SoMachine

Network Variable Configuration
SE_NetVarUdp Library Guide

11/2016

EIO0000001151.03

www.schneider-electric.com

Schneider Electric

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

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

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

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

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

Table of Contents

Safety Information

Important Information

NOTICE

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

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

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

⚠ DANGER

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

⚠ WARNING

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

⚠ CAUTION

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

NOTICE

NOTICE is used to address practices not related to physical injury.

PLEASE NOTE

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

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

About the Book

At a Glance

Document Scope

This guide explains the data exchange between controllers within a network via network variables.

Validity Note

This document has been updated for the release of SoMachine V4.2.

Product Related Information

⚠ WARNING

LOSS OF CONTROL

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

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

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

Terminology Derived from Standards

The technical terms, terminology, symbols and the corresponding descriptions in this manual, or that appear in or on the products themselves, are generally derived from the terms or definitions of international standards.

In the area of functional safety systems, drives and general automation, this may include, but is not limited to, terms such as *safety*, *safety function*, *safe state*, *fault*, *fault reset*, *malfunction*, *failure*, *error*, *error message*, *dangerous*, etc.

Among others, these standards include:

Standard	Description
EN 61131-2:2007	Programmable controllers, part 2: Equipment requirements and tests.
ISO 13849-1:2008	Safety of machinery: Safety related parts of control systems. General principles for design.
EN 61496-1:2013	Safety of machinery: Electro-sensitive protective equipment. Part 1: General requirements and tests.
ISO 12100:2010	Safety of machinery - General principles for design - Risk assessment and risk reduction
EN 60204-1:2006	Safety of machinery - Electrical equipment of machines - Part 1: General requirements
EN 1088:2008 ISO 14119:2013	Safety of machinery - Interlocking devices associated with guards - Principles for design and selection
ISO 13850:2006	Safety of machinery - Emergency stop - Principles for design
EN/IEC 62061:2005	Safety of machinery - Functional safety of safety-related electrical, electronic, and electronic programmable control systems
IEC 61508-1:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: General requirements.
IEC 61508-2:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Requirements for electrical/electronic/programmable electronic safety-related systems.
IEC 61508-3:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Software requirements.
IEC 61784-3:2008	Digital data communication for measurement and control: Functional safety field buses.
2006/42/EC	Machinery Directive
2014/30/EU	Electromagnetic Compatibility Directive
2014/35/EU	Low Voltage Directive

In addition, terms used in the present document may tangentially be used as they are derived from other standards such as:

Standard	Description
IEC 60034 series	Rotating electrical machines
IEC 61800 series	Adjustable speed electrical power drive systems
IEC 61158 series	Digital data communications for measurement and control – Fieldbus for use in industrial control systems

Finally, the term *zone of operation* may be used in conjunction with the description of specific hazards, and is defined as it is for a *hazard zone* or *danger zone* in the *Machinery Directive* (*2006/42/EC*) and *ISO 12100:2010*.

NOTE: The aforementioned standards may or may not apply to the specific products cited in the present documentation. For more information concerning the individual standards applicable to the products described herein, see the characteristics tables for those product references.

Chapter 1
Network Variables List - NVL

What Is in This Chapter?

This chapter contains the following topics:

Introduction to Network Variables List (NVL)

Overview

The Network Variables List (NVL) feature consists of a fixed list of variables that can be sent or received through a communication network. This enables data exchange within a network via network variables, if supported by the controller (target system).

The list must be defined in the sender and in the receiver controllers (and can be handled in a single or in multiple projects). Their values are transmitted via broadcasting through User Datagram Protocol (UDP) datagrams. UDP is a connectionless Internet communications protocol defined by IETF RFC 768. This protocol facilitates the direct transmission of datagrams on Internet Protocol (IP) networks. UDP/IP messages do not expect a response, and are therefore ideal for applications in which dropped packets do not require retransmission (such as streaming video and networks that demand real-time performance).

The NVL functionality is a powerful feature of SoMachineSoMachine BasicSoMachine Motion. It allows you to share and monitor data between controllers and their applications. However, there are no restrictions as to the purpose of the data exchanged between controllers, including, but not limited to, attempting machine or process interlocking or even controller state changes.

NOTE: The type of the network variable is not shared between different controllers. You have to ensure that the used types have the same definition on all devices; otherwise NVL communication is not possible. This applies, for example, to the types `SEC.ETH_R_STRUCT` or `SEC.PLC_R_STRUCT`. They are available by default in various controllers with different size or fields.

Only you, the application designer and/or programmer, can be aware of all the conditions and factors present during operation of the machine or process and, therefore, only you can determine the proper communication strategies, interlocks and related safeties necessary for your purposes in exchanging data between controllers using this feature. Strict care must be taken to monitor this type of communication feature, and to be sure that the design of the machine or process will not present safety risks to people or property.

⚠ WARNING

LOSS OF CONTROL

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

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

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

You can use Diagnostic *(see page 25)* and Error Management *(see page 28)* function blocks as well as network properties parameters to monitor the health, status and integrity of communications using this feature. This feature was designed for data sharing and monitoring and cannot be used for critical control functions.

Network Variables List (NVL)

The network variables to be exchanged are defined in the following 2 types of lists:

- Global Variables Lists (GVL) in a sending controller (sender)
- Global Network Variables List (GNVL) in a receiving controller (receiver)

The corresponding GVL and GNVL contain the same variable declarations. You can view their contents in the respective editor that opens after double-clicking the GVL or GNVL node in the **Devices** pane.

A GVL contains the network variables of a sender. In the **Network properties** of the sender, protocol and transmission parameters are defined. According to these settings, the variable values are broadcasted within the network. They can be received by all controllers that have a corresponding GNVL.

NOTE: For network variables exchange, the respective network libraries must be installed. This is done automatically for the standard network type UDP as soon as the network properties for a GVL are set.

Network variables are broadcasted from the GVL (sender) to one or more GNVL (receivers). For each controller you can define GVLs as well as GNVLs. Thus each controller can act as sender as well as receiver.

A sender GVL can be provided by the same or by another project. So, when creating a GNVL, the sender GVL can either be chosen from a selection list of all available GVLs within the network, or it can be read from an export file, which previously has been generated (for example, by using the **Link to File** dialog box) from the GVL.

NOTE: An export file is needed if the sender GVL to be used is defined within another project.

