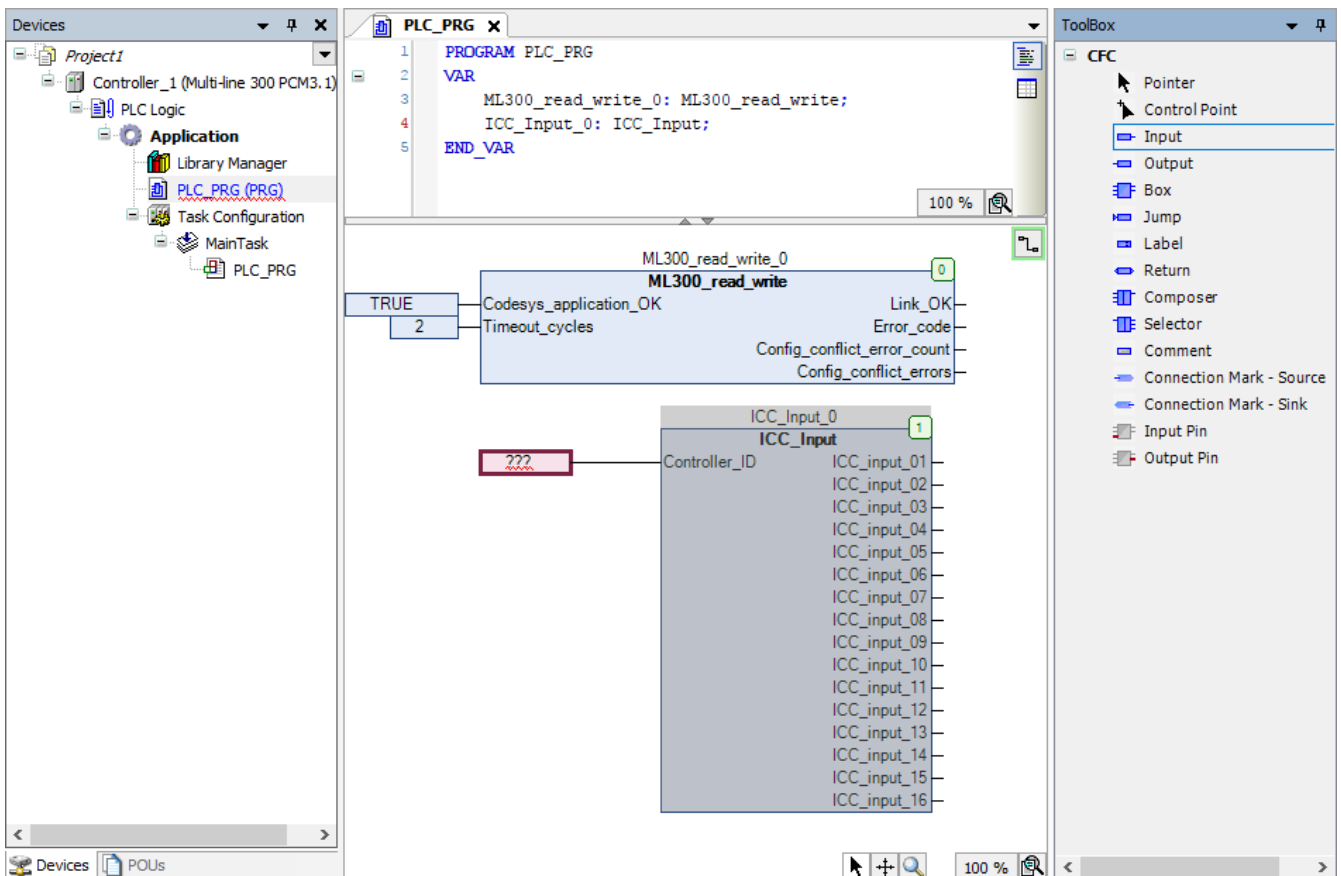
- Where **[Controller library]** is the library for your specific controller type, for example *MULTILINE_300_GPC_GENSET_GB* for a GPC 300 genset controller.
 - Select **OK**.
4. Press *Return* on your keyboard two times to open the *Auto Declare* window for the function block. Select **OK** to declare the variables as they are shown:



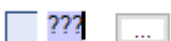
5. The **ICC_Input** function block has been added to the project:



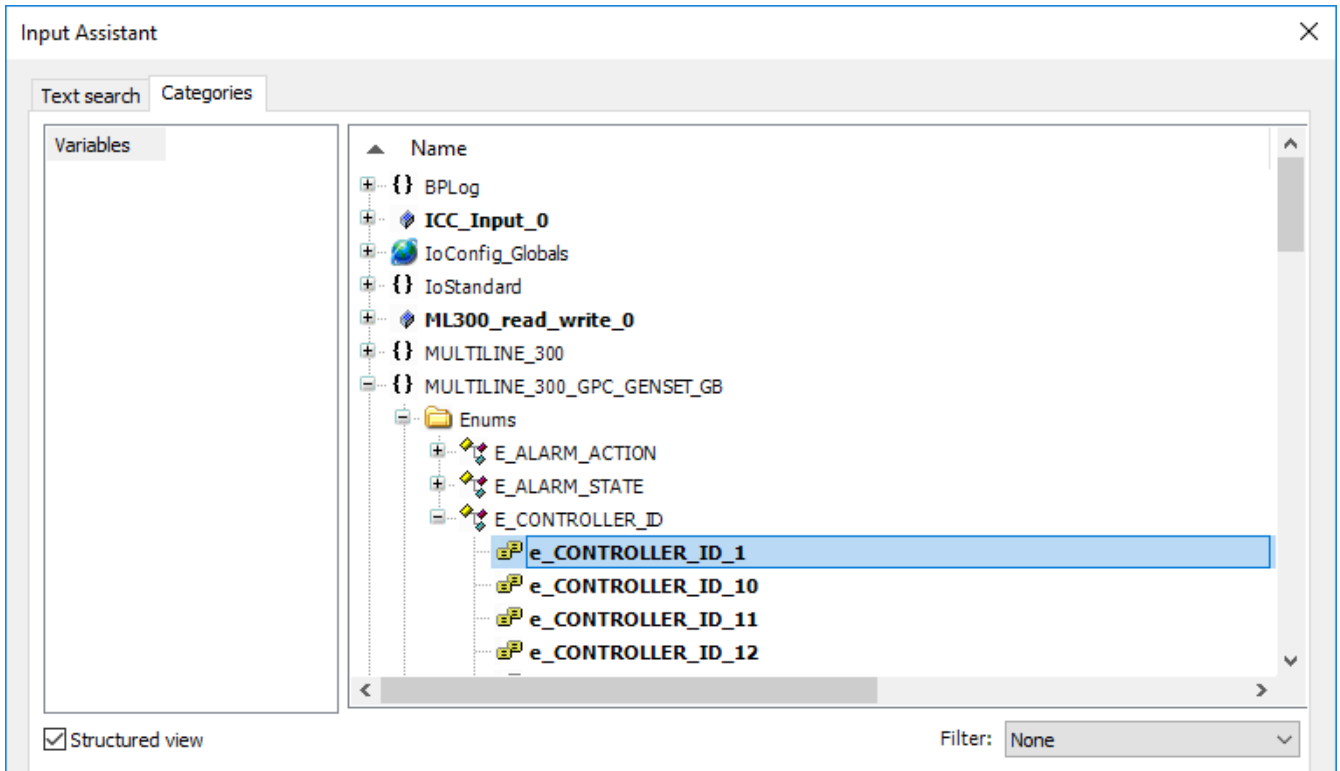
6. Drag an **Input** from the *ToolBox* to the implementation part of the working area and connect it to the *Controller_ID* input:



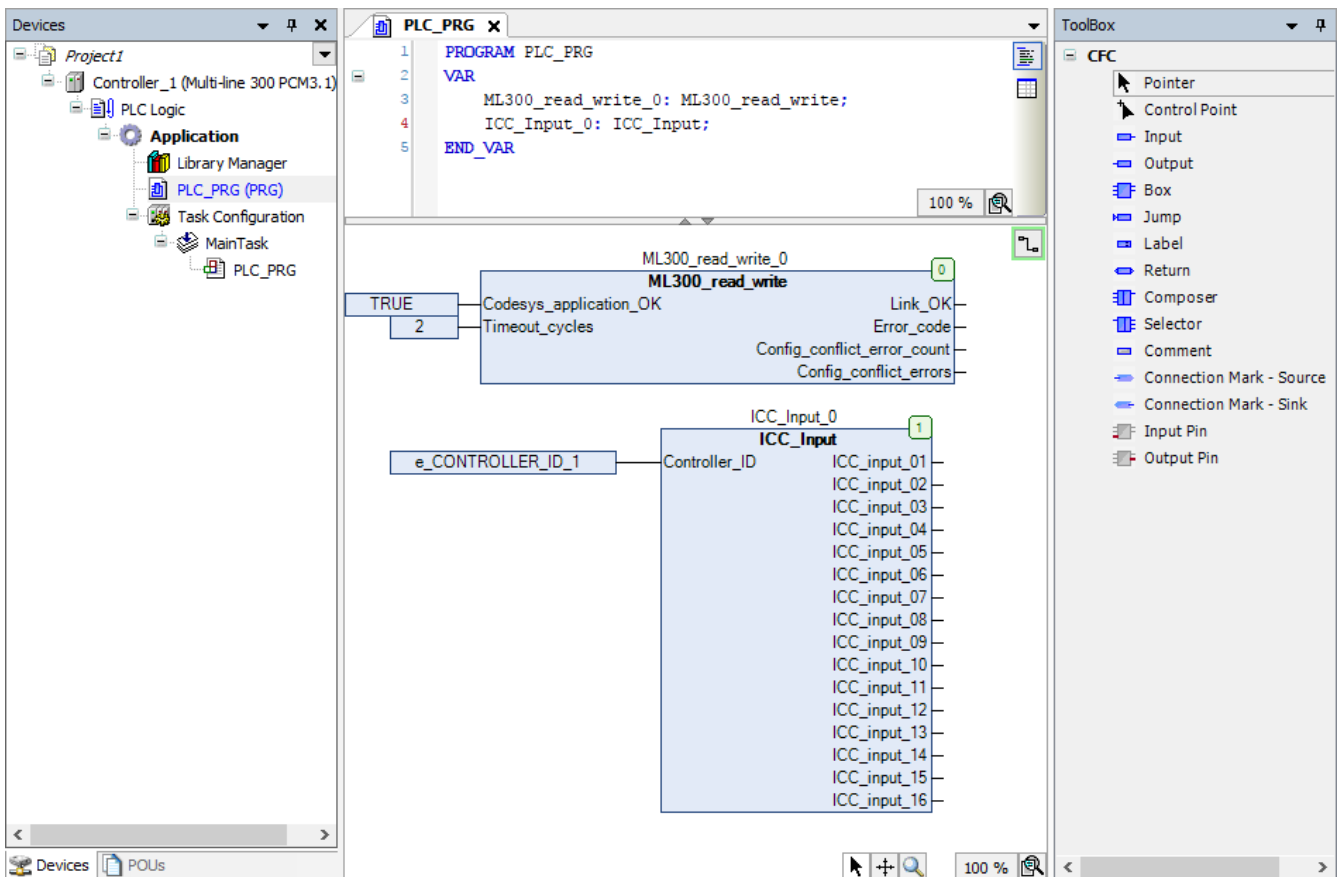
7. Select **???** in the **Input** and then select **...** to open the *Input Assistant* window for the *Input*:



8. Go to **Categories > Variables > [Controller library] > Enums > E_CONTROLLER_ID** and select the controller ID number of the controller where the information is read from (1 to 64):



- The **E_CONTROLLER_ID** of the controller that you read information from must match the **Controller ID** that was assigned to the controller on the single line diagram.
 - Select **OK**.
9. The **ICC_Input** function block is ready to be used in the application:



- You can now assign the Boolean outputs of the **ICC_Input** function block to other function blocks as inputs.

