

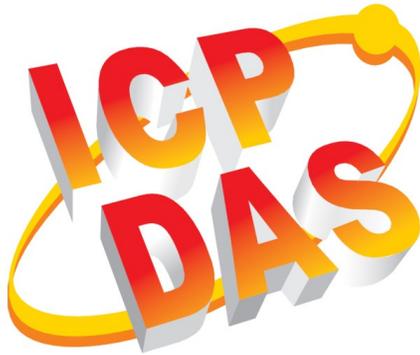
HMI

(eLogger, Indusoft)

for
Win-GRAF Runtime
(Windows Version)

User Manual

(Version 1.0)



WARRANTY

All products manufactured by ICP DAS are warranted against defective materials for a period of one year from the date of delivery to the original purchaser.

WARNING

ICP DAS assumes no liability for damages consequent to the use of this product. ICP DAS reserves the right to change this manual at any time without notice. The information furnished by ICP DAS is believed to be accurate and reliable. However, no responsibility is assumed by ICP DAS for its use, nor for any infringements of patents or other rights of third parties resulting from its use.

COPYRIGHT

Copyright © 2022 by ICP DAS. All rights are reserved.

TRADEMARK

Names are used for identification only and may be registered trademarks of their respective companies.

CONTACT US

If you have any questions, please feel free to contact us via email at:

service@icpdas.com

service.icpdas@gmail.com

Revision

Revision	Date	Description	Author
1.0	24.11.2021	Initial version	M. K.

Contents

1	INTRODUCTION.....	5
2	ELOGGER.....	6
2.1	INTRODUCTION.....	6
2.2	SOFTWARE INSTALLATION.....	7
2.2.1	<i>eLogger.....</i>	<i>7</i>
2.2.2	<i>Win-GRAF.....</i>	<i>7</i>
2.2.3	<i>Win-GRAF - eLogger Communication.....</i>	<i>7</i>
2.3	WIN-GRAF MEMORY MAPPING.....	10
2.4	ELOGGER MEMORY MAPPING.....	15
2.5	SHARED MEMORY UTILITY.....	21
2.6	DEMO PROGRAM.....	22
3	INDUSOFT.....	23
3.1	INTRODUCTION.....	23
3.2	SOFTWARE INSTALLATION.....	23
3.2.1	<i>InduSoft.....</i>	<i>24</i>
3.2.2	<i>Win-GRAF.....</i>	<i>24</i>
3.3	WIN-GRAF - INDUSOFT COMMUNICATION.....	24
3.4	VARIABLE - TAG MAPPING.....	27
3.4.1	<i>Win-GRAF Configuration.....</i>	<i>27</i>
3.4.2	<i>InduSoft Configuration.....</i>	<i>33</i>
3.4.3	<i>Tag Mapping.....</i>	<i>34</i>
3.5	INDUSOFT - WINGRAF MONITOR UTILITY.....	40
3.6	DEMO PROGRAM.....	41
4	PROGRAMMING INTERFACE FOR EXTERNAL PROGRAM.....	42
4.1	INTRODUCTION.....	42
4.2	REQUIRED LIBRARIES.....	43
4.3	COMMUNICATION STRUCTURE.....	43
4.4	WIN-GRAF WORKBENCH CONFIGURATION.....	45
4.4.1	<i>Defining and Creating Shared Memory.....</i>	<i>45</i>
4.5	EXTERNAL PROGRAM.....	51
4.5.1	<i>Create/Link.....</i>	<i>52</i>
4.5.2	<i>Data Exchange.....</i>	<i>55</i>
4.5.3	<i>Close Link.....</i>	<i>57</i>
4.5.4	<i>Error Table.....</i>	<i>57</i>

1 Introduction

Two HMI software packages are provided for the Win-GRAF runtime (Windows version): eLogger and Indusoft. eLogger is an in-house developed software which provides a cost-effective solution for basic HMI requirements. Indusoft is a more advanced HMI solution and supports the development of a more advanced graphical user interface. Both HMI packages enable the developer to directly exchange data with the runtime via shared memory (Figure 1).

In addition programming interfaces are available in standard C which allows the user to create a user interface for the PLC application program. It is possible to create a user interface in any of the most common used programming languages, e.g. Python, C#, LabVIEW etc..

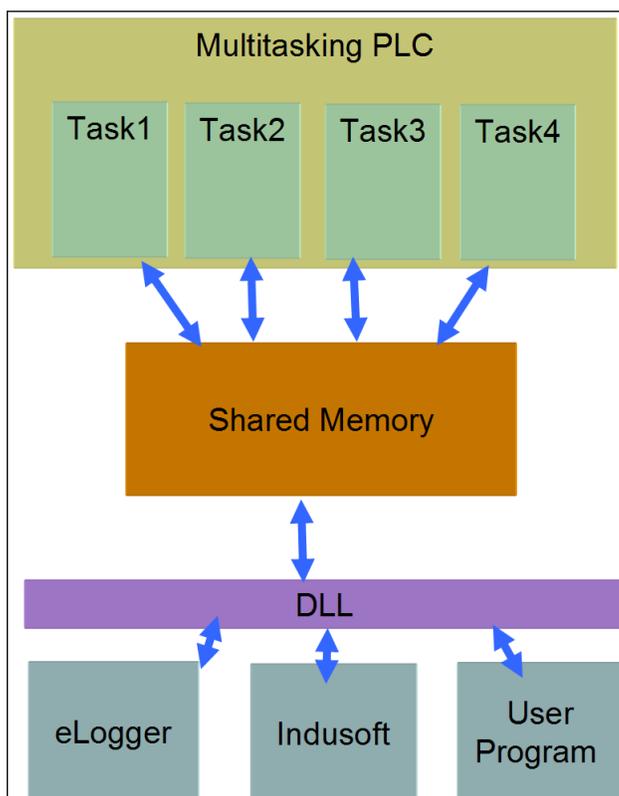


Figure 1: Data exchange between runtime and external programs

2 eLogger

2.1 Introduction

eLogger is an easy-to-use HMI software platform developed by ICPDAS. eLogger's HMI application (Figure 2) can either remotely communicate with the Win-GRAF runtime using Modbus or can be directly installed on the same PC as the Win-GRAF runtime.

This manual main focus is to explain how to setup the system when both eLogger and Win_GRAF runtimes share the same platform. If you want to use eLogger on a remote device consult the Win-GRAF Workbench and the eLogger user manual for more information. In this case it necessary to configure the Modbus register for both eLogger and Win-GRAF runtime which will not be discussed in this manual.

In this manual the configuration of both the Win-GRAF and eLogger runtimes which share the same PC will be shown. When installed on the same device both runtimes exchange data via a shared memory. This chapter introduces the shared memory structure and the PLC HMI data mapping procedure. eLogger has an integrated web server which supports the design of a web server HMI (Figure 2) for remotely controlling the PLC application.

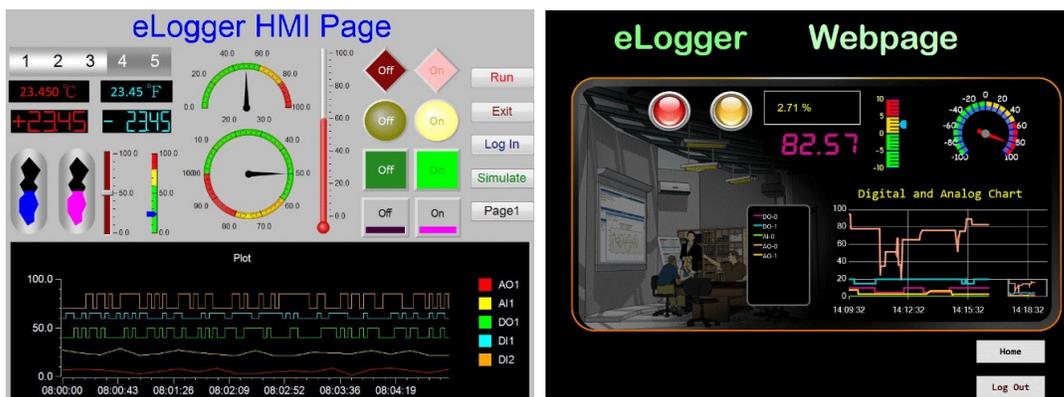


Figure 2: Graphical user interfaces created by eLogger

The eLogger supports the following functions:

- Multi-screen navigation

- Web HMI
- Real time data trend
- Data logging
- MQTT Client
- Modbus TCP Server

2.2 Software Installation

2.2.1 eLogger

The installation files and manual for eLogger has to be downloaded from the ICPDAS website '<https://www.icpdas.com/>'. Enter the name '*eLogger*' into the search box to get to the product website and click the '*Download*' button

- Download the software manual from the 'User Manual' category
- Download the eLogger runtime and developer installation file from the '*Utility&Tools*' category.

Follow the installation instructions described in the '*elogger_user_manual_en.pdf*' to install all the necessary software packages for the eLogger on a Windows PC.

2.2.2 Win-GRAF

In order for the Win-GRAF runtime to be able to communicate with the eLogger runtime the following dynamic libraries files have to be added to the directory of the Win-GRAF runtime execution file:

- eLoggerMem.dll
- ddkc_elogger.dll

By default both libraries are automatically installed by executing the runtime setup program.

2.2.3 Win-GRAF - eLogger Communication

The Win-GRAF and eLogger runtime are two separate programs which communicates via a shared memory.

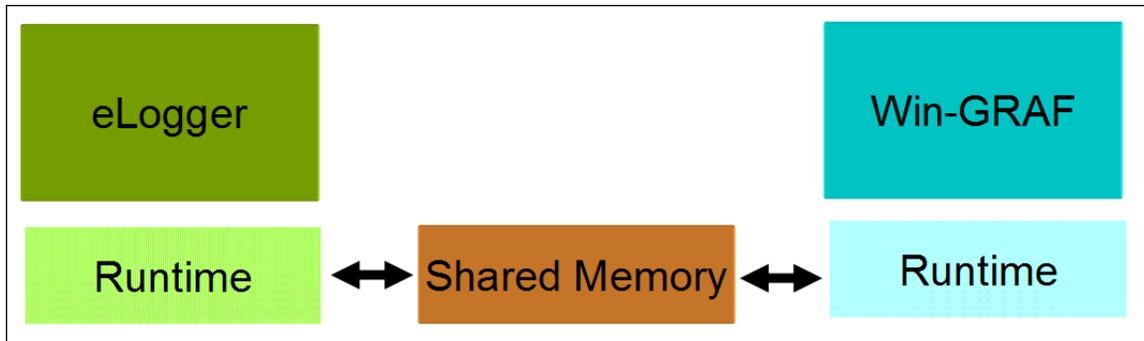


Figure 3: eLogger - Win-GRAF communication structure

The shared memory is divided into several memory segments to decrease the access time and increase the communication speed. It basically consists of three memory types which store different data formats: BOOL, UINT and STRING. The read/write segments determine the access rights of each runtime:

- Read:

- DO: eLogger writes bool data to the shared memory; Win-GRAF can only read this value but not modify it. The memory unit size is one byte.
- AO: eLogger writes two bytes to the shared memory. Win-GRAF on the other side is only allowed to read two bytes from this memory section. The memory unit size is two bytes.

- Write:

- DI: eLogger reads bool data from the shared memory which has been set by the Win-GRAF runtime. Win-GRAF has only write access and the eLogger only read access. The memory unit size is one byte.
- Word: eLogger has only read access and reads two bytes from the shared memory written by Win-GRAF. Win-GRAF can only write to this memory area. The memory unit size is two bytes.

- String: Both eLogger and Win-GRAF are allowed to read and write strings to this memory area.

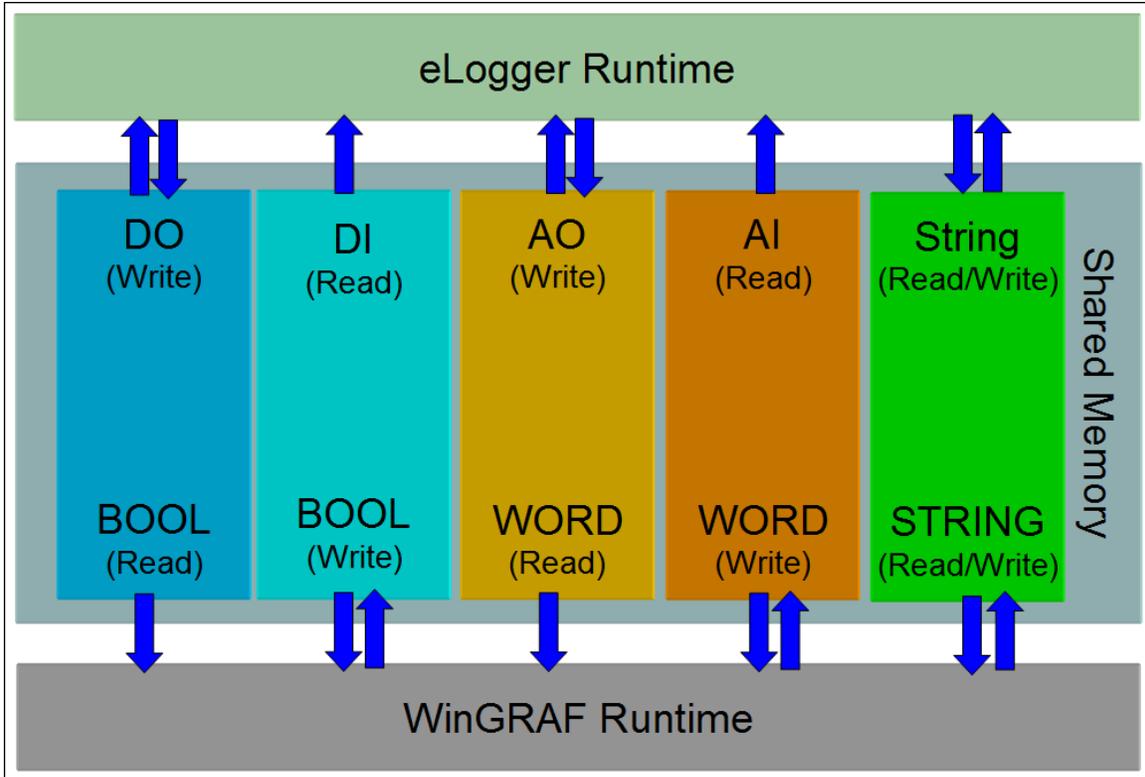


Figure 4: Shared memory segments

Table 1 shows the default data type and access right of the different memory segments for Win-GRAF and eLogger workbench.

Memory Type	Win-GRAF		eLogger	
	Access	Default Data Type	Access	Default Data Type
DO	Read	BOOL	Write	Unsigned Char (TRUE/FALSE)
AO		UINT		32 bit Unsigned Long
String		STRING		Char String
DI	Write	BOOL	Read	Unsigned Char (TRUE/FALSE)
AI		UINT		32 bit Unsigned Long
String		STRING		Char String

Table 1: Default memory data types

Table 2 shows the memory access right from the perspective of the eLogger.

eLogger Graphic Object	Linked Memory Segment	Description
AIO: 	AIO	Only read: <ul style="list-style-type: none"> HMI object only displays data stored in the segment AIO The memory content AIO can not be changed via

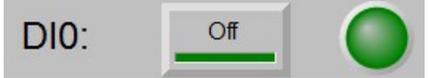
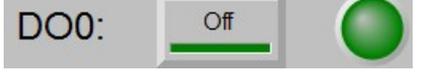
	AO0	the HMI. Read and write: <ul style="list-style-type: none"> • The user can changed the data of the memory by directly editing a new value via the HMI object • Read the data stored in the memory
	DIO	Only read: <ul style="list-style-type: none"> • Data stored in the DI memory segment can not be changed by the HMI operator. For example clicking the button will have no effect. • Displays data currently stored in the memory DI
	DO0	Read and write: <ul style="list-style-type: none"> • HMI user can directly change the DO memory • HMI object displays the content of the DO memory

Table 2: eLogger memory read/write access

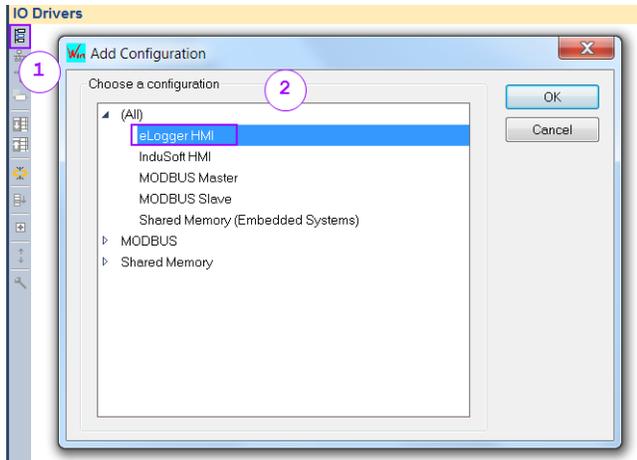
2.3 Win-GRAF Memory Mapping

This section describes how to create a programming interface for the PLC application program to access the eLogger graphical objects. The Win-GRAF workbench provides a graphic plugin which allows the programmer to map the local PLC data to the shared memory.

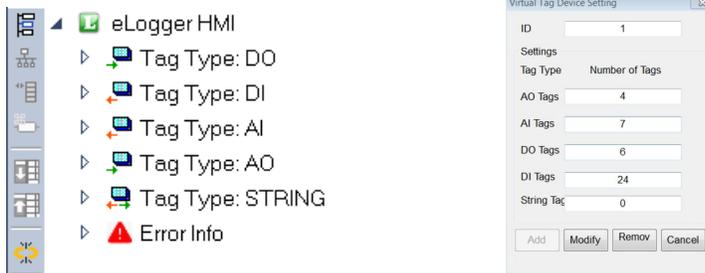
The mapped memory segment determines the data flow direction: A variable mapped to the read memory segment will cause the Win-GRAF runtime to update the variable with the current shared memory data in each task cycle, while mapped to the write segment will result in a shared memory update with the current variable data.

