SIEMENS

SIMOTION

SIMOTION SCOUT Supplement to the CP 340 and CP 341 Modules

Function Manual

04/2014

Legal information

Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

⚠ DANGER
indicates that death or severe personal injury **will** result if proper precautions are not taken.

⚠ WARNING
indicates that death or severe personal injury **may** result if proper precautions are not taken.

⚠ CAUTION
indicates that minor personal injury can result if proper precautions are not taken.

NOTICE
indicates that property damage can result if proper precautions are not taken.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

Proper use of Siemens products

Note the following:

⚠ WARNING
Siemens products may only be used for the applications described in the catalog and in the relevant technical documentation. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient conditions must be complied with. The information in the relevant documentation must be observed.

Trademarks

All names identified by ® are registered trademarks of Siemens AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Siemens AG
Industry Sector
Postfach 48 48
90026 NÜRNBERG
GERMANY

Preface

Contents of the function manual

This document is part of the SIMOTION Programming References documentation package.

This manual is a supplement to the following SIMATIC manuals:

- CP 340 Point-to-Point Communication, *Installation and Parameter Assignment*
- CP 341 Point-to-Point Communication, *Installation and Parameter Assignment*

These documents are included in the SIMOTION SCOUT scope of supply as electronic documentation!

This manual supplement will help you to integrate and start up the CP 340 and CP 341 communication processors in a SIMOTION system.

Differences in handling which result from the software architecture of a SIMOTION system as compared to the software architecture of a SIMATIC system will be described.

Function blocks

The function blocks for communication between the SIMOTION system and the CP 340 and CP 341 modules are part of the program library of the "SIMOTION SCOUT" engineering system.

Sections in this manual

The following chapters of this manual describe the function blocks (FBs) and data structures used in a SIMOTION system.

- General
  This chapter describes the differences and similarities in operation of the various CP modules.
- CP 340 function blocks
  This chapter describes the function blocks required for communication between a SIMOTION system and a CP 340.
- CP 341 function blocks
  This chapter describes the function blocks required for communication between a SIMOTION system and a CP 341.
- Alarm processing
  This chapter describes the differences in alarm processing in the SIMOTION system compared to the SIMATIC system.
- SIMATIC and SIMOTION Names
  This appendix contains a comparison of SIMATIC and SIMOTION names.
- The index allows you to locate information quickly.

SIMOTION Documentation

An overview of the SIMOTION documentation can be found in the SIMOTION Documentation Overview document.

This documentation is included as electronic documentation in the scope of delivery of SIMOTION SCOUT. It comprises ten documentation packages.

The following documentation packages are available for SIMOTION V4.4:

- SIMOTION Engineering System Handling
- SIMOTION System and Function Descriptions
- SIMOTION Service and Diagnostics
- SIMOTION IT
- SIMOTION Programming
- SIMOTION Programming - References
- SIMOTION C
- SIMOTION P
- SIMOTION D
- SIMOTION Supplementary Documentation

Hotline and Internet addresses

Additional information

Click the following link to find information on the the following topics:

- Ordering documentation / overview of documentation
- Additional links to download documents
- Using documentation online (find and search manuals/information)

http://www.siemens.com/motioncontrol/docu

Please send any questions about the technical documentation (e.g. suggestions for improvement, corrections) to the following e-mail address:
docu.motioncontrol@siemens.com

My Documentation Manager

Click the following link for information on how to compile documentation individually on the basis of Siemens content and how to adapt it for the purpose of your own machine documentation:

http://www.siemens.com/mdm

Training

Click the following link for information on SITRAIN - Siemens training courses for automation products, systems and solutions:

http://www.siemens.com/sitrain

FAQs

Frequently Asked Questions can be found in SIMOTION Utilities & Applications, which are included in the scope of delivery of SIMOTION SCOUT, and in the Service&Support pages in Product Support:

http://support.automation.siemens.com

Technical support

Country-specific telephone numbers for technical support are provided on the Internet under Contact:

http://www.siemens.com/automation/service&support

Table of contents

Fundamental safety instructions 1

1.1 General safety instructions

⚠ WARNING

Risk of death if the safety instructions and remaining risks are not carefully observed

If the safety instructions and residual risks are not observed in the associated hardware documentation, accidents involving severe injuries or death can occur.