6. Additional libraries

6.1 Introduction

For further customisation and performance additional libraries can be installed. These are.

- Multiline_300_custom_parameters.compiled-library
- Multiline_300_priority_data.compiled-library.

The ML 300 controller libraries must be installed in CODESYS before you can add them to your application.



More information

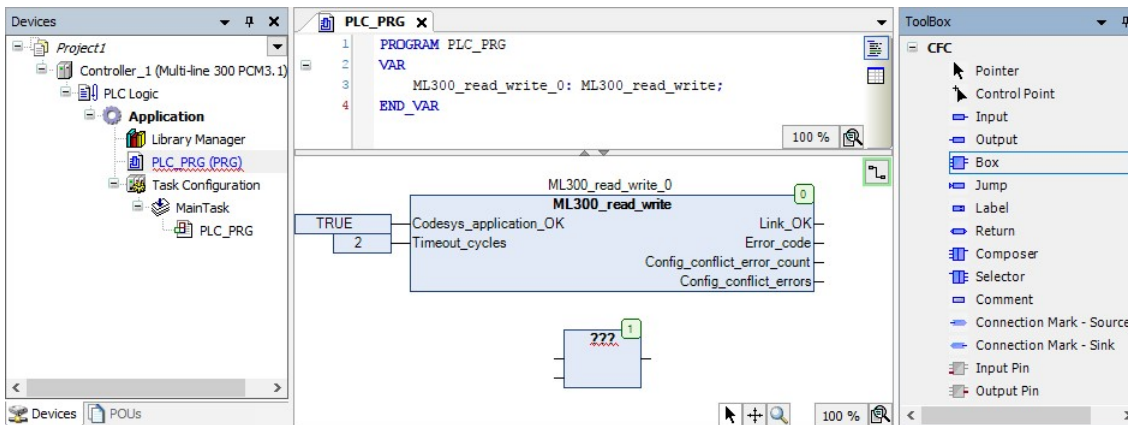
See **Get started with CODESYS, Install, Add the Multi-line 300 libraries to your application** for more information about how to install the ML 300 controller libraries.


6.2 Custom parameters

6.2.1 Setup BOOL function block

Follow these steps to add a Parameter setup Function block to your application:

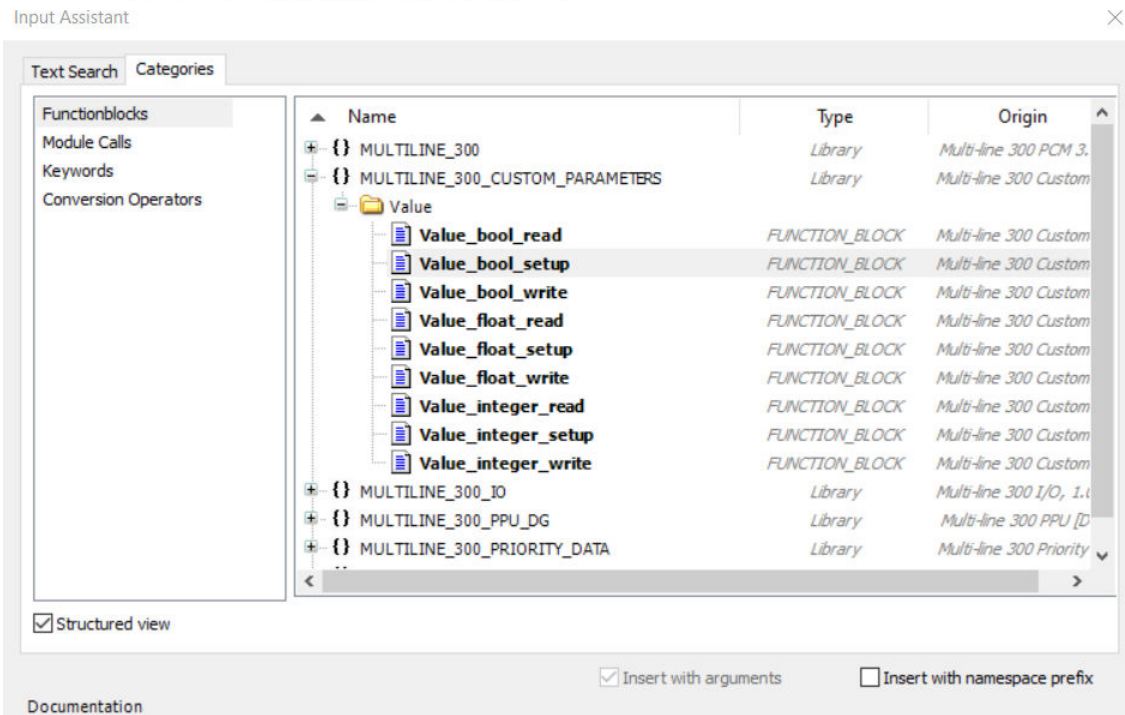
1. Drag a **Box** from the *ToolBox* to the implementation part of the working area:



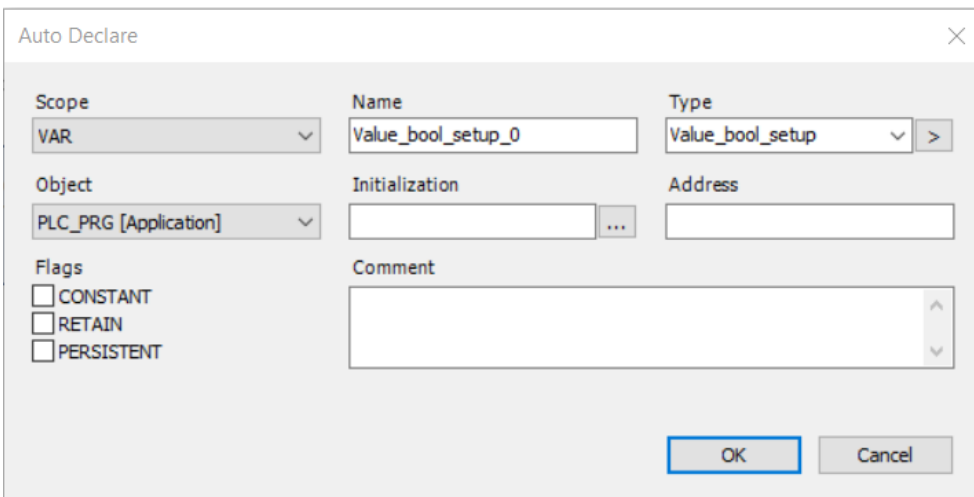
2. Select **???** in the **Box** and then  select to open the *Input Assistant* window:



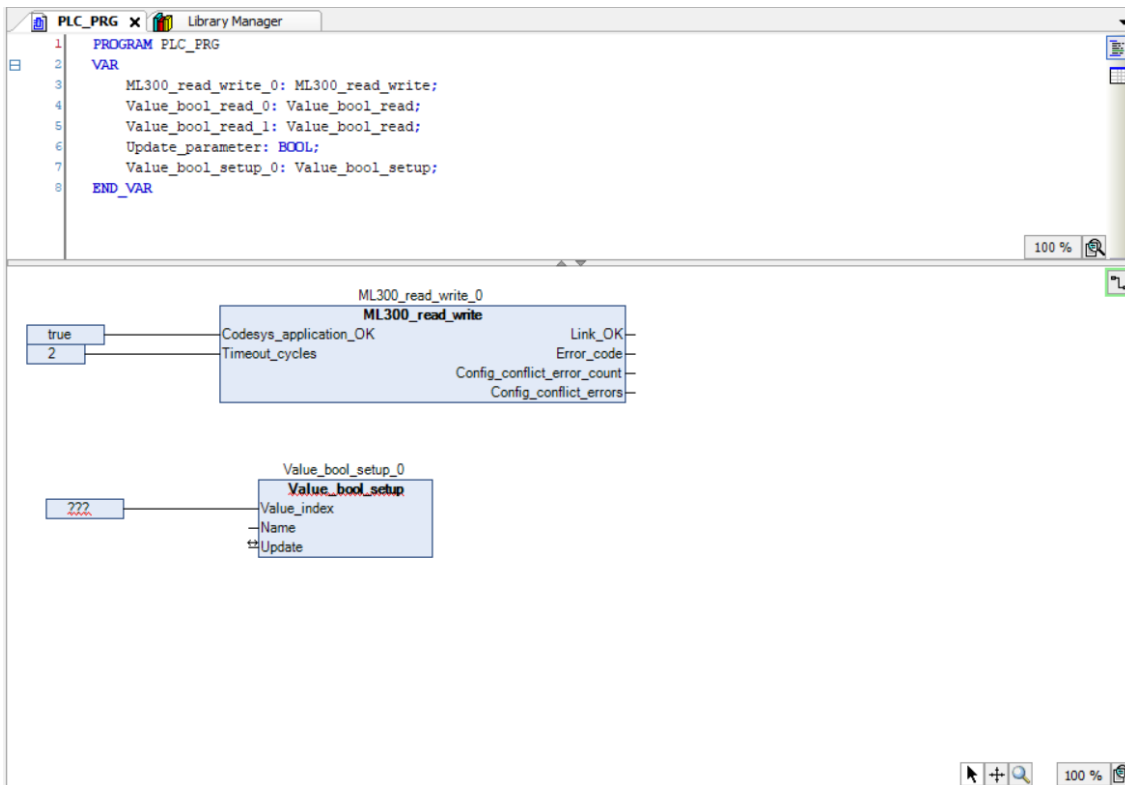
3. Go to **Categories > Functionblocks > MULTILINE_300_CUSTOM_PARAMETERS** and select the value bool setup block:



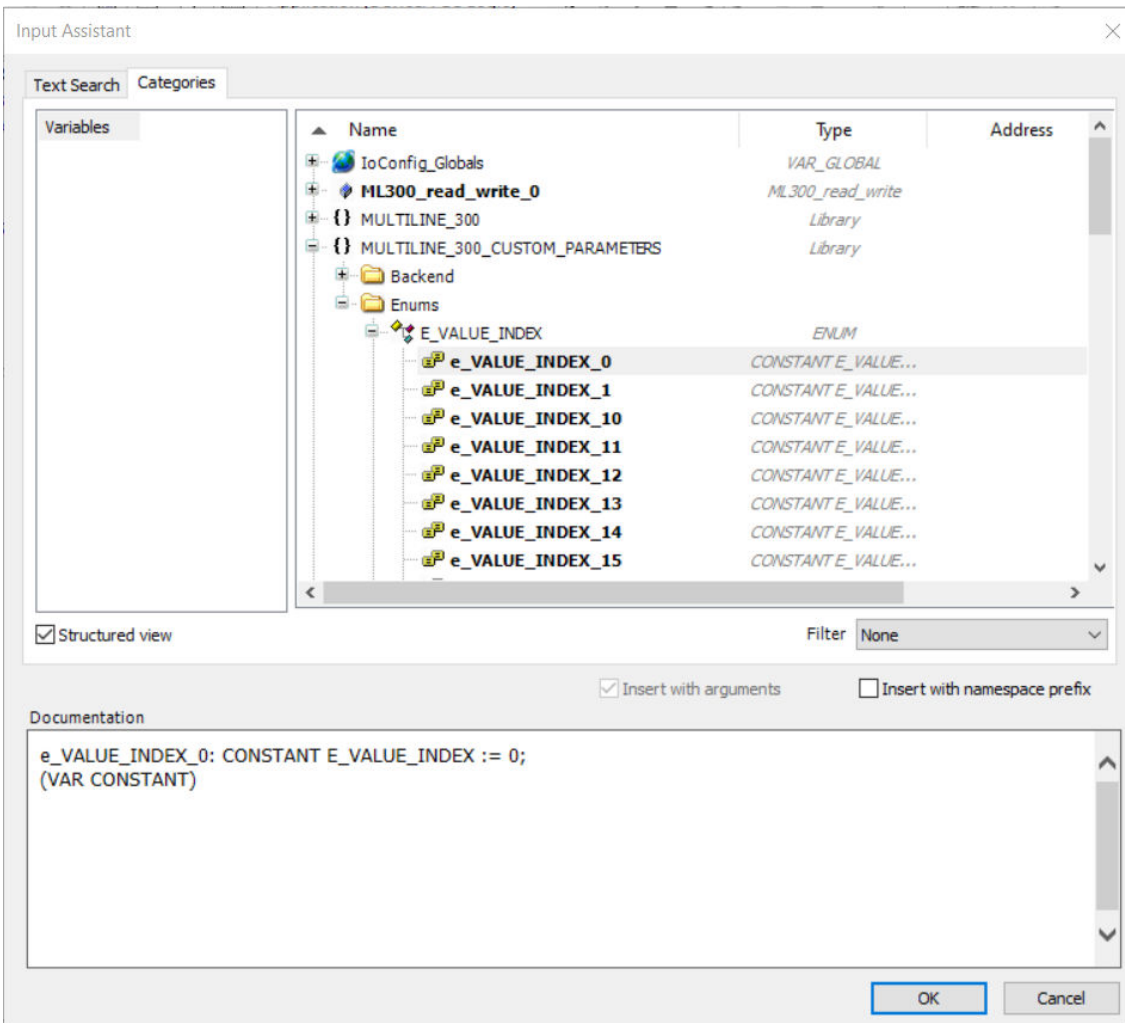
-
- 4. Select **OK**.
- 5. Press *Return* on your keyboard two times to open the *Auto Declare* window for the function block.
- 6. Select **OK** to declare the variables as they are shown:



-
- 7. Drag an **Input** from the *ToolBox* to the implementation part of the working area and connect it to the *Value_index* input:



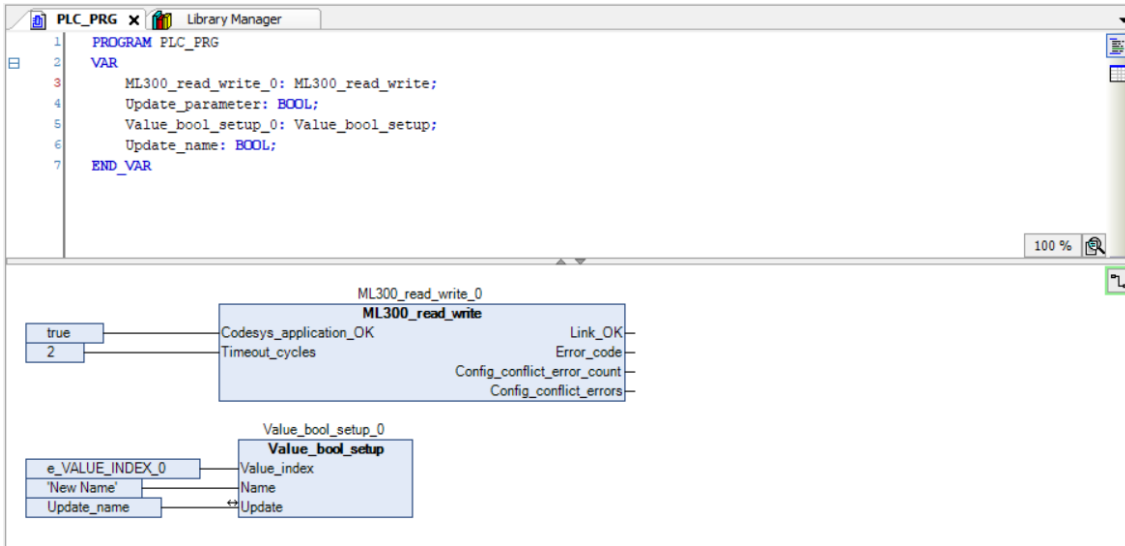
- 8. Go to **Categories > Variables > MULTILINE_300_CUSTOM_PARAMETERS > Enums > E_VALUE_INDEX** and select the index number of the input (0 to ?):



- The *Value_index* value must be unique for the Parameter type/set in your application.

- 9. Select **OK**.

- Drag an **Input** from the *ToolBox* to the implementation part of the working area and connect it to the *Name* input, and one to the *Update* input:

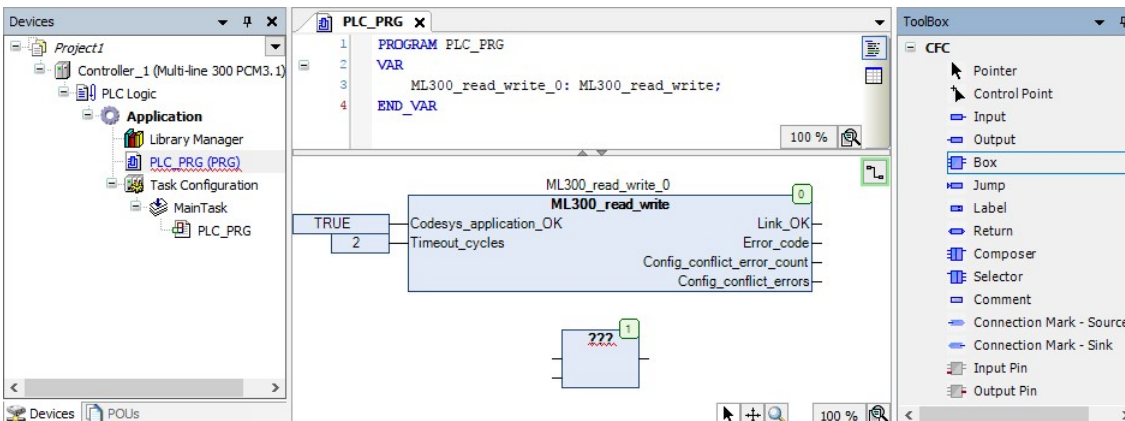


- To assign a **fixed name** to the I/O, type the name **must be surrounded by double quotation marks**.
 - For example, to rename the Parameter with **IO_INDEX 0** to *Procedure 1*, select **???** in the **Input**, type *"Procedure 1"* and press *Return* on your keyboard.
 - The new name is only visible on the controller after it is written to the controller using the *Update* input.
 - To assign a **variable** to the Parameter *Name* input:
 - Select the input that is connected to the *Name* input.
 - Type the variable name (for example, *new_name*) and press *Return* on your keyboard.
 - Select **OK** to declare the variable as it is shown.
 - To rename a Parameter that has a variable assigned to the *Name* input you must set the variable value to the name that you want to display, then write the new name to the controller using the *Update* input.
 - If you connect an input to the *Name* input, you must also add a variable to the *Update* input. This is used to write the selected name to the controller.
- The Parameter function block has been added to the project and can be assigned in the ML 300 controller after the application is downloaded to the controller.

