SoMachine V3.0
ASCII Communications - Send/Receive Text
Communication_ASCII.project
Example Guide
04/2011

www.schneider-electric.com
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.

No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying, without express written permission of Schneider Electric.

All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to 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.

© 2011 Schneider Electric. All rights reserved.
# Table of Contents

SAFETY INFORMATION .................................................................................. 5

ABOUT THE BOOK ......................................................................................... 7

1. DESCRIPTION .............................................................................................. 13
   1.1. Presentation ............................................................................................ 13
   1.2. Main Features ........................................................................................ 14
   1.3. Functional Analysis of the Example ...................................................... 15
   1.4. Functions Used in this Example .............................................................. 16
   1.5. Hardware Installation .......................................................................... 17

2. DESCRIPTION OF THE EXAMPLE’S CONTENT .......................................... 18

3. CREATION OF THE PROJECT .................................................................... 20

4. SERIAL LINE ASCII MANAGER .................................................................. 22

5. LIBRARY MANAGER .................................................................................... 25

6. GLOBAL VARIABLE LIST (GVL) ................................................................ 28

7. ST, LD, OR CFC PROGRAM .......................................................................... 29
   7.1. ST Program ............................................................................................ 30
   7.2. LD Program ........................................................................................... 36
   7.3. CFC Program ........................................................................................ 45

8. CREATION OF THE VISU_ASCII VISUALIZATION .................................... 54

9. CONFIGURATION OF THE HYPERTERMINAL .......................................... 57

10. RUNNING THE EXAMPLE ........................................................................... 59
   10.1. MAST Task Configuration .................................................................. 59
   10.2. Downloading the Example to the Controller ...................................... 61
   10.3. Running the Example on the Controller ............................................. 64
Safety Information

Important Information

NOTICE

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

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

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

⚠️ DANGER

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

⚠️ WARNING

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

⚠️ CAUTION

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

⚠️ CAUTION

CAUTION, used without the safety alert symbol, indicates a potentially hazardous situation which, if not avoided, can result in equipment damage.

PLEASE NOTE

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

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

Document Scope

This document describes one of the SoMachine examples.

Since the example described in this document is intended for learning purposes only, it must not be run, nor tested, on products that are part of a machine or process.

Validity Note

This document has been updated with the release of SoMachine V3.0.

The technical characteristics of the device(s) described in this manual also appear online. To access this information online:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Go to <a href="http://www.schneider-electric.com">www.schneider-electric.com</a></td>
</tr>
<tr>
<td>2</td>
<td>In the Search box on the home page, type a model number. Do not type any blank spaces in the model number. To get information on a grouping of similar modules, you can use the characters **; do not use dots or xx's.</td>
</tr>
<tr>
<td>3</td>
<td>Under All, click Products → Product Datasheets and select the model number that interests you.</td>
</tr>
<tr>
<td>4</td>
<td>To save or print a data sheet as a .pdf file, click Export to PDF.</td>
</tr>
</tbody>
</table>

The characteristics presented in this manual should be the same as those that appear online. In line with our policy of constant improvement we may revise content over time to improve clarity and accuracy. In the event that you see a difference between the manual and online information, use the online information as your reference.

Related Documents

<table>
<thead>
<tr>
<th>Title of Documentation</th>
<th>Reference Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modicon LMC058 Motion Controller Hardware Guide</td>
<td>EIO0000000438 (ENG);</td>
</tr>
<tr>
<td></td>
<td>EIO0000000439 (FRE);</td>
</tr>
<tr>
<td></td>
<td>EIO0000000440 (GER);</td>
</tr>
<tr>
<td></td>
<td>EIO0000000441 (SPA);</td>
</tr>
<tr>
<td></td>
<td>EIO0000000442 (ITA);</td>
</tr>
<tr>
<td></td>
<td>EIO0000000443 (CHS)</td>
</tr>
</tbody>
</table>
### About the Book

<table>
<thead>
<tr>
<th>Title of Documentation</th>
<th>Reference Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoMachine Modbus and ASCII Read/Write Functions PLCCommunication Library Guide</td>
<td>EIO0000000361 (ENG); EIO0000000742 (FRE); EIO0000000743 (GER); EIO0000000744 (SPA); EIO0000000745 (ITA); EIO0000000746 (CHS)</td>
</tr>
<tr>
<td>Modicon LMC058 Motion Controller Programming Guide</td>
<td>EIO0000000408 (ENG); EIO0000000409 (FRE); EIO0000000410 (GER); EIO0000000411 (ITA); EIO0000000412 (SPA); EIO0000000413 (CHS)</td>
</tr>
</tbody>
</table>

### Product Related Information

This document and its related SoMachine project file focus on specific Functions and Function Blocks of the Schneider libraries provided with SoMachine, and on specific features available in SoMachine if these features are related to these libraries. They are intended to help you developing, testing, commissioning, and integrating applicative software of your own design on control systems.

It is intended for new SoMachine users who already have some degree of expertise in the design and programming of control systems.

⚠️ **WARNING**

**UNINTENDED EQUIPMENT OPERATION**

- Only use software approved by Schneider Electric for use with this equipment.
- Update your application program every time you change the physical hardware configuration.

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


Before You Begin

The products specified in this document have been tested under actual service conditions. Of course, your specific application requirements may be different from those assumed for this and any related examples described herein. In that case, you will have to adapt the information provided in this and other related documents to your particular needs. To do so, you will need to consult the specific product documentation of the hardware and/or software components that you may add or substitute for any examples specified in this documentation. Pay particular attention and conform to any safety information, different electrical requirements and normative standards that would apply to your adaptation.

WARNING

REGULATORY INCOMPATIBILITY

Be sure that all equipment applied and systems designed comply with all applicable local, regional and national regulations and standards.

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

The use and application of the information contained herein require expertise in the design and programming of automated control systems. Only the user or integrator can be aware of all the conditions and factors present during installation and setup, operation, and maintenance of the machine or process, and can therefore determine the automation and associated equipment and the related safeties and
interlocks which can be effectively and properly used. When selecting automation and control equipment, and any other related equipment or software, for a particular application, the user or integrator must also consider any applicable local, regional or national standards and/or regulations.

Some of the major software functions and/or hardware components used in the proposed architectures and examples described in this document cannot be substituted without significantly compromising the performance of your application. Further, any such substitutions or alterations may completely invalidate any proposed architectures, descriptions, examples, instructions, wiring diagrams and/or compatibilities between the various hardware components and software functions specified herein and in related documentation. You must be aware of the consequences of any modifications, additions or substitutions. A residual risk, as defined by EN/ISO 12100-1, Article 5, will remain if:

- it is necessary to modify the recommended logic and if the added or modified components are not properly integrated in the control circuit.
- you do not follow the required standards applicable to the operation of the machine, or if the adjustments to and the maintenance of the machine are not properly made (it is essential to strictly follow the prescribed machine maintenance schedule).
- the devices connected to any safety outputs do not have mechanically-linked contacts.

**CAUTION**

**EQUIPMENT INCOMPATIBILITY**

Read and thoroughly understand all device and software documentation before attempting any component substitutions or other changes related to the application examples provided in this document.

**Failure to follow these instructions can result in injury or equipment damage.**

**Start-up and Test**

Before using electrical control and automation equipment after design and installation, the application and associated functional safety system must be subjected to a start-up test by qualified personnel to verify correct operation of the equipment. It is important that arrangements for such testing be made and that enough time is allowed to perform complete and satisfactory testing.
**CAUTION**

**EQUIPMENT OPERATION HAZARD**
- Verify that all installation and set up procedures have been completed.
- Before operational tests are performed, remove all blocks or other temporary holding means used for shipment from all component devices.
- Remove tools, meters and debris from equipment.

*Failure to follow these instructions can result in injury or equipment damage.*

Verify that the completed system, including the functional safety system, is free from all short circuits and grounds, except those grounds installed according to local regulations. If high-potential voltage testing is necessary, follow the recommendations in equipment documentation to help prevent injury or equipment damage.

**Operation and Adjustments**

Regardless of the care exercised in the design and manufacture of equipment or in the selection and ratings of components, there are hazards that can be encountered if such equipment is improperly installed and operated.

In some applications, such as packaging machinery, additional operator protection such as point-of-operation guarding must be provided. This is necessary if the hands and other parts of the body are free to enter the pinch points or other hazardous areas where serious injury can occur. Software products alone cannot protect an operator from injury. For this reason, the software cannot be substituted for or take the place of point-of-operation protection.

**WARNING**

**UNGUARDED MACHINERY CAN CAUSE SERIOUS INJURY**
- Do not use this software and related automation equipment on equipment which does not have point-of-operation protection.
- Do not reach into machinery during operation.

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

Ensure that appropriate safeties and mechanical/electrical interlocks related to point-of-operation protection have been installed and are operational before placing the equipment into service. All interlocks and safeties related to point-of-operation protection must be coordinated with the related automation equipment and software programming.

**NOTE:** Coordination of safeties and mechanical/electrical interlocks for point-of-operation protection is outside the scope of the examples and implementations suggested herein.
It is sometimes possible to adjust the equipment incorrectly and this produce unsatisfactory or unsafe operation. Always use the manufacturer instructions as a guide to functional adjustments. Personnel who have access to these adjustments must be familiar with the equipment manufacturer instructions and the machinery used with the electrical equipment.

Only those operational adjustments actually required by the machine operator should be accessible to the operator. Access to other controls should be restricted to help prevent unauthorized changes in operating characteristics.

User Comments

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

1.1. Presentation

This example presents communications between two Data Terminal Equipments (DTE) using the ASCII protocol on a RS232 serial line. In this example, these exchanges are performed between an LMC058 Motion Controller and a PC. On the PC, a terminal such as HyperTerminal is used for exchanging text with the controller.

The controller’s program is created using SoMachine software.

In this example, the purpose of these communications consists in receiving and sending text characters.

Related SoMachine project: Communication_ASCII.project

Supported SoMachine Languages:
- ✔ CFC
- ✔ ST
- ✔ LD
- ✔ IL
- ✔ FBD
- ✔ SFC

Key features: ASCII communications for receiving and sending text characters on a RS232 serial line

Requirements: To use this example, the user must have:
- installed SoMachine V3.0 on a PC;
- run at least one Basic SoMachine example.

Note: Because it is required to run the communication functions used in this example’s programs on the controller, do not run this example in SIMULATION mode.
1.2. Main Features

The main features of this example include:

- Configuration of an RS232 ASCII serial connection using SoMachine.
- Addressing a serial connection using the PLCCommunication library.
- Continuous reception of ASCII text sent by a terminal (e.g. HyperTerminal) on this serial connection.
- Emission of ASCII text at the intent of the very same terminal (with abortion of the reception during this emission) on this serial connection.
- Use example of management of the ASCII texts sent to and received from this serial connection.
1.3. Functional Analysis of the Example

The program described in this example, whatever its programming language, performs the following treatments:

In this diagram, the **green boxes** indicate where the functions of the **PLCCommunication** library are used.