The following steps describes the procedure for configuring the eLogger plugin and variable mapping:

- Step 1:** Open the Fieldbus configuration by clicking the fieldbus  button of the toolbar or the fieldbus node in the workspace.
- Step 2:** Click 'Insert Configuration'  button of the toolbar on the left of the 'IO Drivers' editor and select 'eLogger HMI' from the 'Add Configuration' dialog.



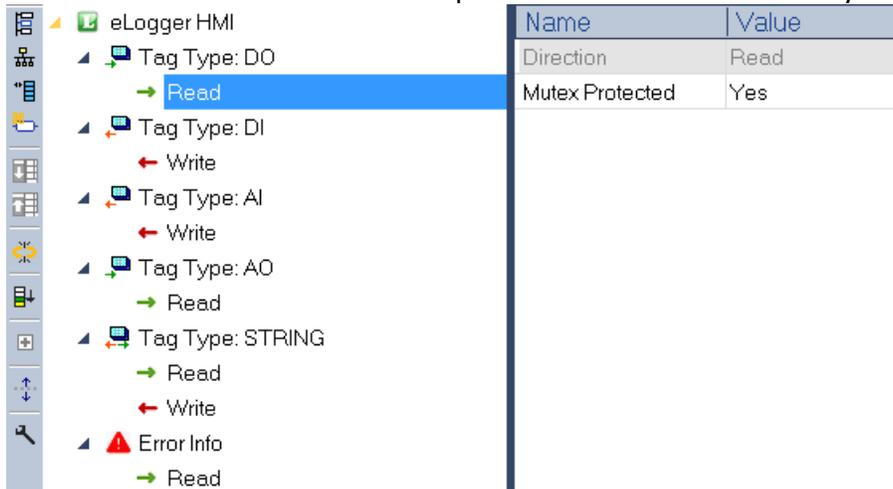
A tree with six different nodes will be generated. Each node represents a shared memory segment. The last node is reserved for error handling. It indicates whether an error occurred during data exchange of the Win-GRAF and eLogger runtime. The left image shows the workbench tree view and the right image the corresponding eLogger developer configuration interface.



Expand the tree by clicking on one of the node:

The 'Read'/'Write' shows the direction of the data flow.

- 'Read': sets the direction from the shared memory to the local variable
- 'Write': the PLC variable will be update with the shared memory data

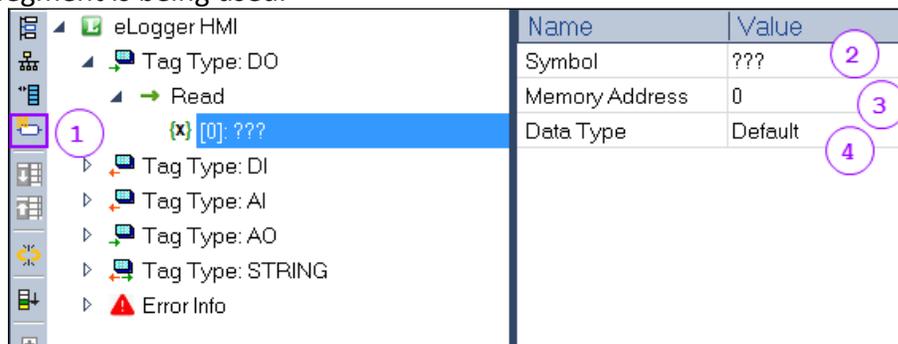


Step 3: Set the memory synchronization:

The 'Mutex Protect' properties, if enabled, synchronizes the memory access by ensuring that the Win-GRAF runtime first writes all the data from the PLC application to the shared memory before the eLogger runtime can access it. If memory synchronization is not required disable the function to increase the communication speed and decrease the cycle time.

Step 4: Variable mapping: Add variables to the nodes to represent the content of the shared memory:

1. Double click the 'Insert Variable'  button.
2. Map a variable to the shared memory by double click the 'Symbol' box ('???') to open the variable editor. Either select an existing variable or create a new variable.
3. Select the offset position of the target shared memory segment. Each segment consist of an array of the data type it represents. The offset 0 indicates the first position of the segment and 1 the second position, etc. The offset length is fixed for each segment, e.g. 1 byte (BOOL) for 'DI' and 'DO'; and two bytes (UINT) for 'AI' and 'AO' data segment.
4. Set the storage length. This setting is necessary if the data length of the variable to be mapped is longer than the storage unit size of the selected memory section. For example to store a data type of four bytes lengths to a 'AI' memory section two UINT memory spaces have to be used. It is therefore necessary to select either UDINT, DINT or REAL. The 'Default' option indicates that one storage unit of the corresponding memory segment is being used.



It is possible to map more than one variable at a time. The following example shows the multi variable mapping procedure.

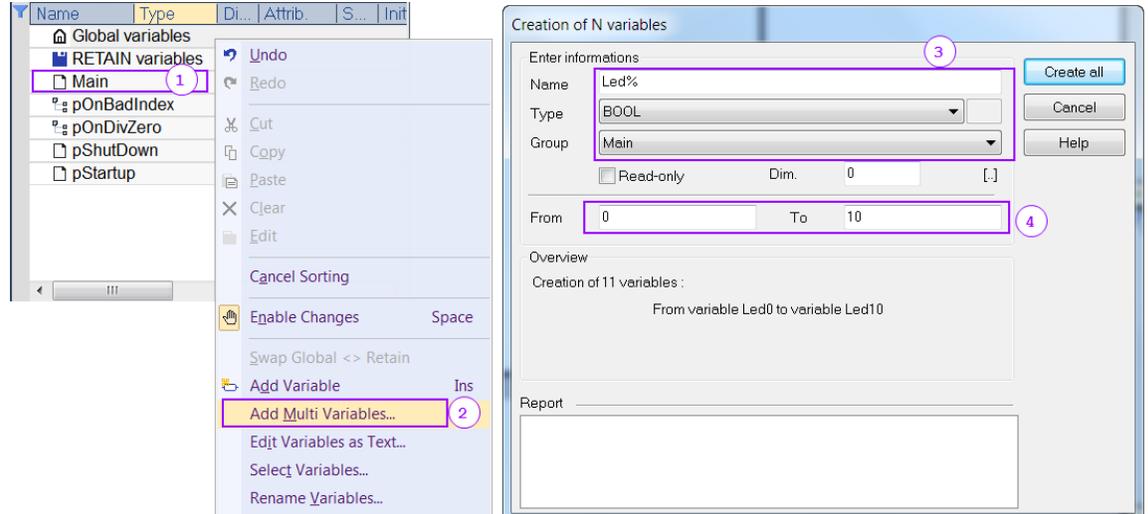
Example:

Declare 10 variables of type BOOL with names Led0, Led1, ..., Led10.

1. Right click the program in the variable editor where to declare the new variables.
2. Select 'Add Mulit variables...'
3. Enter the variable name ('Led'). The symbol '%' next to the name indicates that a number will be attached to the name. The number

will increment by one for each new declaration.

4. Enter the start and end value for the '%' symbol. In this example the number attached to the variable name will start from 0 and end with 10 (Led0, Led1, Led2,...,Led10).



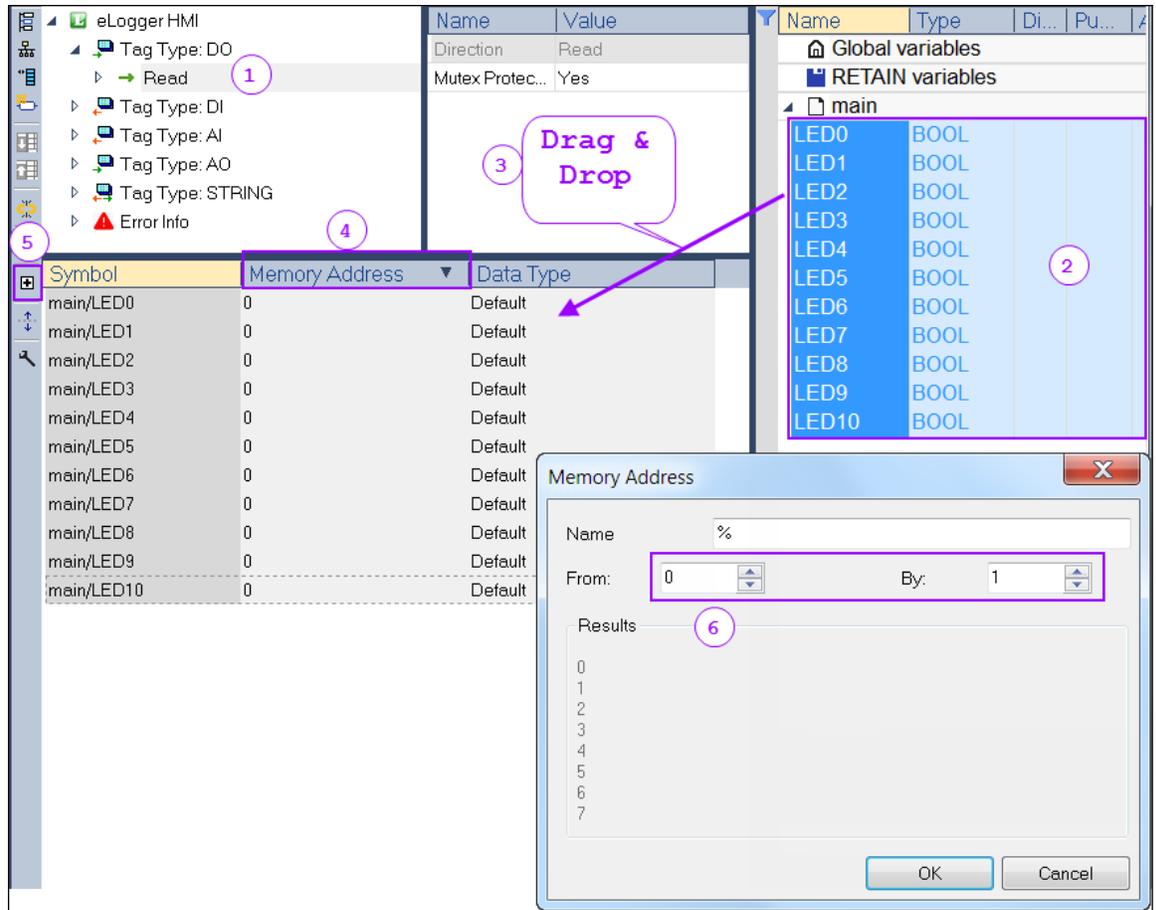
Map the declared variables via drag and drop to the shared memory:

1. Select the memory segment
2. In the variable editor select all the variables to map
3. Drag the variables to the bottom window of the 'IO Drivers'

The offset of the dragged variables are all set to zero which is incorrect.

The following steps describes how to quickly renumber the offsets:

4. Click on the 'Memory Address' header to select the column. Make sure the arrow in the header shows downwards.
5. Click the 'Iterate Property' button to open the offset editor.
6. Set the start and increment value for the offset and confirm the setting. All the offset will now be renumbered.



Tag type nodes can be left empty if the memory segment is not being accessed by the PLC program. In this example the DI, AI, AO, STRING are not being used therefore no variable mapping is required and the nodes are empty.

Step 5: Assign a variable to the 'Error Info' node. Only one variable node can be mapped to the 'Error Info' node. You can leave this node empty if the error information is not required. Table 3 lists the error code with its descriptions.

Step 6: The shared memory can now be directly accessed via the mapped variable within the PLC program.

Error Info Code	Description
0	No error
100	Register type error: The memory or tag type is not being supported
101	Data range error: Attempt to access a memory outside the shared memory area. Check whether the variable offset settings are too large and exceeds the supported range.
102	Timeout error: The accessing the memory was blocked by a mutex. If

Error Info Code	Description
	the access is blocked for more than 200 milliseconds it will be aborted and an timeout error will be generated
103	Alignment error: To read/write data to the AO/AI the variable size needs to be a multiple of two bytes
104	Shared memory initialization error: Failed to initialize the shared memory. Make sure that either the eLogger or Win-GRAF runtime has been run as administrator

Table 3: Communication errors

Important:

In order to initialize the shared memory correctly it is necessary that the first program which is going to access it is being run as administrator. Therefore one of the following three program has to be started as administrator before data can be exchanged: eLogger runtime, Win-GRAF runtime, eLoggerMonitor.

2.4 eLogger Memory Mapping

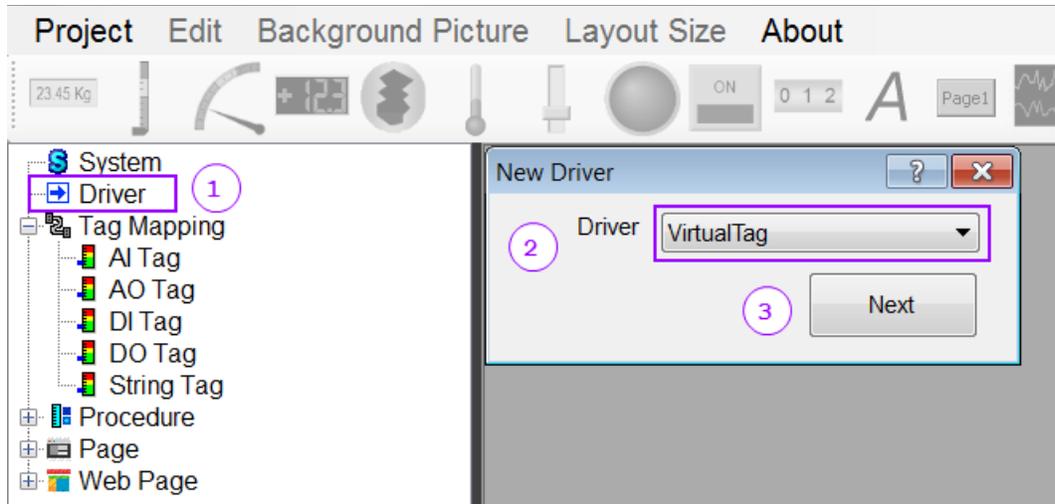
This section shows how to configure the eLogger in order to access the PLC variables. The eLogger communicates with the shared memory via virtual tags. It is therefore necessary to first install the virtual driver and create virtual tags before linking the graphic to the shared memory. Five types of the virtual tags are available: DI, DO, AI, AO, String (Figure 4). Each type accesses a different shared memory segment. Table 1 list all the types and the corresponding PLC data types,

The following steps describes the procedure for installing the virtual driver and setting up the virtual tags:

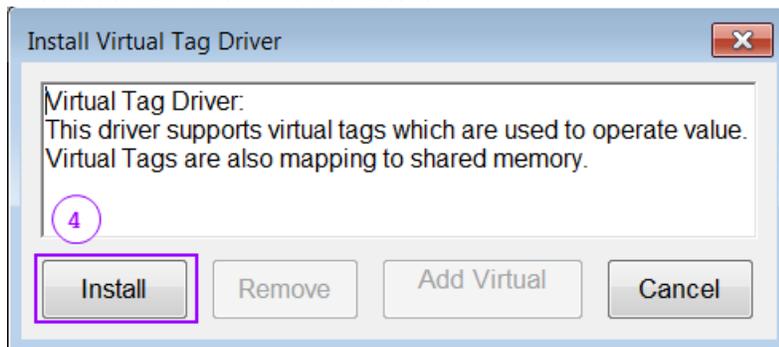
Step 1: Create a new project: *'Project/New'*

Step 2: Install the virtual tag driver:

1. Click *'Driver'* in the tree view
2. Select *'VirtualTag'* from the drop box.
3. Click *'Next'*.

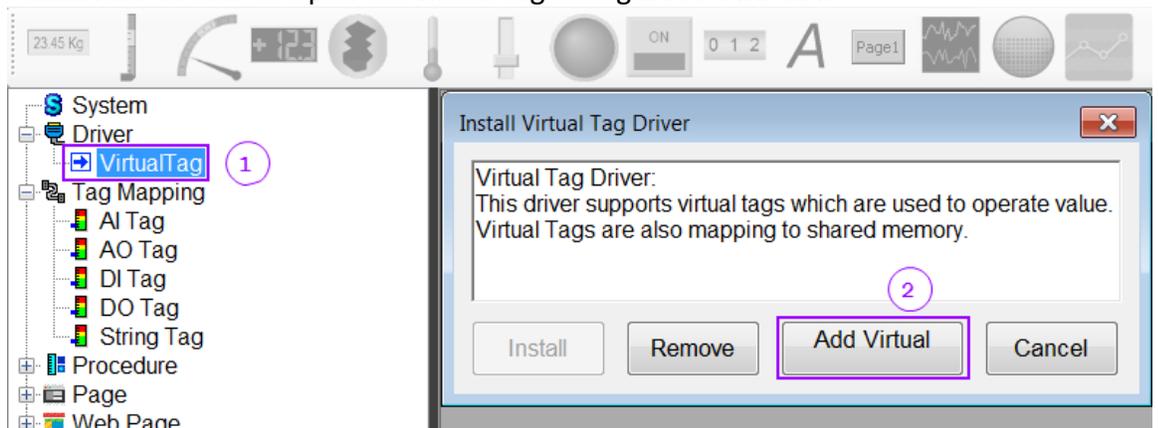


4. Click 'Install' to install the driver



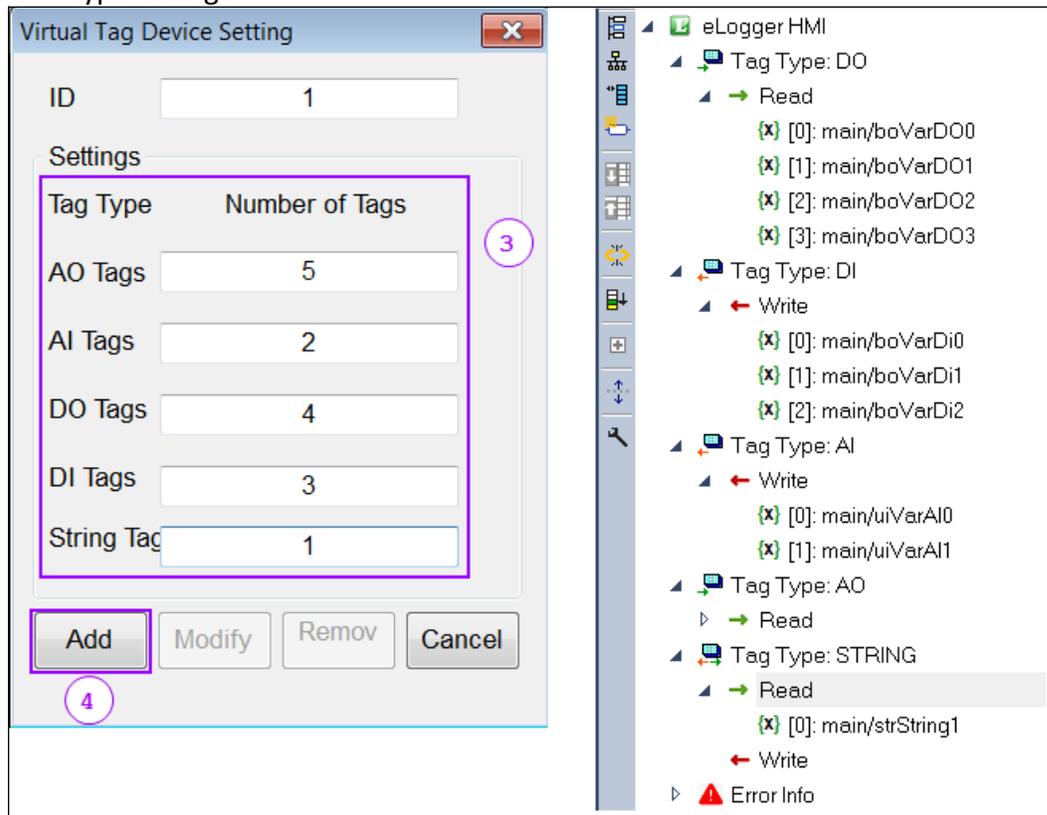
Step 3: Create virtual tags

1. Click the 'VirtualTag' node in the tree view which has been created in the previous step.
2. Click 'Add Virtual' to open the virtual tag configuration window

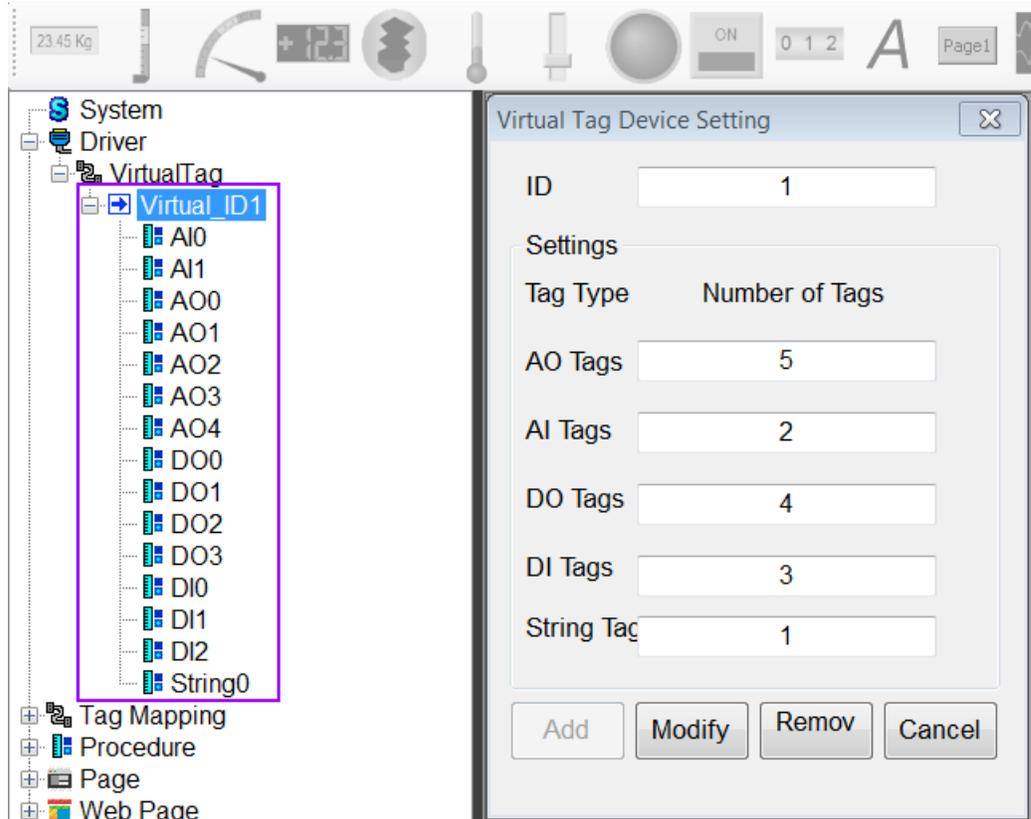


3. Create virtual tags. By default the tag quantity for each type is zero. The tag type and quantity depends on the memory segment and the number of position to access in the shared memory. For example:

- i) PLC declares three 'Read' variable of type 'BOOL': Create three virtual tags of type 'DI'.
- ii) PLC declares four 'Write' variable of type 'BOOL': Create four virtual tags of type 'DO'.
- iii) PLC declares two 'Read' variable of type 'UINT': Create two virtual tags of type 'AI'.
- iv) PLC declares five 'Write' variable of type 'UINT': Create five virtual tags of type 'AI'.
- v) PLC declares one variable of type 'STRING': Create one virtual tags of type 'String'.



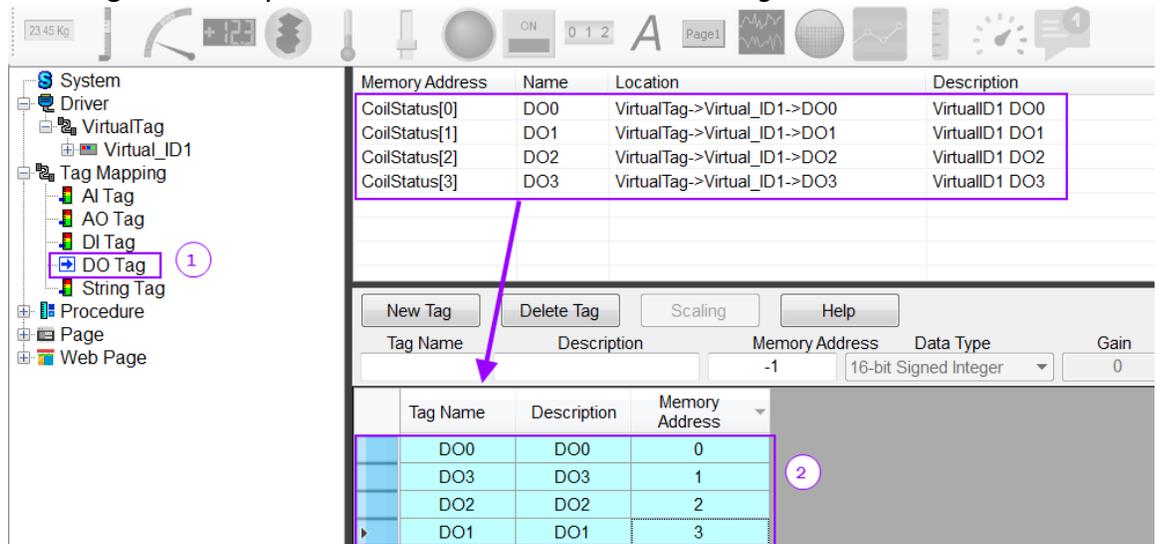
4. Click 'Add' to create the virtual tags. The new tags are added to the tree view. You can at any time add or remove the virtual tags by entering new values in the 'Virtual Tag Device Setting' dialog and clicking 'Modify'.



Step 4: Map the tag to a address of the corresponding shared memory segment.

Example 1: Map the 'DO' virtual tag to the shared memory.

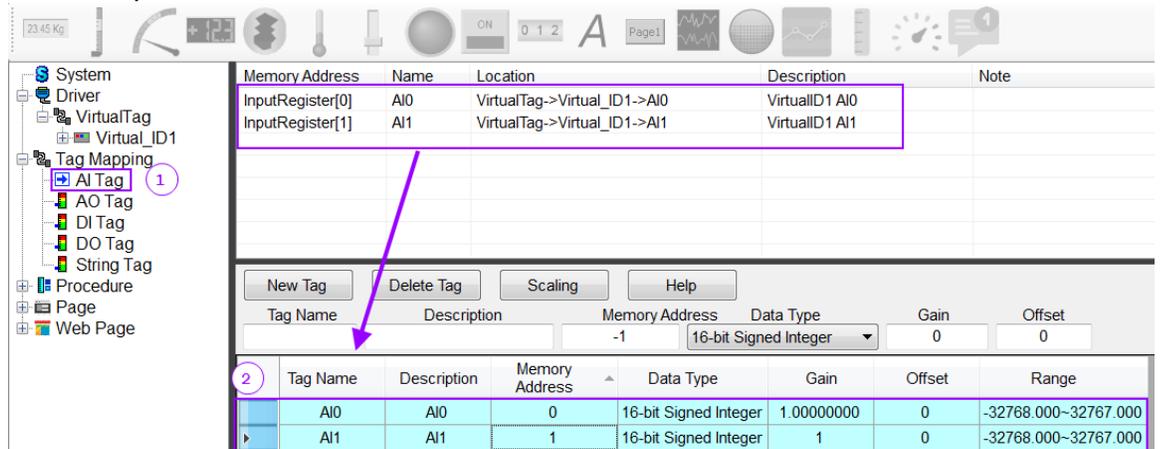
1. Expand the '*Tag Mapping*' node of the tree view menu and click '*DO Tag*' sub node to display the memory mapping window.
2. Assign a memory offset address for each virtual '*DO tag*'.



Example 2: Map the 'AI' virtual tag to the shared memory.

1. Click the '*AI Tag*' sub node to display its memory mapping window.

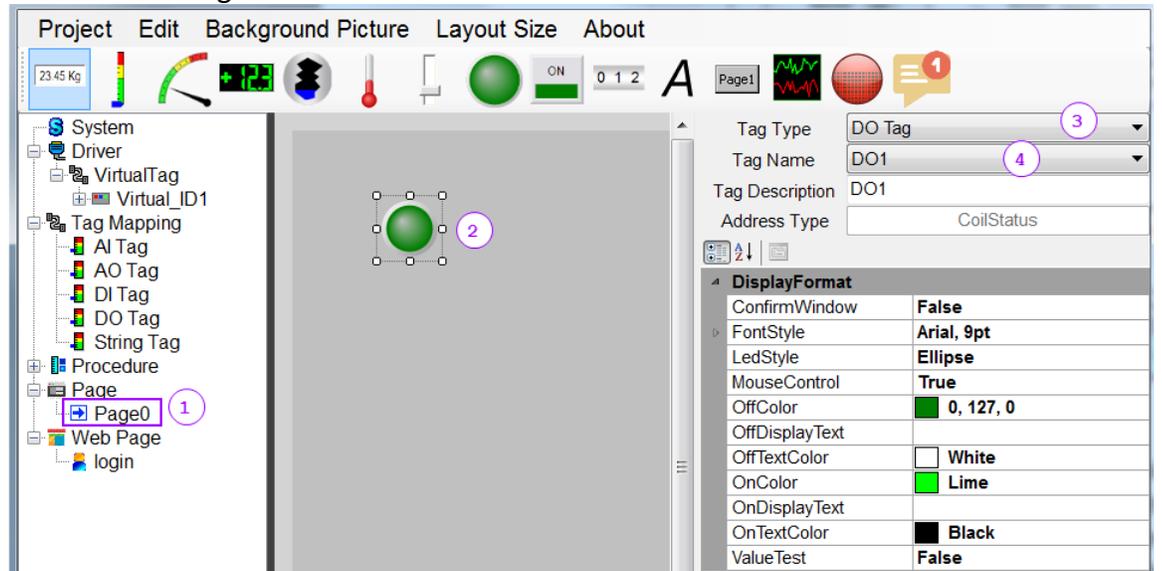
2. Add the tag to the mapping area and set the memory address, data type, etc. for each entry. Consult the eLogger user manual for a detailed description.



Step 5: Link the virtual tag to a graphic object. The tag can be linked to a HMI or/and web HMI object. In the example below it is shown how to link a tag to a HMI objects. The procedure for creating a web HMI is very similar. Consult the eLogger user manual for more information.

Example 1: Map the 'DO' virtual tag to a LED object.

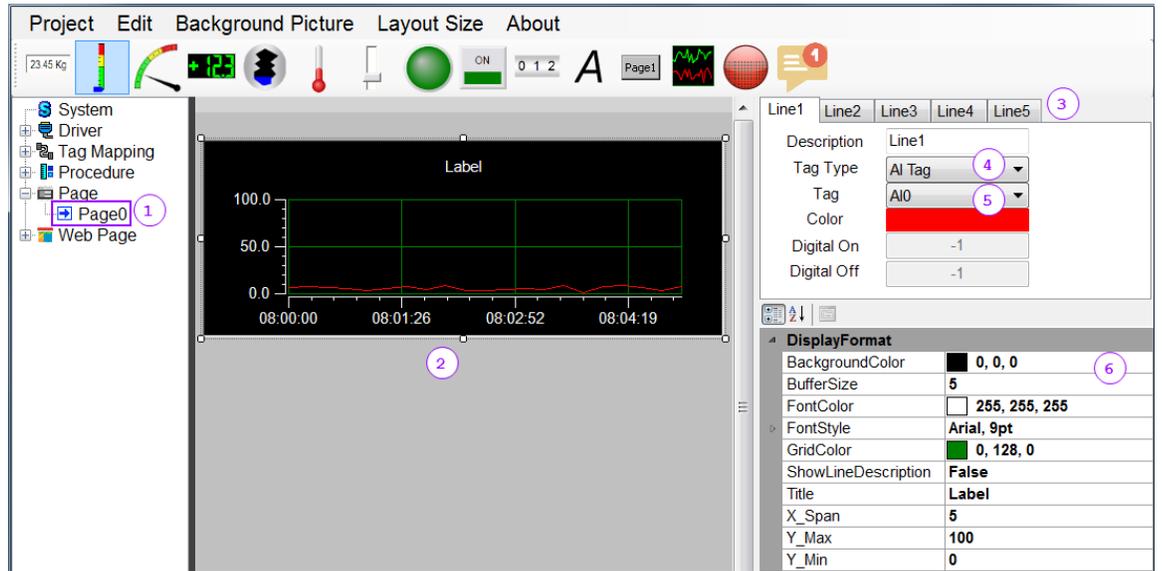
1. Expand the 'Page' node of the tree view menu and click the 'Page' sub node to display the graphical design area.
2. Add a LED object to the design area.
3. Select the tag type: 'DO Tag'.
4. Select the tag name: 'DO1'.



Example 2: Map a 'AI' virtual tag to a plot object.

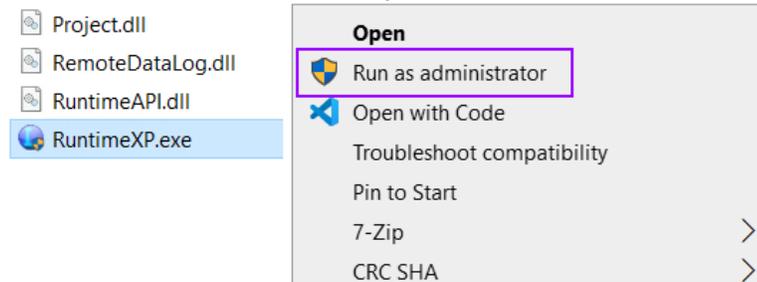
1. Expand the 'Page' node of the tree view menu and click the 'Page' sub node to display the graphical design area.
2. Add a 'Plot' object to the design area.
3. Pick a 'Line' tab and set the line properties.
4. Select the tag type: 'AI Tag'.
5. Select the tag name: 'AI0'.
6. Set the display format for the plot, e.g. back ground color, title, font color, etc.

To add multiple line to the plot repeat procedure 3 to 5 for a different 'Line' tab selection.



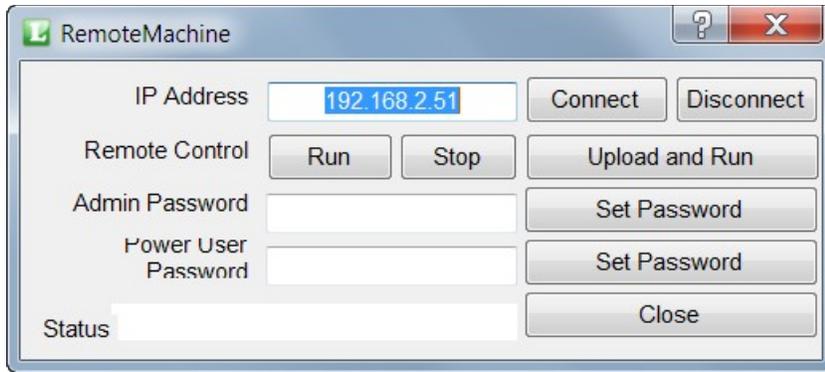
Step 6: Start the eLogger runtime.

ATTENTION: Run the runtime as administrator otherwise the shared memory will not be initialized correctly.