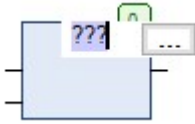
6.2.2 Setup float/integer function block

Follow these steps to add an Parameter setup Function block to your application:

- Drag a **Box** from the *ToolBox* to the implementation part of the working area:



- Select **???** in the **Box** and then **...** select to open the *Input Assistant* window:



- Go to **Categories > Functionblocks > MULTILINE_300_CUSTOM_PARAMETERS** and select the value float/integer setup block:

Input Assistant

Text Search Categories

Functionblocks
Module Calls
Keywords
Conversion Operators

Name	Type	
{ } DED	Library	CAA De...
{ } MULTILINE_300	Library	Multi-line
{ } MULTILINE_300_CUSTOM_PARAMETERS	Library	Multi-line
Value		
Value_bool_read	FUNCTION_BLOCK	Multi-line
Value_bool_setup	FUNCTION_BLOCK	Multi-line
Value_bool_write	FUNCTION_BLOCK	Multi-line
Value_float_read	FUNCTION_BLOCK	Multi-line
Value_float_setup	FUNCTION_BLOCK	Multi-line
Value_float_write	FUNCTION_BLOCK	Multi-line
Value_integer_read	FUNCTION_BLOCK	Multi-line
Value_integer_setup	FUNCTION_BLOCK	Multi-line
Value_integer_write	FUNCTION_BLOCK	Multi-line
{ } MULTILINE_300_IO	Library	Multi-line
{ } MULTILINE_300_PPU_DG	Library	Multi-line

Structured view

Insert with arguments Insert with namespace prefix

Documentation

FUNCTION_BLOCK Value_float_setup

Value_index	E_VALUE_INDEX	VAR_INPUT
Name	WSTRING	VAR_INPUT
Unit	E_VALUE_UNIT	VAR_INPUT
Min_value	REAL	VAR_INPUT
Max_value	REAL	VAR_INPUT

OK Cancel

- Select **OK**.
- Press *Return* on your keyboard two times to open the *Auto Declare* window for the function block. Select **OK** to declare the variables as they are shown:

Auto Declare

Scope: VAR

Name: Value_integer_setup_0

Type: Value_integer_setup

Object: PLC_PRG [Application]

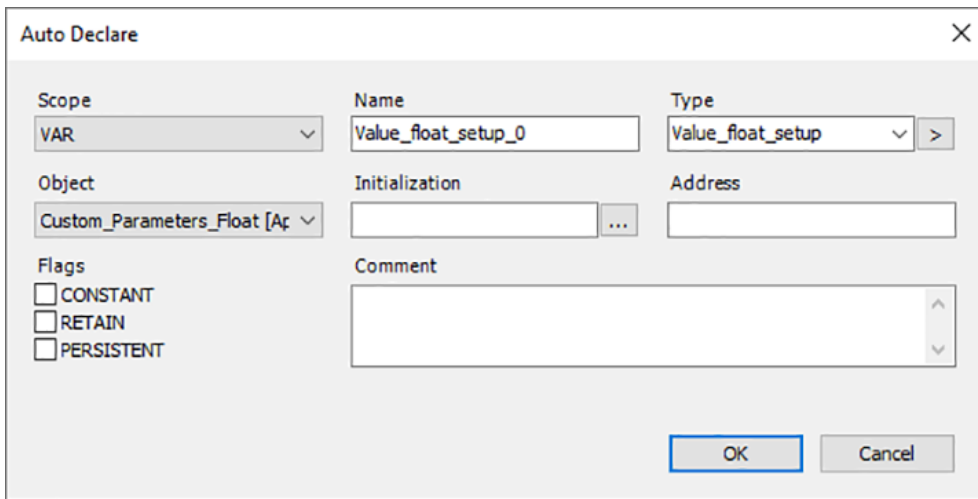
Initialization: [] ...

Address: []

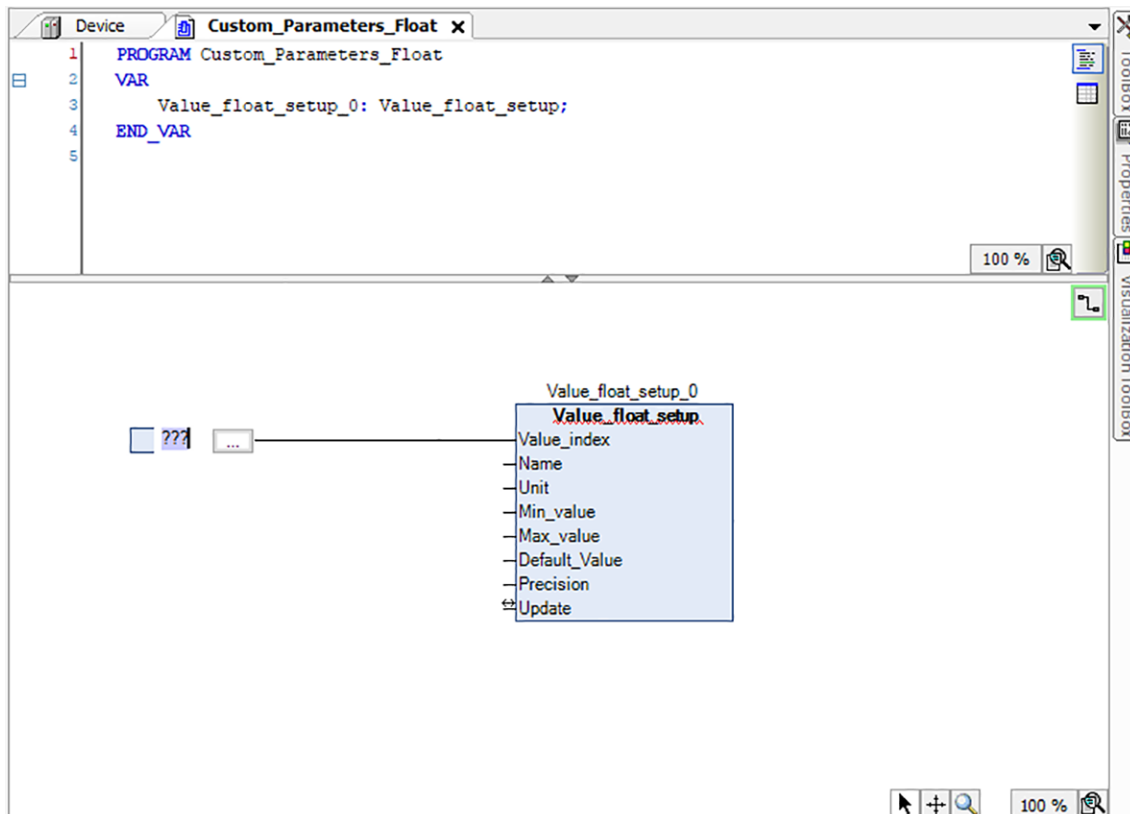
Flags: CONSTANT RETAIN PERSISTENT

Comment: []

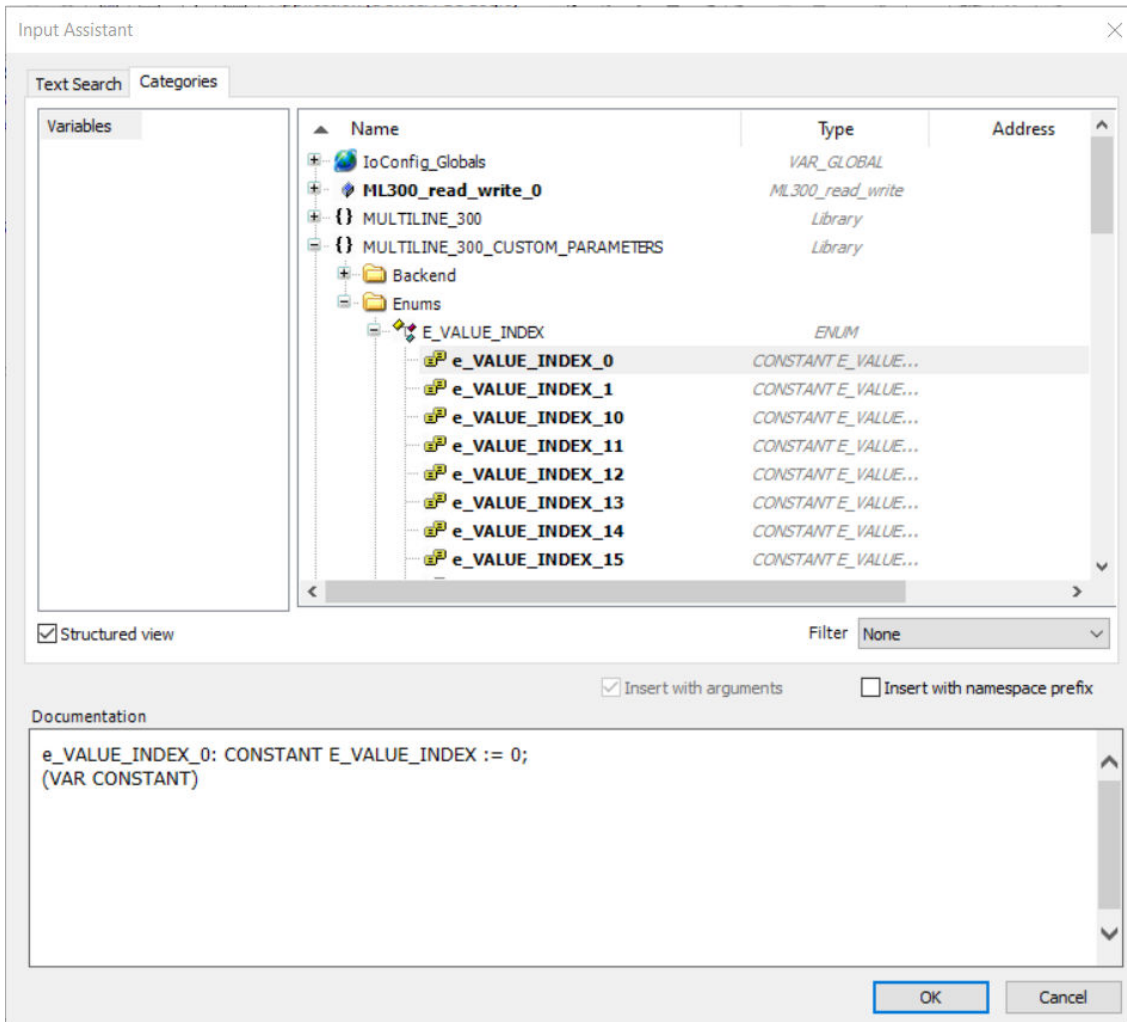
OK Cancel



6. Drag an **Input** from the *ToolBox* to the implementation part of the working area and connect it to the *Value_index* input:



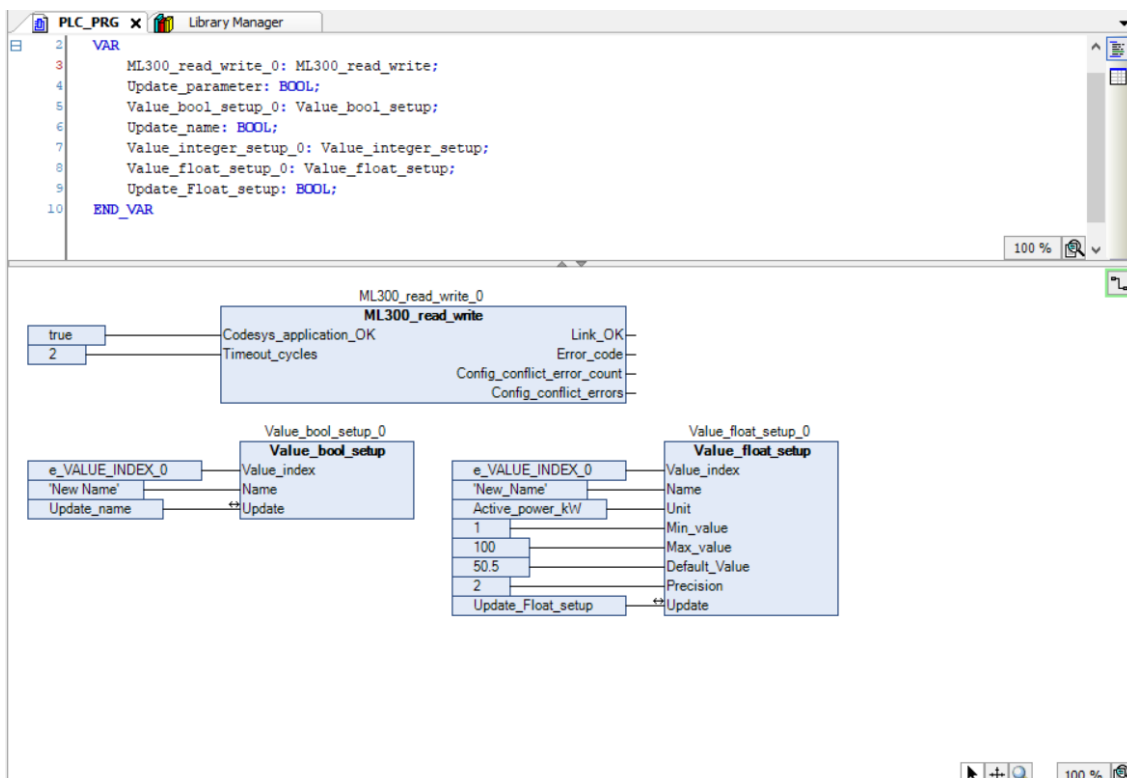
7. Go to **Categories > Variables > MULTILINE_300_CUSTOM_PARAMETERS > Enums > E_VALUE_INDEX** and select the index number of the input (0 to ?):



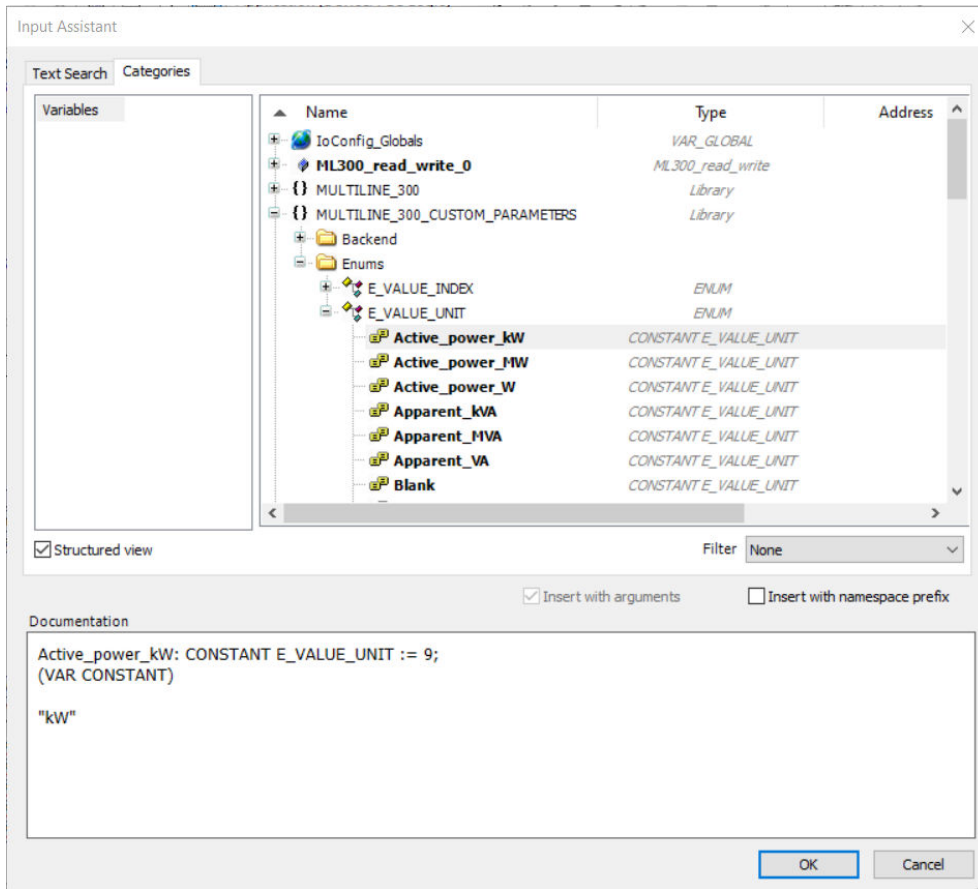
- The *Value_index* value must be unique for the Parameter type/set in your application.

8. Select **OK**.

9. Drag an **Input** from the *ToolBox* to the implementation part of the working area and connect it to the **Name, Unit, Min_value, Max_Value, Default_Value, precision (Float only) and Update** input:



- To assign a **fixed name** to the parameter, type the name **must be surrounded by double quotation marks**.
 - For example, to rename the Parameter with **IO_INDEX 0** to *Procedure 1*, select **???** in the **Input**, type *"Procedure 1"* and press *Return* on your keyboard.
- To assign a unit, represented in PICUS, Go to **Categories > Variables > MULTILINE_300_CUSTOM_PARAMETERS > Enums > E_VALUE_Unit** and select the unit you need.



- Define a minimum value, this restricts PICUS from writing values lower than the min_Value.
 - Define a maximum value, this restricts PICUS from writing values higher than the max_Value.
 - Define a default value, this value will be the default.
 - Define Precision, this defines the amount of decimals shown in PICUS. (Precision parameter is only available for Custom Parameter of type *Float*, not for *Integer*.)
- The new name is only visible on the controller after it is written to the controller using the *Update* input.
- To assign a **variable** to the Parameter *Name* input:
 - Select the input that is connected to the *Name* input.
 - Type the variable name (for example, *new_name*) and press *Return* on your keyboard.
 - Select **OK** to declare the variable as it is shown.
- To rename an Parameter that has a variable assigned to the *Name* input you must set the variable value to the name that you want to display, then write the new name to the controller using the *Update* input.
- If you connect an input to the *Name* input, you must also add a variable to the *Update* input. This is used to write the selected name to the controller.

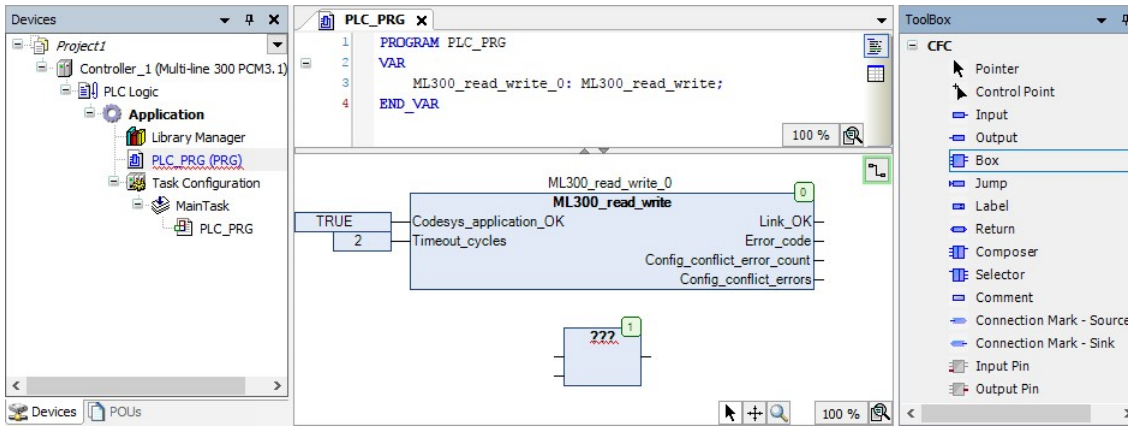
10. The Parameter function block has been added to the project and can be assigned in the ML 300 controller after the application is downloaded to the controller.

6.2.3 Setup read function block

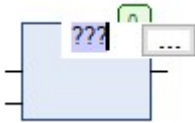
The Read Function Block has the same parameters for all three block types, *value-index*, *value* and *update*.

