EcoStruxure Machine Expert

Modem Functions
Modem Library Guide

06/2019

EIO0000000552.05

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.

You agree not to reproduce, other than for your own personal, noncommercial use, all or part of this document on any medium whatsoever without permission of Schneider Electric, given in writing. You also agree not to establish any hypertext links to this document or its content. Schneider Electric does not grant any right or license for the personal and noncommercial use of the document or its content, except for a non-exclusive license to consult it on an "as is" basis, at your own risk. All other rights are reserved.

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.

QUALIFICATION OF PERSONNEL

A qualified person is one who has the following qualifications:

- Skills and knowledge related to the construction and operation of electrical equipment and the installation.
- Knowledge about providing machine functionality in software implementation.
- Received safety-related training to recognize and avoid the hazards involved.

The qualified person must be able to detect possible hazards that may arise from parameterization, modifying parameter values and generally from mechanical, electrical, or electronic equipment. The qualified person must be familiar with the standards, provisions, and regulations for the prevention of industrial accidents, which they must observe when designing and implementing the system.

PROPER USE

This product is a library to be used together with the control systems and servo amplifiers intended solely for the purposes as described in the present documentation as applied in the industrial sector.

Always observe the applicable safety-related instructions, the specified conditions, and the technical data.

Perform a risk evaluation concerning the specific use before using the product. Take protective measures according to the result.

Since the product is used as a part of an overall system, you must ensure the safety of the personnel by means of the design of this overall system (for example, machine design).

Any other use is not intended and may be hazardous.

About the Book

At a Glance

Document Scope

This document describes the modem functions library for EcoStruxure Machine Expert controllers.

Validity Note

This document has been updated for the release of EcoStruxure™ Machine Expert V1.1.

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.

Before you attempt to provide a solution (machine or process) for a specific application using the POUs found in the library, you must consider, conduct and complete best practices. These practices include, but are not limited to, risk analysis, functional safety, component compatibility, testing and system validation as they relate to this library.

⚠ WARNING

IMPROPER USE OF PROGRAM ORGANIZATION UNITS

- Perform a safety-related analysis for the application and the devices installed.
- Ensure that the Program Organization Units (POUs) are compatible with the devices in the system and have no unintended effects on the proper functioning of the system.
- Use appropriate parameters, especially limit values, and observe machine wear and stop behavior.
- Verify that the sensors and actuators are compatible with the selected POUs.
- Thoroughly test all functions during verification and commissioning in all operation modes.
- Provide independent methods for critical control functions (emergency stop, conditions for limit values being exceeded, etc.) according to a safety-related analysis, respective rules, and regulations.

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

Chapter 1
Modem Principles

Introduction

You can connect an PSTN (Public Switched Telephone Network) or GSM (Global System for Mobile Communications) Modem to a Serial Line to open transparent communications with a remote device to allow ASCII, Modbus, or Machine Expert protocol exchanges. GSM modems can also be used for sending and receiving SMS.

The modem library offers a set of function blocks to handle these features.

What Is in This Chapter?

This chapter contains the following topics:

Modem Functions on Controllers

Introduction

This topic describes management and operations of the controllers' modem communication functions.

NOTE: Communication functions are processed asynchronously with regard to the application task that called the function.

Available Function Blocks

This table describes the modem function blocks available to controllers:

Function	Description
Dial *(see page 23)*	`Dial` establishes communications through a modem to a remote device.
HangUp *(see page 25)*	`HangUp` closes a previously opened connection.
SendSMS *(see page 27)*	`SendSMS` sends SMS.
ReceiveSMS *(see page 27)*	`ReceiveSMS` allows the controller to receive SMS.
ConfigSim *(see page 31)*	Use `ConfigSim` when your SIM card requires either a PIN code, an SMS center phone number, or an initialization command.

Generic Parameters

Introduction

This topic describes the management and operations of the modem communication functions using the `Dial` function block as an example.

Graphical Representation

The parameters that are common to all function blocks in the modem library are highlighted in this graphic:

Common Parameters

These parameters are shared by several function blocks in the modem library:

Input	Type	Comment
`xExecute`	BOOL	The function is executed on the rising edge of this input. **NOTE:** When `xExecute` is set to TRUE at the first task cycle in RUN after a cold or warm reset, the rising edge is not detected.
`xAbort`	BOOL	aborts the ongoing operation at the rising edge
`serialLineNb`	BYTE	the number of the serial line (1 or 2)
`timeout`	WORD	exchange timeout is a multiple of 100 ms (0 for infinite)
NOTE: A function block operation may require several exchanges. The timeout applies to each exchange between the controller and the modem. Therefore, the global duration of the function block can be longer than the timeout.		

Output	Type	Comment
`xDone`	BOOL	The `xDone` output is set to `TRUE` when the function is completed successfully.
`xBusy`	BOOL	The `xBusy` output is set to `TRUE` while the function is ongoing.
`xAborted`	BOOL	The `xAborted` output is set to `TRUE` when the function is aborted with the `xAbort` input.