The **yellow box** manages the abortion of these functions in order to interrupt them during their operation.

Please note that this diagram is a simplified version of what is actually performed by the example described herein. Many intermediary actions are not shown above.
1. Description

1.4. Functions Used in this Example

The Functions (and Function Blocks) used in this example are listed below, grouped by library:

- **PLCCommunication** library (Schneider Electric)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Location in the Input Assistant</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDM</td>
<td>Convert a string into an address</td>
<td>Function Blocks ➜ {} SEN</td>
</tr>
<tr>
<td>SEND_RECV_MSG</td>
<td>Send and/or receive free messages</td>
<td></td>
</tr>
</tbody>
</table>

- **Toolbox** library (Schneider Electric)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Location in the Input Assistant</th>
</tr>
</thead>
<tbody>
<tr>
<td>String_TO_ArrayOfByte</td>
<td>Output array and ASCII value of the input string</td>
<td>Module Calls ➜ {} SE_TBX ➜ Numerical conversions</td>
</tr>
</tbody>
</table>

- **Standard** library (System)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Location in the Input Assistant</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONCAT</td>
<td>Do a concatenation (combination) of two strings</td>
<td>Module Calls ➜ {} Standard ➜ String Functions</td>
</tr>
<tr>
<td>DELETE</td>
<td>Remove a partial string from a larger string at a defined position</td>
<td></td>
</tr>
<tr>
<td>LEN</td>
<td>Return the length of a string</td>
<td></td>
</tr>
</tbody>
</table>

Please refer to the SoMachine online help.

Note: In the rest of this document, the former sentence instructs you to refer to the online help of SoMachine, accessible through its upper-right help button.

Please refer to the SoMachine online help for detailed information on these Functions and Function Blocks: Function description, Graphical representation, I/O Variables description, and more.

To install these libraries in your own project, please refer to *Library Manager* (page 25).
1.5. Hardware Installation

Required Devices

<table>
<thead>
<tr>
<th>N°</th>
<th>Designation</th>
<th>Reference</th>
<th>Use or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SoMachine Software</td>
<td>MSD CHNLMUA</td>
<td>SoMachine Software, 1-station license, installed on a PC</td>
</tr>
<tr>
<td></td>
<td>SoMachine Solution Extension Software</td>
<td>MSD CHLLMU30S0</td>
<td>SoMachine Solution Extension Software, 1-station license, installed on the same PC than SoMachine Software</td>
</tr>
<tr>
<td>2a</td>
<td>Terminal port/USB port cordset</td>
<td>TCS XCN AM UM3P</td>
<td>From the mini B USB port on the Modicon M238 base to the type A USB port on the PC terminal for programming and updating firmware; length: 3 m (10 ft)</td>
</tr>
<tr>
<td>2b</td>
<td>Programming cable</td>
<td>BMX XCA USB H018</td>
<td>Same as TCS XCN AM UM3P, but with two ground connections along the cable; length: 1.8 m (6 ft)</td>
</tr>
<tr>
<td>3</td>
<td>Motion controller Modicon LMC058</td>
<td>LMC058 •••••••</td>
<td>Compact base motion controller</td>
</tr>
<tr>
<td>4</td>
<td>Cordset for DTE terminal (printer) with 1 RJ45 connector and 1 SUB-D9 female connector</td>
<td>TCS MCN 3M4F3C2</td>
<td>From the Serial port marked MBS on the LMC058 controller to the SUB-D9 male RS-232 connector of a PC (COM1 port); length 3 m (10 ft)</td>
</tr>
<tr>
<td>5</td>
<td>HyperTerminal software</td>
<td></td>
<td>Communication software furnished with any version of the Windows operating system</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The PC on which this software is used must have a SUB-D9 male RS-232 communication port (COM1)</td>
</tr>
</tbody>
</table>

Please refer to the *Modicon LMC058 Motion Controller Hardware Guide* for the hardware setup of this device.
2. Description of the Example’s Content

In SoMachine, the configuration of the example is made with the following devices:

- **1 Motion Controller**: LMC058LF42S0
- **1 ASCII Manager** on the controller to configure its serial line as a RS232 point-to-point connection

The program of the LMC058 controller is made of the following items:

- **Library Manager**: List of the libraries linked to the programs of this example.
- **Global variable List (GVL)**: Definitions of the constants used by the CFC, LD, and ST programs. It also includes the definitions of the variables exchanged between the **Visu_ASCII visualization** and the CFC, LD, or ST program actually run on the controller.
- **ST program**: Contains the source code for implementing what is described in *Functional Analysis of the Example (page 15)*. This is the default program since it is called by the MAST task of the controller.
- **LD program**: Translation of the ST program into LD language. To run this program on the controller, instead of the ST program, change the POU called by the MAST task of the controller from **PLC_PRG_ST** to **PLC_PRG_LD**.
- **CFC program**: Translation of the ST program into CFC language. To run this program on the controller, instead of the ST program, change the POU called by the MAST task of the controller from **PLC_PRG_ST** to **PLC_PRG_CFC**.
- **Task Configuration**: The standard MAST task, cyclically called every 20ms.
- **Visualization Manager**: Settings of the visualization of this example.
- **Visu_ASCII visualization**: This visualization runs on SoMachine, in online mode, to exchange user-level data and commands with the controller. It is specifically designed for testing this example.
- **Serial Line**: Configured as an **ASCII_Manager**.
Example’s content, visible in the Devices panel of the Program tab:
3. Creation of the Project

The steps listed in the following table describe how to create the SoMachine project, and how to set up the device(s) used in this example. No details are given here since it is assumed that you already know the Basic commands of SoMachine.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the Create new machine part of the Home tab, select Start with empty project to create a new SoMachine project. Give this new project the following name: Communication_ASCII.</td>
</tr>
<tr>
<td>2</td>
<td>In the Configuration tab, add an LMC058LF424S0 Motion Controller.</td>
</tr>
</tbody>
</table>

Note: Details on the selected controller are displayed in the Information section of SoMachine.

SoMachine controller version: Defines the version of the selected controller; it is displayed in the Information section of SoMachine.

Target controller firmware version: Defines the firmware version of your controller. This version is shown when you select your controller's node, as shown as in Downloading the Example to the Controller (see page 62).

For compatibility purposes between a SoMachine controller version and a target controller firmware version, only the first three numbers of a version must be identical. In the preceding picture, the 2.0.2.27 SoMachine controller version is compatible with any 2.0.2.•• target controller firmware version.

For each controller model, SoMachine only presents the latest available version. If you check the Display all versions (for expert only) option, SoMachine will list all supported controller firmware versions. However, a good practice consists in using the latest available version and updating the firmware of your controller, if required. Please refer to the Modicon LMC058 Motion Controller Programming Guide.
3. Creation of the Project

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Rename this controller to <strong>LMC058_Controller</strong>.</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Diagram of LMC058_Controller" /></td>
</tr>
<tr>
<td></td>
<td><strong>Information</strong></td>
</tr>
<tr>
<td></td>
<td>Name: LMC058_Controller(LMC058LF42S0)</td>
</tr>
<tr>
<td></td>
<td>Vendor: Schneider Electric</td>
</tr>
<tr>
<td></td>
<td>Version: 2.0.2.27</td>
</tr>
<tr>
<td></td>
<td>Order #: LMC056LF42S0</td>
</tr>
<tr>
<td></td>
<td>Description: LMC058 Performance Motion controller for Solutions - 2 x 5 fast sink input (200kHz, 24Vdc), 2 x 2 fast push-pull outputs (100kHz, 24Vdc, 0.2 A), 2 x 2 sink inputs (24Vdc), 12 sink inputs (24Vdc) and 12 source outputs (24Vdc, 0.5A), 1 Ethernet port, 1 serial line port, 1 CANopen master, 1 CANmotion master and 1 Encoder connector. Timer and calendar. Removable terminal blocks.</td>
</tr>
<tr>
<td>4</td>
<td>Save your new project.</td>
</tr>
</tbody>
</table>
4. Serial Line ASCII Manager

The steps listed in the following table describe how to add and configure the ASCII Manager of the LMC058 controller.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Click on the <strong>SoMachine-Network_Manager</strong> Serial Line port of the LMC058 controller:</td>
</tr>
</tbody>
</table>

![Diagram of LMC058 Controller](image1)

This will remove this Manager and propose you to replace it by another Serial Line Manager.

| 2    | Confirm the removal of the **SoMachine-Network_Manager** by clicking on the **OK** button: |

![Confirmation Window](image2)

| 3    | The **Add device** window is then automatically displayed to allow you to replace the Manager you just removed. |

In this window, select the **ASCII_Manager** device and click on the **Add and close** button:

![Add Device Window](image3)

This will configure the Serial Line port of the LMC058 controller as an **ASCII_Manager**.
### 4. Serial Line ASCII Manager

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>To open the hardware configuration of the LMC058 controller, double-click on its picture:</td>
</tr>
</tbody>
</table>
| 5    | In the left-hand panel, select:  
  - **Communication** menu  
  - **Serial Line**  
  - **Physical Settings** |
| 6    | In the **Configuration** tab of the central panel, set the configuration shown below:  
  - Baud rate ............... 19200  
  - Parity ................ None  
  - Data bits .............. 8  
  - Stop bits .............. 1  
  - Physical Medium ...... RS 232 |
| 7    | In the left-hand panel, select:  
  - **Communication** menu  
  - **Serial Line**  
  - **Protocol Settings** |
### 4. Serial Line ASCII Manager

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 8    | In the **Configuration** tab of the central panel, set the configuration shown below:  
  - Start Character ....................... 0 (no *start of frame* character)  
  - First End Character .............. 13 (*end of frame* = Carriage Return (CR))  
  - Second End Character .......... 0 (no additional *end of frame* character)  
  - Frame Length Received .......... 0 (not used)  
  - Frame Received Timeout (ms) .... 0 (not used)  
  
  **Note:** The *First End Character* is used by the controller to detect the end of any frame (or message) received on the serial line. It is also appended by the controller at the end of any frame (or message) sent on the serial line. In this example, the default *First End Character* has been changed from 10 (for Line Feed (LF)) to 13 (for Carriage Return (CR)). |

![](image1.png)

Depending on the needs of your own applications, you may have to change these settings.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Click on the <strong>Back</strong> button to revert to the main <strong>Configuration</strong> tab.</td>
</tr>
</tbody>
</table>
5. Library Manager

The steps listed in the following table describe how to add and/or check the list of the libraries linked to this example.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select the <strong>Program</strong> tab. In the <strong>Devices</strong> tree view, double-click on the <strong>Library Manager</strong> to open the list of the libraries linked to the <strong>Application</strong> software of this example.</td>
</tr>
<tr>
<td>2</td>
<td>Check that the <strong>PLCCommunication</strong> and <strong>Standard</strong> libraries are already linked, as shown below:</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Library Manager screenshot" /></td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: These libraries are grayed to inform that they have been automatically linked to the program upon addition of the LMC058 controller to the project and that they cannot be removed.</td>
</tr>
<tr>
<td>3</td>
<td>Click on the <strong>Add library...</strong> command of the <strong>Library Manager</strong>.</td>
</tr>
</tbody>
</table>
5. Library Manager

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>In the Add Library window:</td>
</tr>
<tr>
<td></td>
<td>- Select the Toolbox library, located in the Util category of libraries provided by Schneider Electric.</td>
</tr>
<tr>
<td></td>
<td>- Click on OK.</td>
</tr>
</tbody>
</table>

In the Add Library window:
- Select the Toolbox library, located in the Util category of libraries provided by Schneider Electric.
- Click on OK.
This adds the Toolbox library to the list of libraries linked to the project.