Follow these steps to add an Parameter Read Function block to your application:

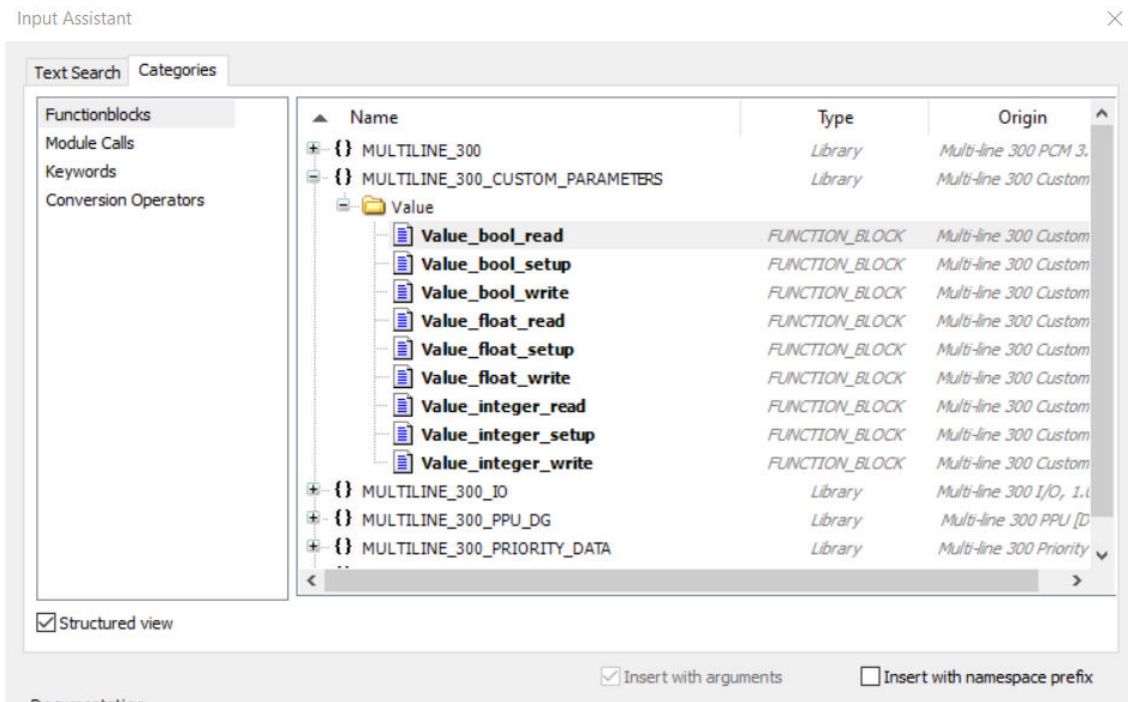
1. Drag a **Box** from the *ToolBox* to the implementation part of the working area:



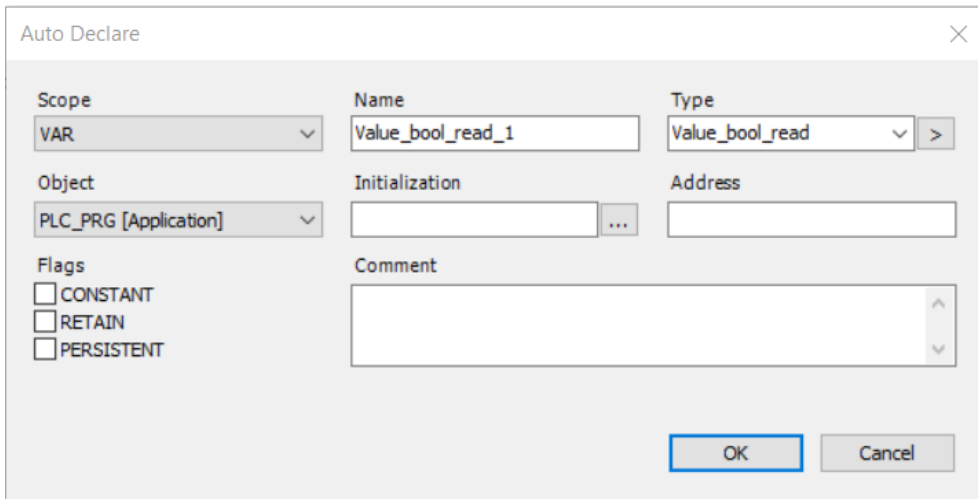
2. Select **???** in the **Box** and then **...** select to open the *Input Assistant* window:



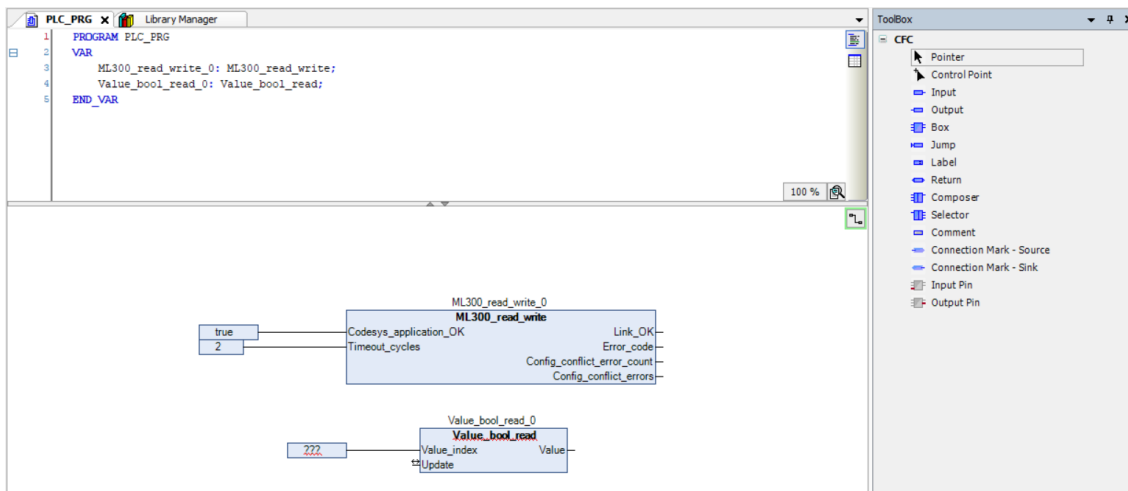
3. Go to **Categories > Functionblocks > MULTILINE_300_CUSTOM_PARAMETERS** and select the value read block you need:



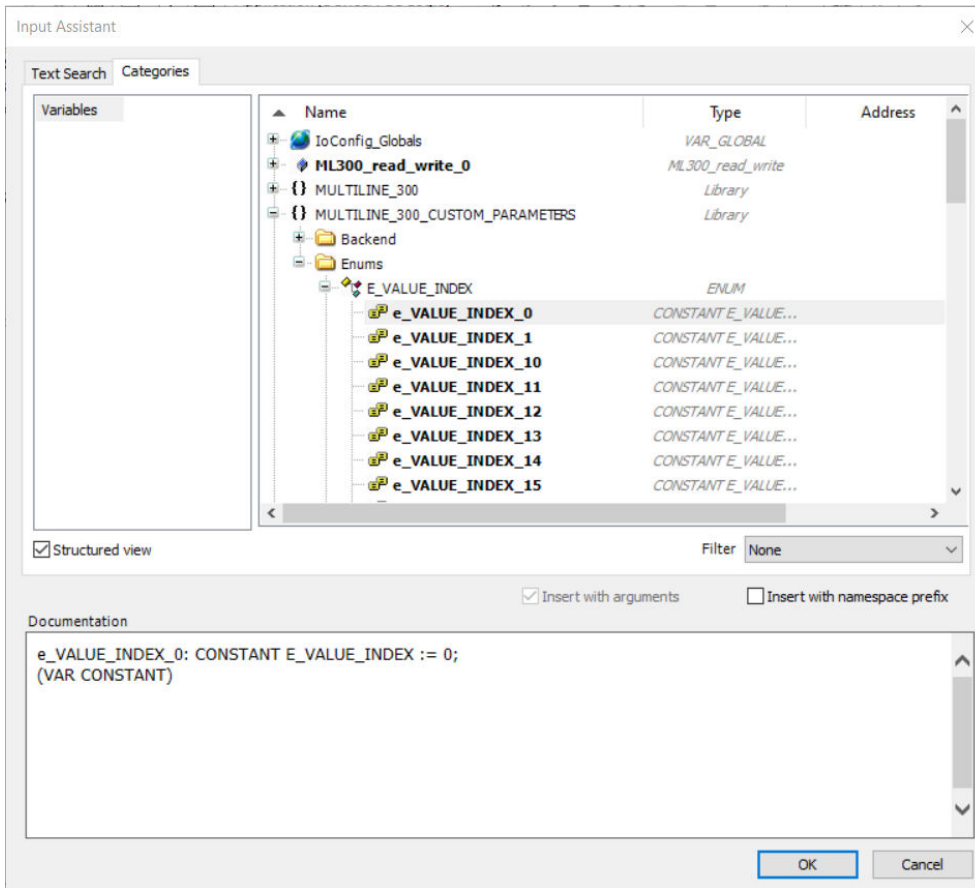
4. Select **OK**.
5. Press *Return* on your keyboard two times to open the *Auto Declare* window for the function block. Select **OK** to declare the variables as they are shown:



6. Drag an **Input** from the *ToolBox* to the implementation part of the working area and connect it to the *Value_index* input:



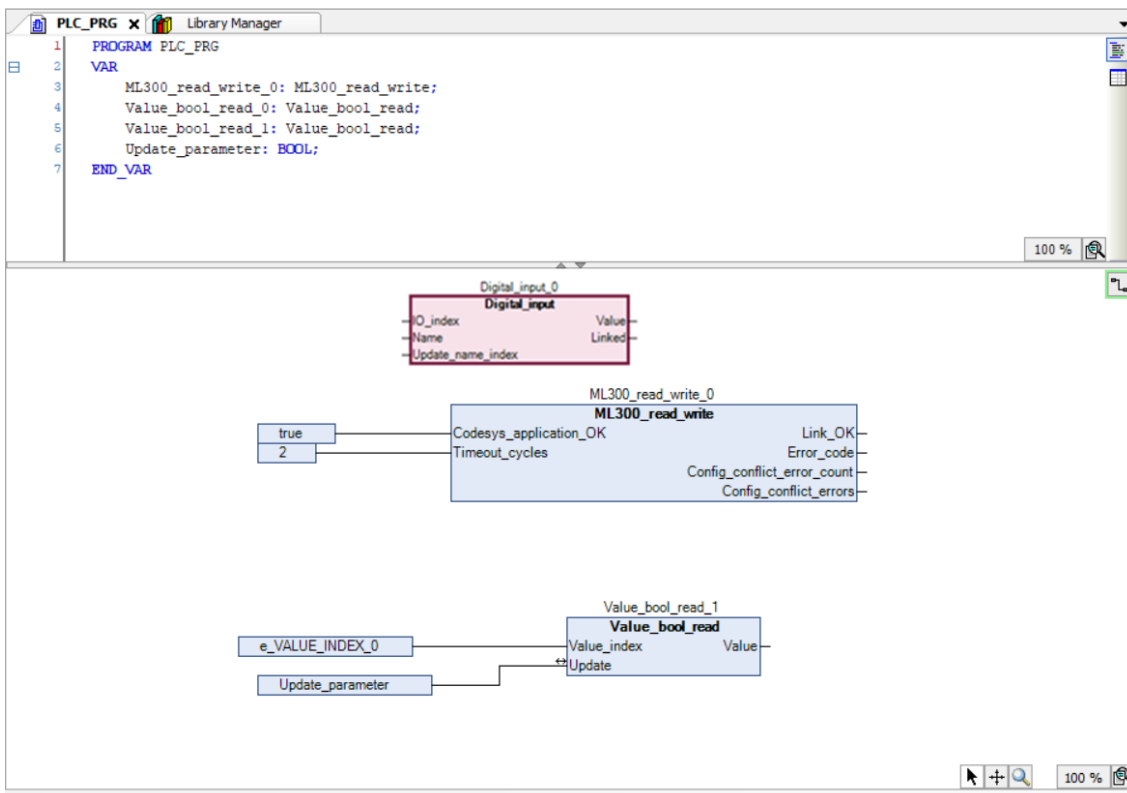
7. Go to **Categories > Variables > MULTILINE_300_CUSTOM_PARAMETERS > Enums > E_VALUE_INDEX** and select the index number of the input (0 to ?):



- The *Value_index* value must be unique for the Parameter type/set in your application.