Output	Type	Comment
xError	BOOL	The xError output is set to TRUE when the function stops because of a detected error. When there is a detected error, nCommError and nOperError contain information about the detected error.
nCommError	SEN.Comm_ErrCodes	The nCommError output contains communication error codes. The ENUM type comes from the associated PLCCommunication library. (For details, refer to the guide *Communication Functions: PLCCommunication Library*.)
nOperError	OperationErrorCodesExt	The nOperError output contains operation error codes *(see page 13)*.
NOTE: As soon as the xBusy output is reset to FALSE, one (and only one) of these outputs is set to TRUE: • xDone • xError • xAborted		

Function blocks require a rising edge in order to be initiated. The function block needs to first see the xExecute input as FALSE in order to detect a subsequent rising edge.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Always make the first call to a function block with its xExecute input set to FALSE so that it will detect a subsequent rising edge.

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

OperationErrorCodesExt: Operation Error Codes (ENUM Type)

Enumerated Type Description

The `OperationErrorCodesExt` enumerated type contains codes that correspond to detected errors. The value of the communication error code returned by the `nCommError` output affects the meaning of the operation error code returned by the `nOperError` output.

nCommError = CommunicationOK

When the communication error code is 00 hex (correct transaction), the `OperationErrorCodesExt` enumerated type can return these values:

Enumerator	Value (hex)	Description
`OperationOK`	`00`	The exchange is valid.
`NotProcessed_or_TargetResourceMissing`	`01`	The request has not been processed, or the target system's resource is absent.
`BadResponse`	`02`	The received response is incorrect.
`ModemConfSLAsciiFailed`	`100`	The configuration of the serial line in ASCII failed. (See note 1.)
`ModemReconfSLFailed`	`200`	The configuration of the serial line back to the user configuration failed. (See note 1.)
`ModemBusy`	`300`	The modem answers "BUSY" to the `Dial` command. (See note 2.)
`ModemNoDialtone`	`400`	The modem answers "NO DIALTONE" to the `Dial` command. (See note 2.)
`ModemNoCarrier`	`500`	The modem's carrier signal has been lost or disconnected. The modem answers "NO CARRIER" to the `Dial` command. (See note 2.)
`ModemBadAnswer`	`600`	The modem's response in not valid.
`SimConfigurationFailed`	`1000`	The SIM card configuration failed. (For example, a PUK code is requested or the initSimString is not valid. *(see page 33)*) (See note 3.)
`SimPinCodeInvalid`	`2000`	The PIN code is not valid. (See note 3.)
`SimSmsCenterInvalid`	`4000`	The SMS center phone number is not valid. (See note 3.)
NOTE 1: These enumerated codes are dedicated to `Dial` and `HangUp` function blocks.		
NOTE 2: These enumerated codes are dedicated to the `Dial` function block.		
NOTE 3: These enumerated codes are dedicated to the `ConfigSim` function block		

nCommError = Refused

When the communication error code is FF hex (message refused), the `OperationErrorCodesExt` enumerated type can return these values:

Enumerator	Value (hex)	Description
`NotProcessed_or_TargetResourceMissing`	`01`	The request has not been processed, or the target system's resource is absent.
`BadLength`	`05`	The length is incorrect.
`CommChannelErr`	`06`	The communication channel is associated with a detected error.
`BadAddr`	`07`	The address is incorrect.
`SystemResourceMissing`	`0B`	A system resource is missing.
`TargetCommInactive`	`0C`	A target communication function is not active.
`TargetMissing`	`0D`	The target is absent.
`ChannelNotConfigured`	`0F`	The channel is not configured.

Chapter 2
Configuration

Adding a Modem to Your Application

Introduction

The modem library is automatically included in your program when you add a modem to a serial line manager with EcoStruxure Machine Expert. Once the library is added, you can use modem-specific function blocks in any of your application's POUs.

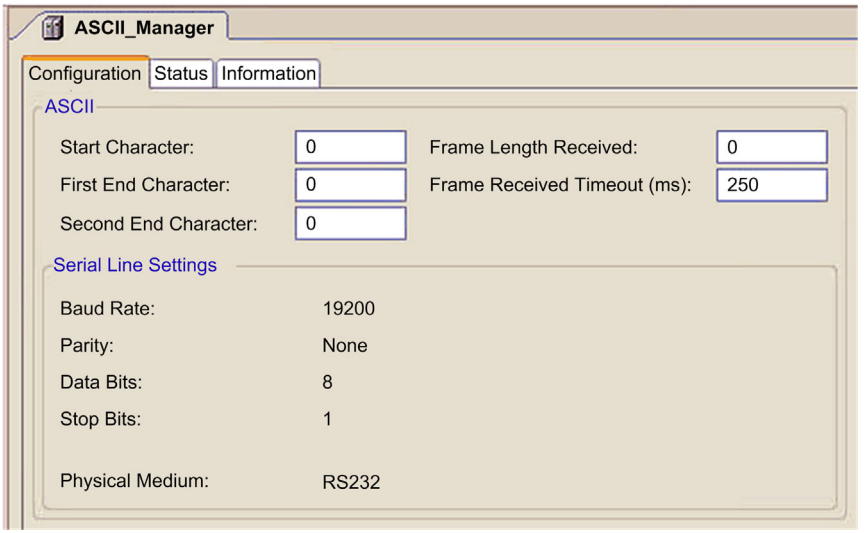
This table lists the serial line managers and their compatible functions:

Serial Line Manager	Transparent Communication	SMS
ASCII	PSTN or GSM modem	GSM modem SR2MOD03 with a specific ASCII configuration
Modbus	PSTN or GSM modem	no
EcoStruxure Machine Expert	Use a modem that can ignore protocol frames when a connection is not established. (Use a TDW33 PSTN modem.)	no

Adding a Manager to Your Serial Line

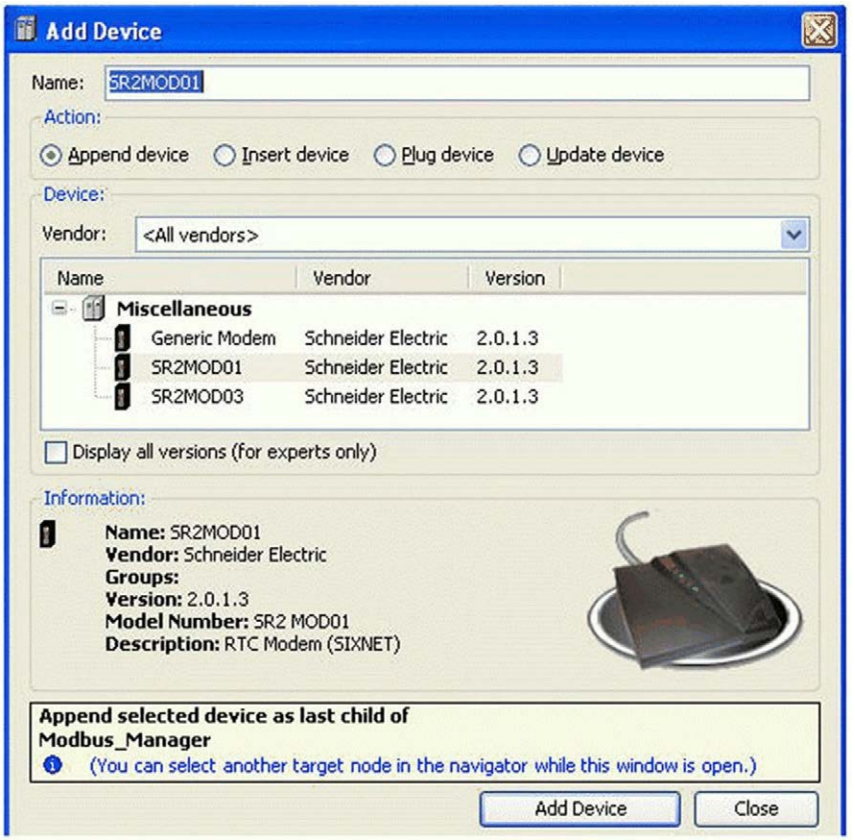
Follow this process to send or receive SMS through a GSM modem:

Step	Action
1	Add the appropriate manager to the serial line. (Refer to your controller's programming manual for details.)
2	Configure the manager for any required transparent communications.
3	An ASCII manager is required for SMS functionality. It is recommended that you set an End of Frame detection on Timeout (no End Character and no Frame Length). (Refer to the figure below.)