Step 7: Download the project to the runtime:

1. Click 'Project/Remote Machine'
2. Set the IP address of the local or remote eLogger runtime.
3. Connect to the runtime by clicking 'Connect'
4. Download and run the HMI project by clicking 'Upload and Run'



2.5 Shared Memory Utility

A utility 'eLoggerMonitor' is provided which allows you to read and change the content of the five shared memory sections at any time. The utility assists you in testing either the PLC application or the eLogger HMI during the development by monitoring the data exchange between both runtimes. With this utility for example you can directly manipulate the PLC application without running the eLogger runtime by modifying the contents of the shared memory to which the PLC variable is mapped. For each memory segment the displaying data format can be set.

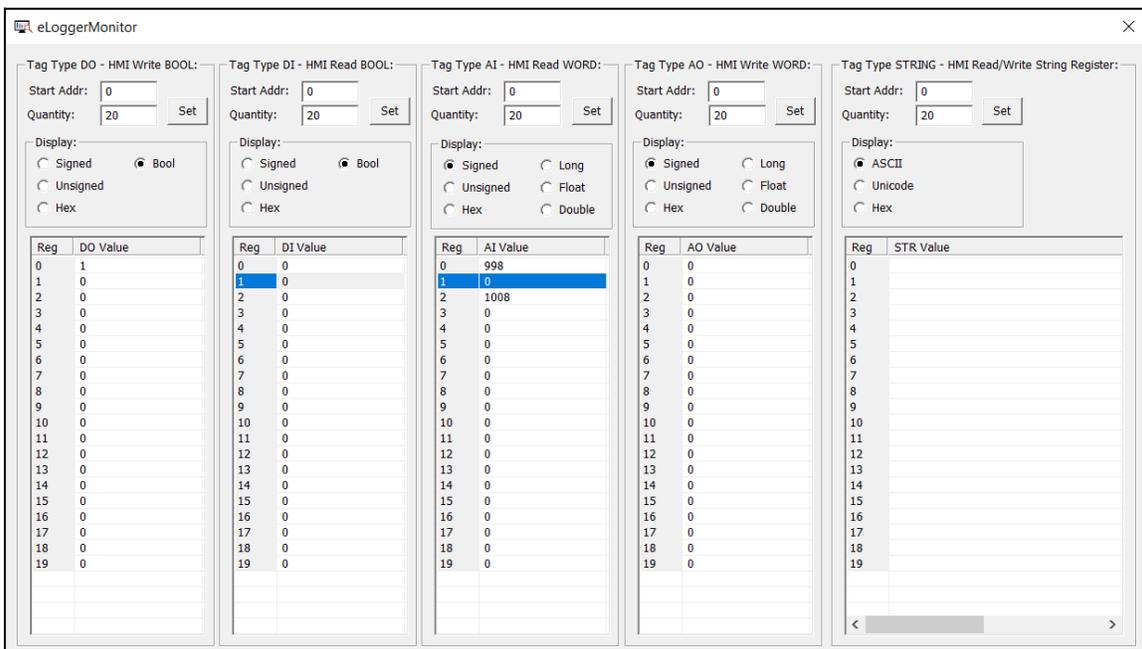


Figure 5: eLogger Utility

Remember to start this utility as administrator if the eLogger or Win-GRAF runtime has not been started prior otherwise the shared memory will not be initialized correctly.

2.6 Demo Program

Demo programs which shows the PLC and eLogger data mapping and exchanged can be found in the following directory:

C:\Users\Public\Documents\Win-GRAF Workbench\Win-GRAF Wb xx.x\Projects\eLogger



Figure 6: HMI for motion control

3 InduSoft

3.1 Introduction

InduSoft Web Studio is a powerful collection of automation tools that provides all the automation building blocks to develop HMIs, SCADA systems and embedded instrumentation solutions. InduSoft is being sold by ICPDAS.



Figure 7: Graphical user interfaces created by eLogger

InduSoft supports the following functions:

- Trend creation
- Recipes
- Alarm management
- Multiple test
- Modbus TCP
- OPC UA (server & client)

3.2 Software Installation

3.2.1 InduSoft

First you need to install the InduSoft Web Studio and copy the below mentioned files to their target directories.

1. Copy the following Win-GRAF driver files:
 - WGRAF.dll
 - WGRAF.ini
 - WGRAF.msgto the to the InduSoft subdirectory:
 - *C:\Program Files (x86)\InduSoft Web Studio vX.X\Drv*whereby 'vX.X' represents the InduSoft version
2. Copy the '*WGrafSharedMem.dll*' file to the following directory:
 - Windows 64 bit: '*C:\Windows\SysWOW64*'
 - Windows 32 bit: '*C:\Windows\System32*'

3.2.2 Win-GRAF

The Win-GRAF runtime requires the following dynamic libraries to be able to communicate with the InduSoft runtime:

- *ddkc_indusoft.dll*
- *WGrafSharedMem.dll*

By default both libraries are automatically installed by runtime setup program. These files can be found in the execution directory of the Win-GRAF runtime.

3.3 Win-GRAF - InduSoft Communication

The schematic (Figure 8) shows the basic communication structure of the Win-GRAF and InduSoft runtime. Both runtimes exchange data via a shared memory.

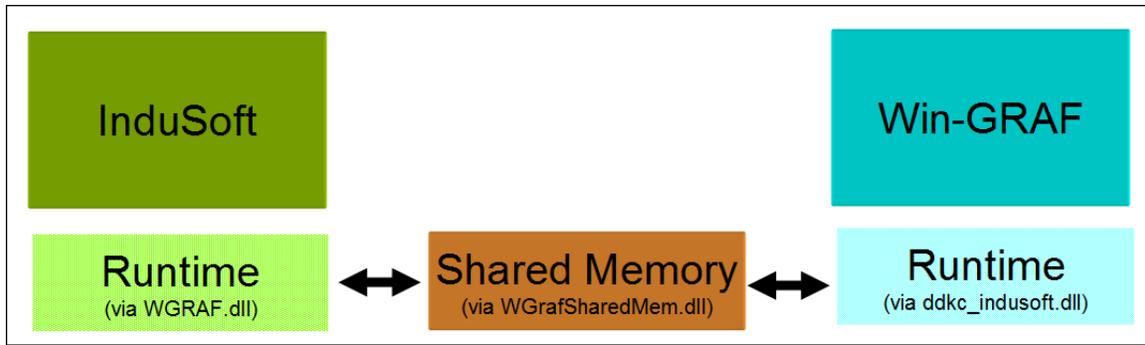


Figure 8: InduSoft - Win-GRAF communication structure

The following PLC data types can be directly exchanged (Table 4):

T5 Runtime Data Type	Write	Read
BOOL	●	●
INT	●	●
UINT	●	●
DINT	●	●
UDINT	●	●
REAL	●	●
LREAL	●	●
TIME	●	●
STRING	●	●

Table 4: Data Types supported:

For each of these PLC data type a separate shared memory is being created (Figure 9). Both runtimes have read and write access to the memory. It is therefore important to define for each memory an area range which restricts the write access of one of the runtime.

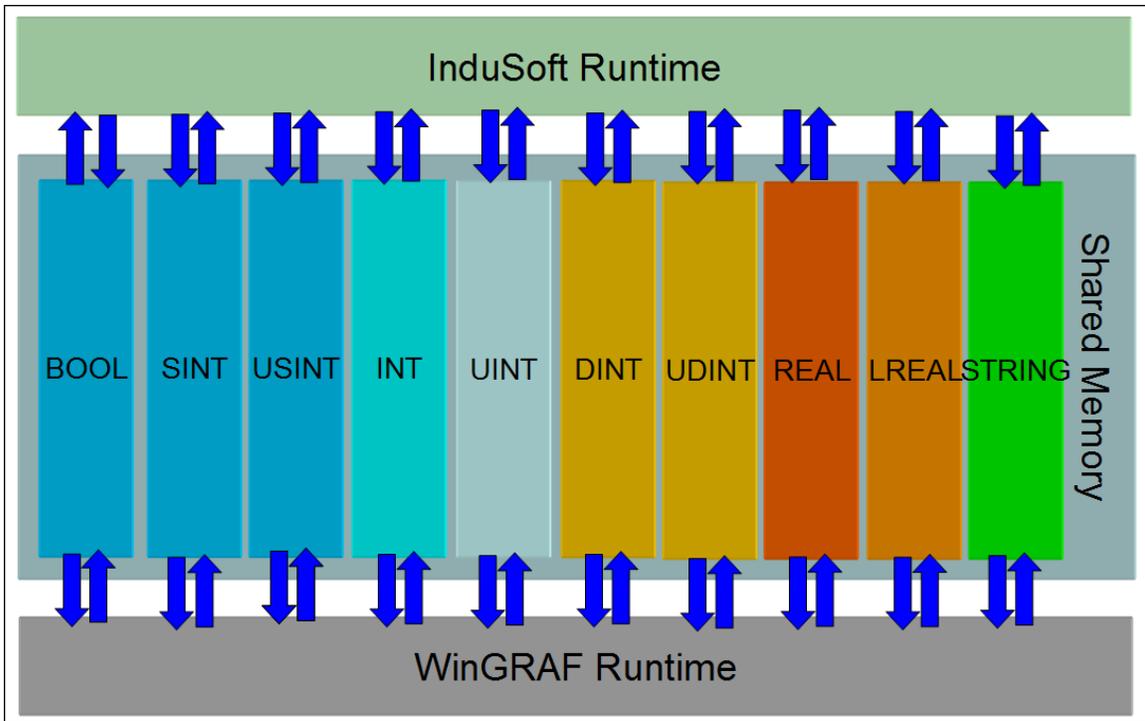


Figure 9: PLC data type memory

3.4 Variable - Tag Mapping

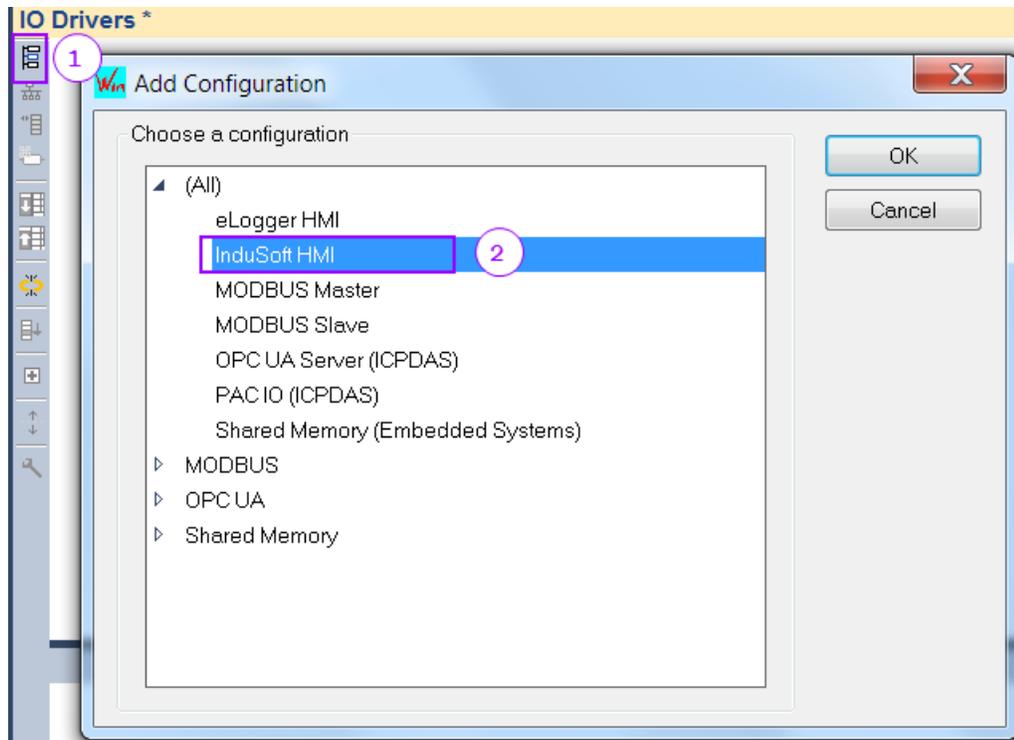
Mapping of the PLC variable to the InduSoft tag or vice versa will be shown in this chapter. The first section describes the configuration procedure of the Win-GRAF workbench to add the InduSoft plugin and set up the communication interface. The second part described the steps needed to be taken for configuring the InduSoft Studio for accessing the PLC data.

3.4.1 Win-GRAF Configuration

Mapping PLC variables to the InduSoft tags has to be done via the InduSoft plugin provided for the Win-GRAF workbench. This plugin is a graphical interface and allows the user via drag and drop to select the PLC variables to be accessed by the InduSoft runtime. The access right set for each variable determines which side, InduSoft or Win-GRAF, can alter the variable content or just read it. The variable content will be update in each cycle of both runtimes.

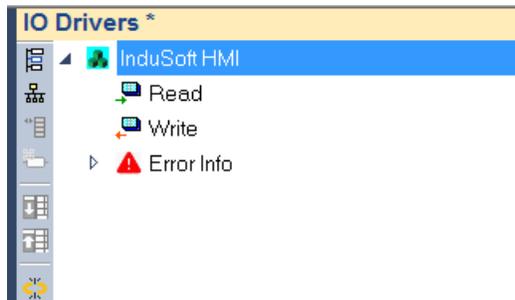
The mapping procedure for the InduSoft plugin will be described below:

- Step 1:** Open the Fieldbus configuration by clicking the fieldbus  button of the toolbar or the fieldbus node in the workspace.
- Step 2:** Click 'Insert Configuration'  button of the toolbar on the left of the 'IO Drivers' editor and select 'InduSoft HMI' from the 'Add Configuration' dialog.



A tree with three different nodes will be displayed:

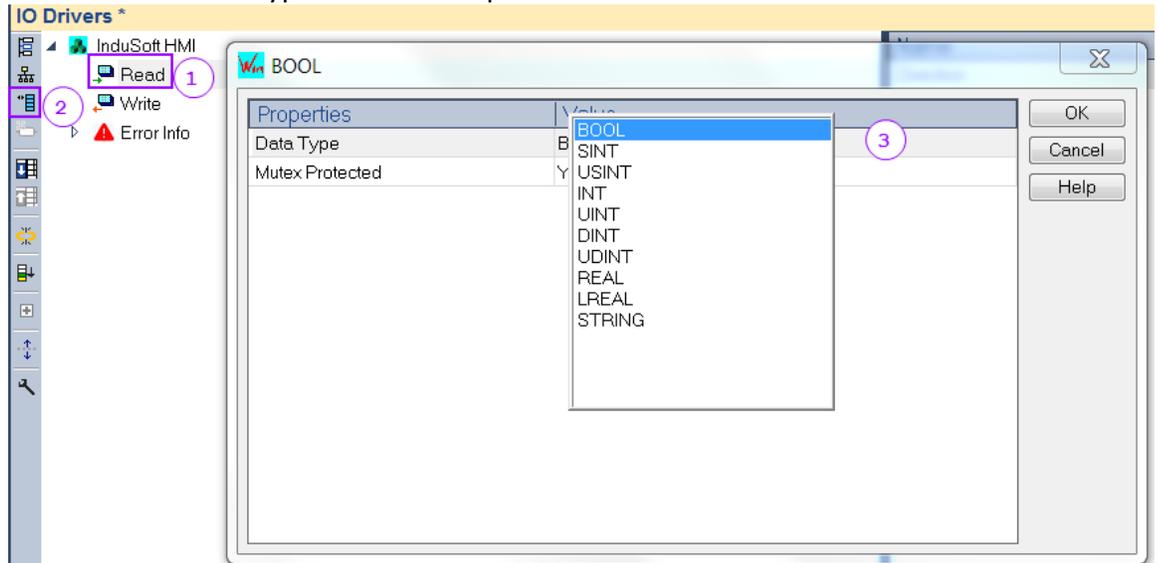
- Read node: Variable which should only be read by the PLC application should be added to this node. In each cycle the variables listed in the node are updated with the data of the mapped InduSoft tag.
- Write node: Variable assigned to the 'Write' node can be written to by the PLC application. The InduSoft runtime has only read access these variables.
- Error Info node: Only one DINT variable can be added to this node. It displays any initialization or communication errors between InduSoft and Win-GRAF.



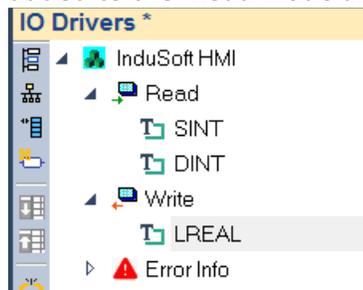
Step 3: Select the data type categories to map. It is necessary to first set the data type of the variable to map. The data type format determines the internal memory sized reserved for this variable (Figure 9).

1. Select either the 'Read' or 'Write' node. It determines whether the variable data is either being sent to or received from the InduSoft runtime

2. Insert a new data type node by clicking on the 'Insert Slave/Data Block'
3. Select the data type from the drop list



For example in the node configuration below two data type nodes has been added to the 'Read' node and one to the 'Write'.



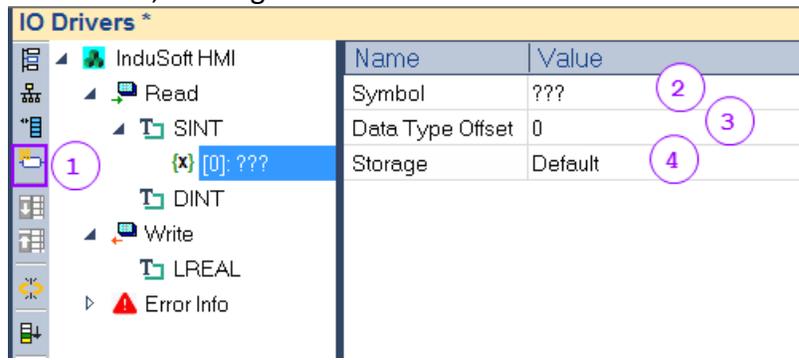
Step 4: Set the memory synchronization ('*Mutex Protected*'): Select whether all the variables listed under the same data type node should be protected by mutex lock during the data exchanged between Win-GRAF and InduSoft. The mutex ensures data consistency by synchronizing the access of the two runtimes to the mapped variable list. It makes sure that all the variables listed under the same node are sent together in a block to the target runtime. The mutex will prevent the target runtime from accessing any variables in the block until the entire block has been received. If data synchronization is not required disable the function to increase the communication speed and decrease the cycle time.