NVL Considerations

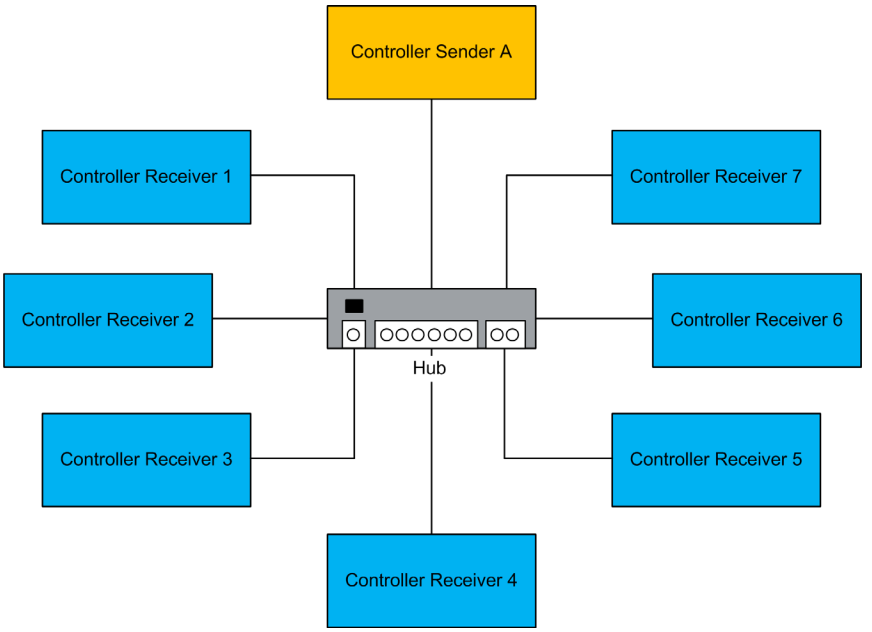
The following table shows the list of controllers that support the network variables list (NVL) functionality:

Function Name	M238	M241	M251	M258 LMC058	XBTGC XBT GK XBT GT	ATV IMC
Network Variables List	No	Yes	Yes	Yes	Yes	Yes*
*ATV IMC supports NVL only in freewheeling tasks.						

The following table shows the list of controllers that support the network variables list (NVL) functionality:

Function Name	LMC Eco LMC Pro LMC Pro2
Network Variables List	Yes

The figure shows a network consisting of 1 sender and the maximum of 7 receivers:



Controller Sender A: Sender with the global variables list (GVL) and receiver controller with global network variables lists (GNVLs)

Controller Receiver 1...7: Receivers (with GNVL) from A and sender controller (GVL) only for A

Configuring the Network Variables Exchange

Overview

To exchange network variables between a sender and a receiver, one sender and one receiver controller must be available in the SoMachine **Devices tree**. These are the controllers that are assigned the network properties described below.

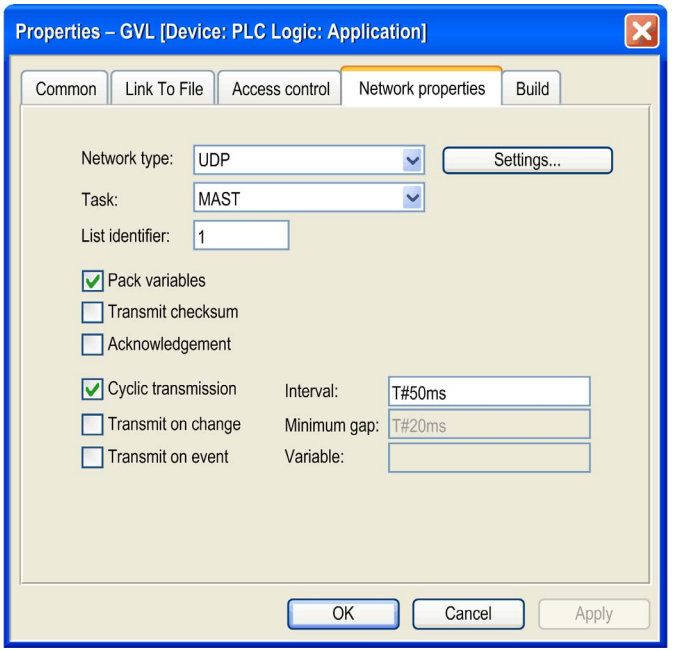
Proceed as follows to configure the network variables list:

Step	Action
1	Create a sender and a receiver controller in the **Devices tree**.
2	Create a program (POU) for the sender and receiver controller.
3	Add a task for the sender and receiver controller. **NOTE:** In order to maintain performance transparency, you should set the task priority of the dedicated NVL task to something greater than 25, and regulate communications to avoid saturating the network unnecessarily.
4	Define the global variables list (GVL) for the sender.
5	Define the global network variables list (GNVL) for the receiver.

An example with further information is provided in the Appendix *(see page 31)*.

Global Variables List

To create the GVL for the sender, define the following network properties in the **GVL → Properties → Network properties** dialog box:



Description of parameters

Parameter	Default Value	Description
Network type	**UDP**	Only the standard network type **UDP** is available. To change the **Broadcast Address** and the **Port**, click the **Settings...** button.
Task	**MAST**	Select the task you configured below the **Task Configuration** item for executing NVL code. To help maintain performance transparency, we recommend to configure a cycle time **Interval** ≥50 ms for this task. **NOTE:** In order to maintain performance transparency, you should set the task priority of the dedicated NVL task to something greater than 25, and regulate communications to avoid saturating the network unnecessarily.
List identifier	**1**	Enter a unique number for each GVL on the network. It is used by the receivers for identifying the variables list *(see page 22)*.

Parameter	Default Value	Description
Pack variables	activated	With this option activated, the variables are bundled in packets (datagrams) for transmission. If this option is deactivated, one packet per variable is transmitted.
Transmit checksum	deactivated	Activate this option to add a checksum to each packet of variables during transmission. Receivers will then check the checksum of each packet they receive and will reject those with a non-matching checksum. A notification will be issued with the `NetVarError_CHECKSUM` parameter *(see page 26)*.
Acknowledgement	deactivated	Activate this option to prompt the receiver to send an acknowledgement message for each data packet it receives. A notification will be issued with the `NetVarError_ACKNOWLEDGE` parameter *(see page 26)* if the sender does not receive this acknowledgement message from the receiver before it sends the next data packet.
Cyclic transmission • **Interval**	activated	Select this option for cyclic data transmission at the defined **Interval**. This **Interval** should be a multiple of the cycle time you defined in the task for executing NVL code to achieve a precise transmission time of the network variables.
Transmit on change • **Minimum gap**	deactivated • **T#20ms**	Select this option to transmit variables whenever their values have changed. **NOTE:** After the first download or using of **Reset Cold** or **Reset Warm** command in Online Mode the receiver controllers are not updated and keep their last value, whereas the sender controller value becomes 0 (zero). The **Minimum gap** parameter defines a minimum time span that has to elapse between the data transfer.
Transmit on event • **Variable**	deactivated • –	Select this option to transmit variables as long as the specified **Variable** equals TRUE. The variable is checked with every cycle of the task for executing NVL code.