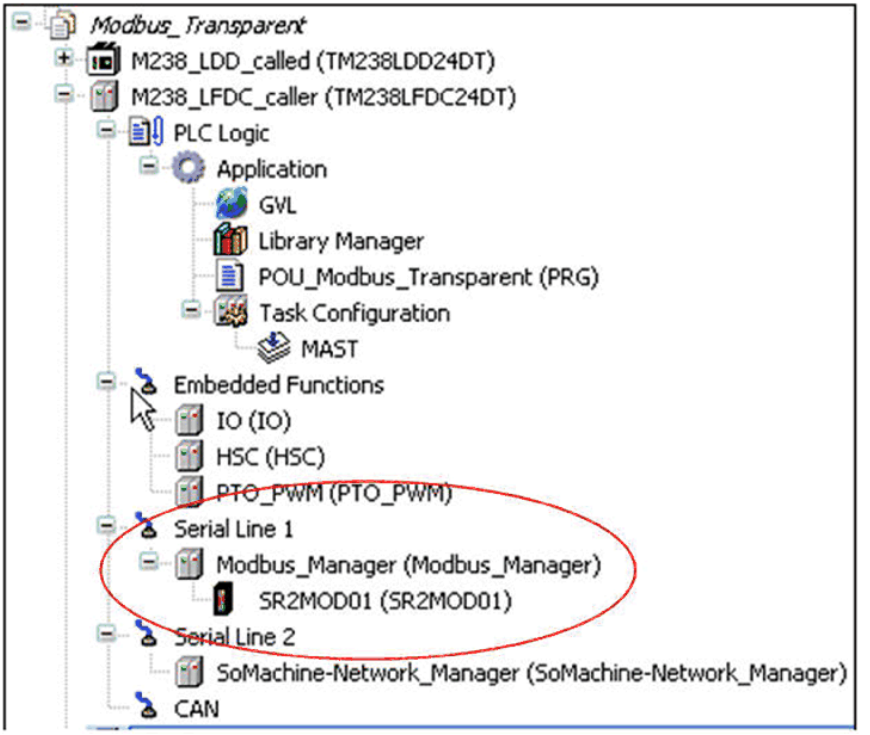


Adding a Modem to the Manager

Add the selected modem to the serial line manager configured in the **Add Device** dialog box:

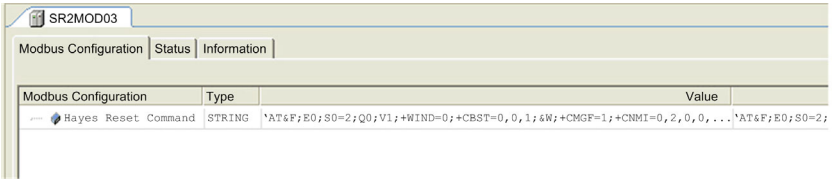


The modem appears under the serial line manager in the **Devices** tree:



Modem Editor

Double-click on the modem to open the device editor:



In the Configuration view, the `Hayes Reset Command` string is set by default.

For modems SR2MOD01 and TDW33 supported by Schneider Electric, this default command string is set to be used with the following serial line configuration:
Baud rate 19 200
Parity none
Data bits 8
Stop bit 1

If the serial line configuration is different, the command string must be adapted accordingly.

NOTE: The `Hayes Reset Command` is the modem initialization string that consists of a series of commands which are called Hayes ("AT") commands. This string is sent on the serial line during the application configuration (that is, after the controller's power on, application download, and reset warm or reset cold commands). If the modem answers **OK**, the connected modem appears with no error (green) in the Devices tree in online mode. Otherwise it appears as a detected error (red triangle).

NOTE: The modem can take several seconds to be ready.

Chapter 3
Function Block Descriptions

Introduction

This chapter describes the function blocks in the modem library.

What Is in This Chapter?

This chapter contains the following sections:

Section 3.1
Opening and Closing Transparent Communications

Introduction

Use the `Dial` and `HangUp` function blocks to open and close transparent communications between a controller and a remote device. In such a case, a modem is required at each end.

There are three types of managers you can add to a serial line to exchange requests:

- Modbus manager
- ASCII manager
- Machine Expert network manager

NOTE: An example that uses `Dial` and `HangUp` to open a Modbus connection between 2 controllers is included in EcoStruxure Machine Expert's examples. This file (**Modem_Modbus.project**) is accessible from the **Home** menu.

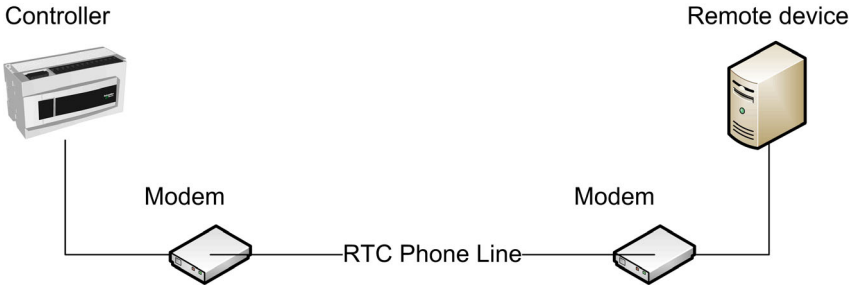
What Is in This Section?

This section contains the following topics:

Dial: Open Transparent Communications

Introduction

The controller can establish communications through a modem to a remote device with the `Dial` function block:



NOTE: Transparent communications are also possible using a GSM modem.

The Dial function block executes the `Dial` command to establish transparent communication between modems. Once `xDone` is TRUE, communication can start with the configured protocol (Modbus, EcoStruxure Machine Expert, or ASCII).