Step 5: Variable mapping: Add variables to the data type nodes to be exchanged between InduSoft and Win-GRAF:

1. Double click the '*Insert Variable*' button.
2. Map a variable to the shared memory by double click the '*Symbol*' box

('???') to open the variable editor. Either select an existing variable or create a new variable.

3. Select the offset position of the selected shared memory type. Each type consist of an array of the data type it represents. The offset 0 indicates the first position of the memory and 1 the second position, etc. The element size is different for each memory type, e.g. the element size of BOOL is one byte and two byte for UINT.
4. Set the storage length. This setting is necessary if the data size of the variable to be mapped is greater than the element size of the selected memory section. For example to store a data type of four bytes lengths to a UINT memory type two elements are required. The 'Default' option indicates that element size of the selected data type, indicated by the parent node, is being used.



More than one variable can be mapped at a time. The following example shows a array mapping procedure.

Example:

Declare an five element array of type BOOL with names 'readBool' by using the variable editor on the right hand side of the workbench.

Name	Type	Dim.	Attrib.	S...	Init value
Global variables					
RETAIN variables					
Main					
readBool	BOOL	[0..4]		<input type="checkbox"/>	5(false)

Drag the array via drag and drop to the data type node:

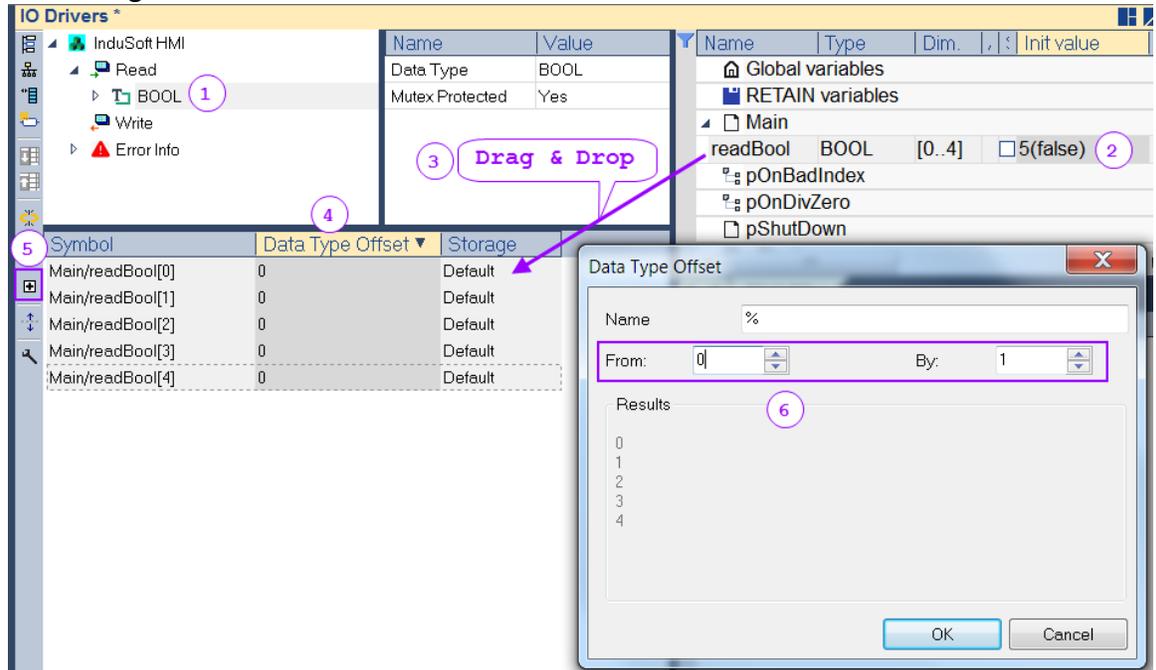
1. Select the data type node
2. In the variable editor select all the array to map
3. Drag the variables to the bottom window of the 'IO Drivers'

The offset of the dragged variables are all set to zero which is incorrect.

The following steps describes how to quickly renumber the offsets:

4. Click on the 'Memory Address' header to select the column. Make

- sure the arrow in the header shows downwards.
- Click the '*Iterate Property*' button to open the offset editor.
 - Set the start and increment value for the offset and confirm the setting. All the offset will now be renumbered.



Nodes of the tree can be left empty if no mapping is required. For example the 'Write' node can be left empty if the PLC application does not need to send data to the InduSoft runtime.

- Step 6:** Assign a variable to store the communication status. Only one variable node can be mapped to the '*Error Info*' node. You can leave this node empty if the error information is not required. Table 6 lists the error code with its descriptions.

Error Info Code	Description
0	No error
-1	Shared memory error (not in use, reserved).
-2	Internal system error: Shared memory name is invalid.
-3	Internal system error: Shared memory name not found.
-4	Internal system error: Shared memory not linked.
-5	Blocking error: Mutex blocking error encountered.
-6	Memory outside the shared memory is being accessed.
-7	Internal system warning: Shared memory is being accessed with a data type of zero size.
-8	Internal system error: Shared memory could not be linked.

Error Info Code	Description
-9	Internal system error: Failed to create a shared memory.
-10	Internal system warning: Link to shared memory already exists.
-11	Internal system warning: Shared memory already exists.
-12	Internal system error: Shared memory mapping failed.

Table 5: Communication errors

Important:

1. Make sure that the same variable in the 'Read' and 'Write' node do not access the offset position of the same memory type otherwise the data will be overwritten (Figure 10).

In Figure 10 the BOOL memory is being accessed by 'Read' and 'Write' variables read at the same offset. This may cause BOOL data received from the InduSoft runtime to be overwritten by the Workbench runtime. Set the 'Read' and 'Write' variables to different offset to prevent this error from occurring.

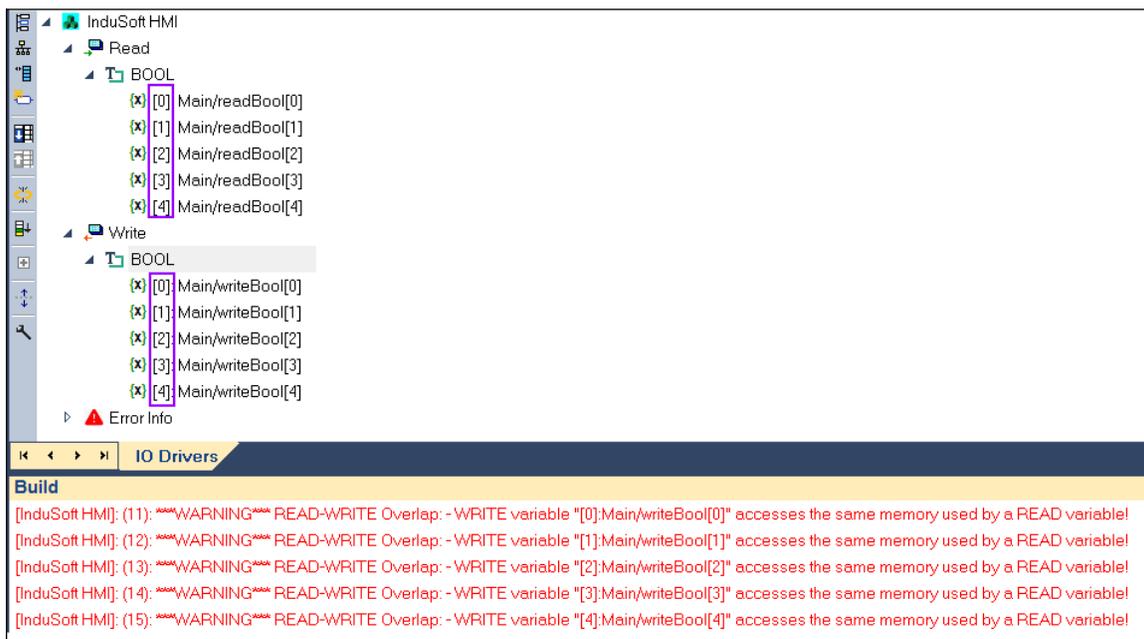


Figure 10: Read/write overlap

With the above described steps the configuration of the variable mapping on the PLC side is complete. Within the PLC program the mapped variable can be accessed like any other variables. In the next step the InduSoft tag mapping has to be set up as described in the next chapter.

3.4.2 InduSoft Configuration

The Win-GRAF driver installation and the driver workpage setup for the InduSoft Studio are described.

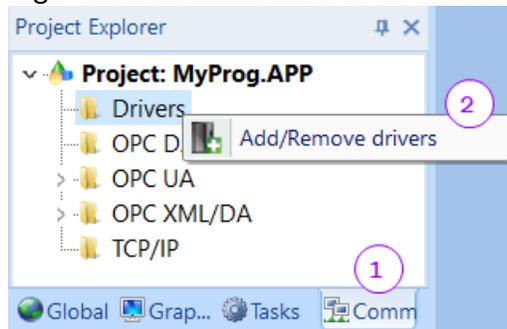
3.4.2.1 Install and Configure the Communication Driver

After a new project has been created the Win-GRAF driver has to be added to the project.

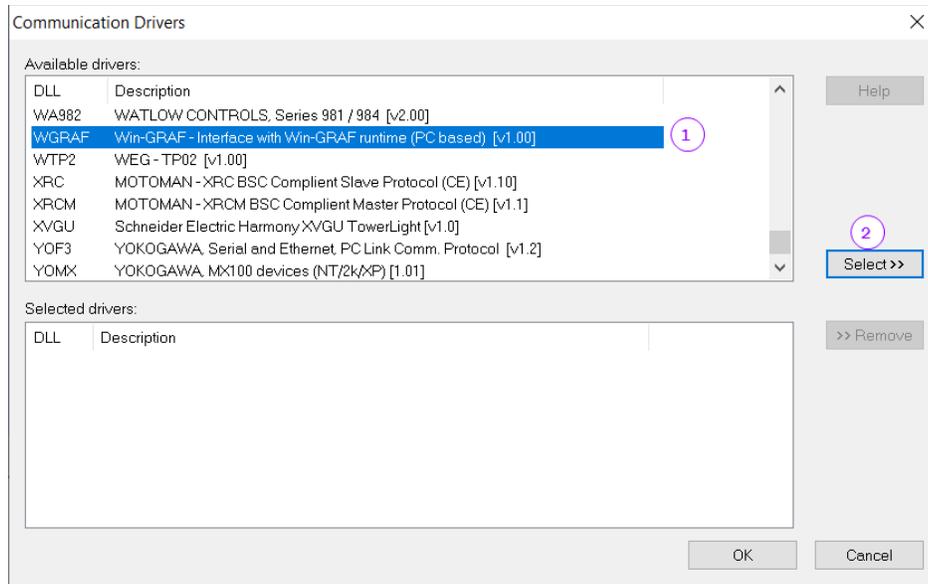
The following procedure shows how to include the driver to the project:

Step 1: Select the 'Comm' tab at the bottom of the 'Project Explorer'

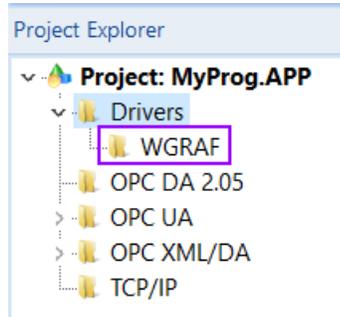
Step 2: Right click the 'Driver' node and select 'Add/Remove drivers'



Step 3: Select the 'WGRAF' driver and click the 'Select>>' button. Confirm the selection by clicking 'OK'.



The added 'WGRAF' driver is shown in the 'Driver' directory



Note:

If the 'WGRAF' driver is not shown in the 'Communication Drivers' list then make sure that the 'WGRAF.dll', 'WGRAF.ini' and 'WGRAF.msg' files are in the 'Drv' directory (see 3.2.1).

3.4.3 Tag Mapping

This section describes how to declare a variable tags and map them to the Win-GRAF memory using InduSoft.

The following steps describes the procedure for installing the virtual driver and setting up the virtual tags:

Step 1: Declare tags.

1. Select the 'Global' tab in the 'Project Explorer'
2. Double click the 'Datasheet View' in the 'Project Tags' subdirectory
3. Declare tag arrays of different data type as shown in the figure below.

Tags for reading and writing PLC data are declared. The prefix "read" indicates that the tag is constantly updated with the mapped PLC data; the prefix "write" on the other hand forces the InduSoft runtime to update the mapped Win-GRAF variable with its current value.

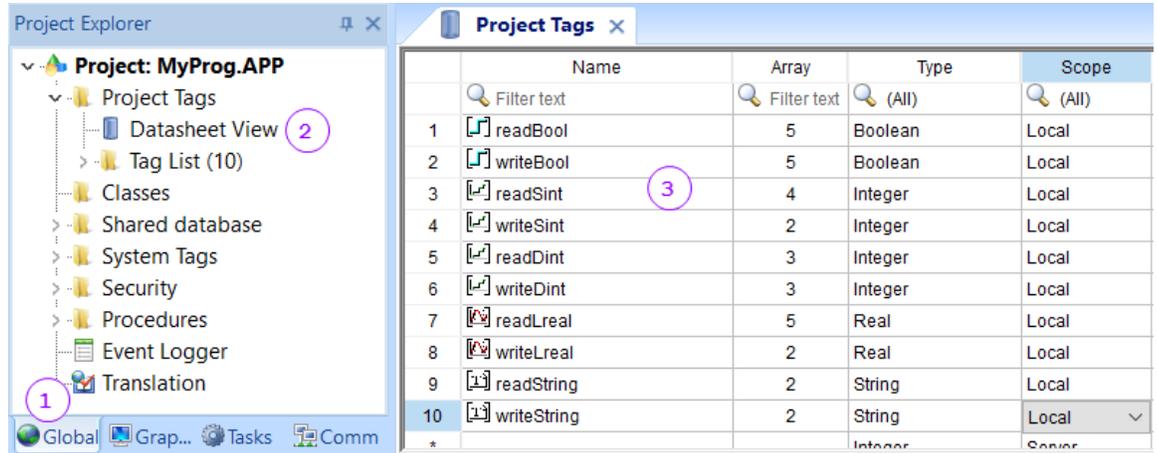
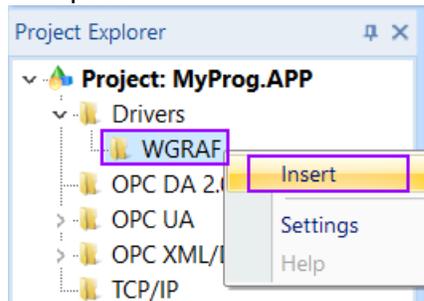


Figure 11: Tag arrays of different data types

Step 2: Map the tags to the PLC memory:

Map the 'MyBool' array to the BOOL memory:

1. Add a new mapping sheet by right clicking the 'WGRAF' driver in the 'Comm' tab and clicking 'Insert'. A new sheet it being added to the workspace.



2. The 'WGRAF' driver supports all the fields shown on the worksheet (Figure 12), except 'Station'. The InduSoft 'Studio User Guid' describes the main purpose of each field. In the following only the four basic field entries are described required for the communication between Win-GRAF and InduSoft.

The screenshot shows a configuration window titled 'WGRAF001.DRV'. It contains several input fields and checkboxes. The 'Description' field is filled with 'HMI_READ_BOOL'. There are checkboxes for 'Increase priority', 'Enable Read when Idle', 'Enable Write on Tag Change', 'Min', and 'Max'. The 'Read Trigger' and 'Write Trigger' fields are empty. The 'Station' field is empty. The 'Header' field is filled with 'BOOL'. The 'Read Completed' and 'Write Completed' fields are empty. The 'Read Status' and 'Write Status' fields are empty.

Figure 12: Driver configuration fields

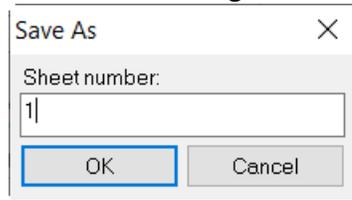
- **Description (1):**
It is for your own reference to identify the sheet in your project. This description will be shown in the 'WGRAF' folder of the 'Driver' directory. You can choose any name but to make the program readable it is suggested to clearly indicate whether it reads data from the PLC or write data to the PLC. Here the name 'HMI_READ_BOOL' is chosen to indicate that this sheet is only be used for mapping Boolean tags which displays data read from the PLC.
- **Enable read when Idle (2):** If the tag or value of this field is greater than one then the InduSoft runtime will continuously read the mapped tags of this worksheet when not busy.
- **Enable Write on Tag Change (3):** If the tag or value of this field is greater than one then the InduSoft runtime will update the PLC memory if one of the mapped tags of this worksheet has been changed.
- **Header (4):** Enter the PLC data type the mapped tags in the worksheet are representing. One of the following entries are supported: BOOL; SINT; USINT; INT; UINT; DINT; UDINT; REAL; LREAL; STRING. InduSoft only supports four data type it is therefore suggested to just use the PLC data type listed in the table below to prevent any error due to data type casting.

InduSoft		Win-GRAF
Tag Types	Length	Data Types
Boolean	1 bit	BOOL
Integer	32 bits	DINT
Real	64 bits	LREAL
String	80 ASCII characters	STRING(80)

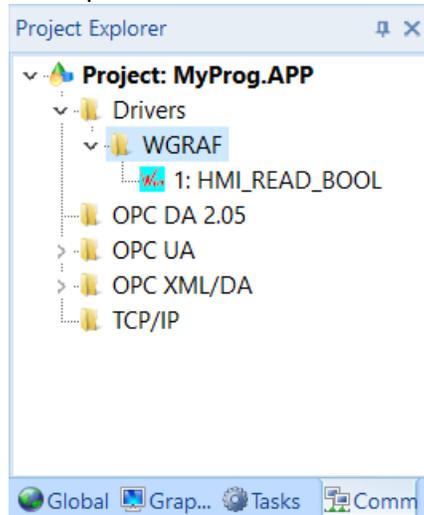
Table 6: InduSoft and Win-GRAF corresponding data types

- Save the worksheet. A dialog will pop up with a suggested sheet number under which it will be saved. Use the suggested number and

confirm the setting.