8. Select **OK**.

9. Drag an **Input** from the *ToolBox* to the implementation part of the working area and connect it to the *Update* input:



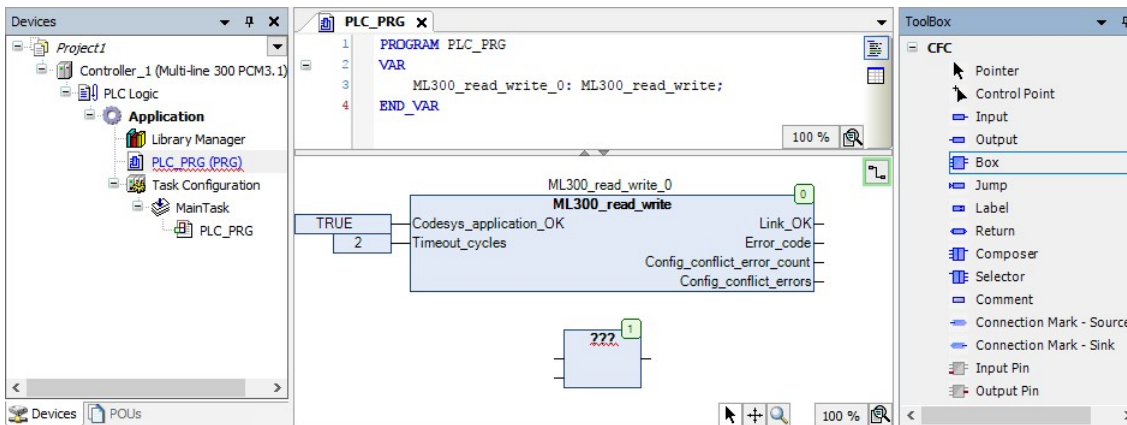
- The update input is bi-directional and will return your command to 0 after update.


6.2.4 Setup write function block

The Write Function Block has the same parameters for all three block types, *value-index*, *value* and *update*.

Follow these steps to add a Parameter Write Function block to your application:

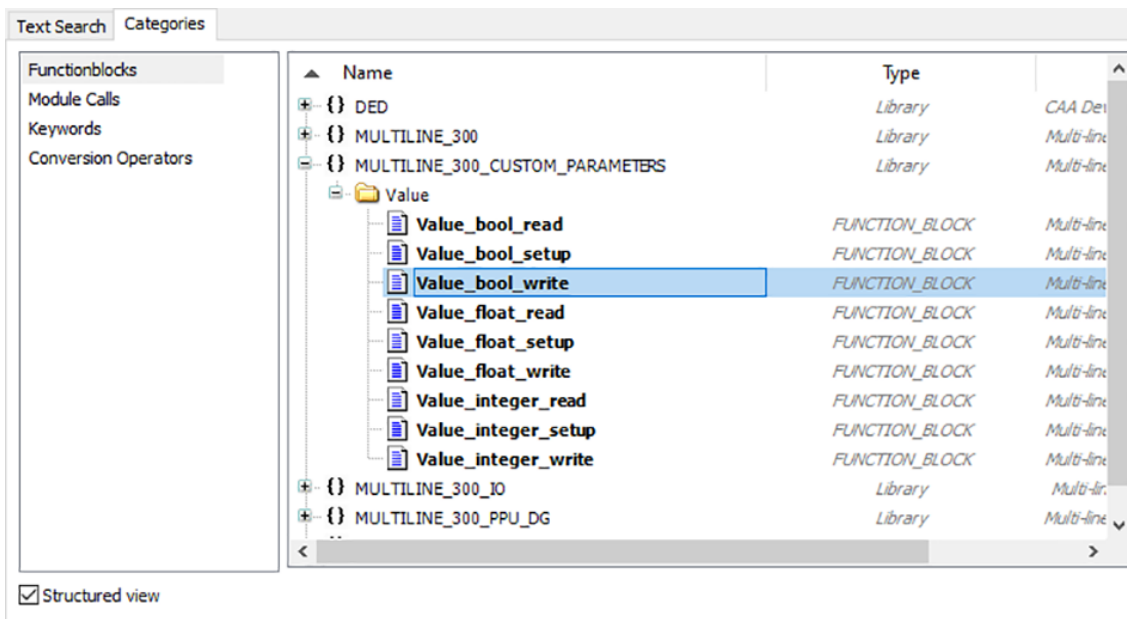
1. Drag a **Box** from the *ToolBox* to the implementation part of the working area:



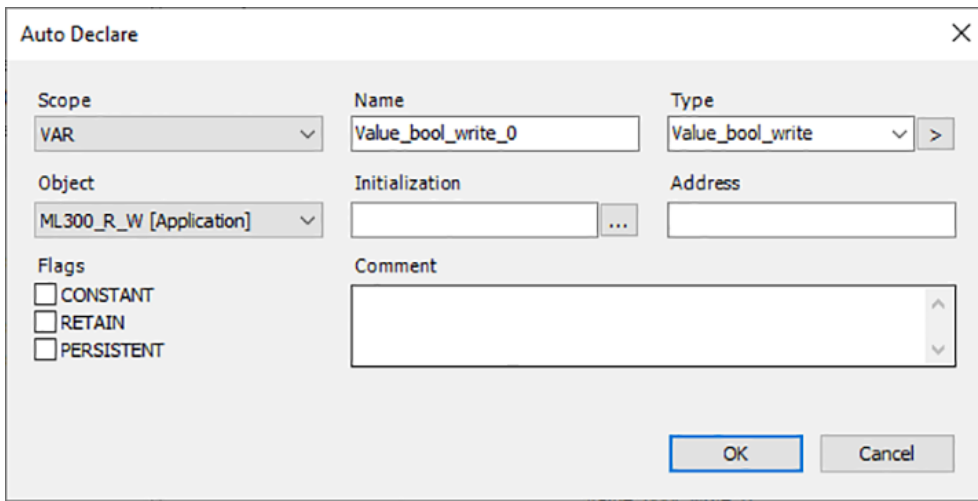
2. Select **???** in the **Box** and then  select to open the *Input Assistant* window:



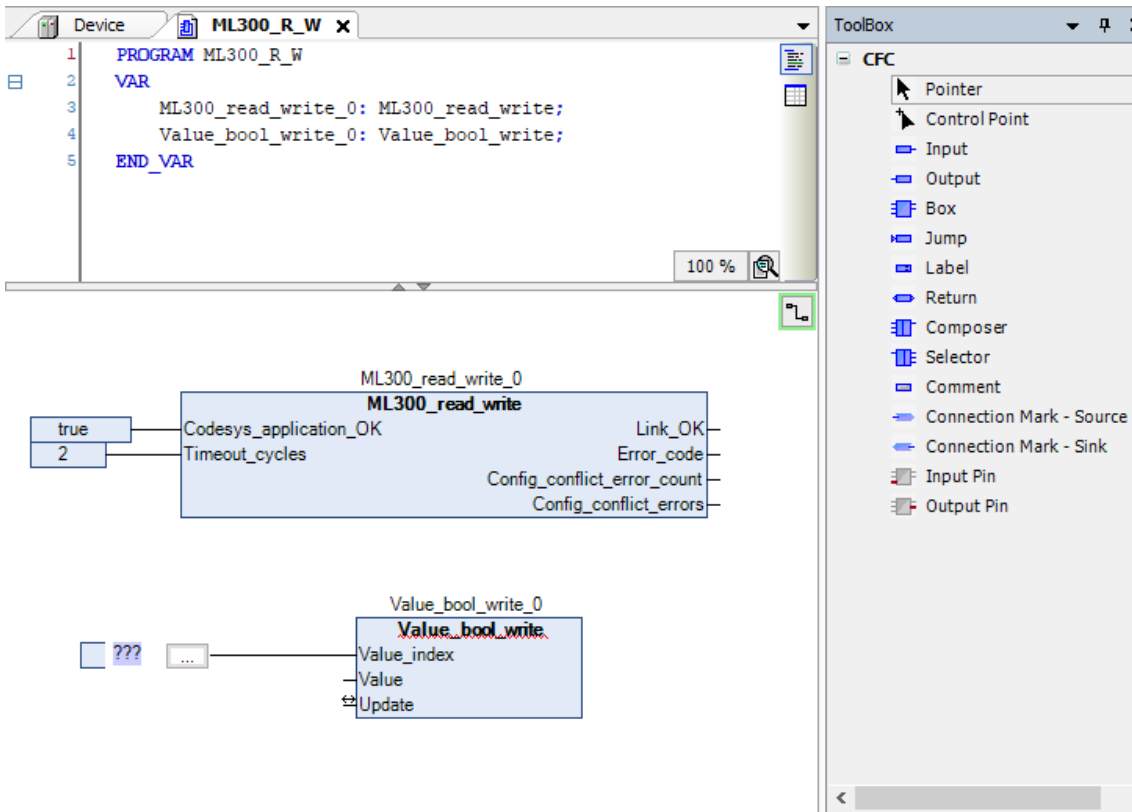
3. Go to **Categories > Functionblocks > MULTILINE_300_CUSTOM_PARAMETERS** and select the value write block you need:



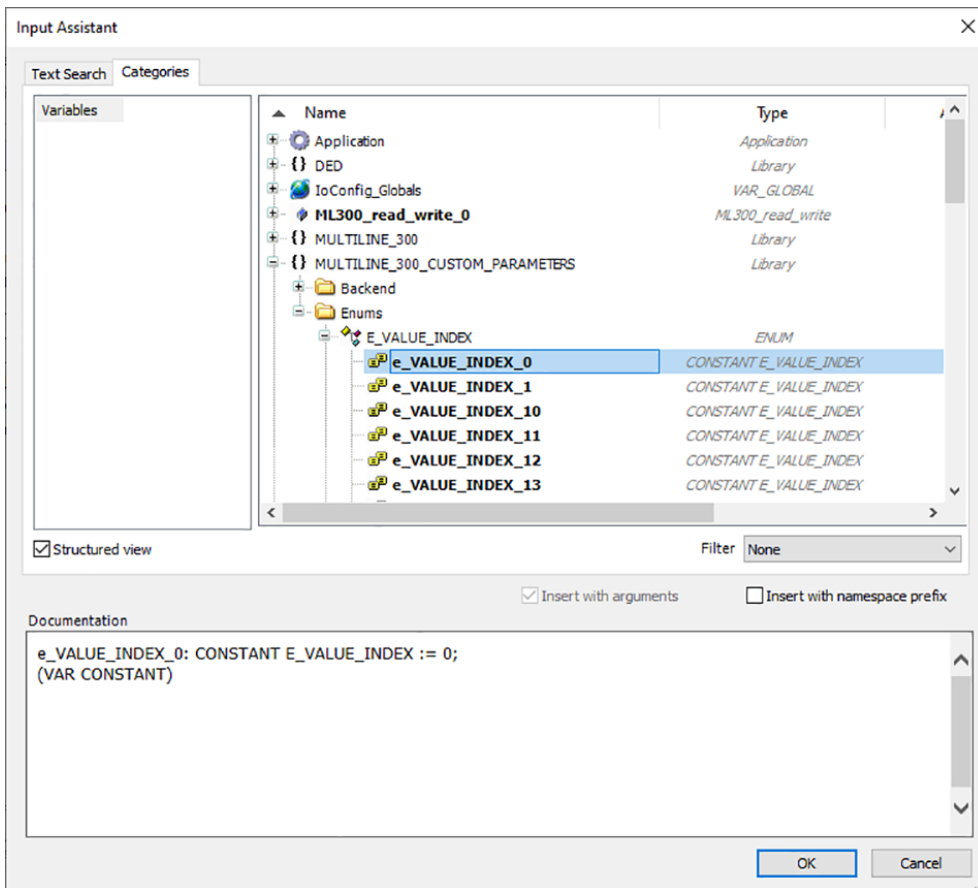
4. Select **OK**.
5. Press *Return* on your keyboard two times to open the *Auto Declare* window for the function block. Select **OK** to declare the variables as they are shown:



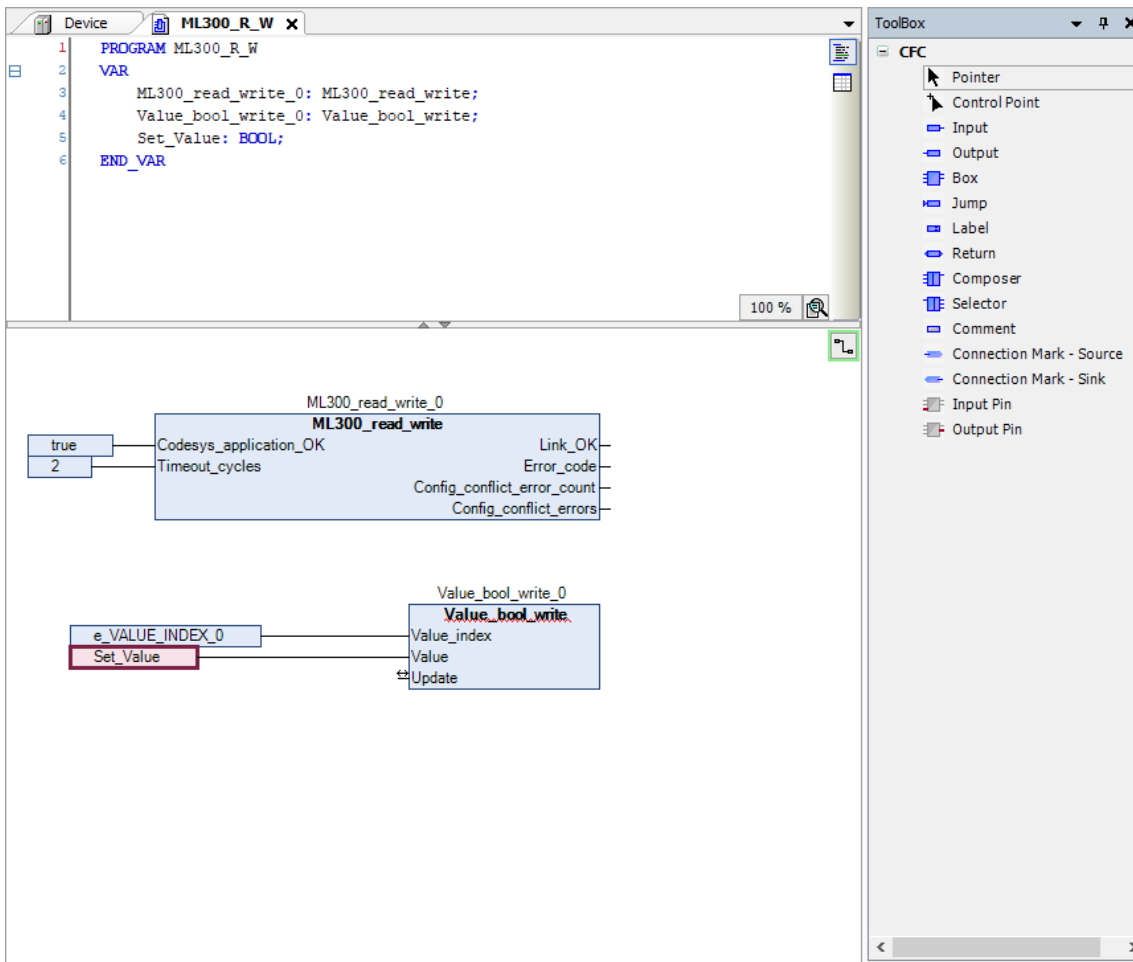
6. Drag an **Input** from the *ToolBox* to the implementation part of the working area and connect it to the *Value_index* input:



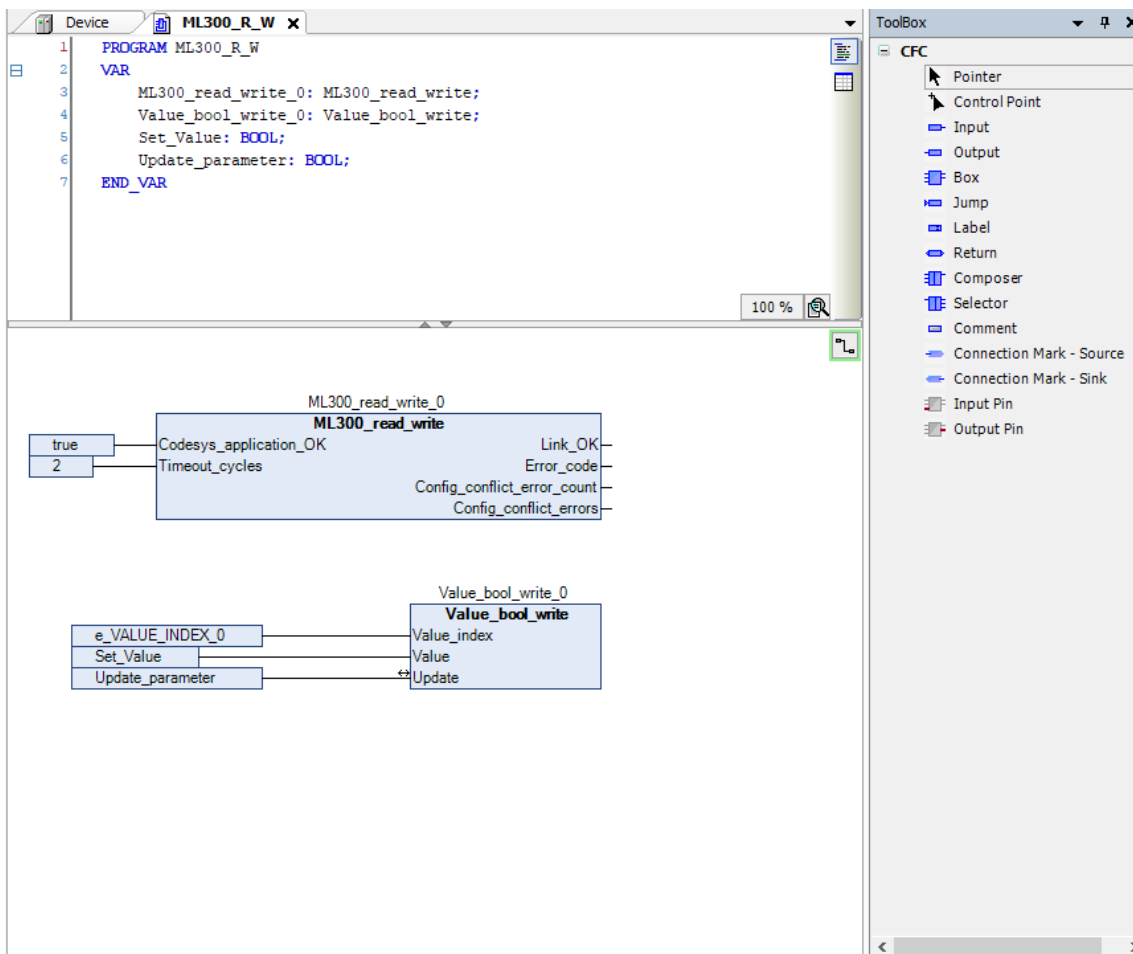
7. Go to **Categories > Variables > MULTILINE_300_CUSTOM_PARAMETERS > Enums > E_VALUE_INDEX** and select the index number of the input (0 to 49):



- The *Value_index* value must be unique for the Parameter type/set in your application.
8. Select **OK**.
 9. Drag an **Input** from the *ToolBox* to the implementation part of the working area and connect it to the *Value* input:



10. Drag an **Input** from the *ToolBox* to the implementation part of the working area and connect it to the *Update* input:



- The update input is bi-directional and will return your command to 0 after update.

6.3 Assign a CODESYS I/O function in the controller

To see the custom CODESYS I/Os in the controller:

1. The POU containing the I/O function block must be added to **MainTask** in the project tree.
2. The program must be downloaded to the controller.
3. The program must have run at least once.



More information

See **ML 300 CODESYS projects, Download the application to the controller** for more information about how to download and run your program on an ML 300 controller.

Follow these steps to assign the CODESYS I/O function block to a controller terminal:

1. Use PICUS to log on to the ML 300 controller to which the I/O is assigned, and go to **Configure > Input/output:**