Graphical Representation

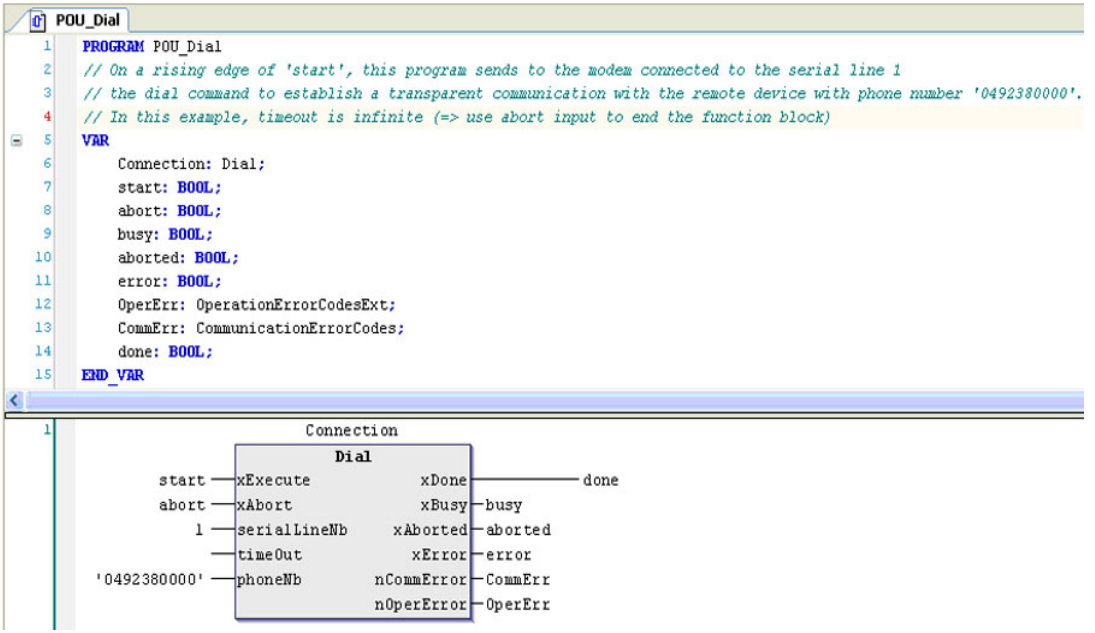
I/O Variables Description

Input	Type	Description
`phoneNb`	`STRING`	The `phoneNb` input contains the phone number of the modem being called.

NOTE: The input and output parameters that are common to all modem library function blocks are described elsewhere *(see page 11)*.

Example

This figure shows the declaration and use of the `Dial` function:



HangUp: Close Transparent Communications

Introduction

The HangUp function allows a controller to close a previously opened connection.

Graphical Representation

I/O Variables Description

The input and output parameters in the HangUp function block are those that are common to all modem library function blocks. They are described elsewhere *(see page 11)*.

Section 3.2
Sending and Receiving SMS

Introduction

The SMS-specific function blocks (`SendSMS` and `ReceiveSMS`) are used to establish a connection with a GSM Modem to send and receive SMS.

NOTE: An example of an SMS program that uses both the `SendSMS` and `ReceiveSMS` function blocks is included in EcoStruxure Machine Expert's examples. This file (**Modem_SMS.project**) is accessible from the **Home** menu.

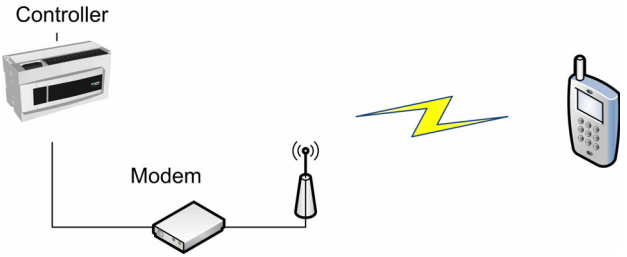
What Is in This Section?

This section contains the following topics:

SendSMS: Send SMS

Introduction

The SendSMS function block is used to establish a connection with a GSM modem and send an SMS to a specified receiver. For example, the controller can send SMS when a trigger is raised to transmit an alarm to a specified cell phone:



NOTE:
Be sure to have your GSM modem properly configured as follows:

- Make sure the SIM card in the modem is unlocked.
- Make sure the telephone number of the SMS center is valid.

You can use the `ConfigSim` function block to properly set these parameters from your application program.

Graphical Representation

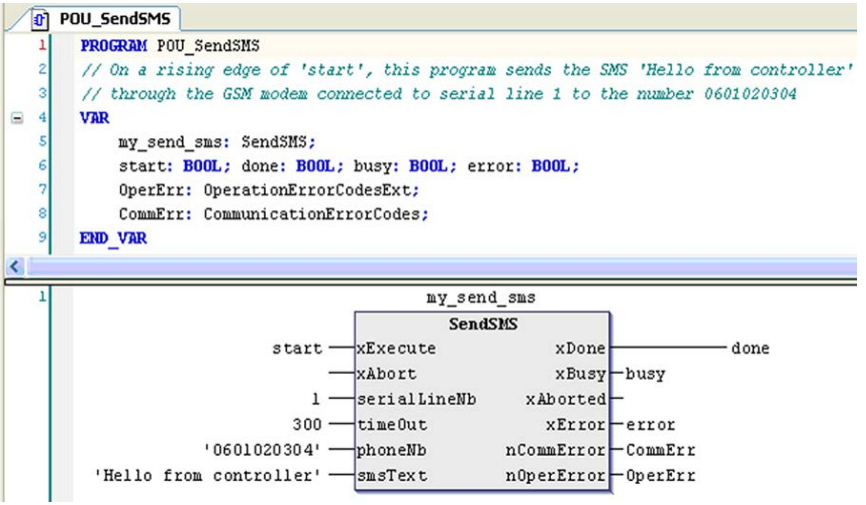
I/O Variables Description

Input	Type	Description
`phoneNb`	`STRING`	The `phoneNb` input contains the phone number of the receiver.
`smsText`	`STRING(255)`	The `smsText` input contains the body of the text message (255-character maximum).