Description of the button **Settings...**

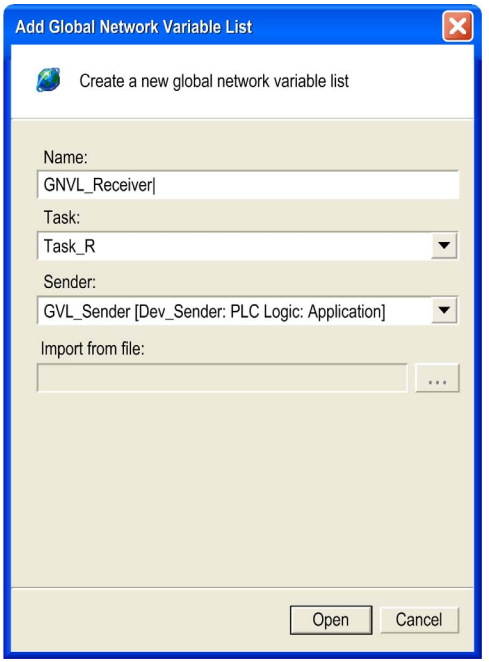
Parameter	Default Value	Description
Port	1202	Enter a unique port number (≥ 1202) for each GVL sender.
Broadcast Address	255.255.255.255	Enter a specific broadcast IP address for your application.

Global Network Variables List (GNVL)

A global network variables list can only be added in the **Devices** tree. It defines variables, which are specified as network variables in another controller within the network.

Thus, a GNVL object can only be added to an application if a global variables list (GVL) with network properties (network variables list) has already been created in one of the other network controllers. These controllers may be in the same or different projects.

To create the GNVL, define the following parameters in the **Add Object → Global Network Variable List** dialog box:



Description of parameters

Parameter	Default Value	Description
Name	**NVL**	Enter a name for the GNVL.
Task	task defined in the **Task Configuration** node of this **Application**	Select a task from the list of tasks which will receive the frames from the sender that are available under the **Task Configuration** node of the receiver controller.
Sender	1 of the GVLs currently available in the project	Select the sender's GVL from the list of all sender GVLs with network properties currently available in the project. Select the entry **Import from file** from the list to use a GVL from another project. This activates the **Import from file:** parameter below.

Parameter	Default Value	Description
Import from file:	–	This parameter is only available after you selected the option **Import from file** for the parameter **Sender**. The **...** opens a standard Windows Explorer window that allows you to browse to the export file **.gvl* you created from a GVL in another project. For further information refer to the *How to Add a GNVL From a Different Project* paragraph below.

How to Add a GNVL in the Same Project

When you add a GNVL via the **Add Object** dialog box, all appropriate GVLs that are found within the current project for the current network are provided for selection in the **Sender** list box. GVLs from other projects must be imported (see the *How to Add a GNVL From a Different Project* paragraph below).

Due to this selection, each GNVL in the current controller (sender) is linked to 1 specific GVL in another controller (receiver).

Additionally, you have to define a name and a task, that is responsible for handling the network variables, when adding the GNVL.

How to Add a GNVL From a Different Project

Alternatively to directly choosing a sender GVL from another controller, you can also specify a GVL export file you had generated previously from the GVL by using the **Link to file** properties. This allows you to use a GVL that is defined in another project.

To achieve this, select the option **Import from file** for the **Sender:** parameter and specify the path in the **Import from file:** parameter.

You can modify the settings later on via the **Properties - GVL** dialog box.

GNVL Properties

If you double-click a **GNVL** item in the **Devices** tree, its content will be displayed on the right-hand side in an editor. But the content of the GNVL cannot be edited, because it is only a reference to the content of the corresponding GVL. The exact name and the path of the sender that contains the corresponding GVL is indicated at the top of the editor pane together with the type of network protocol used. If the corresponding GVL is changed, the content of the GNVL is updated accordingly.

Network Variables List (NVL) Rules

Rules on the Amount of Data

Because of some performance limitations, respect the following rules:

Number	Rule
1	Data transmission from one GVL (sender) to one GNVL (receiver) should not exceed 200 bytes.
2	Data exchange between several GVLs (senders) of one controller and their associated GNVLs (receivers) should not exceed 1000 bytes of variables.

Rules on the Number of Datagrams

To limit the maximum cycle time of NVL tasks, respect the following rules:

Number	Rule	Description
1	Limit the number of received datagrams per cycle to 20.	When the limit is exceeded, the remaining datagrams are treated in the next cycle. A notification **Received overflow** is raised in the diagnostics data *(see page 26)* when the limit is reached. One datagram can contain up to 256 bytes. That means that you should not exceed the limit of 5120 bytes of data received by one receiver.
2	Limit the number of transmitted datagrams per cycle to 20.	When the limit is exceeded, the remaining datagrams are treated in the next cycle. A notification **Transmit overflow** is raised in the diagnostics data *(see page 26)* when the limit is reached. One datagram can contain up to 256 bytes. That means that you should not exceed the limit of 5120 bytes of data transmitted on one sender controller.

If the number of received / transmitted datagrams per cycle exceeds the limit several times, the following may happen:

- loss of UDP (user datagram protocol) datagrams
- incoherent or inconsistent exchange of variables

Adapt the following parameters according to your needs:

- cycle time of sender controller
- cycle time of receiver controller
- number of senders in the network

NOTICE
LOSS OF DATA Thoroughly test your application for proper transmission and reception of UDP datagrams prior to placing your system into service. **Failure to follow these instructions can result in equipment damage.**

Maximum Number of GVLs (Senders)

Define a maximum of 7 GVLs per controller (sender) to help maintain performance transparency.

Rules on Task Cycle Times of GVLs (Senders) and GNVLs (Receivers)

To help avoid reception overflow, you must define a cycle time for the task that manages the GVL transmission that is at least two times greater than the cycle time of the task that manages the GNVL reception.

Rules on the List Identifier Protection

The NVL function includes a list identifier checking:

The list identifier helps to avoid that a GVL (sender) from two separate controllers with the same list identifier (see dialog box **GVL → Properties → List identifier:**) sends datagrams to the same global network variables list (GNVL) of any controller. If the **List Identifier** is not unique, this can cause an interruption in the exchange of variables.

NOTICE
LOSS OF COMMUNICATION Ensure that the list identifier in the network is only used by one IP address. **Failure to follow these instructions can result in equipment damage.**

The list identifier checking function is implemented in the receiver controller.

If a GNVL (receiver) detects that two different IP addresses are using the same list identifier, the receiver immediately stops to receive datagrams.

Furthermore, a notification is issued in the `NETVARGETDIAGINFO` function block. The IP addresses of the two senders are provided in the output parameters `dwDuplicateListIdIp1` and `dwDuplicateListIdIp2` of this function block *(see page 25)*.

With the function block `NETVARRESETERROR` the detected NVL errors are reset and the communication is restarted.

Consistency in the Type of Network Variables

NOTE: The type of the network variable is not shared between different controllers. You have to ensure that the used types have the same definition on all devices; otherwise NVL communication is not possible.

This applies, for example, to the types `SEC.ETH_R_STRUCT` or `SEC.PLC_R_STRUCT`. They are available by default in various controllers with different size or fields.

Operating State of the Sender and the Receiver

Operating State

Operating State of the...		Network Variables Behavior
Sender	Receiver	
RUN	RUN	Network variables are exchanged between the sender and the receiver.
STOP	RUN	The sender is no longer sending variables to the receiver. The network variables are not exchanged between sender and receivers.
RUN	STOP	The receiver is not processing network variables from the sender. When the receiver returns to RUN state, the network variables are again processed by the receiver.
STOP	STOP	No variables are exchanged.