In this example, only the `String_TO_ArrayOfByte` function of this library is used.

**Note:** In the Library Manager, the name of this library is colored in **black** to inform that it has been manually linked to the program and that it can be removed.
6. Global Variable List (GVL)

The steps listed in the following table describe how to implement the Global variable List (GVL) used in this example.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the <strong>Devices</strong> tree view, double-click on the <strong>GVL</strong> to open it.</td>
</tr>
<tr>
<td>2</td>
<td>In the <strong>GVL</strong> editor, replace its default content, shown below…</td>
</tr>
</tbody>
</table>

… by the following variables and constants declarations:

```plaintext
VAR_GLOBAL CONSTANT
// Size of the Buffer for receiving ASCII text from the Serial Link
c_Size_ReceptionBuffer  : INT := 80;
// Size of the Buffer for sending ASCII text on the Serial Link
c_Size_EmissionBuffer   : INT := 255;
// Number of lines used to memorized the text received from the serial link
c_NumberOfLines         : INT := 10;
// Number of characters per line of memorized text received from the serial link
c_NumberOfCharacters    : INT := 80;
END_VAR

VAR_GLOBAL
// USER command: Clear the contents of the Cumulated Reception STRING
g_xCmdClearReceivedText : BOOL := FALSE;

// USER command: Send the current STRING to send
g_xCmdSendCurrentText   : BOOL := FALSE;

// Cumulated Reception STRING: Array of 80-character lines for the text received
// from the serial link
g_sTextReceivedFromSL   : ARRAY [1..c_NumberOfLines] OF STRING(c_NumberOfCharacters) := [10('')];

// STRING to send on the serial link (default size of 80 characters)
g_sTextToSendOnSL       : STRING := '';

// Error on Text reception
g_xErrorReception       : BOOL := FALSE;

// Error on Text emission
g_xErrorEmission        : BOOL := FALSE;
END_VAR
```
7. ST, LD, or CFC Program

Each of the following three chapters describes how to create the program used in the example. Choose the language of your program (CFC, LD, or ST) and go to the corresponding chapter:

- ST Program ........................................................................................................... 30
- LD Program .......................................................................................................... 36
- CFC Program ...................................................................................................... 45

You only need to write your SoMachine program in one of these three languages.

In addition, each of these three chapters begins with explanations on the difficulties you may face, if any, in the form of optional steps.
## 7.1. ST Program

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Creation of the POU:</strong> Create a new POU in ST language, called <strong>PLC_PRG_ST</strong>.</td>
</tr>
</tbody>
</table>

Upon creation of this POU, it is automatically opened by SoMachine.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td><strong>ST variables:</strong> In the upper part of the ST editor, declare the following variables:</td>
</tr>
</tbody>
</table>

```plaintext
PROGRAM PLC_PRG_ST
VAR
  // Command to execute the rest of this program; initially at FALSE; set to TRUE at
  // the end of the 1st cycle
  v_xExecuteProgram          : BOOL := FALSE;

  **********************
  (** ADDM Variables **)
  **********************

  // ADDM Function Block for formatting the address of the ASCII serial link
  ADDM_ASCII_SL               : ADDM;

  // ADDRESS structure for the address of the ASCII serial link
  v_addressAsciiSL            : ADDRESS;

  // "Done" result of the Address conversion
  v_xAddressDone              : BOOL := FALSE;

  // "Error" result of the Address conversion
  v_xAddressError             : BOOL := FALSE;

  // Result of the Address conversion: OK if "Done" without any "Error"
  v_xSLAddressIsOK            : BOOL := FALSE;

  **********************
  (** RECEIVE ASCII text **)
  **********************

  // Flag used to enable (TRUE) or disable (FALSE) the Reception
  v_xEnableReception          : BOOL := TRUE;

  // Command to process the SEND_RECV_MSG function block for receiving text
  v_xReceiveText              : BOOL := FALSE;

  // Command to abort the current reception
```


<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>v_xAbortReception : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// SEND RECV MSG Function Block for receiving text from the ASCII serial link</td>
</tr>
<tr>
<td></td>
<td>SEND_RECV_MSG_RECEIVE : SEND_RECV_MSG;</td>
</tr>
<tr>
<td></td>
<td>// Buffer for receiving BYTEs from the Serial Link</td>
</tr>
<tr>
<td></td>
<td>v_abReceptionBuffer : ARRAY [1..GVL.c_Size_ReceptionBuffer] OF BYTE := [80(16#00)];</td>
</tr>
<tr>
<td></td>
<td>// &quot;Done&quot; result of the ASCII reception</td>
</tr>
<tr>
<td></td>
<td>v_xReceptionDone : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// &quot;Busy&quot; output of the ASCII reception</td>
</tr>
<tr>
<td></td>
<td>v_xReceptionBusy : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// &quot;Aborted&quot; result of the ASCII reception</td>
</tr>
<tr>
<td></td>
<td>v_xReceptionAborted : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// &quot;Error&quot; result of the ASCII reception</td>
</tr>
<tr>
<td></td>
<td>v_xReceptionError : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Number of character(s) to delete</td>
</tr>
<tr>
<td></td>
<td>v_iNumberOfCharToDelete : INT := 0;</td>
</tr>
<tr>
<td></td>
<td>// Number of BYTEs received in the Reception Buffer</td>
</tr>
<tr>
<td></td>
<td>v_wNumberOfReceivedBytes : WORD := 0;</td>
</tr>
<tr>
<td></td>
<td>// Index of the currently received and analyzed BYTE</td>
</tr>
<tr>
<td></td>
<td>v_wCurrentByteIndex : WORD := 0;</td>
</tr>
<tr>
<td></td>
<td>// Total length of the ASCII text received from the Serial Link</td>
</tr>
<tr>
<td></td>
<td>v_iTextReceivedFromSLLength : INT := 0;</td>
</tr>
<tr>
<td></td>
<td>// Character currently appended at the end of the Cumulated Reception STRING</td>
</tr>
<tr>
<td></td>
<td>v_sCurrentCharacter : STRING(1) := '';</td>
</tr>
<tr>
<td></td>
<td>// Pointer on this character</td>
</tr>
<tr>
<td></td>
<td>v_pbCurrentCharacter : POINTER TO BYTE;</td>
</tr>
<tr>
<td></td>
<td>// Current line for receiving text from the serial link</td>
</tr>
<tr>
<td></td>
<td>v_iCurrentLine : INT := 0;</td>
</tr>
<tr>
<td></td>
<td>// Temporary line index</td>
</tr>
<tr>
<td></td>
<td>v_iLine : INT := 0;</td>
</tr>
<tr>
<td></td>
<td>(***) SEND ASCII text ***</td>
</tr>
<tr>
<td></td>
<td>(***) SEND ASCII text ***</td>
</tr>
<tr>
<td></td>
<td>// Flag used to enable (TRUE) or disable (FALSE) the Emission</td>
</tr>
<tr>
<td></td>
<td>v_xEnableEmission : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Command to process the SEND_RECV_MSG function block for sending text</td>
</tr>
<tr>
<td></td>
<td>v_xSendText : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Length of the ASCII text to send on the serial link</td>
</tr>
<tr>
<td></td>
<td>v_iTextToSendOnSLLength : INT := 0;</td>
</tr>
<tr>
<td></td>
<td>// Number of BYTEs to send on the serial link</td>
</tr>
<tr>
<td></td>
<td>v_uiNumberOfBytesToSend : UINT := 0;</td>
</tr>
<tr>
<td></td>
<td>// SEND RECV MSG Function Block for sending text on the ASCII serial link</td>
</tr>
<tr>
<td></td>
<td>SEND_RECV_MSG_SEND : SEND_RECV_MSG;</td>
</tr>
<tr>
<td></td>
<td>// Buffer for receiving BYTEs from the Serial Link</td>
</tr>
<tr>
<td></td>
<td>v_abEmissionBuffer : ARRAY [1..GVL.c_Size_EmissionBuffer] OF BYTE := [255(16#00)];</td>
</tr>
<tr>
<td></td>
<td>// &quot;Done&quot; result of the ASCII emission</td>
</tr>
<tr>
<td></td>
<td>v_xEmissionDone : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// &quot;Busy&quot; output of the ASCII emission</td>
</tr>
<tr>
<td></td>
<td>v_xEmissionBusy : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// &quot;Error&quot; result of the ASCII emission</td>
</tr>
<tr>
<td></td>
<td>v_xEmissionError : BOOL := FALSE;</td>
</tr>
</tbody>
</table>