The input and output parameters that are common to all modem library function blocks are described elsewhere *(see page 11)*.

Example

This figure shows the declaration and use of the `ReceiveSMS` function:



ReceiveSMS: Receive SMS

Introduction

The `ReceiveSMS` function block is used to wait for SMS received by a GSM Modem. For example, the controller can handle a command received in SMS from a specified cell phone.

NOTE:
Be sure to have your GSM modem properly configured as follows:

- Make sure the SIM card in the modem is unlocked.
- Make sure the telephone number of the SMS center is valid.

You can use the `ConfigSim` function block to properly set these parameters from your application program.

Graphical Representation

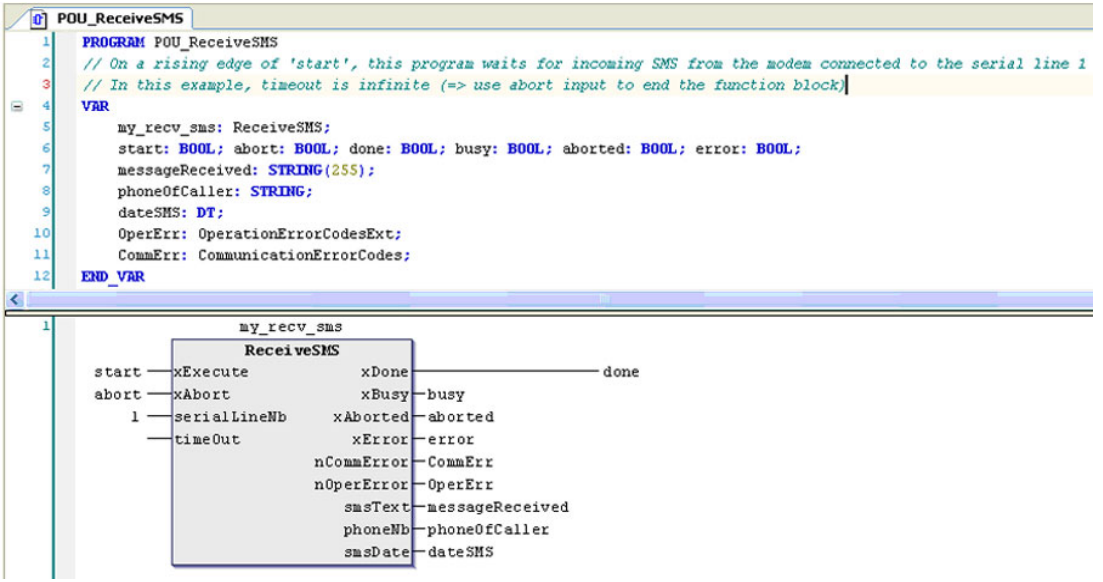
I/O Variables Description

Output	Type	Description
`smsText`	`STRING(255)`	The `smsText` output contains the body of the text message.
`phoneNb`	`STRING`	The `phoneNb` output contains the number of the phone that sent the SMS.
`smsDate`	`DATE_AND_TIME`	The `smsDate` output contains the date of the communication.

The input and output parameters that are common to all modem library function blocks are described elsewhere *(see page 11)*.

Example

This figure shows the declaration and use of the `ReceiveSMS` function



Section 3.3
GSM Modem SIM Card Services

ConfigSim

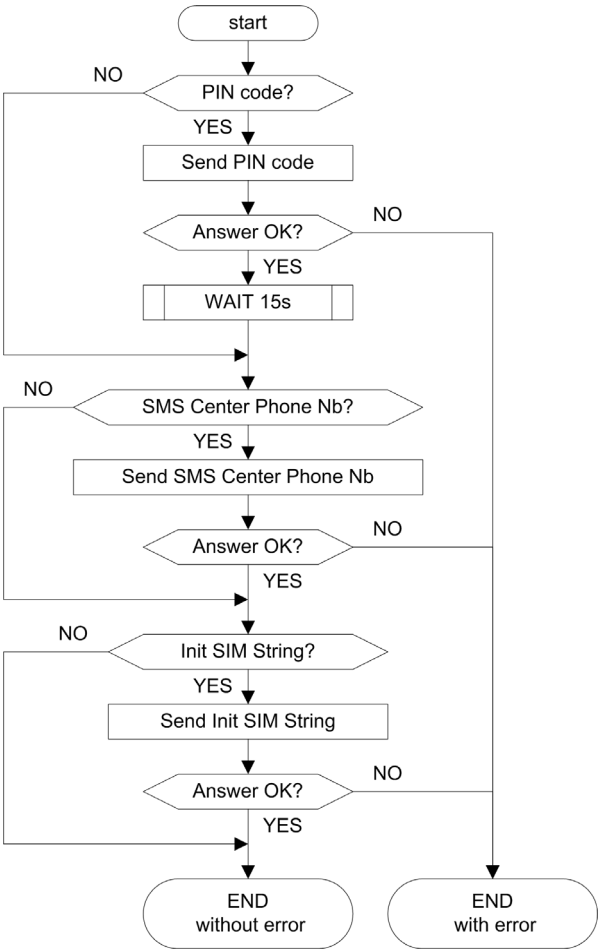
Introduction

Before using any other function block in the MODEM library, use the `ConfigSim` function block only when your GSM modem's SIM card requires one of these:

- Enter the PIN code.
- Configure the SMS center phone number.
- Send an initialization command.