NOTE: Several communication initialization errors (`NetVarError_INITCOMM`) are detected, when you change the operating state of the sender from STOP to RUN.

Events in the Task that Manages NVL

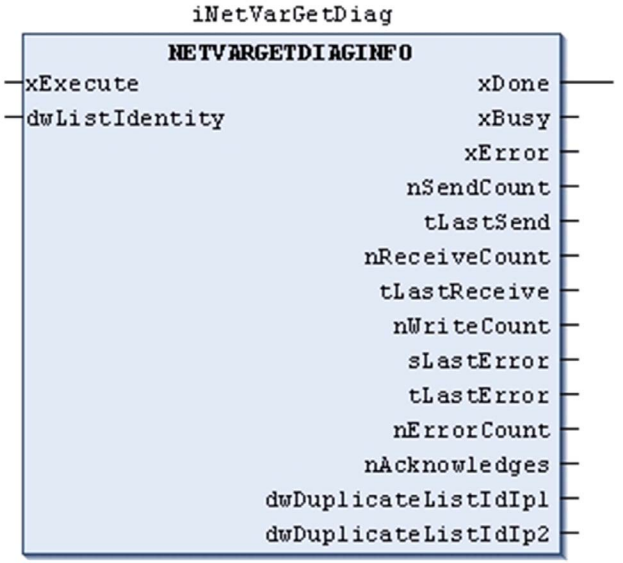
If the following events occur in the task that manages NVL, the behavior expected for the NVL is the same as if the controller was in STOP state in the array above:

- an exception occurs in the application which suspends the task
- a breakpoint is hit or a single cycle is processed on the task

Diagnosis of Network Variables List (NVL)

Function Block Description

The function block NETVARGETDIAGINFO gets information about the network variables list (NVL) functionality.



Input Parameters

Parameter	Type	Comment
xExecute	BOOL	rising edge: Diagnosis information is available. falling edge: The outputs of the function block are reset.
dwListIdentity	DWORD	List identifier of the GVL / GNVL where information should be retrieved from.

Output Parameters

Parameter	Type	Comment
xDone	BOOL	Information has been successfully retrieved.
xBusy	BOOL	The function block is active.
xError	BOOL	TRUE: An unknown list identifier has been used. FALSE: No error has been detected in the NETVARGETDIAGINFO function block execution.

Parameter	Type	Comment
nSendCount	UDINT	Number of UDP datagrams sent.
tLastSend	TIME	Date when the last UDP datagram was sent.
nReceiveCount	UDINT	Number of UDP datagrams received.
tLastReceive	TIME	Date when the last UDP datagram was received.
nWriteCount	UDINT	Number of variables written.
sLastError	NetVarUDPError	Last error detected on the network variable protocol. See description of the enumerated parameter NetVarUDPError below.
tLastError	TIME	Date when the last error was detected.
nErrorCount	UINT	Number of errors detected.
nAcknowledges	UINT	Number of acknowledges received.
dwDuplicateListIdIp1	DWORD	If a duplicate list identifier has been detected, this parameter indicates the first IP address of the sender which has the same list identifier as dwDuplicateListIdIp1.
dwDuplicateListIdIp2	DWORD	If a duplicate list identifier has been detected, this parameter indicates the second IP address of the sender which has the same list identifier as dwDuplicateListIdIp2.

NetVarUDPError Structure

The NetVarUDPError structure is an enumerated parameter and is defined by the following array:

Parameter	Value	Comment
NetVarError_NOERROR	0	no error detected
NetVarError_SENDDATA	1	unsuccessful data transfer
NetVarError_ACKNOWLEDGE	2	acknowledgement error detected **NOTE:** This error is only detected when the **Properties - GVL_Sender** dialog box option **Acknowledgement** is activated.
NetVarError_INIT_COMM	3	communication initialization error detected
NetVarError_CHECKSUM	4	checksum error detected **NOTE:** This error is only detected when the **Properties - GVL_Sender** dialog box option **Transmit checksum** is activated.
NetVarError_LAYOUT	5	layout error detected The number of variables exchanged does not match between sender and receiver.

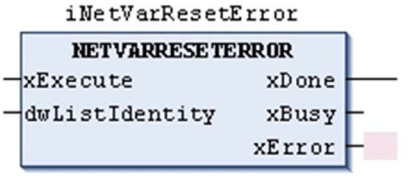
Parameter	Value	Comment
NetVarError_LISTID	6	duplicate list identifier detected There are 2 senders on the network with the same list identifier.
NetVarError_TRANSMIT_OVERFLOW	7	The number of datagrams sent per cycle has reached the limit of 20 *(see page 21)*.
NetVarError_RECEIVE_OVERFLOW	8	The number of datagrams received per cycle has reached the limit of 20 *(see page 21)*.

Error Management of Network Variable List (NVL)

Function Block Description

The function block NETVARRESETERROR restarts the communication exchange of variables after an NVL error is detected.

This function block resets detected duplicate list identifiers which are indicated by the output parameters dwDuplicateListIdIp1 or dwDuplicateListIdIp2 of the function block NETVARGETDIAGINFO.



Input Parameters

Parameter	Type	Comment
xExecute	BOOL	rising edge: The detected NVL errors are reset and the communication is restarted. falling edge: The outputs of the function block are reset.
dwListIdentity	DWORD	List identifier used by the GVLs on which the errors need to be reset.

Output Parameters

Parameter	Type	Comment
xDone	BOOL	The reset after detected error(s) has been completed successfully.
xBusy	BOOL	The function block is active.
xError	BOOL	TRUE: NETVARRESETERROR function block execution error has been detected. FALSE: No NETVARRESETERROR function block execution error has been detected.

Appendices

Overview

This appendix provides examples for defining GVLs and GNVLs. It shows an example for exchanging network variables between applications of different programming system versions, and it gives general information on functions and function blocks.

What Is in This Appendix?

The appendix contains the following chapters:

Appendix A
Example of a Simple Network Variable Exchange

Example

Overview

In the following example, a simple network variables exchange is established. In the sender controller a global variables list (GVL) is created. In the receiver controller the corresponding global network variables list (GNVL) is created.

Perform the following preparations in a standard project, where a sender controller **Dev_Sender** and a receiver controller **Dev_Receiver** are available in the **Devices** tree:

- Create a POU (program) **prog_sender** below the **Application** node of **Dev_Sender**.
- Under the **Task Configuration** node of this application, add the task **Task_S** that calls **prog_sender**.
- Create a POU (program) **prog_rec** below the **Application** node of **Dev_Receiver**.
- Under the **Task Configuration** node of this application, add the task **Task_R** that calls **prog_rec**.