END_VAR
### 7. ST, LD, or CFC Program

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td><strong>ST program:</strong> In the lower part of the ST editor, implement the following program:</td>
</tr>
</tbody>
</table>

```plaintext
(* ******************************************************************************
(* Formatting the Address of the ASCII Serial Link *)
(* Function Block(s): ADDM *)
(* ******************************************************************************)

ADDM_ASCII_SL(
    AddrTable := v_addressAsciiSL ,  // IN/OUT - Resulting ADDRESS structure
    Execute   := v_xExecuteProgram,  // IN     - Rising Edge signal that triggers this Function Block
    Addr      := '1'              ,  // IN     - ASCII Serial Address Format = '<communication port number>' ('1' for the 1st Serial Link)
    Done      => v_xAddressDone   ,  // OUT    - Resulting address is OK
    Error     => v_xAddressError  ,  // OUT    - Resulting address is not OK
    (*CommError*)                 ); // OUT    - This error code is not used in this program

(* Result of the Address conversion: OK if 'Done' without any 'Error' *)
v_xSLAddressIsOK := v_xAddressDone AND NOT v_xAddressError;

(* ******************************************************************************
(* Receiving Text from the ASCII Serial Link *)
(* Function Block(s): SEND_RECV_MSG *)
(* ******************************************************************************)

(* Command to process the SEND_RECV_MSG function block for receiving text *)
v_xReceiveText := v_xSLAddressIsOK AND v_xEnableReception;

(* In presence of a USER command to send text on the serial link *)
IF GVL.g_xCmdSendCurrentText THEN
    (* The current reception is ABORTED in order to free the serial link, *)
    (* thus allowing the controller to use the serial link to send text *)
    v_xAbortReception := TRUE;
END_IF

(* SEND_RECV_MSG Function Block for receiving text from the ASCII serial link *)
(* Note: In ST language, this call syntax is required for Function Blocks of the 'PLCCommunication' library that use an 'Addr' INPUT variable (data type = ADDRESS) *)
SEND_RECV_MSG_RECEIVE(
    Execute         := v_xReceiveText,              // IN  - Rising Edge signal that triggers this Function Block
    Abort           := v_xAbortReception,           // IN  - Command for aborting the ongoing operation
    Addr            := v_addressAsciiSL,            // IN  - Formatted address of the controller's Serial Link
    Timeout         := 0,                           // IN  - Exchange timeout = 0 (for infinite) because, in this example, the Serial Link is continuously polled
    QuantityToSend  := 0,                           // IN  - Number of bytes to send = 0 (for a receive-only use of this Function Block)
    BufferToSend    := 0,                           // IN  - Address of the buffer for emission = 0 (for a receive-only use of this Function Block)
    SizeRecvBuffer  := SIZEOF(v_abReceptionBuffer), // IN  - Available size (in bytes) of the reception buffer (TOTAL size)
    BufferToRecv    := ADR(v_abReceptionBuffer),    // IN  - Address of the reception buffer
    Done            => v_xReceptionDone,            // OUT - "Done" result of the ASCII reception
    Busy            => v_xReceptionBusy,            // OUT - "Busy" output of the ASCII reception
    Aborted         => v_xReceptionAborted,         // OUT - "Aborted" result of the ASCII reception
    Error           => v_xReceptionError);          // OUT - "Error" result of the ASCII reception
    (*CommError*)                                   // OUT - This error code is not used
```
<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(* OperError*) // in this program</td>
</tr>
<tr>
<td></td>
<td>(* When the current reception is finished... *)</td>
</tr>
<tr>
<td></td>
<td>IF v_xReceiveText AND NOT v_xReceptionBusy THEN</td>
</tr>
<tr>
<td></td>
<td>(* If it is &quot;Aborted&quot;, then the serial link is considered as being &quot;freed&quot; and can *)</td>
</tr>
<tr>
<td></td>
<td>(* be used by the controller for sending characters without provoking an error *)</td>
</tr>
<tr>
<td></td>
<td>IF v_xReceptionAborted THEN</td>
</tr>
<tr>
<td></td>
<td>(* The Reception is disabled (it will be enabled once the emission is finished*)</td>
</tr>
<tr>
<td></td>
<td>v_xEnableReception := FALSE;</td>
</tr>
<tr>
<td></td>
<td>(* The serial link is &quot;freed&quot; and can be used for emission *)</td>
</tr>
<tr>
<td></td>
<td>v_xEnableEmission := TRUE;</td>
</tr>
<tr>
<td></td>
<td>(* If it is &quot;Done&quot; (OK), the received BYTESs are appended at the end of the Cumulated</td>
</tr>
<tr>
<td></td>
<td>Reception STRING *)</td>
</tr>
<tr>
<td></td>
<td>ELSIF NOT v_xReceptionError AND v_xReceptionDone THEN</td>
</tr>
<tr>
<td></td>
<td>(* Reception state = OK *)</td>
</tr>
<tr>
<td></td>
<td>GVL.g_xErrorReception := FALSE;</td>
</tr>
<tr>
<td></td>
<td>(* Number of BYTESs received in the Reception Buffer (internal variable *)</td>
</tr>
<tr>
<td></td>
<td>(* of the &quot;SEND_RECV_MSG_RECEIVE&quot; Function Block instance) *)</td>
</tr>
<tr>
<td></td>
<td>v_wNumberOfReceivedBytes := SEND_RECV_MSG_RECEIVE.NbRecvBytes;</td>
</tr>
<tr>
<td></td>
<td>(* If the Cumulated Reception STRING is full, the oldest line is removed and all</td>
</tr>
<tr>
<td></td>
<td>lines are shifted up *)</td>
</tr>
<tr>
<td></td>
<td>IF (v_iCurrentLine &gt;= GVL.c_NumberOfLines) THEN</td>
</tr>
<tr>
<td></td>
<td>FOR v_iLine := 1 TO (GVL.c_NumberOfLines - 1) DO</td>
</tr>
<tr>
<td></td>
<td>GVL.g_sTextReceivedFromSL[v_iLine] := GVL.g_sTextReceivedFromSL[v_iLine + 1];</td>
</tr>
<tr>
<td></td>
<td>END_FOR</td>
</tr>
<tr>
<td></td>
<td>(* For updating the last line *)</td>
</tr>
<tr>
<td></td>
<td>v_iCurrentLine := GVL.c_NumberOfLines;</td>
</tr>
<tr>
<td></td>
<td>(* Otherwise, the next line is updated *)</td>
</tr>
<tr>
<td></td>
<td>ELSE</td>
</tr>
<tr>
<td></td>
<td>v_iCurrentLine := v_iCurrentLine + 1;</td>
</tr>
<tr>
<td></td>
<td>END_IF</td>
</tr>
<tr>
<td></td>
<td>(* The current line is reset *)</td>
</tr>
<tr>
<td></td>
<td>GVL.g_sTextReceivedFromSL[v_iCurrentLine] := '';</td>
</tr>
<tr>
<td></td>
<td>(* Pointer on the current character (for BYTE to STRING conversion purposes) *)</td>
</tr>
<tr>
<td></td>
<td>v_pbCurrentCharacter := ADR(v_sCurrentCharacter);</td>
</tr>
<tr>
<td></td>
<td>(* Each BYTE received from the Serial Link is appended at the end of the</td>
</tr>
<tr>
<td></td>
<td>Cumulated Reception STRING *)</td>
</tr>
<tr>
<td></td>
<td>FOR v_wCurrentByteIndex := 1 TO v_wNumberOfReceivedBytes DO</td>
</tr>
<tr>
<td></td>
<td>(* Current length of the Cumulated Reception STRING *)</td>
</tr>
<tr>
<td></td>
<td>v_iTextReceivedFromSLLength := LEN(GVL.g_sTextReceivedFromSL[v_iCurrentLine]);</td>
</tr>
<tr>
<td></td>
<td>(* In this example, if the line of the Cumulated Reception STRING is already</td>
</tr>
<tr>
<td></td>
<td>full (Max. size reached), *)</td>
</tr>
<tr>
<td></td>
<td>(* its oldest character (i.e. the 1st character) is removed for making room</td>
</tr>
<tr>
<td></td>
<td>for the new character *)</td>
</tr>
<tr>
<td></td>
<td>IF (v_iTextReceivedFromSLLength &gt;= GVL.c_NumberOfCharacters) THEN</td>
</tr>
<tr>
<td></td>
<td>(* Removes the 1st character of the Cumulated Reception STRING *)</td>
</tr>
<tr>
<td></td>
<td>GVL.g_sTextReceivedFromSL[v_iCurrentLine] :=</td>
</tr>
<tr>
<td></td>
<td>DELETE(GVL.g_sTextReceivedFromSL[v_iCurrentLine], 1, 1);</td>
</tr>
<tr>
<td></td>
<td>END_IF</td>
</tr>
<tr>
<td></td>
<td>(* Conversion from BYTE to (character of) STRING performed with the help of a</td>
</tr>
<tr>
<td></td>
<td>POINTER *)</td>
</tr>
<tr>
<td></td>
<td>v_pbCurrentCharacter* := v_abReceptionBuffer[v_wCurrentByteIndex];</td>
</tr>
<tr>
<td></td>
<td>(* The currently analyzed character is appended at the end of the line of the</td>
</tr>
<tr>
<td></td>
<td>Cumulated Reception STRING *)</td>
</tr>
<tr>
<td></td>
<td>GVL.g_sTextReceivedFromSL[v_iCurrentLine] :=</td>
</tr>
<tr>
<td></td>
<td>CONCAT(GVL.g_sTextReceivedFromSL[v_iCurrentLine], v_sCurrentCharacter);</td>
</tr>
<tr>
<td></td>
<td>END_FOR</td>
</tr>
<tr>
<td></td>
<td>(* If it is &quot;Erroneous&quot;, an error is reported *)</td>
</tr>
<tr>
<td>Step</td>
<td>Action</td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
</tr>
</tbody>
</table>
| 7. ST, LD, or CFC Program | **ELSIF** v_xReceptionError **THEN**

(* Reception state = ERROR *)
GVL.g_xErrorReception := TRUE;

**END_IF**

(* Since the "Execute" input of the "SEND_RECV_MSG" Function Block requires a RISING EDGE to operate, *)
(* it is mandatory to reset it (at least during one cycle) between two different message receptions; *)
(* thus, the "SEND_RECV_MSG" Function Block will be processed on the next "Execute" RISING EDGE *)
v_xEnableReception := FALSE;

(* In any other case, the reception is ENABLED *)
**ELSE**
(* But only if the emission is DISABLED (i.e. if the controller is not sending text on the serial line) *)
**IF NOT** v_xEnableEmission **THEN**
v_xEnableReception := TRUE;
**END_IF**
**END_IF**

(* USER command: Clear the contents of the Cumulated Reception STRING *)
**IF** GVL.g_xCmdClearReceivedText **THEN**
GVL.g_xCmdClearReceivedText := FALSE;
(* Empty the STRING array *)
**FOR** v_iLine := 1 **TO** GVL.c_NumberOfLines **DO**
GVL.g_sTextReceivedFromSL[v_iLine] := '';
**END_FOR**
(* Start over at 1st line (line #0 because line #1 is the NEXT line) *)
v_iCurrentLine := 0;
**END_IF**

(*********************************************)
(*** Sending Text on the ASCII Serial Link ***)
(*********************************************)

(* USER command: Send the current STRING to send, but ONLY if the *)
(* previous emission is finished AND if the Emission is enabled *)
**IF** GVL.g_xCmdSendCurrentText **AND** v_xEnableEmission **AND** NOT v_xSendText **THEN**
GVL.g_xCmdSendCurrentText := FALSE;
(* Length of the STRING to send on the serial link *)
v_lTextToSendOnSLLength := LEN(GVL.g_sTextToSendOnSL);
(* An emission is asked to the "SEND_RECV_MSG" Function Block only *)
(* IF the STRING is not empty AND IF the Serial Link address is OK *)
**IF** (v_lTextToSendOnSLLength > 0) **AND** v_xSLAddressIsOK **THEN**
(* The STRING to send is converted into BYTEs for the "SEND_RECV_MSG" Function Block *)
v_abEmissionBuffer := String_TO_ArrayOfByte(GVL.g_sTextToSendOnSL, TRUE);
(* Number of BYTES to send on the Serial Link (in ASCII, each character is 1-byte long) *)
v_uiNumberOfBytesToSend := INT_TO_UINT(v_lTextToSendOnSLLength);
(* Command to process the SEND_RECV_MSG function block for sending text *)
v_xSendText := TRUE;
**END_IF**
**END_IF**