The saved worksheet is shown in the 'WGRAF' driver folder with the description name.



3. Add the 'readBool' array to the mapping area: Enter the tag name with the array index to the 'Tag Name' column and the mapped memory address to 'Address' column. Up to a maximum of 128 tags per sheet can be mapped.

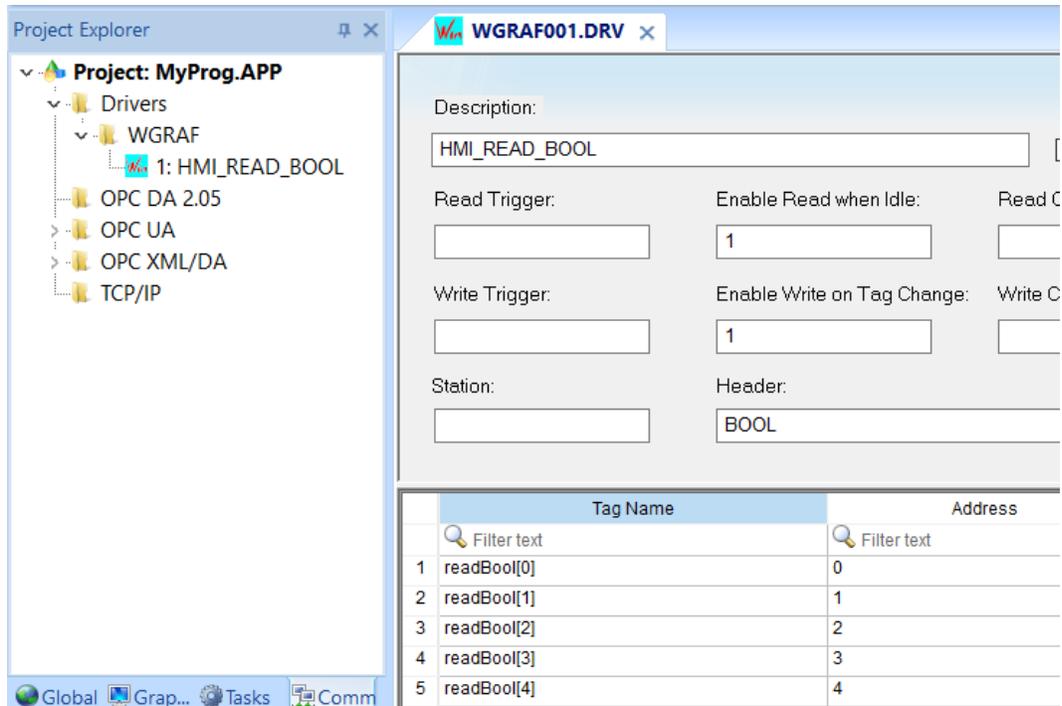


Figure 13: InduSoft mapped BOOL tags

The InduSoft mapping procedure for the 'readBool' array is finished. Figure 14 shows the corresponding Win-GRAF variable mapping setup. The InduSoft runtime will automatically update the HMI with the current values of the PLC BOOL variables to which the tags have been mapped.

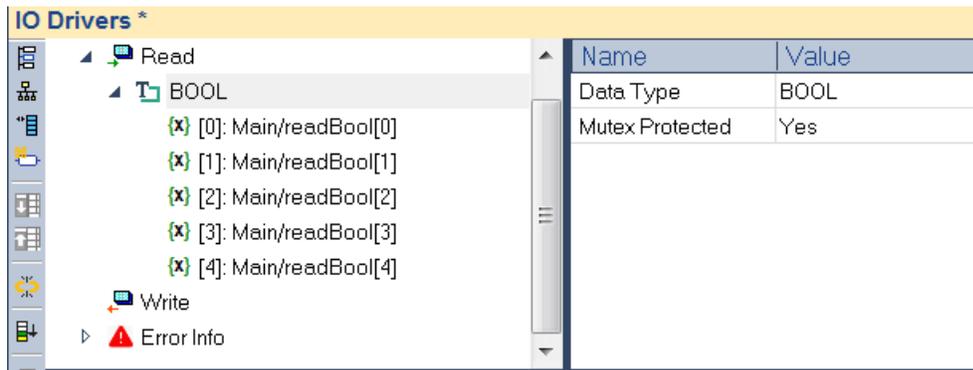


Figure 14: Win-GRAF mapped BOOL variables

Now repeat the process for the remaining tag array types of Figure 11. The following figures shows a couple of setting for the other tag mappings. The tree view on the right bottom shows the necessary Win-GRAF workbench variable mapping.

Important:

Prevent 'Read'/'Write' overlapping by assigning the 'Write' parameters to a different memory offset (Address) than the 'Read' parameters.

To prevent 'Read'/'Write' overlapping in the following worksheet examples the offset 0 to 99 are reserved for 'Read' variables and the offsets above 99 are used for the 'Write' variables.

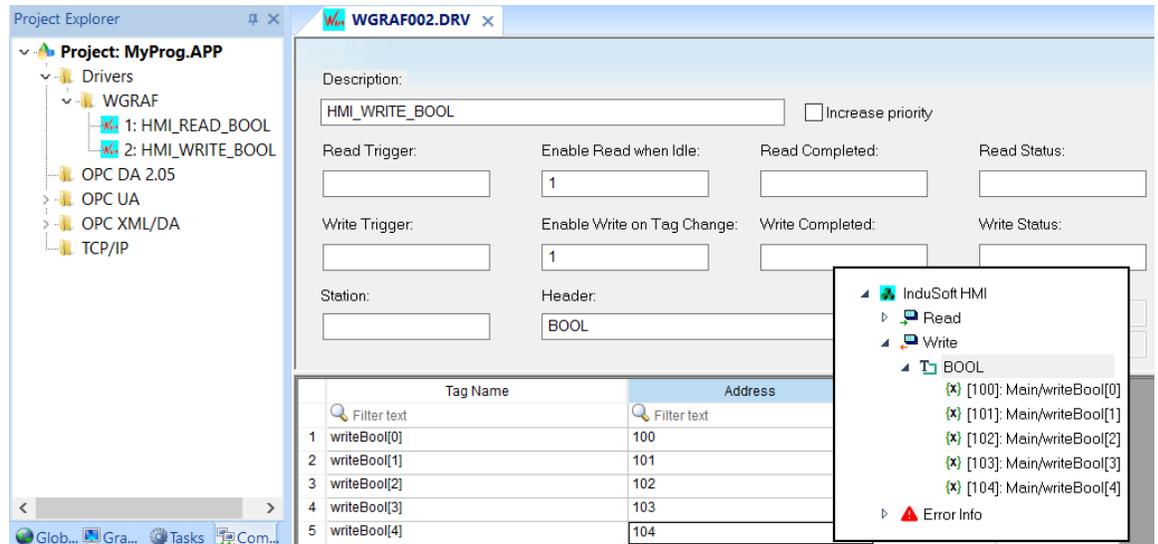


Figure 15: 'writeBool' tag array mapping for writing data to PLC

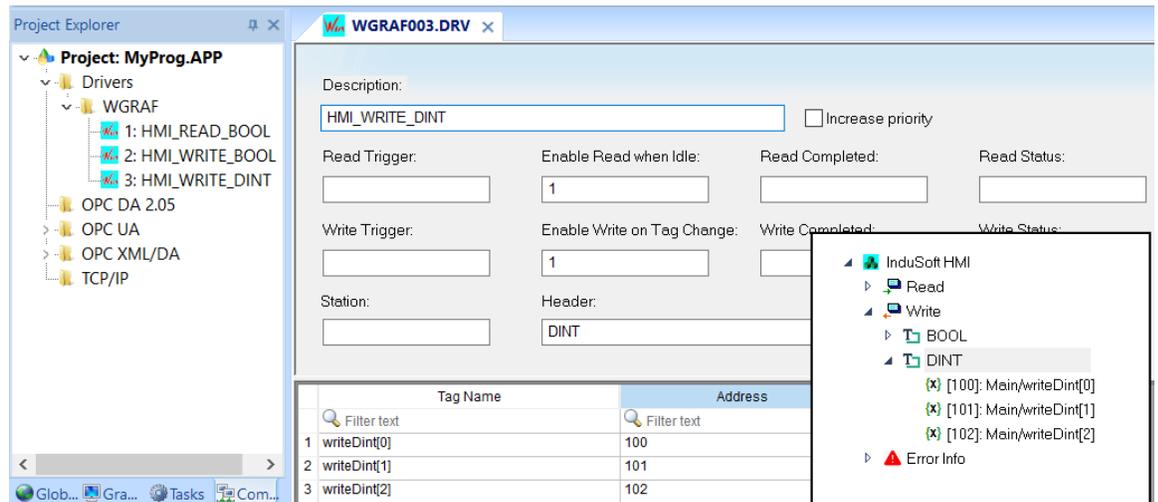


Figure 16: 'writeDint' tag array mapping

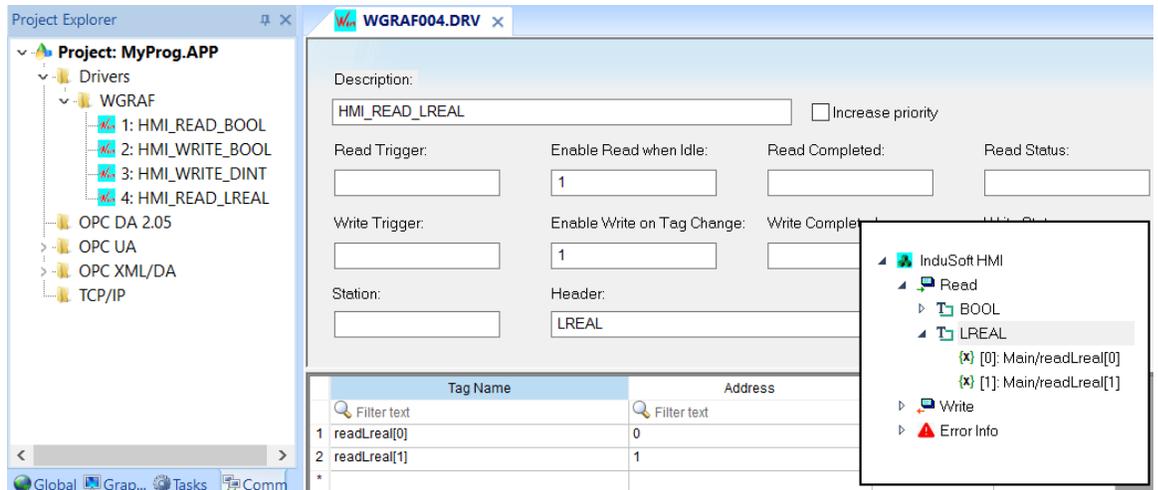
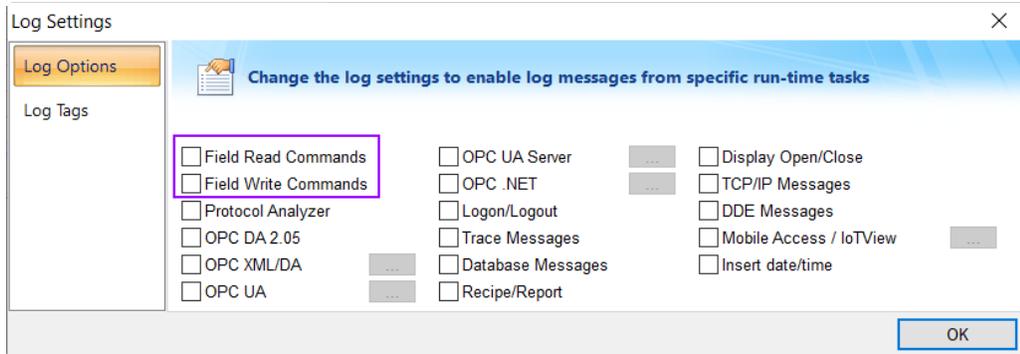


Figure 17: 'MyLreal' tag array mapping

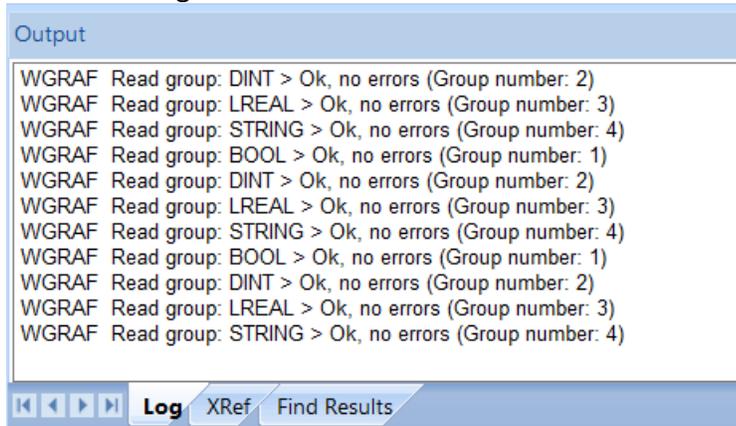
Step 3: Check the communication quality:
 InduSoft Studio allows you to test the communication quality of each mapped tag. Add the tag to monitor to the to one of the database list (e.g. DB1) of the 'Database Spy' at the bottom of the window. The 'Database Spy' will show the current value of the tags and its communication status.

Tag/Expression	Value	Quality	Continuous
readBool[0]	0	GOOD	<input checked="" type="checkbox"/>
writeBool[1]	0	GOOD	<input checked="" type="checkbox"/>
writeBool[2]	0	GOOD	<input checked="" type="checkbox"/>
readDint[0]	0	GOOD	<input checked="" type="checkbox"/>
readDint[1]	0	GOOD	<input checked="" type="checkbox"/>
writeString[0]		GOOD	<input checked="" type="checkbox"/>
readString[1]		GOOD	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

Addition information of the mapped worksheets can be obtained by right clicking the 'Output' window and selecting 'Field Read Commands' and/or 'Field Write Commands'



The runtime will in each cycle display whether and error occurred while worksheet tags data has been send to or received from the PLC.



3.5 InduSoft - WinGRAF Monitor Utility

The utility '*IndusoftMonitor*' monitors the data communication between the Win-GRAF and InduSoft runtime. The current content of each memory type is shown in a grid table. The memory content can be directly changed by double clicking a cell of the table and assigning a new value.

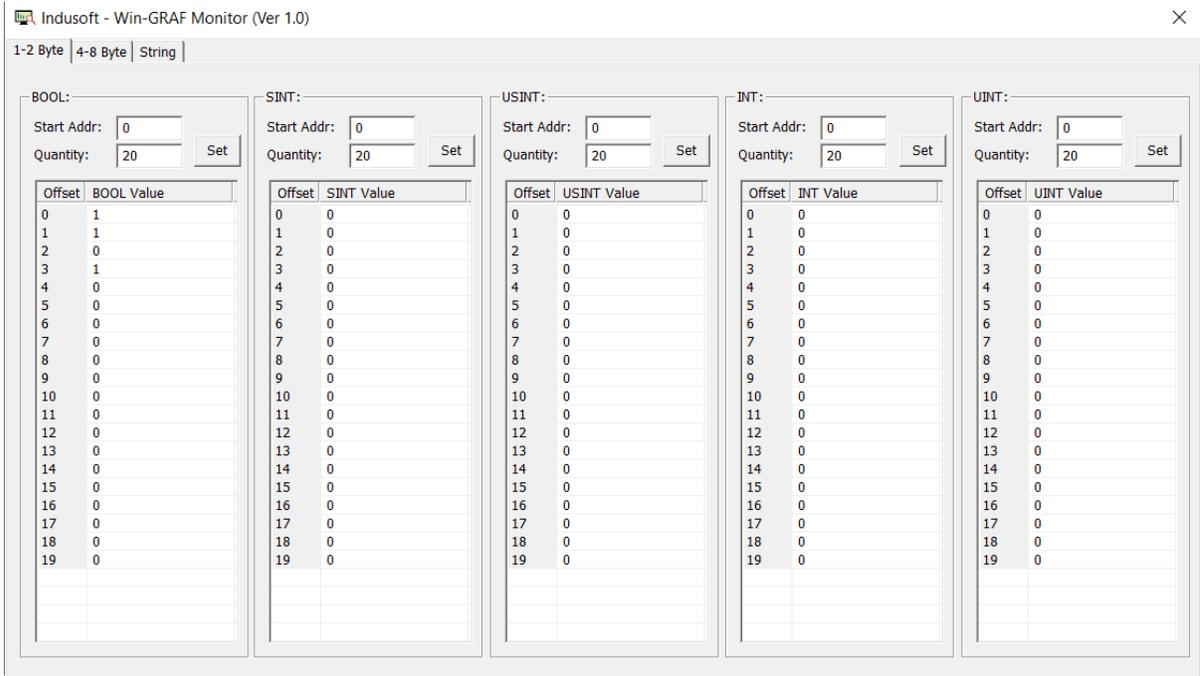


Figure 18: InduSoftMonitor Utility

3.6 Demo Program

Demo programs which shows the PLC and InduSoft data mapping and exchanged can be found in the following directory:

C:\Users\Public\Documents\Win-GRAF Workbench\Win-GRAF Wb xx.x\Projects\InduSoft

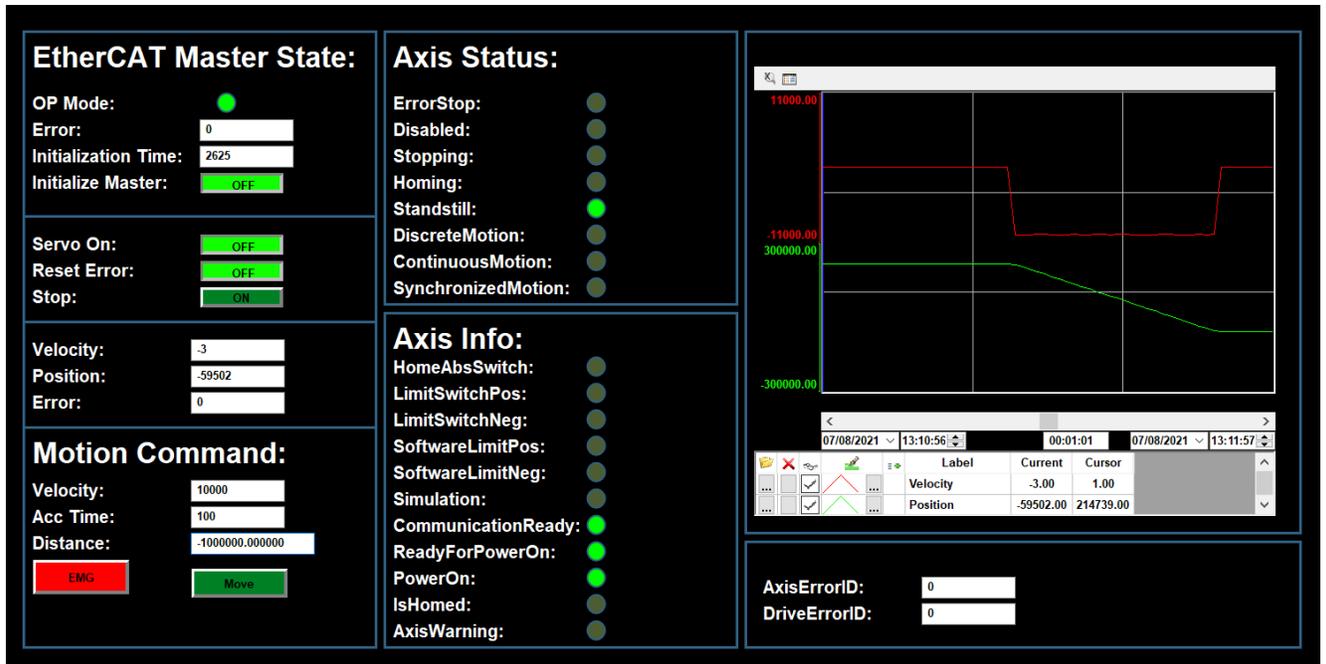


Figure 19: InduSoft HMI motion control demo

4 Programming Interface for External Program

4.1 Introduction

The Win-GRAF runtime (Windows version) provides programming interfaces APIs which allows other programs to directly access variables of the PLC application. This allows the system developer to integrate non-SoftPLC programs. The APIs are only available for embedded programs which means both runtime and program has to run on the same Windows device. The programming APIs are provided in standard c which allows the most common programming language , e.g. Python, C#, LabVIEW to directly access it.

4.2 Required Libraries

In order for the external application program to access the the Win-GRAF runtime the following dynamic library is required:

- WGrdfSharedMem.dll

The runtime setup automatically installs this library file to the Win-GRAF runtime execution directory.

If the execution files of both the runtime and the program accessing the runtime are placed in different directories then it is necessary to copy the 'WGrdfSharedMem.dll' file to the following directory:

- Windows 64 bit: 'C:\Windows\SysWOW64'
- Windows 32 bit: 'C:\Windows\System32'

4.3 Communication Structure

The schematic (Figure 20) shows the basic communication structure between the Win-GRAF runtime and external program which exchange data via the shared memory.

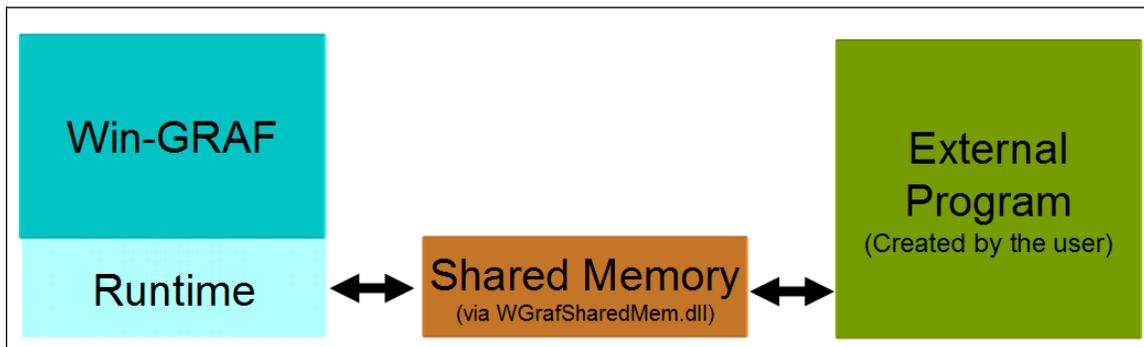


Figure 20: Win-GRAF shared memory communication structure

The following PLC data types can be directly accessed (Table 7):

T5 Runtime Data Type	Write	Read
BOOL	●	●

T5 Runtime Data Type	Write	Read
INT	●	●
UINT	●	●
DINT	●	●
UDINT	●	●
REAL	●	●
LREAL	●	●
TIME	●	●
STRING	●	●

Table 7: Supported data types

4.4 Win-GRAF Workbench Configuration

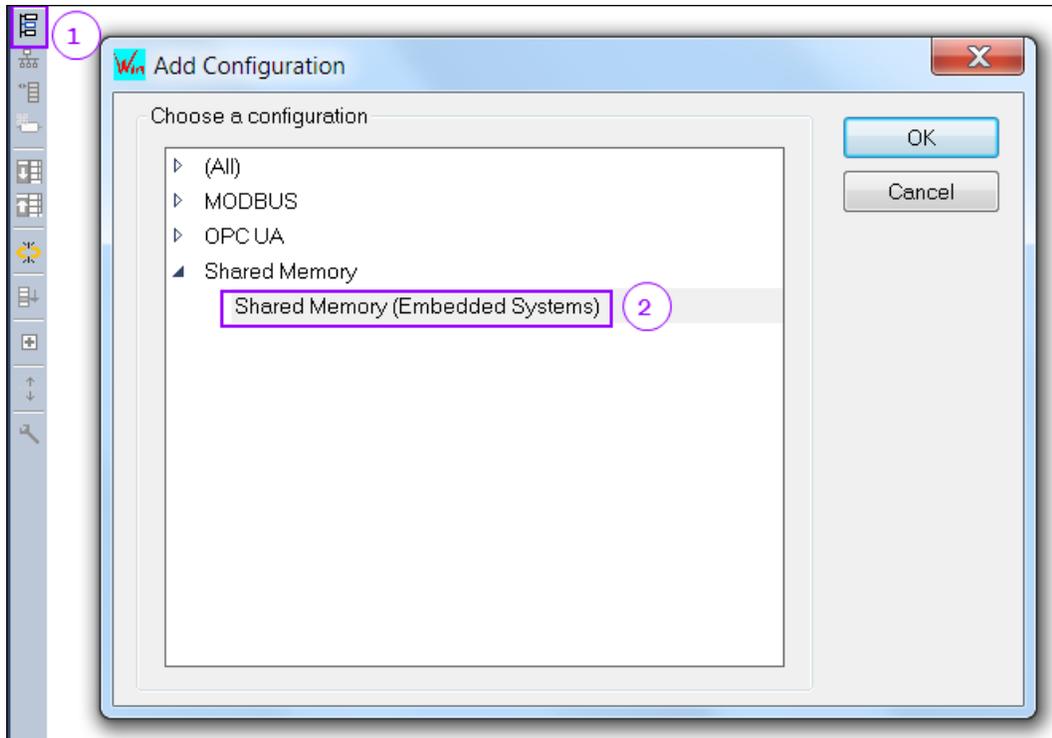
Before an external program can access any PLC data the PLC programmer has to explicitly select all the internal PLC variables for public access. For this first a shared memory with a unique name and memory size has to be declared and then the PLC variable mapped to the memory area. The configuration of the shared memory and assignment of internal variable can be conveniently done using the wizard of the workbench.

4.4.1 Defining and Creating Shared Memory

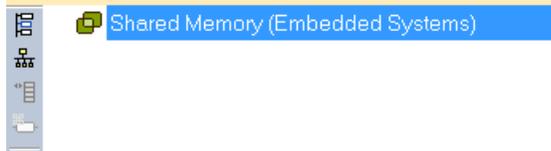
The Fieldbus wizard assist you in setting up a shared memory for embedded external program to be accessed. It allows the user via drag and drop to select the PLC variables to be mapped to the shared memory after the memory has been created. In addition the access right for each variable can be individually set, which determines the variable can be just read or both read and written by the external program. The variables in the memory will be automatically updated in each cycle. More than one shared memory can be created for different purposes.

Procedure to create a shared memory:

- Step 1:** Open the Fieldbus wizard by clicking the fieldbus  button of the toolbar or the fieldbus node in the workspace.
- Step 2:** Click 'Insert Configuration'  button of the toolbar on the left of the 'IO Drivers' editor and select 'Shared Memory (Embedded Systems)' from the 'Add Configuration' dialog. Confirm the selection with 'OK'.



A shared memory item will be added to the tree view editor:



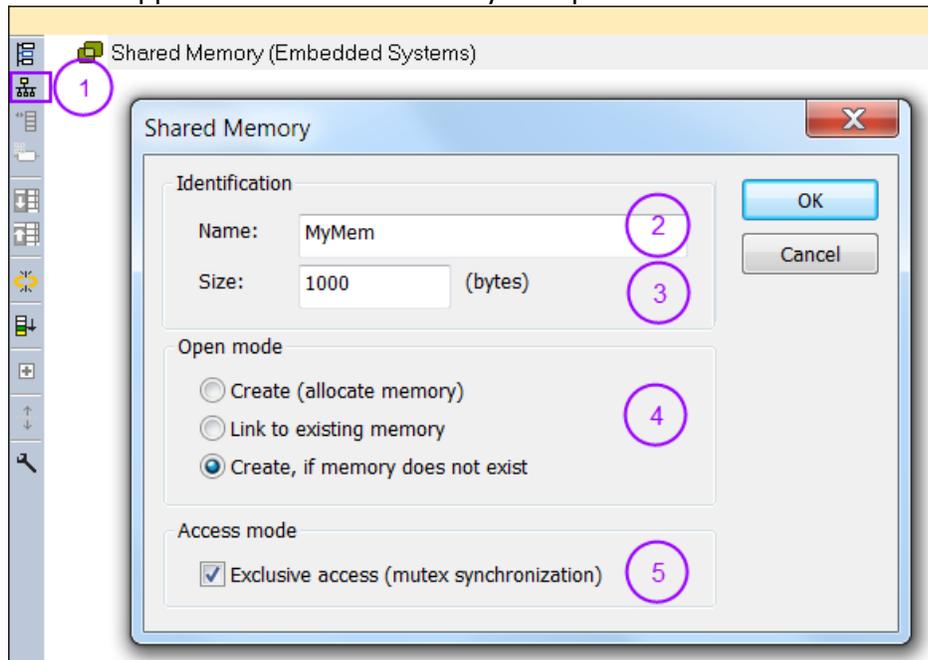
All the shared memories to be created will be listed as a sub item to the shared memory tree.

Step 3: Create a shared memory:

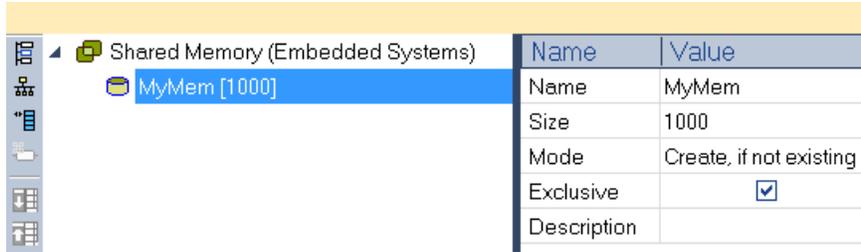
1. Open the 'Shared Memory' dialog by clicking the 'Insert Master/Port' on the left command bar
2. Enter the unique name of the shared memory. The name identifies the memory and it is not allowed to assign the same name to two different shared memories. The external program has to use this name to get access to the memory.
3. Set the size of memory in bytes
4. Set the shared memory creation mode. The mode determines which runtime (Win-GRAF or external program) creates the shared memory. Three modes are available:
 - **'Create (allocated memory)'**: Win-GRAF runtime creates the memory. The shared memory will be automatically created when the PLC application starts running.
 - **'Link to existing memory'**: The shared memory has to be created by the

external program and will not be created by the Win-GRAF runtime. It is therefore important to make that the external program creates this memory with the define name before the Win-GRAF application starts otherwise no data exchange will take place.

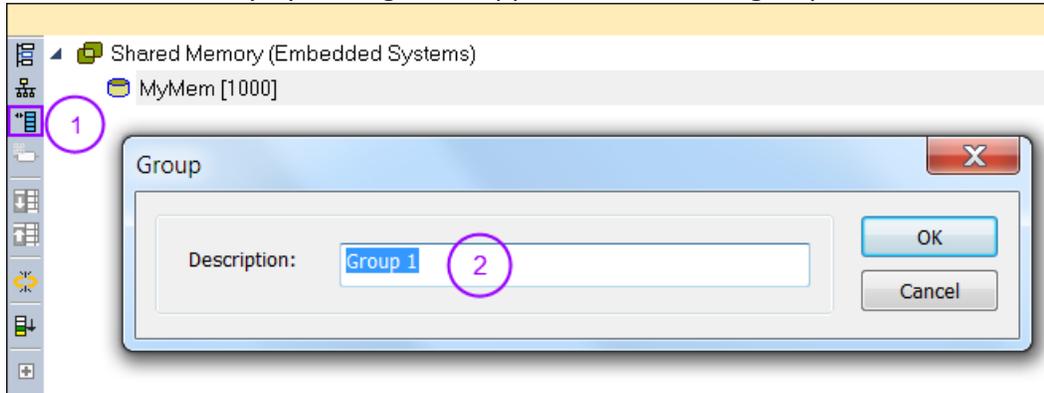
- '**Create, if memory does not exist**': Here either the PLC application or external program will create the memory depending which creates it first. During the initialization phase the PLC application automatically checks whether a shared memory with the defined name exists and creates a new one if it does not exist. By default this mode is enable as it guarantees that always a shared memory exists regardless which execution file started first.
5. Set the access mode. The access mode synchronizes the access to the shared memory. If the '*Exclusive access*' option is enabled only one application has the access right to the memory at a time. For example, once the PLC application has received the access right to the memory, the external program will be forced to wait until the PLC has finished its reading/writing process and gives up its access right before accessing it. The access mode ensures data consistency by ensuring that either the PLC application or the external program finish writing/reading a complete data package. Disable the option if data consistency is not important for the application to increase the system performance.



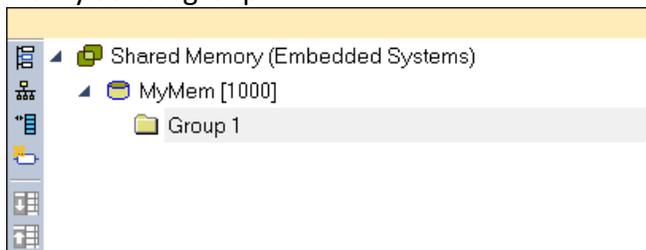
Confirming the setting with 'OK' generates a new sub item with the memory name to the tree view.



Step 4: Add a new group. Its purpose is to increase the readability of the content of the shared memory by sorting the mapped variables into groups.



Newly added group:



Step 5: Map variables to a locations in the memory space.

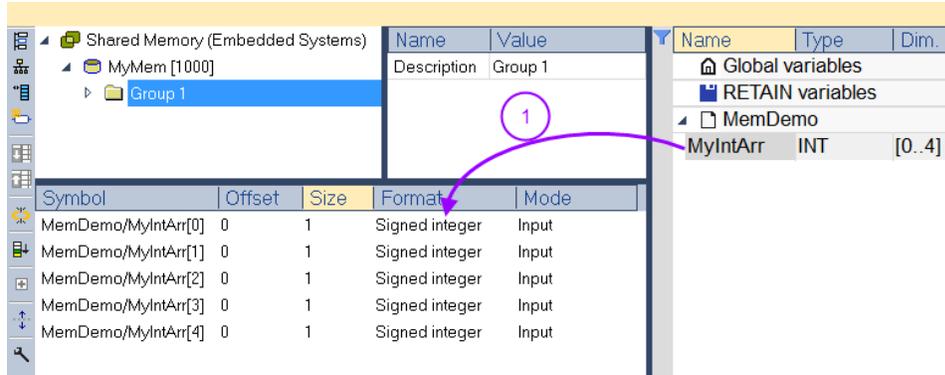
Example:

This example shows how to map an array to the shared memory.

Declare an INT array using the variable editor on the right hand side of the workbench.

Name	Type	Dim.	P	At...	S..	Init value	Us.
Global variables							
RETAIN variables							
MemDemo							
MyIntArr	INT	[0..4]				<input type="checkbox"/> 5(INT#0)	

1. Drag the array via drag and drop from the variable editor to the mapping area. The element of each array will be listed separately.