You can then directly use one of the dedicated SMS function blocks.

Different commands are sent to the GSM Modem according to this flowchart:



⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

If an SR2MOD03 modem with a SIM card protected by a PIN code is used, the default initialization string must be modified in the Modem configuration editor. Replace the value of the Hayes Reset Command with this:

`'AT&F;E0;S0=2;Q0;V1;+WIND=0;+CBST=0,0,1;&W'`

and use the `ConfigSim` function block to send an additional initialization command with this:

`InitSimString` input = `'AT+CMGF=1;+CNMI=0,2,0,0,0;+CSAS'`.

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

Graphical Representation

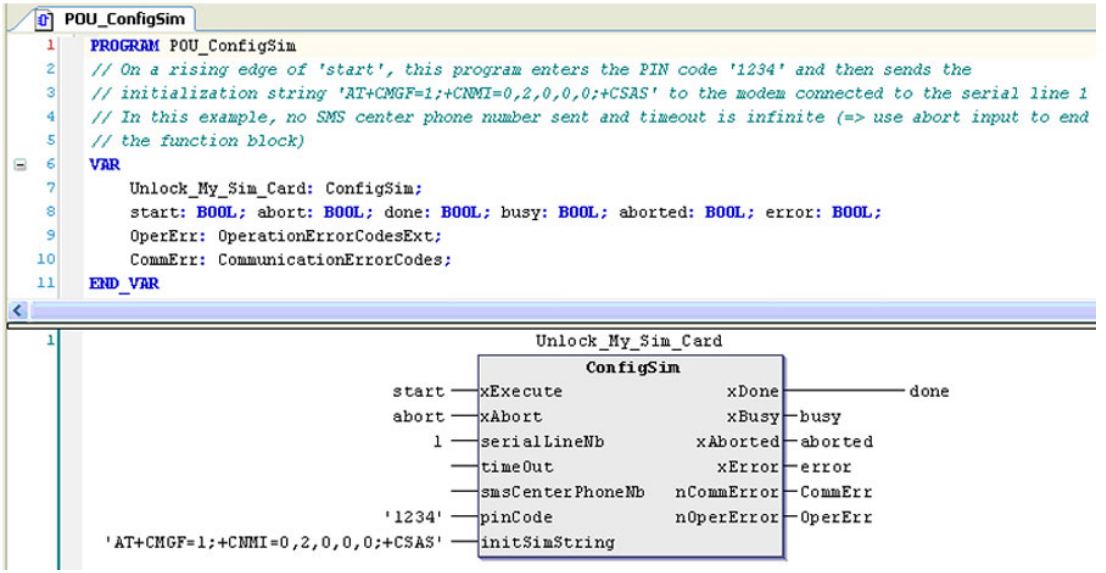
I/O Variables Description

Input	Type	Description
`smsCenterPhoneNb`	`STRING`	The `smsCenterPhoneNb` input contains the phone number of the SMS center to be configured in the SIM card. When empty, the SMS center phone number is not sent and the modem uses the actual number.
`pinCode`	`STRING(4)`	The `pinCode` input represents the SIM card's PIN code to be sent to unlock the SIM card. When `pinCode` is empty, no PIN code is sent.
`initSimString`	`STRING(255)`	The `initSimString` input represents the initialization string of the SIM card that is sent after the PIN and service center phone number have been sent. **NOTE:** For SR2MOD03, use this: `'AT+CMGF=1;+CNMI=0,2,0,0,0;+CSAS'`

The input and output parameters that are common to all modem library function blocks are described elsewhere *(see page 11)*.

Example

This figure shows the declaration and use of the `ConfigSim` function:



Appendices

Appendix A
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.