(* SEND_RECV_MSG Function Block for sending text on the ASCII serial link *)
(* Note: In this example, an "exchange timeout" of 500 ms is used to ensure that an emission *)
(* error does not DEFINITIVELY block the Serial Link receptions & emissions... *)
(* Note: In ST language, this call syntax is required for Function Blocks of the *)
(* 'PLCCommunication' library that use an 'Addr' INPUT variable (data type = ADDRESS) *)
SEND_RECV_MSG_SEND{
<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>:= v_xSendText, // IN - Rising Edge signal that triggers this Function Block</td>
</tr>
<tr>
<td>Abort</td>
<td>:= FALSE, // IN - Function Block not aborted (FALSE)</td>
</tr>
<tr>
<td>Addr</td>
<td>:= v_addressAsciiSL, // IN - Formatted address of the controller’s Serial Link</td>
</tr>
<tr>
<td>Timeout</td>
<td>:= 5, // IN - Exchange timeout = 5 (for 500 ms)</td>
</tr>
<tr>
<td>QuantityToSend</td>
<td>:= v_uiNumberOfBytesToSend, // IN - Number of bytes to send = Number of bytes in the ASCII string</td>
</tr>
<tr>
<td>BufferToSend</td>
<td>:= ADR(v_abEmissionBuffer), // IN - Address of the buffer used for emission</td>
</tr>
<tr>
<td>SizeRecvBuffer</td>
<td>:= 0, // IN - Available size (in bytes) of the reception buffer = 0 (this Function Block is only used to send data)</td>
</tr>
<tr>
<td>BufferToRecv</td>
<td>:= 0, // IN - Address of the reception buffer = 0 (for a send-only use of this Function Block)</td>
</tr>
<tr>
<td>Done</td>
<td>=&gt; v_xEmissionDone, // OUT - “Done” result of the ASCII emission</td>
</tr>
<tr>
<td>Busy</td>
<td>=&gt; v_xEmissionBusy, // OUT - “Busy” output of the ASCII emission</td>
</tr>
<tr>
<td>Error</td>
<td>=&gt; v_xEmissionError); // OUT - “Error” result of the ASCII emission</td>
</tr>
</tbody>
</table>

IF    v_xSendText AND NOT v_xEmissionBusy THEN

(* When the current emission is finished... *)
IF    NOT v_xEmissionError AND v_xEmissionDone THEN

(* Emission state = OK *)
GVL.g_xErrorEmission := FALSE;
(* The previously sent STRING is reset *)
GVL.g_sTextToSendOnSL := ‘’;
(* If it is “Erroneous”, an error is reported *)
ELSIF     v_xEmissionError THEN

(* Emission state = ERROR *)
GVL.g_xErrorEmission := TRUE;

END_IF

(* Resets the command to process the SEND_RECV_MSG function block for sending text;*)
(* this allows for a new, future RISING EDGE for the next STRING to send on the serial link *)
v_xSendText := FALSE;
(* DISables the Emission and ENABLEs the reception (the ABORT command is also reset) *)
v_xEnableEmission := FALSE;
v_xEnableReception := TRUE;
v_xAbortReception := FALSE;

END_IF

*******************************
*** Program General Enable ***
*******************************

(* Sets to TRUE the command to execute the rest of this program *)
(* (for ensuring that a rising edge is correctly detected) *)
(* in this example, this occurs at the end of the 1st cycle. *)
v_xExecuteProgram := TRUE;
7.2. LD Program

Before instructing you how to write down the LD program used in this example, the following table presents optional steps that give you information on how to program in LD language.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Optional step</td>
</tr>
</tbody>
</table>

**How to display titles and comments in LD language**

- Select the **Options** command of the **Tools** menu.
- Select, in the **Options** window, the **FBD, LD and IL editor** section.
- If you wish to add a title and/or a comment for each LD network, check the **Show network title** and/or the **Show network comment** options, as shown below:
<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td><strong>Creation of the POU:</strong> Create a new POU in LD language, called <strong>PLC_PRG_LD</strong>.</td>
</tr>
</tbody>
</table>

Upon creation of this POU, it is automatically opened by SoMachine.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td><strong>LD variables:</strong> In the upper part of the LD editor, declare the following variables:</td>
</tr>
</tbody>
</table>

```plaintext
PROGRAM PLC_PRG_LD
VAR
   // Command to execute the rest of this program; initially at FALSE; set to TRUE at
   // the end of the 1st cycle
   v_xExecuteProgram           : BOOL := FALSE;

   // ADDM Variables
   // ADDM Function Block for formatting the address of the ASCII serial link
   ADDM_ASCII_SL               : ADDM;
   // ADDRESS structure for the address of the ASCII serial link
   v_addressAsciiSL            : ADDRESS;
   // "Done" result of the Address conversion
   v_xAddressDone              : BOOL := FALSE;
   // "Error" result of the Address conversion
   v_xAddressError             : BOOL := FALSE;
   // Result of the Address conversion: OK if "Done" without any "Error"
   v_xSLAddressIsOK            : BOOL := FALSE;

   // RECEIVE ASCII text
   // Flag used to enable (TRUE) or disable (FALSE) the Reception
   v_xEnableReception          : BOOL := TRUE;

   // Command to process the SEND_RECV_MSG function block for receiving text
   v_xReceiveText              : BOOL := FALSE;

   // Command to abort the current reception
   v_xAbortReception           : BOOL := FALSE;

   // SEND_RECV_MSG Function Block for receiving text from the ASCII serial link
   SEND_RECV_MSG_RECEIVE       : SEND_RECV_MSG;
   // Buffer for receiving BYTEs from the Serial Link
   v_abReceptionBuffer         : ARRAY [1..GVL.c_Size_ReceptionBuffer] OF BYTE := [80(16#00)];
```
7. ST, LD, or CFC Program

---

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>// &quot;Done&quot; result of the ASCII reception</td>
</tr>
<tr>
<td></td>
<td>v_xReceptionDone : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// &quot;Busy&quot; output of the ASCII reception</td>
</tr>
<tr>
<td></td>
<td>v_xReceptionBusy : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// &quot;Aborted&quot; result of the ASCII reception</td>
</tr>
<tr>
<td></td>
<td>v_xReceptionAborted : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// &quot;Error&quot; result of the ASCII reception</td>
</tr>
<tr>
<td></td>
<td>v_xReceptionError : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// End of the current reception</td>
</tr>
<tr>
<td></td>
<td>v_xEndOfReception : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Reception &quot;Done OK&quot; flag</td>
</tr>
<tr>
<td></td>
<td>v_xReceptionDoneOk : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Number of character(s) to delete</td>
</tr>
<tr>
<td></td>
<td>v_iNumberOfCharToDelete : INT := 0;</td>
</tr>
<tr>
<td></td>
<td>// Number of BYTEs received in the Reception Buffer</td>
</tr>
<tr>
<td></td>
<td>v_wNumberOfReceivedBytes : WORD := 0;</td>
</tr>
<tr>
<td></td>
<td>// Flag set to TRUE when the Reception Buffer is already FULL (i.e. when all lines</td>
</tr>
<tr>
<td></td>
<td>// are used)</td>
</tr>
<tr>
<td></td>
<td>v_xReceptionBufferIsFull : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Index of the currently received and analyzed BYTE</td>
</tr>
<tr>
<td></td>
<td>v_wCurrentByteIndex : WORD := 0;</td>
</tr>
<tr>
<td></td>
<td>// Flag set to TRUE to allow the recovery of the received STRING (when it is not</td>
</tr>
<tr>
<td></td>
<td>// EMPTY)</td>
</tr>
<tr>
<td></td>
<td>v_xRecoverNonEmptyString : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Total length of the ASCII text received from the Serial Link</td>
</tr>
<tr>
<td></td>
<td>v_iTextReceivedFromSLLength : INT := 0;</td>
</tr>
<tr>
<td></td>
<td>// Flag set to TRUE for deleting the first character from the currently received line</td>
</tr>
<tr>
<td></td>
<td>v_xDeleteFirstCharacter : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Character currently appended at the end of the Cumulated Reception STRING</td>
</tr>
<tr>
<td></td>
<td>v_sCurrentCharacter : STRING(1) := '';</td>
</tr>
<tr>
<td></td>
<td>// Pointer on this character</td>
</tr>
<tr>
<td></td>
<td>v_pbCurrentCharacter : POINTER TO BYTE;</td>
</tr>
<tr>
<td></td>
<td>// Current line for receiving text from the serial link</td>
</tr>
<tr>
<td></td>
<td>v_iCurrentLine : INT := 0;</td>
</tr>
<tr>
<td></td>
<td>// Temporary line index</td>
</tr>
<tr>
<td></td>
<td>v_iLine : INT := 0;</td>
</tr>
<tr>
<td></td>
<td>(***) SEND ASCII text ***</td>
</tr>
<tr>
<td></td>
<td>(***) SEND ASCII text ***</td>
</tr>
<tr>
<td></td>
<td>// Flag used to enable (TRUE) or disable (FALSE) the Emission</td>
</tr>
<tr>
<td></td>
<td>v_xEnableEmission : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Command to process the SEND_RECV_MSG function block for sending text</td>
</tr>
<tr>
<td></td>
<td>v_xSendText : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Flag set to TRUE when an USER command to send a STRING is present</td>
</tr>
<tr>
<td></td>
<td>v_xSendCommandPresent : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Flag set to TRUE when an USER command to send a STRING is accepted</td>
</tr>
<tr>
<td></td>
<td>v_xSendCommandAccepted : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Length of the ASCII text to send on the serial link</td>
</tr>
<tr>
<td></td>
<td>v_iTextToSendOnSLLength : INT := 0;</td>
</tr>
<tr>
<td></td>
<td>// Number of BYTEs to send on the serial link</td>
</tr>
<tr>
<td></td>
<td>v_uiNumberOfBytesToSend : UINT := 0;</td>
</tr>
<tr>
<td></td>
<td>// SEND_RECV_MSG Function Block for sending text on the ASCII serial link</td>
</tr>
<tr>
<td></td>
<td>SEND_RECV_MSG_SEND : SEND_RECV_MSG;</td>
</tr>
<tr>
<td></td>
<td>// Buffer for receiving BYTEs from the Serial Link</td>
</tr>
<tr>
<td></td>
<td>v_abEmissionBuffer : ARRAY [1..GVL.c_Size_EmissionBuffer] OF BYTE := [255(16#00)];</td>
</tr>
<tr>
<td></td>
<td>// &quot;Done&quot; result of the ASCII emission</td>
</tr>
</tbody>
</table>
7. ST, LD, or CFC Program

4  **LD program:** In the lower part of the LD editor, implement the following program:

```plaintext
*** Formatting the Address of the ASCII Serial link ***