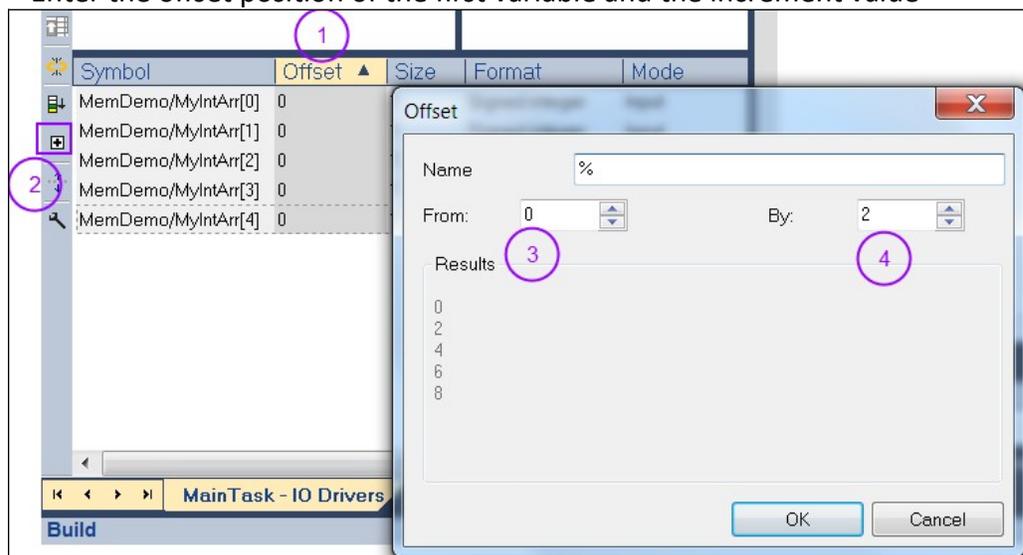
It is now necessary to set for each variable (element) the shared memory position (offset) is has to be mapped to, the occupied memory space (size), the data format it represents and the access right.

2. Memory position mapping: Double click the 'Offset' and set the offset number. The shared memory starts with the offset position of zero and each memory offset has the size of one byte. Make sure that the offset number does not exceed the maximum memory size. INT data type occupies two byte therefore the offset has to be incremented by two.

Symbol	Offset	Size	Format	Mode
MemDemo/MyIntArr[0]	0	1	Signed integer	Input
MemDemo/MyIntArr[1]	2	1	Signed integer	Input
MemDemo/MyIntArr[2]	4	1	Signed integer	Input
MemDemo/MyIntArr[3]	6	1	Signed integer	Input
MemDemo/MyIntArr[4]	8	1	Signed integer	Input

A quick way to set the offset values for all variables:

- Select all offset parameters by clicking on the 'Offset' header
- Click the 'Iterate property' command on the left vertical command bar
- Enter the offset position of the first variable and the increment value

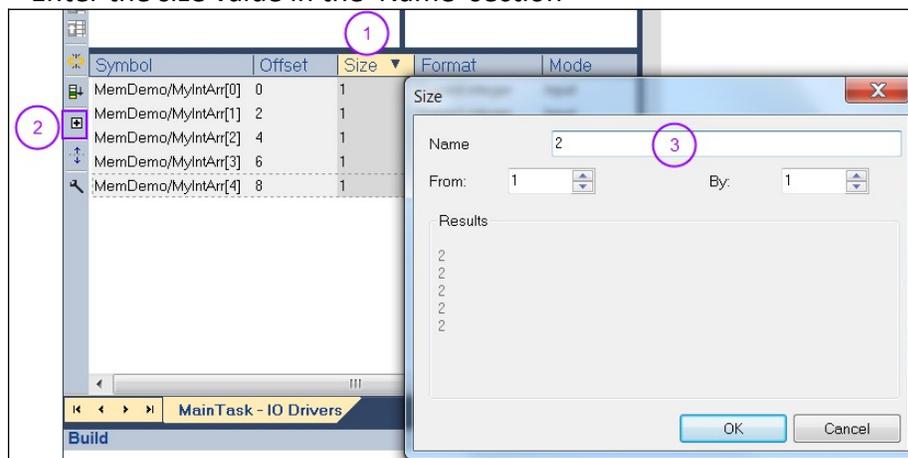


- Set the size each variable occupies in the memory. INT data size is two bytes therefore enter the value 2 in the 'Size' column.

Symbol	Offset	Size	Format	Mode
MemDemo/MyIntArr[0]	0	2	Signed integer	Input
MemDemo/MyIntArr[1]	2	2	Signed integer	Input
MemDemo/MyIntArr[2]	4	2	Signed integer	Input
MemDemo/MyIntArr[3]	6	2	Signed integer	Input
MemDemo/MyIntArr[4]	8	2	Signed integer	Input

A quick way for setting the size values for all variables:

- Select all size parameters by clicking on the 'Size' header
- Click the 'Iterate property' command on the left vertical command bar
- Enter the size value in the 'Name' section



- Set the data format type. For types a supported:

Signed integer
 Unsigned integer
 Float
 String

The default value is 'Signed integer'. In this example the INT array is a integer type therefore keep the setting.

Symbol	Offset	Size	Format	Mode
MemDemo/MyIntArr[0]	0	2	Signed integer	Input
MemDemo/MyIntArr[1]	2	2	Signed integer	Input
MemDemo/MyIntArr[2]	4	2	Signed integer	Input
MemDemo/MyIntArr[3]	6	2	Signed integer	Input
MemDemo/MyIntArr[4]	8	2	Signed integer	Input

- Set the access right. Three types are supported:

- **Input:** PLC application can only read the value from the memory
- **Output:** PLC application can only write to the memory section of the mapped variable
- **In/Out:** PLC application has both read/write access to memory location

represented by the mapped variable.

In this example some element are have only read and others only write access as shown below.

Symbol	Offset	Size	Format	Mode
MemDemo/MyIntArr[0]	0	2	Signed integer	Input
MemDemo/MyIntArr[1]	2	2	Signed integer	Output
MemDemo/MyIntArr[2]	4	2	Signed integer	Output
MemDemo/MyIntArr[3]	6	2	Signed integer	Input
MemDemo/MyIntArr[4]	8	2	Signed integer	Input

A quick way for setting the mode for all variables:

- All variable have to be set to the same mode: Select all modes by clicking on the 'Mode' header and press the ENTER key to open a drop box with the available mode. After selecting a mode all variable entries are automatically updated with this mode.
- Variables have to be set to different mode: Select the variables which needs to be assigned to the mode by clicking each individual entry in the 'Mode' section. Press the ENTER key to open a drop box with the available modes and select one. All the selected variables will be updated to the new mode setting.

The configuration of the shared memory is complete. Write your own external program which exchanges data with the memory. The APIs needed for this purpose are described in the next section.

4.5 External Program

In order for the external program to access the PLC application it has to call the shared memory APIs (Figure 21) to initial/link and gain read/write access. The APIs are written in standard c and can be accessed by all standard programming languages (e.g. c#, VB.net, Python, LabVIEW, etc.).

The required library is in the following directory

`C:\Users\Public\Documents\Win-GRAF Runtime\Win-GRAF Rt 10.0.0\API\Lib`

Create/Link Shared Memory	Read/Write Memory	Close/Disconnect
<ul style="list-style-type: none"> •WG_CreateSharedMem •WG_LinkToSharedMem •WG_LinkCreateSharedMem 	<ul style="list-style-type: none"> •WG_ReadFromSharedMem •WG_WriteToSharedMem 	<ul style="list-style-type: none"> •WG_CloseLinkToSharedMem

Figure 21: Shared Memory APIs

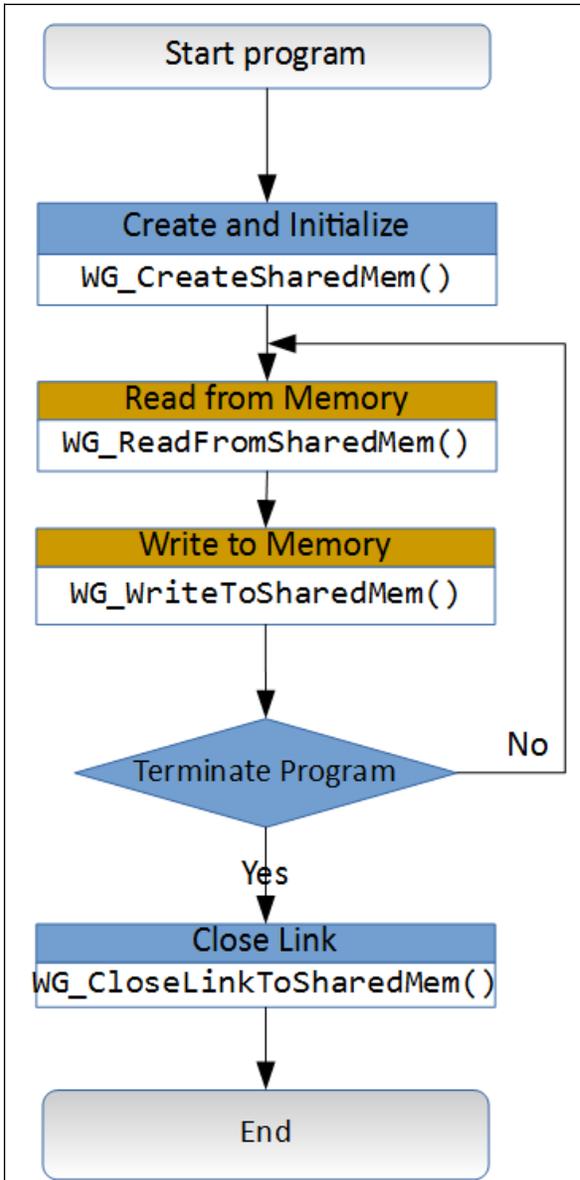


Figure 22: Shared memory API calling procedure

4.5.1 Create/Link

Either the PLC application, external program or both can be set to create the shared memory. It is suggested to allow both execution programs to create the shared memory. In this way the shared memory access is guaranteed regardless which program executes first.

Table 8 list the APIs available to create or link to a shared memory. The APIs fulfill the same purpose as the corresponding workbench UI function (Figure 23).

Win-GRAF Workbench (Figure 23)	Win-GRAF
Option mode	API
Create (allocate memory)	WG_CreateSharedMem()
Link to existing memory	WG_LinkToSharedMem()
Create, if memory does not exist	WG_LinkCreateSharedMem()

Table 8: Comparison of Workbench user interface functions and programming APIs

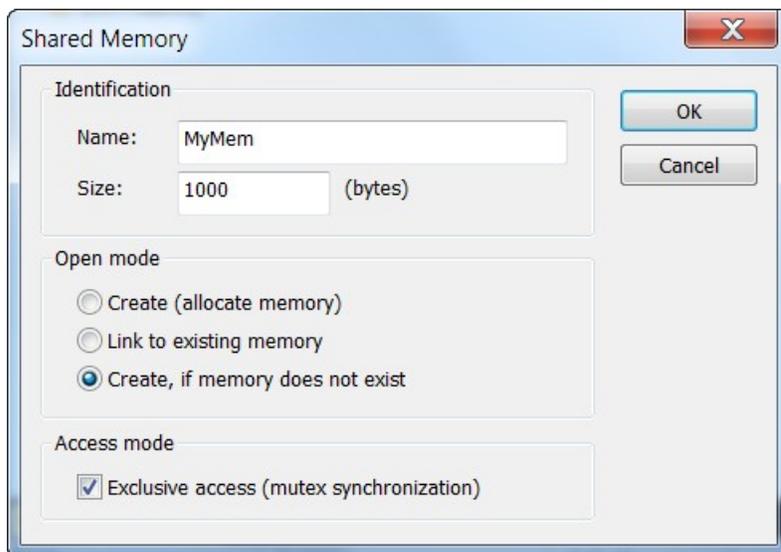


Figure 23: Workbench shared memory configuration UI

4.5.1.1 WG_CreateSharedMem

Creates a new shared memory with the configured name and size. The function fails if a memory with the same name already exists.

Syntax:

```

long WG_CreateSharedMem (    const wchar_t *szMemName,
                             DWORD dwMemSize,
                             BYTE bExclusiveAccess,
                             WORD *pwMemID);

```

Parameters:

Name	Description
<i>szMemName:</i>	Name of the shared memory: <ul style="list-style-type: none"> • Name has to be the same as the workbench setting
<i>dwMemSize:</i>	<ul style="list-style-type: none"> • Size of the shared memory in bytes
<i>bExclusiveAccess:</i>	Set memory access synchronization by using mutex. If enabled only one execution can access at a time. <ul style="list-style-type: none"> • 0: synchronization disabled • >0: synchronization enabled. Make sure the workbench mutex setting is enabled as well otherwise no synchronization takes place.
<i>pwMemID:</i>	<ul style="list-style-type: none"> • Handle to the shared memory

Return:

- 0: SHMEM_SUCCESS
- Others: see error table

Remarks:

-

4.5.1.2 WG_LinkToSharedMem

This function links to an existing shared memory with the set name, but does not create a new memory.

Syntax:

```

long WG_LinkToSharedMem (    const wchar_t *szMemName,
                             DWORD dwMemSize,
                             BYTE bExclusiveAccess,
                             WORD *pwMemID);

```

Parameters:

Name	Description
<i>szMemName:</i>	Name of the shared memory to link to: <ul style="list-style-type: none"> • Name has to be the same as the workbench setting
<i>dwMemSize:</i>	<ul style="list-style-type: none"> • Size of the shared memory in bytes • The size has to be set lower or
<i>bExclusiveAccess:</i>	Set memory access synchronization by using mutex. If enabled only one execution can access at a time. <ul style="list-style-type: none"> • 0: synchronization disabled • >0: synchronization enabled. Make sure the workbench mutex setting is enabled as well otherwise no synchronization takes place.
<i>pwMemID:</i>	<ul style="list-style-type: none"> • Handle to the shared memory

Return:

- 0: SHMEM_SUCCESS
- Others: see error table

Remarks:

-

4.5.1.3 WG_LinkCreateSharedMem

This function links to a shared memory and if it does not exist automatically creates it.

Syntax:

```
Long WG_LinkCreateSharedMem (    const wchar_t *szMemName,
                                DWORD dwMemSize,
                                BYTE bExclusiveAccess,
                                WORD *pwMemID);
```

Parameters:

Name	Description
<i>szMemName:</i>	Name of the shared memory: <ul style="list-style-type: none"> • Name has to be the same as the workbench setting
<i>dwMemSize:</i>	<ul style="list-style-type: none"> • Size of the shared memory in bytes • The size has to be set lower or
<i>bExclusiveAccess:</i>	Set memory access synchronization by using mutex. If enabled only one execution can access at a time. <ul style="list-style-type: none"> • 0: synchronization disabled • >0: synchronization enabled. Make sure the workbench mutex setting is enabled as well otherwise no synchronization takes place.
<i>pwMemID:</i>	<ul style="list-style-type: none"> • Shared memory ID

Return:

- 0: SHMEM_SUCCESS
- Others: see error table

Remarks:

-

4.5.2 Data Exchange

Either the PLC application, external program or both can be set to create the shared

4.5.2.1 WG_WriteToSharedMem

This function writes data to the shared memory to the set offset position.

Syntax:

```
Long WG_WriteToSharedMem ( WORD MemID,  
                           DWORD dwOffset,  
                           BYTE* pBuff,  
                           DWORD dwBuffSize);
```

Parameters:

Name	Description
<i>MemID:</i>	Shared memory ID
<i>dwOffset:</i>	Shared memory offset position in bytes.
<i>pBuff:</i>	Pointer to a buffer which contains the data to be written to the share memory.
<i>dwBuffSize:</i>	Number of data bytes to write

Return:

- 0: SHMEM_SUCCESS
- Others: see error table

Remarks:

-

4.5.2.2 WG_ReadFromSharedMem

This function reads data from the shared memory.

Syntax:

```
Long WG_ReadFromSharedMem ( WORD MemID,  
                             DWORD dwOffset,  
                             BYTE* pBuff,  
                             DWORD dwBuffSize);
```

Parameters:

Name	Description
<i>MemID:</i>	Shared memory ID
<i>dwOffset:</i>	Shared memory offset position in bytes.

<i>pBuff</i> :	Pointer to a buffer which store the data read from the shared memory.
<i>dwBuffSize</i> :	Number of data bytes to read

Return:

- 0: SHMEM_SUCCESS
- Others: see error table

Remarks:

-

4.5.3 Close Link

4.5.3.1 WG_CloseLinkToSharedMem

This function releases the share memory resource. It should be called when the external program terminates. The shared memory resources will only be release once all the application hast terminated their memory link.

Syntax:

```
Long WG_CloseLinkToSharedMem ( WORD MemID);
```

Parameters:

Name	Description
<i>MemID</i> :	Shared memory ID

Return:

- 0: SHMEM_SUCCESS
- Others: see error table

Remarks:

-

4.5.4 Error Table

Error Code		Description
SHMEM_SUCCESS	0	No error
SHMEM_LINK_ALREADY_EXISTS	1	No error.

Error Code		Description
		A link to the shared memory does already exist. Function returns the shared memory ID of the existing link.
SHMEM_ERROR	-1	reserved
SHMEM_INVALID_MEM_NAME	-2	Invalid shared memory name. The name length is either too short or long
SHMEM_NAME_NOT_FOUND	-3	Linking to an shared memory failed because no memory with the given name could be identified
SHMEM_NOT_LINKED_TO_MEMORY	-4	Invalid shared memory ID. The ID given is not linked to a shared memory.
SHMEM_BLOCKING_ERR	-5	Memory access synchronization failed.
SHMEM_EXCEED_MEMORY_LIMIT	-6	Access a location which it outside the shared memory range.
SHMEM_BUFFER_SIZE_ZERO	-7	The shared memory size is zero
SHMEM_LINK_TO_MEMORY_FAILED	-8	Linking to shared memory failed
SHMEM_CREATE_MEMORY_FAILED	-9	Creating a shared memory failed
SHMEM_LINK_TO_MEMORY_ALREADY_EXISTS	-10	reserved
SHMEM_MEMORY_ALREADY_EXISTS	-11	Failed creating the shared memory because it already exists
SHMEM_MAP_TO_MEMORY_FAILED	-12	Shared memory mapping failed

Table 9: Communication errors