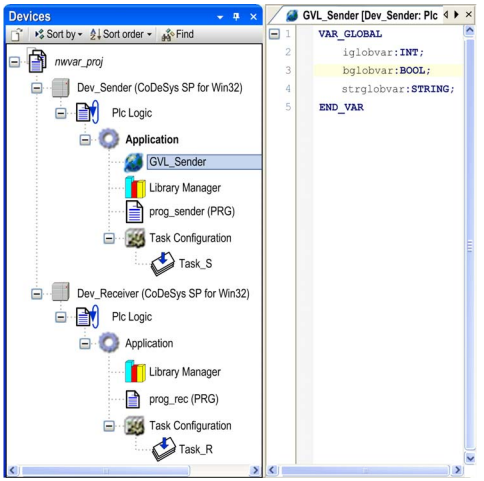
NOTE: The 2 controllers must be configured in the same subnet of the Ethernet network.

Defining the GVL for the Sender

Step 1: Define a global variable list in the sender controller:

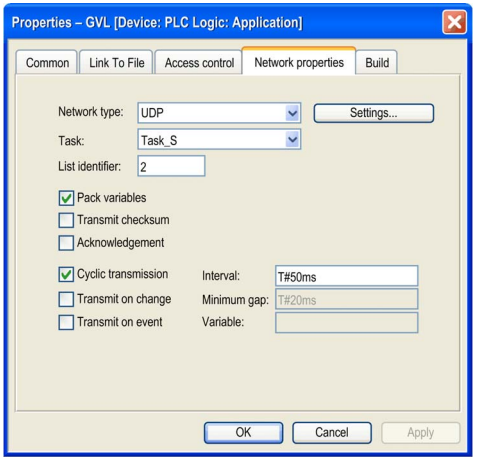
Step	Action	Comment
1	In the **Devices** pane, right-click the **Application** node of the controller **Dev_Sender** and select the commands **Add Object → Global Variable List...**.	The **Add Global Variable List** dialog box is displayed.
2	Enter the **Name** `GVL_Sender` and click **Open** to create a new global variable list.	The **GVL_Sender** node appears below the **Application** node in the **Devices** pane and the editor is opened on the right-hand side.

Step	Action	Comment

3	In the editor on the right-hand side, enter the following variable definitions:
	`VAR_GLOBAL`
	`iglobvar:INT;`
	`bglobvar:BOOL;`
	`strglobvar:STRING;`
	`END_VAR`
	 
		–

Step 2: Define the network properties of the sender GVL:

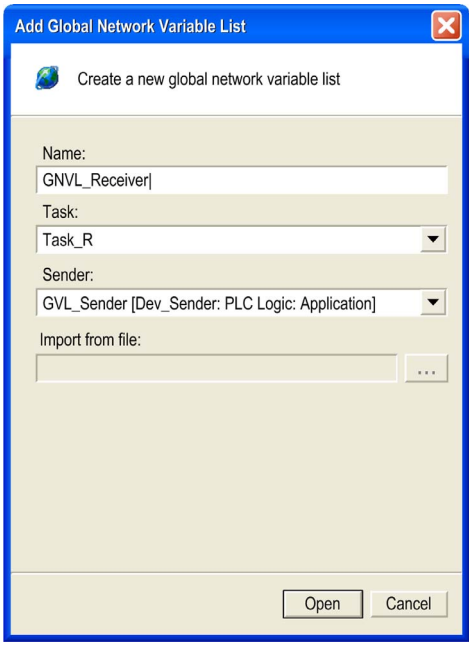
Step	Action	Comment
1	In the **Devices** pane, right-click the **GVL_Sender** node and select the command **Properties...**.	The **Properties - GVL_Sender** dialog box is displayed.

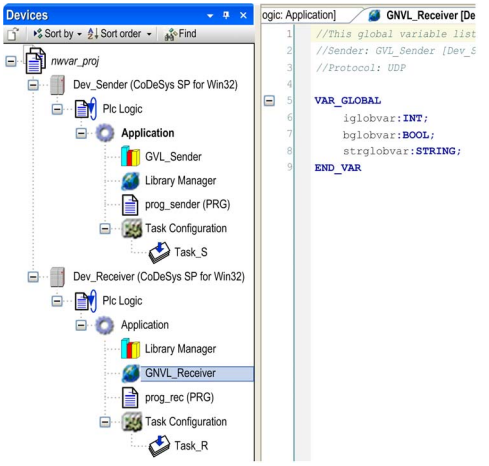
Step	Action	Comment
2	Open the **Network properties** tab and configure the parameters as shown in the graphic: 	–
3	Click **OK**.	The dialog box is closed and the GVL network properties are set.

Defining the GNVL for the Receiver

Step 1: Define a global network variable list in the receiver controller:

Step	Action	Comment
1	In the **Devices** pane, right-click the **Application** node of the controller **Dev_Receiver** and select the commands **Add Object → Global Network Variable List...**.	The **Add Global Network Variable List** dialog box is displayed.

Step	Action	Comment
2	Configure the parameters as shown in the graphic. 	This global network variable list is the counterpart of the GVL defined for the sender controller.

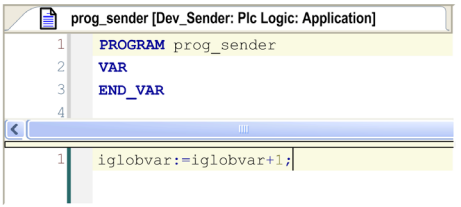
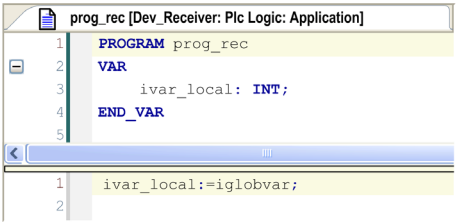
Step	Action	Comment
3	Click **Open**.	The dialog box is closed and the **GNVL_Receiver** appears below the **Application** node of the **Dev_Receiver** controller:  This GNVL automatically contains the same variable declarations as the **GVL_Sender**.

Step 2: View and / or modify the network settings of the GNVL:

Step	Action	Comment
1	In the **Devices** pane, right-click the **GNVL_Receiver** node and select the command **Properties...**.	The **Properties - GNVL_Receiver** dialog box is displayed.
2	Open the **Network settings** tab.	–

Step 3: Test the network variables exchange in online mode:

Step	Action	Comment
1	Under the **Application** node of the controller **Dev_Sender**, double-click the POU **prog_sender**.	The editor for **prog_sender** is opened on the right-hand side.

Step	Action	Comment
2	Enter the following code for the variable `iglobvar`: 	–
3	Under the **Application** node of the controller **Dev_Receiver**, double-click the POU **prog_rec**.	The editor for **prog_rec** is opened on the right-hand side.
4	Enter the following code for the variable `ivar_local`: 	–
5	Log on with sender and receiver applications within the same network and start the applications.	The variable `ivar_local` in the receiver gets the values of `iglobvar` as currently shown in the sender.

Appendix B
Compatibility

Compatibility

Introduction

Even if the controllers work with applications of different versions of the programming system (for example, V2.3 and V3.x), communication via network variables is possible.

However, the file formats of the different export files between versions (*.exp versus *.gvl) makes it impossible to simply import and export these files between projects.

If a reading global network variables list (GNVL) is set up in the latest version (for example, V3.x), the required network parameters configuration must be provided by a sender of the latest version (for example, V3.x). An export file **.exp* created from a sender by an earlier version (for example, V2.3) does not contain this information.