```FUNCTION_BLOCK ABBE

v_xAddressDone := FALSE;

v_xAddressBusy := FALSE;

v_xAddressError := FALSE;

v_xEndOfEmission := FALSE;

END_VAR

```
### 7. ST, LD, or CFC Program

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>When the current reception is finished... If it is &quot;Abort&quot;, then the serial line is considered as being &quot;free&quot; and can be used by the controller for sending characters without provoking an error. If the reception is disabled (it will be enabled once the reception is finished) and the serial line is &quot;free&quot; and can be used for emission.</td>
</tr>
<tr>
<td>8</td>
<td>When the current reception is finished... (continued...)</td>
</tr>
<tr>
<td>9</td>
<td>Since the &quot;Pause&quot; flag of the &quot;SEND_RECV_MSG&quot; Function Block requires a 255 SENDING EDGE to operate, it is mandatory to insert it (at least during one cycle) between two different message receptions. Thus, the &quot;SEND_RECV_MSG&quot; Function Block will be processed on the next &quot;Execute&quot; SILENT EDGE.</td>
</tr>
<tr>
<td>10</td>
<td>In any other case, the reception is ENABLED. But only if the emission is DISABLED (i.e., if the controller is not sending test on the serial line).</td>
</tr>
<tr>
<td>11</td>
<td>Reception &quot;Done OK&quot; flag -&gt; Management of the reception.</td>
</tr>
<tr>
<td>12</td>
<td>Reception &quot;Done OK&quot; flag -&gt; Management of the reception (continued...)</td>
</tr>
<tr>
<td>13</td>
<td>Note: representation of &lt; MBR LINO &gt; using &lt; LABEL &gt; BEGIN</td>
</tr>
</tbody>
</table>
7. ST, LD, or CFC Program

---

Select ST, LD, or CFC program depending on the current situation.

1. Check if the current program is ST, LD, or CFC.
2. If the program is not ST, LD, or CFC, proceed with the next step.
3. If the program is ST, LD, or CFC, proceed with the next step.

---

Step | Action
---|---
1. | Note: Implementation of a < MCB LOG > using a LABEL = BEGIN
2. | For updating the last line
3. | v_RecPerceivedNullableNull
4. | v_RecPerceivedNullableNull
5. | v_RecPerceivedNullableNull
6. | v_RecPerceivedNullableNull
7. | v_RecPerceivedNullableNull
8. | v_RecPerceivedNullableNull
9. | v_RecPerceivedNullableNull
10. | v_RecPerceivedNullableNull
11. | v_RecPerceivedNullableNull
12. | v_RecPerceivedNullableNull
13. | v_RecPerceivedNullableNull
14. | v_RecPerceivedNullableNull
15. | v_RecPerceivedNullableNull
16. | v_RecPerceivedNullableNull
17. | v_RecPerceivedNullableNull
18. | v_RecPerceivedNullableNull
19. | v_RecPerceivedNullableNull
20. | v_RecPerceivedNullableNull
21. | v_RecPerceivedNullableNull
22. | v_RecPerceivedNullableNull
23. | v_RecPerceivedNullableNull
24. | v_RecPerceivedNullableNull

---

In this example, if the line of the Cumulated Reception STRING is already full (Max. size reached), its oldest character (i.e., the 1st character) is removed for making room for the new character.

---

Conversion from BYTE to (character) of STRING performed with the help of a POINTER.

---

EIO0000000911.00 04/2011 41
7. ST, LD, or CFC Program

Step | Action
--- | ---
25 | The currently analyzed character is appended at the end of the line of the Cumulated Receptive STRING

```
v_xBidOnReadyString unplugged

0vL_g_xTeReceivedFromL[v_iCurrentLine] = STR1
v_iCurrentCharacter = STR2
```

26 | Note: Representation of a < MBR LOOP > using a LABEL - END

**USER command:** Clear the contents of the Cumulated Receptive STRING
--- Empty the STRING array

```
0vL_g_xCmdClearReceivedText
```

27 | Note: Representation of a < PCC LOOP > using a LABEL - BEGIN

**USER command:** Clear the contents of the Cumulated Receptive STRING
--- (continued...)
--- Start over at 1st line (line #0 because line #1 is the NEXT line)
--- Reset of this USER command (must be done as last position in ID)

```
0vL_g_xCmdClearReceivedText
```

28 | Note: Representation of a < PCC LOOP > using a LABEL - BEGIN

**USER command:** Send the current STRING to send, but ONLY if the previous emission is finished AND if the Emission is enabled

```
0vL_g_xCmdSendCurrentText v_xSendText v_xEnableEmission
```

29 | --- Send Command on the ASCII Serial Link ---

**Function block:** SEND ASCII MSG

**USER command:** Send the current STRING to send, but ONLY if the previous emission is finished AND if the Emission is enabled

```
0vL_g_xCmdSendCurrentText v_xSendText v_xEnableEmission
```

30 | --- Send Command on the ASCII Serial Link ---

**Function block:** SEND ASCII MSG

**USER command:** Send the current STRING to send, but ONLY if the previous emission is finished AND if the Emission is enabled

```
0vL_g_xCmdSendCurrentText v_xSendText v_xEnableEmission
```

31 | --- Send Command on the ASCII Serial Link ---

**Function block:** SEND ASCII MSG

**USER command:** Send the current STRING to send, but ONLY if the previous emission is finished AND if the Emission is enabled

```
0vL_g_xCmdSendCurrentText v_xSendText v_xEnableEmission
```

32 | --- Send Command on the ASCII Serial Link ---

**Function block:** SEND ASCII MSG

**USER command:** Send the current STRING to send, but ONLY if the previous emission is finished AND if the Emission is enabled

```
0vL_g_xCmdSendCurrentText v_xSendText v_xEnableEmission
```

33 | --- Send Command on the ASCII Serial Link ---

**Function block:** SEND ASCII MSG

**USER command:** Send the current STRING to send, but ONLY if the previous emission is finished AND if the Emission is enabled

```
0vL_g_xCmdSendCurrentText v_xSendText v_xEnableEmission
```

34 | --- Send Command on the ASCII Serial Link ---

**Function block:** SEND ASCII MSG

**USER command:** Send the current STRING to send, but ONLY if the previous emission is finished AND if the Emission is enabled

```
0vL_g_xCmdSendCurrentText v_xSendText v_xEnableEmission
```

35 | --- Send Command on the ASCII Serial Link ---

**Function block:** SEND ASCII MSG

**USER command:** Send the current STRING to send, but ONLY if the previous emission is finished AND if the Emission is enabled

```
0vL_g_xCmdSendCurrentText v_xSendText v_xEnableEmission
```

36 | --- Send Command on the ASCII Serial Link ---

**Function block:** SEND ASCII MSG

**USER command:** Send the current STRING to send, but ONLY if the previous emission is finished AND if the Emission is enabled

```
0vL_g_xCmdSendCurrentText v_xSendText v_xEnableEmission
```
7. ST, LD, or CFC Program

--- The STRING to send is converted into ASCII for the "SEND REVV MSG" Function Block
--- Number of bytes to send on the Serial line (in ASCII, each character is 2-byte long)
--- Command to process the SEND REVV MSG function block for sending text

--- Reset of this UNIX command (must be done in last position an LD)

--- SEND_REVV_MSG Function Block for sending text on the ASCII serial link
Note: In this example, an "exchange timeout" of 100 as is used to ensure that an emission error does not DEFINITELY block the Serial Link receptions & emissions...
Execute = Rising Edge signal that triggers this Function Block
Abort = FALSE (the ongoing SEND operation cannot be aborted)
Addr = Predefined address of the controller’s Serial Link
Timeout = 5 (for 500 ms)
NumberOfBytes = Number of bytes to send (Number of bytes in the ASCII string)
BufferToSend = Address of the buffer used for emission
BufferSize = Available size in bytes of the reception buffer = 0 (this Function Block is only used to send data)
BufferToRecv = Address of the reception buffer = 0 (for a send-only use of this Function Block)

--- When the current emission is finished...
--- If it is "Done" (OK), the next STRING is sent to inform the USER
--- that it has been sent and that a new STRING can be typed in
--- Emission state = OK
--- The previously sent STRING is erased
--- If it is "Errored", an error is reported
--- Emission state = ERROR

--- MOVE
### 7. ST, LD, or CFC Program

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 39   | When the current emission is finished... (continued...)  
--- Resets the command to process the SEND_RECEIPT function block for sending test;  
   this allows for a new future SEND/RECEIVE for the next ST,LD, or CFC to send on the serial link  
--- DISABLES the emission and ENABLES the reception (the ABORT command is also reset)  

```c
v_xSendEmission
```

```c
v_xSendRecx
```

```c
v_yEnableEmission
```

```c
v_yEnableRecx
```

```c
v_xEnableRecx
```

```c
v_xAbortRecx
```

| 40   | *** Program General Enable ***  
Sets to THINK the commands to execute the rest of this program  
(for ensuring that a rising edge is correctly detected);  
in this example, this occurs at the end of the 1st cycle.  