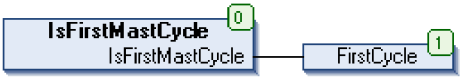
Functions `IsFirstMastCycle` and `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

Using a Function in IL Language

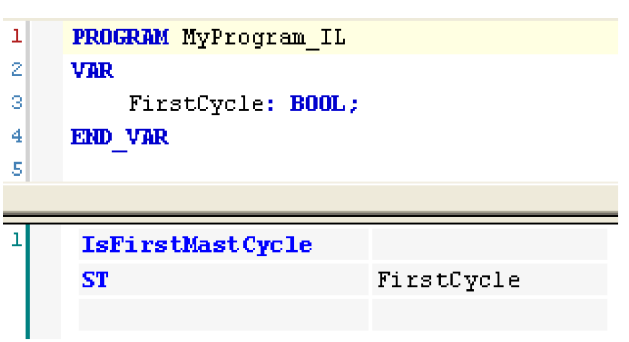
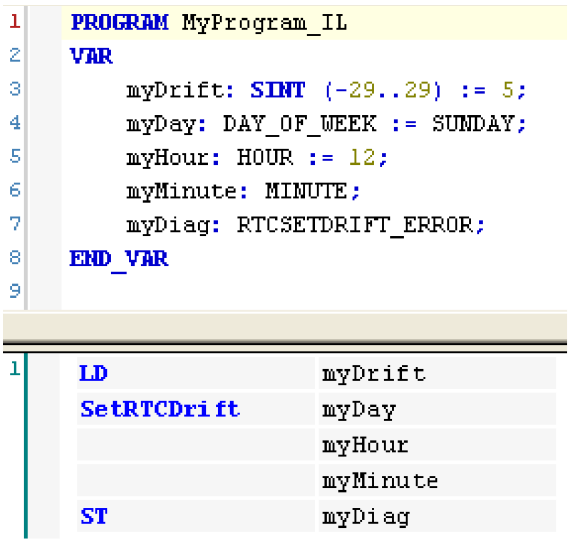
This procedure describes how to insert a function in IL language:

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 EcoStruxure Machine Expert, 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 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:

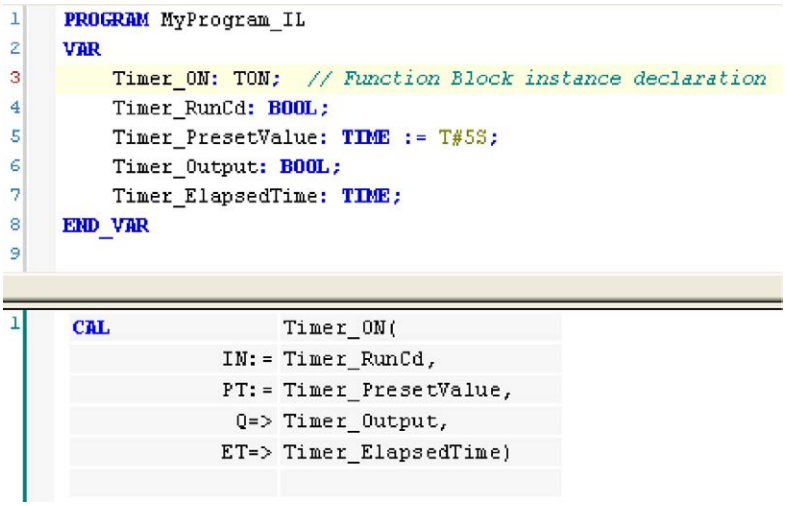
Step	Action
1	Open or create a new POU in Instruction List language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs *(see EcoStruxure Machine Expert, Programming Guide)*.

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 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 EcoStruxure Machine Expert, 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 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:

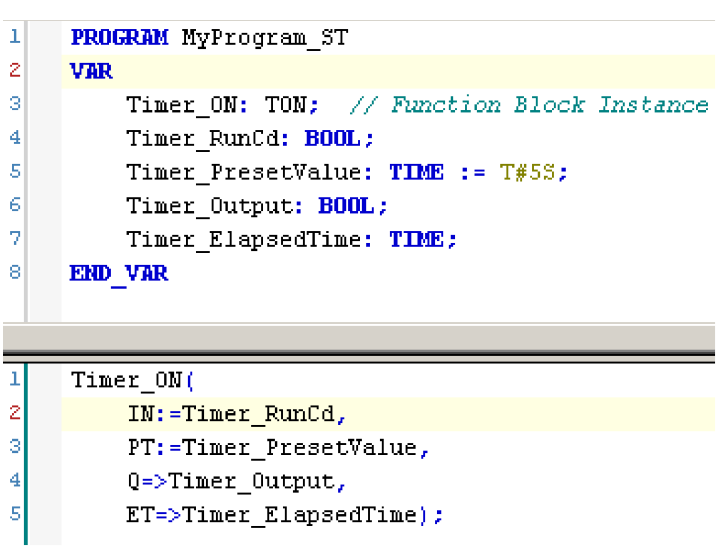
Step	Action
1	Open or create a new POU in Structured Text language. **NOTE:** The procedure to create a POU is not detailed here. For more information on adding, declaring and calling POUs, refer to the related documentation *(see EcoStruxure Machine Expert, Programming Guide)*.
2	Create the input and 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 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

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.

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

A programming unit that has 1 or more inputs and returns 1 or more outputs. FBs are called through an instance (function block copy with dedicated name and variables) and each instance has a persistent state (outputs and internal variables) from 1 call to the other.

Examples: timers, counters

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.

I

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.

S

SMS

(*short message service*) A standard communication service for telephones (or other devices) that send short text messages over the mobile communications system.

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.

V

variable

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

Index