A solution for exchanging network variables between applications of different programming system versions is provided in the following paragraphs.

Updating the Global Network Variables List

To exchange network variables between applications of different programming system versions (for example, V2.3 and V3.x), update the global network variables list by performing the following steps:

Step	Action	Comment
1	Re-create the network variables list (NVL) that is already available in the earlier version (V2.3) in the latest version (V3.x).	To achieve this, add a global variables list (GVL) with network properties, containing the same variables declarations as in the NVL of the earlier version (V2.3).
2	Export the new GVL to a **.exp* file by using the **Link to File** tab.	**NOTE:** Activate the option **Exclude from build** in the **Build** tab to keep the GVL in the project without getting precompile events and ambiguous names. Deactivate the option to re-create the **.exp* file again in case any modifications on the GVL are required.
3	Re-import the list.	To achieve this, create a new global network variables list (GNVL) by using the previously generated **.exp* file in order to create an appropriately configured receiver list.

These steps are illustrated by the following example.

Example

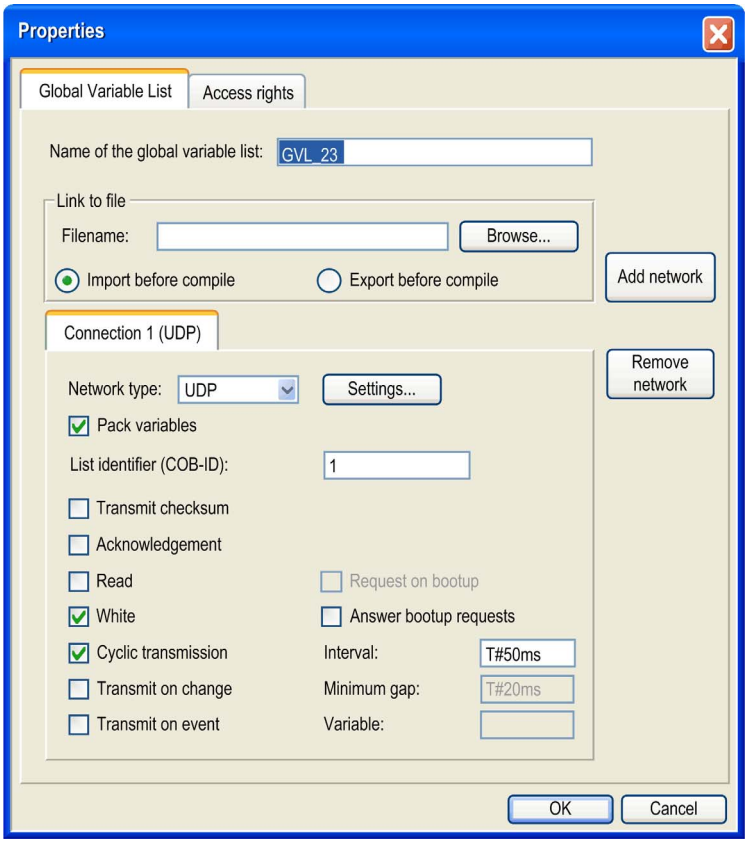
In this example, the variable `trans23`, that is defined in a V2.3 application, is made available for a later version (V3.x).

The following conditions are defined:

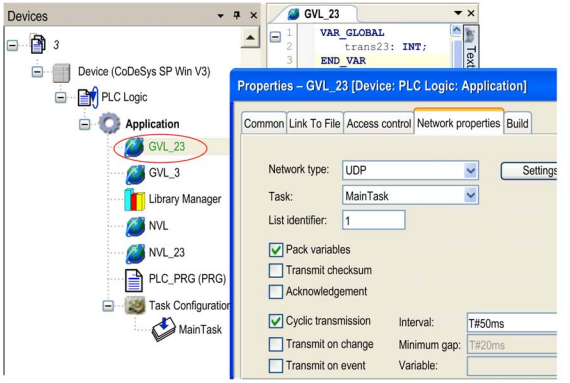
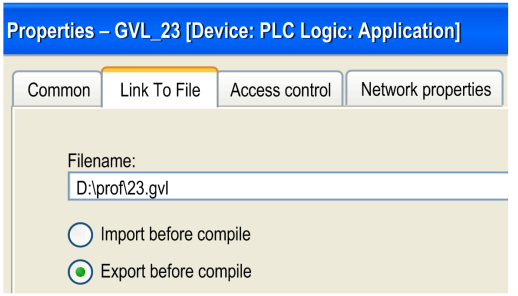
Condition	Description

1	In the earlier programming system version (V2.3) the project *23.pro* contains a global variables list **GVL_23** with the following declaration:
	`VAR_GLOBAL`
	`trans23:INT;`
	`END_VAR`

Condition	Description

2	The network properties of **GVL_23** are configured as follows:
	 
	
	NOTE: The export of this **GVL_23** creates a **.exp* file, that only contains the following variable declaration:
	`VAR_GLOBAL`
	`trans23:INT;`
	`END_VAR`
	The **.exp* file does not contain any configuration settings.

The following table shows the next steps to be executed for re-creating **GVL_23** in the latest version (V3.x):

Step	Action	Comment
1	Add a GVL object named **GVL_23** to an application.	–
2	Set the network properties as defined within the *23.pro* project.	
3	In the **Link to File** tab, configure a target export file *23.gvl*.	
4	In the **Build** tab, set the **Exclude from Build** option.	This setting allows you to keep the file on disk for later modifications. 

Step	Action	Comment

5	Compile the project.	The *23.gvl* file is generated and contains variable and configuration settings:
		`<GVL>`
		`<Declarations><![CDATA[VAR_GLOBAL  trans23: INT;`
		`END_VAR]]></Declarations>`
		`<NetvarSettings Protocol="UDP">`
		`<ListIdentifier>1</ListIdentifier>`
		`<Pack>True</Pack>`
		`<Checksum>False</Checksum>`
		`<Acknowledge>False</Acknowledge>`
		`<CyclicTransmission>True</CyclicTransmission>`
		`<TransmissionOnChange>False</TransmissionOnChange>`
		`<TransmissionOnEvent>False</TransmissionOnEvent>`
		`<Interval>T#50ms</Interval>`
		`<MinGap>T#20ms</MinGap>`
		`<EventVariable>`
		`</EventVariable>`
		`<ProtocolSettings>`
		`<ProtocolSetting Name="Port" Value="1202" />`
		`<ProtocolSetting Name="Broadcast Adr." Value="192.168.101.167" />`
		`</ProtocolSettings>`
		`</NetvarSettings>`
		`</GVL>`

6	Add a GNVL object in the V3.x project from the *23.gvl* export file (with the **Import from file:** command).	This serves to read the variable `trans23` from the controller of the earlier programming system (V.2.3). If both, the project from the earlier version (V2.3) as well as the application from the latest version (V3.x), are running within the network, the application from the latest version (V3.x) can read variable `trans23` from project *23.pro*.