```c
v_yExecuteProgram
```
### 7.3. CFC Program

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Creation of the POU:</strong> Create a new POU in CFC language, called <strong>PLC_PRG_CFC</strong>.</td>
</tr>
</tbody>
</table>

Upon creation of this POU, it is automatically opened by SoMachine.

<table>
<thead>
<tr>
<th>Program PLC_PRG_CFC</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR</td>
</tr>
<tr>
<td>v_xExecuteProgram   : BOOL := FALSE;</td>
</tr>
<tr>
<td>(**********************)</td>
</tr>
<tr>
<td>(*** ADDM Variables ***)</td>
</tr>
<tr>
<td>(**********************)</td>
</tr>
<tr>
<td>ADDM_ASCII_SL       : ADDM;</td>
</tr>
<tr>
<td>ADDRADDRESS         : ADDRESS;</td>
</tr>
<tr>
<td>v_xAddressDone      : BOOL := FALSE;</td>
</tr>
<tr>
<td>v_xAddressError     : BOOL := FALSE;</td>
</tr>
<tr>
<td>(**********************)</td>
</tr>
<tr>
<td>(*** RECEIVE ASCII text ***)</td>
</tr>
<tr>
<td>(**********************)</td>
</tr>
<tr>
<td>v_xEnableReception  : BOOL := TRUE;</td>
</tr>
<tr>
<td>v_xReceiveText      : BOOL := FALSE;</td>
</tr>
<tr>
<td>v_xAbortReception   : BOOL := FALSE;</td>
</tr>
<tr>
<td>Step</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
|      | (*** SEND ASCII text ***)
|      | (***********************) |
|      | // Flag used to enable (TRUE) or disable (FALSE) the Emission |
|      | v_xEnableEmission : BOOL := FALSE; |
|      | // Command to process the SEND_RECV_MSG function block for sending text |
|      | v_xSendText : BOOL := FALSE; |
|      | // Length of the ASCII text to send on the serial link |
|      | v_iTextToSendOnSLLength : INT := 0; |
|      | // Number of BYTEs to send on the serial link |
|      | v_uiNumberOfBytesToSend : UINT := 0; |
|      | // SEND_RECV_MSG Function Block for sending text on the ASCII serial link |
|      | SEND_RECV_MSG_SEND : SEND_RECV_MSG; |
|      | // Buffer for receiving BYTEs from the Serial Link |
|      | v_abEmissionBuffer : ARRAY [1..GVL.c_Size_EmissionBuffer] OF BYTE := [255(16#00)]; |
|      | // "Done" result of the ASCII emission |
|      | v_xEmissionDone : BOOL := FALSE; |
|      | // "Busy" output of the ASCII emission |
|      | v_xEmissionBusy : BOOL := FALSE; |
|      | // "Error" result of the ASCII emission |
|      | v_xEmissionError : BOOL := FALSE; |
|      | // End of the current emission |
|      | v_xEndOfEmission : BOOL := FALSE; |
|      | END_VAR |
3. **CFC program:** In the lower part of the CFC editor, implement the following program:
4 **Order By Data Flow**

When you have finished implementing this CFC program, perform the following steps to correctly set the execution order of its blocks:

- Right-click on an empty spot of the central worksheet.
- In the **Execution Order** part of the contextual menu, execute the **Order By Data Flow** command.

5 **Change the execution order of LABEL_LINE_REMOVAL**

After setting the execution order, you must manually set the execution order of the three labels located in the program.

The first label is **LABEL_LINE_REMOVAL**; it must be processed before the 
\[GVL.g_sTextReceivedFromSL[v_iLine] := GVL.g_sTextReceivedFromSL[v_iLine + 1]\] instruction. Currently, its execution order has been set to an inadequate value, as shown in the following example:

In this example, the **LABEL_LINE_REMOVAL** block is the 71\(^{st}\) block, and the \(:=\) block is the 37\(^{th}\) block; their execution order shall be changed to 37\(^{th}\) block and 38\(^{th}\) block, respectively.
To fix the execution order of these two blocks:

- Right-click on the **LABEL_LINE_REMOVAL -** block.
- In the **Execution Order** part of the contextual menu, execute the **Set Execution Order…** command.

- In the **Edit Working Sheet** window, change the **New Execution Order** from its current value to the execution order of the **:=** block (from 71 to 37 in the case of the example given above):

  - Click on **OK**
  - Check that the **LABEL_LINE_REMOVAL -** block is now located before the **:=** block, with respect to their execution order, as shown below:

Now, the **LABEL_LINE_REMOVAL -** block will be processed just before the **:=** block by the controller.
<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 6    | **Change the execution order of LABEL_GET_BYTE**  
Fix the execution order of the second label, LABEL_GET_BYTE, the same way as described for the first label. The block of this second label must be processed by the controller just before the **LEN** block, as shown below: |
| 7    | **Change the execution order of LABEL_EMPTY_ARRAY**  
Fix the execution order of the third label, LABEL_EMPTY_ARRAY, the same way as described for the first label. The block of this third label must be processed by the controller just before the **:=** block, as shown below: |
8. Creation of the Visu_ASCII Visualization

The steps listed in the following table describe how to create the Visu_ASCII visualization used for testing this example.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 1    | Create a new Visualization, called Visu_ASCII.  
Upon creation of this Visualization, it is automatically opened by SoMachine.  
**Note:** This action adds the following specific libraries to the Library Manager; these libraries come in addition to the libraries described in Library Manager (see page 25):  
- System_VisuElems  
- System_VisuElemMeter  
- System_VisuElemsWinControls  
- System_VisuElemTrace  
- System_VisuInputs |
| 2    | In the top part of the Properties panel, check the [Expert] option to have access to all properties of your visualization elements. |
| 3    | Create the elements of the following visualization: |

---

**Table:**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ElementName</td>
<td>displayTextReceivedLine01</td>
</tr>
<tr>
<td>Position</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>10</td>
</tr>
<tr>
<td>Y</td>
<td>50</td>
</tr>
<tr>
<td>Width</td>
<td>600</td>
</tr>
<tr>
<td>Height</td>
<td>15</td>
</tr>
</tbody>
</table>
These elements are created using the Toolbox and their main Properties must be configured as follows:

**1) LED (Light Emitting Diode) for the state of the last reception**

Type of the **Element**.......................... Toolbox ▶ Basic ▶ Ellipse

**Properties:**
- ElementName............................................ ledReceptionState
- Type of element....................................... VISU_ST_CIRCLE
- Colors ▶ Normalstate ▶ Fillcolor.................. 0; 255; 0
- Colors ▶ Alarmstate ▶ Fillcolor.................... 255; 0; 0
- Color variables ▶ ToggleColor...................... GVL.g_xErrorReception

**2) LED (Light Emitting Diode) for the state of the last emission**

Type of the **Element**.......................... Toolbox ▶ Basic ▶ Ellipse

**Properties:**
- ElementName............................................ ledEmissionState
- Type of element....................................... VISU_ST_CIRCLE
- Colors ▶ Normalstate ▶ Fillcolor.................. 0; 255; 0
- Colors ▶ Alarmstate ▶ Fillcolor.................... 255; 0; 0
- Color variables ▶ ToggleColor...................... GVL.g_xErrorEmission

**3) Title of the Text Reception part**

Type of the object ............................... Toolbox ▶ Basic ▶ Rectangle

**Properties:**
- ElementName............................................ textTextReceived
- Type of element....................................... VISU_ST_RECTANGLE
- Elementlook ▶ Frameattributes.................. PS_HOLLOW
- Texts ▶ Text ........................................... ASCII Text Received from Serial Line

**4) Title of the Text to Send part**

Type of the object ............................... Toolbox ▶ Basic ▶ Rectangle

**Properties:**
- ElementName............................................ textTextToSend
- Type of element....................................... VISU_ST_RECTANGLE
- Elementlook ▶ Frameattributes.................. PS_HOLLOW
- Texts ▶ Text ........................................... ASCII Text to Send on Serial Line

**5) Button for resetting the contents of the last 10 lines received from the serial link**

Type of the object ............................... Toolbox ▶ Windows Controls ▶ Button

**Properties:**
- ElementName............................................ btnClearTextReceived
- Texts ▶ Text ........................................... Clear
- State variables ▶ Invisible......................... GVL.g_xCmdClearReceivedText
- Inputs ▶ Toggle ▶ Variable........................ GVL.g_xCmdClearReceivedText
- Inputs ▶ Toggle ▶ Toggle on up if captured ....

**6) Button for sending a 1-line text on the serial link**

Type of the object ............................... Toolbox ▶ Windows Controls ▶ Button

**Properties:**
- ElementName............................................ btnSaveText
- Texts ▶ Text ........................................... Send
- State variables ▶ Invisible......................... GVL.g_xCmdSendCurrentText
- Inputs ▶ Toggle ▶ Variable........................ GVL.g_xCmdSendCurrentText
- Inputs ▶ Toggle ▶ Toggle on up if captured ....
8. Creation of the Visu_ASCII Visualization

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>7]</td>
<td>Last 10 lines received from the serial link</td>
</tr>
<tr>
<td></td>
<td>Each of these 10 lines is represented by a separate element.</td>
</tr>
<tr>
<td></td>
<td>Type of the object: Toolbox ▶ Windows Controls ▶ Textfield</td>
</tr>
<tr>
<td>Properties:</td>
<td></td>
</tr>
<tr>
<td>ElementName: displayTextReceivedLine01 to displayTextReceivedLine10</td>
<td></td>
</tr>
<tr>
<td>Colors ▶ NormalState ▶ FillColor: 211; 211; 211</td>
<td></td>
</tr>
<tr>
<td>Elementlook ▶ m_ShadowType: DEEPENED</td>
<td></td>
</tr>
<tr>
<td>Texts ▶ Text: %s</td>
<td></td>
</tr>
<tr>
<td>Text properties ▶ HorizontalAlignment: LEFT</td>
<td></td>
</tr>
<tr>
<td>Text variables ▶ Text Variable: GVL.g_sTextReceivedFromSL[1] to GVL.g_sTextReceivedFromSL[10]</td>
<td></td>
</tr>
<tr>
<td>State variables ▶ Deactivate inputs: TRUE</td>
<td></td>
</tr>
</tbody>
</table>

8] 1-line text to send on the serial link

| Properties: | 
| ElementName: displayTextToSend |
| Colors ▶ NormalState ▶ FillColor: 255; 255; 255 |
| Elementlook ▶ m_ShadowType: DEEPENED |
| Texts ▶ Text: %s |
| Text properties ▶ HorizontalAlignment: LEFT |
| Text variables ▶ Text Variable: GVL.g_sTextToSendOnSL |
| Inputs ▶ OnMouseClick: Write a variable: GVL.g_sTextToSendOnSL (see below) |
9. Configuration of the HyperTerminal

The steps listed in the following table describe how to configure the HyperTerminal software. This Windows software must be run on a PC with a RS-232 communication port (COM1).

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Close any other software that may use the COM1 port.</td>
</tr>
</tbody>
</table>
| 2    | Start the HyperTerminal Windows software.  
Up to Windows XP, this Microsoft software can be started from the Windows Start menu, at the following location: 
Start menu ► Programs ► Accessories ► Communications ► HyperTerminal 
**Note:** HyperTerminal is not delivered with Windows Vista or Windows 7. If you use one of these operating systems, then you must apply one of the following alternatives:  
- Search and download from the Internet an archive containing the two files that make up HyperTerminal: hypertrm.dll and hypertrm.exe. After extracting these two files on your PC, you can run hypertrm.exe to launch the HyperTerminal Windows software.  
- Use another communication software. PuTTY, a free implementation of Telnet, is a good example of communication software that can replace HyperTerminal. However, the configuration and use of this software are not described here. |
| 3    | Select the Properties command of the File menu. 
This command will open the properties of the current HyperTerminal connection. |
| 4    | Configure the current HyperTerminal connection as follows:  
- Select COM1;  
- Click on the Configure... button;  
- Set the following Port Settings:  
  - Click on OK. |
9. Configuration of the HyperTerminal

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 5    | **Configure the settings of the current HyperTerminal connection as follows:**  
  - Switch to the **Settings** tab;  
  - Configure the following Settings (left-hand picture);  
  - Click on the **ASCII Setup...** button;  
  - Configure the following ASCII setup (right-hand picture):  
    - Click on **OK** to close the ASCII Setup window;  
    - Click on **OK**. |
| 6    | **Select the Save command of the File menu to save this connection for future use.** |