Appendix C
Function and Function Block Representation

Overview

Each function can be represented in the following languages:

- IL: Instruction List
- ST: Structured Text
- LD: Ladder Diagram
- FBD: Function Block Diagram
- CFC: Continuous Function Chart

This chapter provides functions and function blocks representation examples and explains how to use them for IL and ST languages.

What Is in This Chapter?

This chapter contains the following topics:

Differences Between a Function and a Function Block

Function

A function:

- is a POU (Program Organization Unit) that returns one immediate result.
- is directly called with its name (not through an instance).
- has no persistent state from one call to the other.
- can be used as an operand in other expressions.

Examples: boolean operators (`AND`), calculations, conversion (`BYTE_TO_INT`)

Function Block

A function block:

- is a POU (Program Organization Unit) that returns one or more outputs.
- needs to be called by an instance (function block copy with dedicated name and variables).
- each instance has a persistent state (outputs and internal variables) from one call to the other from a function block or a program.

Examples: timers, counters

In the example, `Timer_ON` is an instance of the function block `TON`:

```
PROGRAM MyProgram_ST
VAR
    Timer_ON: TON;  // Function Block Instance
    Timer_RunCd: BOOL;
    Timer_PresetValue: TIME := T#5S;
    Timer_Output: BOOL;
    Timer_ElapsedTime: TIME;
END_VAR
```

```
Timer_ON(
    IN:=Timer_RunCd,
    PT:=Timer_PresetValue,
    Q=>Timer_Output,
    ET=>Timer_ElapsedTime);
```

How to Use a Function or a Function Block in IL Language

General Information

This part explains how to implement a function and a function block in IL language.

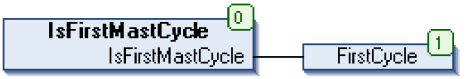
Functions `IsFirstMastCycle` and `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

Using a Function in IL Language

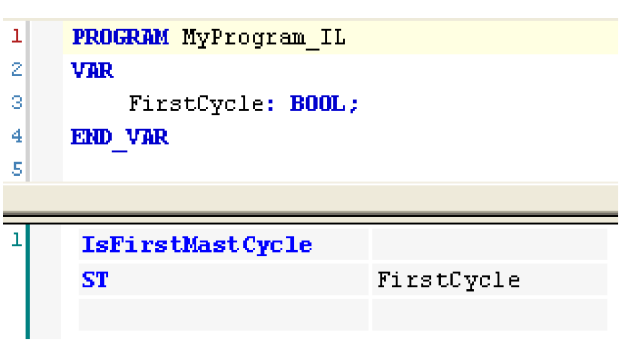
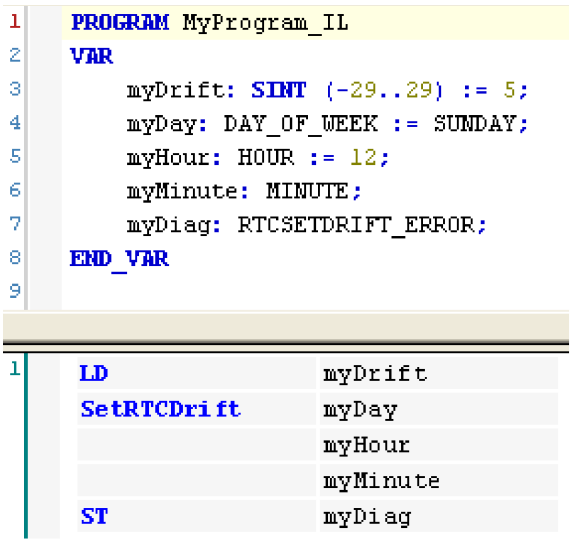
This procedure describes how to insert a function in IL language:

Step	Action
1	Open or create a new POU in Instruction List language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs *(see SoMachine, Programming Guide)*.
2	Create the variables that the function requires.
3	If the function has 1 or more inputs, start loading the first input using LD instruction.
4	Insert a new line below and: ● type the name of the function in the operator column (left field), or ● use the **Input Assistant** to select the function (select **Insert Box** in the context menu).
5	If the function has more than 1 input and when Input Assistant is used, the necessary number of lines is automatically created with `???` in the fields on the right. Replace the `???` with the appropriate value or variable that corresponds to the order of inputs.
6	Insert a new line to store the result of the function into the appropriate variable: type ST instruction in the operator column (left field) and the variable name in the field on the right.

To illustrate the procedure, consider the Functions `IsFirstMastCycle` (without input parameter) and `SetRTCDrift` (with input parameters) graphically presented below:

Function	Graphical Representation
without input parameter: `IsFirstMastCycle`	
with input parameters: `SetRTCDrift`	

In IL language, the function name is used directly in the operator column:

Function	Representation in SoMachineSoMachine BasicSoMachine Motion POU IL Editor
IL example of a function without input parameter: `IsFirstMastCycle`	
IL example of a function with input parameters: `SetRTCDrift`	

Using a Function Block in IL Language

This procedure describes how to insert a function block in IL language:

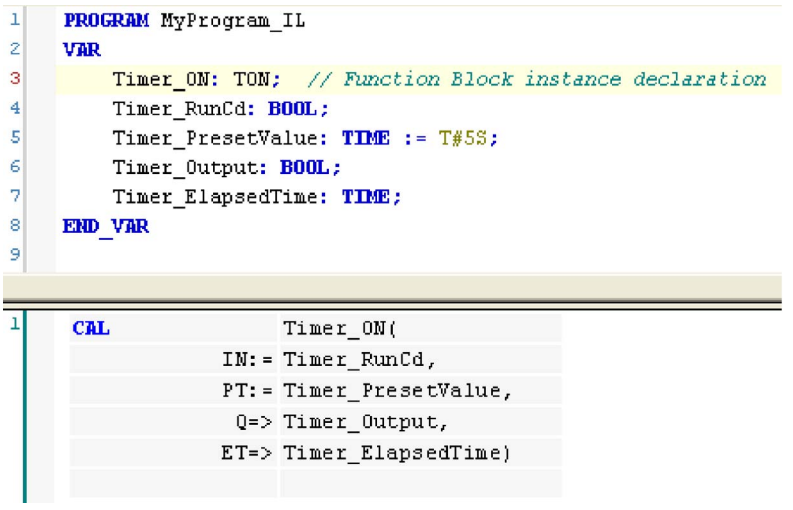
Step	Action
1	Open or create a new POU in Instruction List language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs *(see SoMachine, Programming Guide)*.

Step	Action
2	Create the variables that the function block requires, including the instance name.
3	Function Blocks are called using a `CAL` instruction: ● Use the **Input Assistant** to select the FB (right-click and select **Insert Box** in the context menu). ● Automatically, the `CAL` instruction and the necessary I/O are created. Each parameter (I/O) is an instruction: ● Values to inputs are set by ":=". ● Values to outputs are set by "=>".
4	In the `CAL` right-side field, replace `???` with the instance name.
5	Replace other `???` with an appropriate variable or immediate value.

To illustrate the procedure, consider this example with the `TON` Function Block graphically presented below:

Function Block	Graphical Representation
TON	

In IL language, the function block name is used directly in the operator column:

Function Block	Representation in SoMachineSoMachine BasicSoMachine Motion POU IL Editor
TON	

How to Use a Function or a Function Block in ST Language

General Information

This part explains how to implement a Function and a Function Block in ST language.

Function `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

Using a Function in ST Language

This procedure describes how to insert a function in ST language:

Step	Action
1	Open or create a new POU in Structured Text language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs *(see SoMachine, Programming Guide)*.
2	Create the variables that the function requires.
3	Use the general syntax in the **POU ST Editor** for the ST language of a function. The general syntax is: `FunctionResult:= FunctionName(VarInput1, VarInput2,.. VarInputx);`

To illustrate the procedure, consider the function `SetRTCDrift` graphically presented below:

Function	Graphical Representation
`SetRTCDrift`	

The ST language of this function is the following:

Function	Representation in SoMachineSoMachine BasicSoMachine Motion POU ST Editor

`SetRTCDrift`	`PROGRAM MyProgram_ST`
	`VAR myDrift: SINT(-29..29) := 5;`
	`myDay: DAY_OF_WEEK := SUNDAY;`
	`myHour: HOUR := 12;`
	`myMinute: MINUTE;`
	`myRTCAdjust: RTCDRIFT_ERROR;`
	`END_VAR`
	`myRTCAdjust:= SetRTCDrift(myDrift, myDay, myHour, myMinute);`

Using a Function Block in ST Language

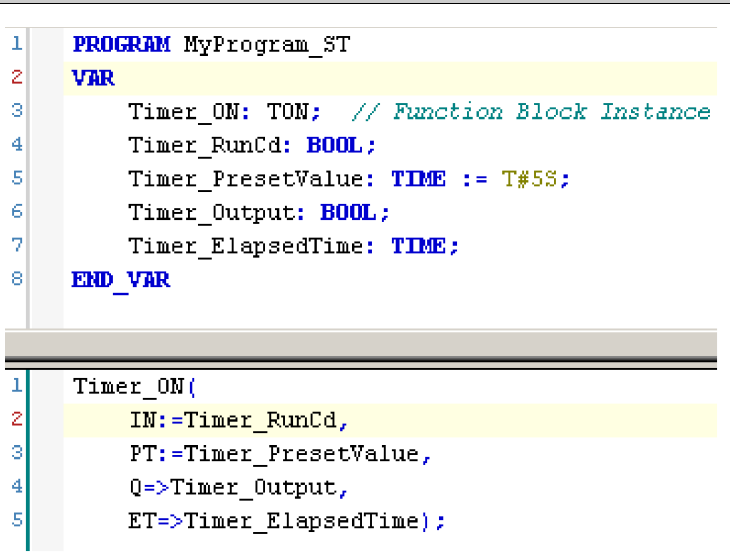
This procedure describes how to insert a function block in ST language:

Step	Action
1	Open or create a new POU in Structured Text language. **NOTE:** The procedure to create a POU is not detailed here. For more information on adding, declaring and calling POUs, refer to the related documentation *(see SoMachine, Programming Guide)*.
2	Create the input and output variables and the instance required for the function block: ● Input variables are the input parameters required by the function block ● Output variables receive the value returned by the function block
3	Use the general syntax in the **POU ST Editor** for the ST language of a Function Block. The general syntax is: `FunctionBlock_InstanceName(Input1:=VarInput1, Input2:=VarInput2,... Ouput1=>VarOutput1, Ouput2=>VarOutput2,...);`

To illustrate the procedure, consider this example with the `TON` function block graphically presented below:

Function Block	Graphical Representation
`TON`	

This table shows examples of a function block call in ST language:

Function Block	Representation in SoMachineSoMachine BasicSoMachine Motion POU ST Editor
TON	

```
PROGRAM MyProgram_ST
VAR
    Timer_ON: TON;  // Function Block Instance
    Timer_RunCd: BOOL;
    Timer_PresetValue: TIME := T#5S;
    Timer_Output: BOOL;
    Timer_ElapsedTime: TIME;
END_VAR
```

```
Timer_ON(
    IN:=Timer_RunCd,
    PT:=Timer_PresetValue,
    Q=>Timer_Output,
    ET=>Timer_ElapsedTime);
```

Glossary

A

application

A program including configuration data, symbols, and documentation.

B

byte

A type that is encoded in an 8-bit format, ranging from 00 hex to FF hex.

C

CFC

(*continuous function chart*) A graphical programming language (an extension of the IEC 61131-3 standard) based on the function block diagram language that works like a flowchart. However, no networks are used and free positioning of graphic elements is possible, which allows feedback loops. For each block, the inputs are on the left and the outputs on the right. You can link the block outputs to the inputs of other blocks to create complex expressions.

configuration

The arrangement and interconnection of hardware components within a system and the hardware and software parameters that determine the operating characteristics of the system.

controller

Automates industrial processes (also known as programmable logic controller or programmable controller).

E

expansion bus

An electronic communication bus between expansion I/O modules and a controller.

F

FB

(*function block*) A convenient programming mechanism that consolidates a group of programming instructions to perform a specific and normalized action, such as speed control, interval control, or counting. A function block may comprise configuration data, a set of internal or external operating parameters and usually 1 or more data inputs and outputs.

function block diagram

One of the 5 languages for logic or control supported by the standard IEC 61131-3 for control systems. Function block diagram is a graphically oriented programming language. It works with a list of networks where each network contains a graphical structure of boxes and connection lines representing either a logical or arithmetic expression, the call of a function block, a jump, or a return instruction.

G

GVL

(*global variable list*) Manages global variables within a SoMachineSoMachine BasicSoMachine Motion project.

I

I/O

(*input/output*)

IL

(*instruction list*) A program written in the language that is composed of a series of text-based instructions executed sequentially by the controller. Each instruction includes a line number, an instruction code, and an operand (refer to IEC 61131-3).

INT

(*integer*) A whole number encoded in 16 bits.

L

LD

(*ladder diagram*) A graphical representation of the instructions of a controller program with symbols for contacts, coils, and blocks in a series of rungs executed sequentially by a controller (refer to IEC 61131-3).

P

POU

(*program organization unit*) A variable declaration in source code and a corresponding instruction set. POUs facilitate the modular re-use of software programs, functions, and function blocks. Once declared, POUs are available to one another.

program

The component of an application that consists of compiled source code capable of being installed in the memory of a logic controller.

S

ST

(*structured text*) A language that includes complex statements and nested instructions (such as iteration loops, conditional executions, or functions). ST is compliant with IEC 61131-3.

U

UDP

(*user datagram protocol*) A connectionless mode protocol (defined by IETF RFC 768) in which messages are delivered in a datagram (data telegram) to a destination computer on an IP network. The UDP protocol is typically bundled with the Internet protocol. UDP/IP messages do not expect a response, and are therefore ideal for applications in which dropped packets do not require retransmission (such as streaming video and networks that demand real-time performance).

V

variable

A memory unit that is addressed and modified by a program.

Index