| 7    | The two following commands will be used in **Running the Example on the Controller (see page 66)** to connect and disconnect **HyperTerminal** to the COM1 communication port:  
  - Click on the **icon** to connect **HyperTerminal** to the COM1 port  
  - Click on the **icon** to disconnect **HyperTerminal** from the COM1 port |
### 10. Running the Example

#### 10.1. MAST Task Configuration

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the Devices tree view, double-click on the MAST task of the Task Configuration item:</td>
</tr>
<tr>
<td></td>
<td><img src="image1" alt="Device View" /></td>
</tr>
<tr>
<td>2</td>
<td>Click on the Add POU command and select your program.</td>
</tr>
<tr>
<td></td>
<td><img src="image2" alt="POU Selection" /></td>
</tr>
</tbody>
</table>

In the following picture, the ST program described in ST Program (see page 30), PLC_PRG_ST, is selected:
### 10. Running the Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Click on <strong>OK</strong>. This adds the selected POU to the list of programs run by the MAST task of the controller.</td>
</tr>
</tbody>
</table>
10.2. Downloading the Example to the Controller

The steps listed in the following table describe how to download the example to the controller. If needed, please refer to the SoMachine online help for further information on these steps: search for Communication Settings.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Connect the USB programming cable between your PC and the LMC058 controller. Please refer to Hardware Installation (see page 17) for the reference and usage of this cable.</td>
</tr>
<tr>
<td>2</td>
<td>In the Devices tree view, double-click on the LMC058_Controller (LMC058LF42S0) item to open its configuration panel.</td>
</tr>
</tbody>
</table>
| 3    | In the Communication Settings tab of this panel.  
  - Click on the Gateway-1 node  
  - Click on the Scan network button |

If the controller is switched on and connected to your PC with the USB programming cable, it will be detected by SoMachine as shown below:

In this example, the **LMC058LF42S0 @0080F4400CBC** controller has been detected.
## 10. Running the Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td><strong>Select this controller’s node and click on the Set active path button.</strong></td>
</tr>
</tbody>
</table>

This will show SoMachine the way to reach your controller.

**Option:** Uncheck the **Secure online mode** box to avoid validation messages during future online modifications.

**Note:** The firmware version of your controller is displayed as the **Target Version**.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 5    | The following window will appear and ask you to confirm your choice:  
- Read the hazard message;  
- Press on both `<Alt>` and `<F>` keys to validate your choice and close this window;  
- Or, click on the **Cancel** button if you can not comply with the statements in the hazard message. |

**WARNING**

**UNINTENDED EQUIPMENT OPERATION**

Ensure that the software application being downloaded is installed on the intended device. Confirm that you have entered the correct device designation or device address.

Ensure guards are in place so that unintended equipment operation will not cause injury to personnel or damage to equipment.

Read and understand the software User Manual, and know how to operate the equipment.

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

If you agree to follow these instructions, press `ALT+F`.

---

![Diagram of software interface showing controller selection and online mode options.](image)
## 10. Running the Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Click on the <strong>Login</strong> button of the toolbar to establish a connection from SoMachine to the controller.</td>
</tr>
<tr>
<td>7a</td>
<td>If you download the current project to the controller for the first time and if there is no project on the controller, the following window appears:</td>
</tr>
<tr>
<td></td>
<td><img src="image1.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>Click on <strong>Yes</strong> to download the <strong>Application</strong> software to the controller.</td>
</tr>
<tr>
<td>7b</td>
<td>If you download the current project to the controller for the first time and if there is another project on the controller, the following window appears:</td>
</tr>
<tr>
<td></td>
<td><img src="image2.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>Click on <strong>Yes</strong> to download the <strong>Application</strong> software to the controller.</td>
</tr>
<tr>
<td>7c</td>
<td>If you already have downloaded the current project to the controller, the following window appears if you have brought new modifications to this project:</td>
</tr>
<tr>
<td></td>
<td><img src="image3.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>➢ Select the <strong>Login with download</strong> choice.</td>
</tr>
<tr>
<td></td>
<td>➢ Click on <strong>OK</strong> to download the <strong>Application</strong> software to the controller.</td>
</tr>
<tr>
<td>7d</td>
<td>If you already have downloaded the current project to the controller, the connection is immediate if you brought no modification to this project.</td>
</tr>
<tr>
<td>8</td>
<td>Wait for the completion of the download operation.</td>
</tr>
<tr>
<td></td>
<td>Once it is finished, the status bar of SoMachine displays the state of the controller: <strong>STOP</strong>.</td>
</tr>
<tr>
<td>9</td>
<td>Click on the <strong>Start</strong> button of the toolbar to run the <strong>Application</strong> software on the controller.</td>
</tr>
<tr>
<td></td>
<td>The state of the controller switches from <strong>STOP</strong> to <strong>RUN</strong>.</td>
</tr>
</tbody>
</table>
10. Running the Example

10.3. Running the Example on the Controller

The steps listed in the following table describe how to use the example, once it has been downloaded to the controller.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 1    | In the **Devices** panel, double-click on your **PLC_PRG** program to open it (e.g. **PLC_PRG_ST** if you previously selected this program as the program run by the **MAST** task of your controller).  
This command opens your program in the central panel of SoMachine; the online values of the variables used by this program are also displayed, as shown below: |
| 2    | **Addressing the ASCII Serial Line using the **ADDM** function block:**  
**Name of the ADDM function block instance:** **ADDM_ASCII_SL**  
To format the address of the ASCII serial line (serial line #1 since the LMC058 controller has only one serial line), an **ADDM_ASCII_SL** instance is used.  
Check that this function block instance works correctly:  
- its **Done** output must be equal to TRUE;  
- its **Error** output must be equal to FALSE. |
| 3    | **Continuous reception of ASCII text from the serial line using the **SEND_RECV_MSG** function block:**  
**Name of the SEND_RECV_MSG function block instance:** **SEND_RECV_MSG_RECEIVE**  
A **SEND_RECV_MSG_RECEIVE** instance is continuously activated for receiving ASCII text from the serial line. It is only disabled on one of the two following events:  
- A frame of ASCII text has been received, thus resetting the **Done** output of this function block; this inhibits this function block for one cycle in order to generate a new rising edge on its **Execute** input. This allows this function block to receive future frame(s) of ASCII text.  
- You order the controller to send ASCII text on the serial line through the activation of the **g_xCmdSendCurrentText** command (i.e. by clicking on the **Send** button of the **Visu_ASCII** visualization). In this case, the current execution of this function block is aborted. |
10. Running the Example

### Step 10.1: Note

**Note:** This abortion is intended to free the serial line; otherwise, the ASCII text emission would have resulted in an error because the `SEND_RECV_MSG_RECEIVE` instance would have continued to monopolize the serial line’s resources.

Each time a frame is received by the `SEND_RECV_MSG_RECEIVE` instance, it resets its `Busy` output to `FALSE` and sets its `Done` to `TRUE` (i.e. a variable length string of `BYTEs`, terminated by a Carriage Return character (CR) and a Line Feed character (LF)).

In this example’s program, this triggers the conversion of this string of `BYTEs` and its memorization into a 10-line array of `STRINGs`: `GVL.g_sTextReceivedFromSL`. This array is displayed in the `Visu_ASCII` visualization.

### Step 4: Emission of ASCII text on the serial line using the `SEND_RECV_MSG` function block:

Name of the `RECV_MSG` function block instance: `SEND_RECV_MSG_SEND`

A `SEND_RECV_MSG_SEND` instance is activated for sending ASCII text on the serial line. Actually, it is only processed when the following event occurs:

- You order the controller to send ASCII text on the serial line through the activation of the `g_xCmdSendCurrentText` command (i.e. by clicking on the `Send` button of the `Visu_ASCII` visualization). In addition, a `v_xEnableEmission` `BOOLEAN` variable validates the emission once the reception has been aborted, such notifying that the serial line has been freed and can now be used by another `SEND_RECV_MSG` function block.

In this example’s program, this event triggers the conversion of the text you typed in the `Visu_ASCII` visualization into an ASCII string of `BYTEs`, followed by its emission through the execution of the `SEND_RECV_MSG_SEND` instance.

Once the execution of the `SEND_RECV_MSG_SEND` function block instance is finished, it resets its `Busy` output to `FALSE` and sets its `Done` to `TRUE`.

In this example’s program, this ends your emission order and enables back the reception of frames on the serial link by the `SEND_RECV_MSG_RECEIVE` function block instance.

### Step 5: Open the `Visu_ASCII` visualization

![Visu_ASCII Visualization](image-url)
### 10. Running the Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td><strong>Using the Visu_ASCII visualization:</strong></td>
</tr>
<tr>
<td></td>
<td>- Connect the <strong>HyperTerminal</strong> by clicking on its ![icon] icon.</td>
</tr>
<tr>
<td></td>
<td>- In the <strong>HyperTerminal</strong>, type any text, followed by the <code>&lt;RETURN&gt;</code> key; pressing this key marks the end of the ASCII frame by adding a final CR character (code = 13 or 16#0D), followed by a LF character (code = 10 or 16#0A). This CR character is used by the <code>SEND_RECV_MSG_RECEIVE</code> function block instance for detecting the end of the current ASCII text frame.</td>
</tr>
<tr>
<td></td>
<td>- In the <strong>Visu_ASCII</strong> visualization, this text appears in the first grey line.</td>
</tr>
<tr>
<td></td>
<td>- Continue to type text in the <strong>HyperTerminal</strong>, <code>&lt;RETURN&gt;</code> key included.</td>
</tr>
<tr>
<td></td>
<td>- Each of these texts appears on a new grey line; once all of these 10 grey lines are used, they are shifted up to make room for the last received text.</td>
</tr>
<tr>
<td></td>
<td>- Any time, you can click on the <strong>Clear</strong> button to reset the contents of these 10 grey lines and restart receiving text on the first grey line.</td>
</tr>
<tr>
<td></td>
<td>- In the <strong>Visu_ASCII</strong> visualization, click on the white line; this opens an <strong>edit field</strong>.</td>
</tr>
<tr>
<td></td>
<td>- In this field, type any text, followed by the <code>&lt;RETURN&gt;</code> key; this closes the <strong>edit field</strong>.</td>
</tr>
<tr>
<td></td>
<td>- Click on the <strong>Send</strong> button; this triggers the emission of this text on the serial line. <strong>Note:</strong> A CR character is appended at the end of this text by the <code>SEND_RECV_MSG_SEND</code> function block instance to mark the end of the frame on the serial line.</td>
</tr>
<tr>
<td></td>
<td>- This text appears in the <strong>HyperTerminal</strong>.</td>
</tr>
<tr>
<td></td>
<td>- In the <strong>Visu_ASCII</strong> visualization, the white line is now empty; this is both intended to confirm that this text has been sent and to reset the <strong>edit field</strong> for allowing you to type any new text.</td>
</tr>
<tr>
<td></td>
<td>- Continue to use both the <strong>HyperTerminal</strong> and the <strong>Visu_ASCII</strong> visualization, mixing the previously described operations.</td>
</tr>
<tr>
<td></td>
<td>- When you have finished testing this example, you may disconnect the <strong>HyperTerminal</strong> by clicking on its ![icon] icon.</td>
</tr>
</tbody>
</table>