- Observe the safety instructions given in the hardware documentation.
- Consider the residual risks for the risk evaluation.

⚠ WARNING

Danger to life or malfunctions of the machine as a result of incorrect or changed parameterization

As a result of incorrect or changed parameterization, machines can malfunction, which in turn can lead to injuries or death.

- Protect the parameterization (parameter assignments) against unauthorized access.
- Respond to possible malfunctions by applying suitable measures (e.g. EMERGENCY STOP or EMERGENCY OFF).

1.2 Industrial security

Note

Industrial security

Siemens provides products and solutions with industrial security functions that support the secure operation of plants, solutions, machines, equipment and/or networks. They are important components in a holistic industrial security concept. With this in mind, Siemens' products and solutions undergo continuous development. Siemens recommends strongly that you regularly check for product updates.

For the secure operation of Siemens products and solutions, it is necessary to take suitable preventive action (e.g. cell protection concept) and integrate each component into a holistic, state-of-the-art industrial security concept. Third-party products that may be in use should also be considered. For more information about industrial security, visit http://www.siemens.com/industrialsecurity.

To stay informed about product updates as they occur, sign up for a product-specific newsletter. For more information, visit http://support.automation.siemens.com

⚠ WARNING

Danger as a result of unsafe operating states resulting from software manipulation

Software manipulation (e.g. by viruses, Trojan horses, malware, worms) can cause unsafe operating states to develop in your installation which can lead to death, severe injuries and/or material damage.

- Keep the software up to date.
  Information and newsletters can be found at:
  http://support.automation.siemens.com
- Incorporate the automation and drive components into a state-of-the-art, integrated industrial security concept for the installation or machine.
  For more detailed information, go to:
  http://www.siemens.com/industrialsecurity
- Make sure that you include all installed products into the integrated industrial security concept.

Description 2

2.1 General

This chapter describes the general differences between SIMOTION and SIMATIC systems in terms of the operation of the CP 340 / CP 341 communications processors and in terms of data transfer.

Note

This manual is a supplement to SIMATIC manuals *CP 340 Point-to-Point Connection, Installation and Parameter Assignment/CP 341 Point-to-Point Connection, Installation and Parameter Assignment*.

These documents are shipped with SIMOTION SCOUT in electronic form!

The following software versions are required for the standard functions described in this documentation:

- SIMOTION SCOUT V4.2 or higher
- SIMOTION Kernel V4.2 or higher

2.2 Product description

The communications processor (CP) enables data to be exchanged between your SIMOTION system and another communications partner.

Function blocks are required for data exchange between the SIMOTION device and the communications processors. The function blocks for the SIMOTION system are described in this manual; these function blocks are handled differently than the function blocks for SIMATIC S7.

Functionality of the CP 340 / CP 341

The functionality of the function blocks and the CPs in a SIMOTION system is the same as in the SIMATIC S7 with the exception of special protocols for the CP 341.

The special protocols for "Modbus Slave", "Modbus Master" and "Data Highway" for the CP 341 are **not** currently supported by SIMOTION function blocks. For detailed information, see the "CP 340 function blocks" and "CP 341 function blocks" chapters.

Possible applications

In addition to the possible applications described in SIMATIC manuals *CP 340 Point-to-Point Connection, Installation and Parameter Assignment and CP 341 Point-to-Point Connection, Installation and Parameter Assignment*, these communications processors (CPs) can also be used in a SIMOTION system. The communications processors can be used as centralized modules (on the SIMOTION C2xx only) or as distributed modules (SIMOTION C2xx, SIMOTION P350 and SIMOTION D4xx).

More than one CP 340/CP 341 can be used on one SIMOTION device.

The figure below shows the connection of an ET 200M distributed I/O device with IM 153-1 and CP 340 or CP 341 to a SIMOTION device (e.g. SIMOTION C2xx).

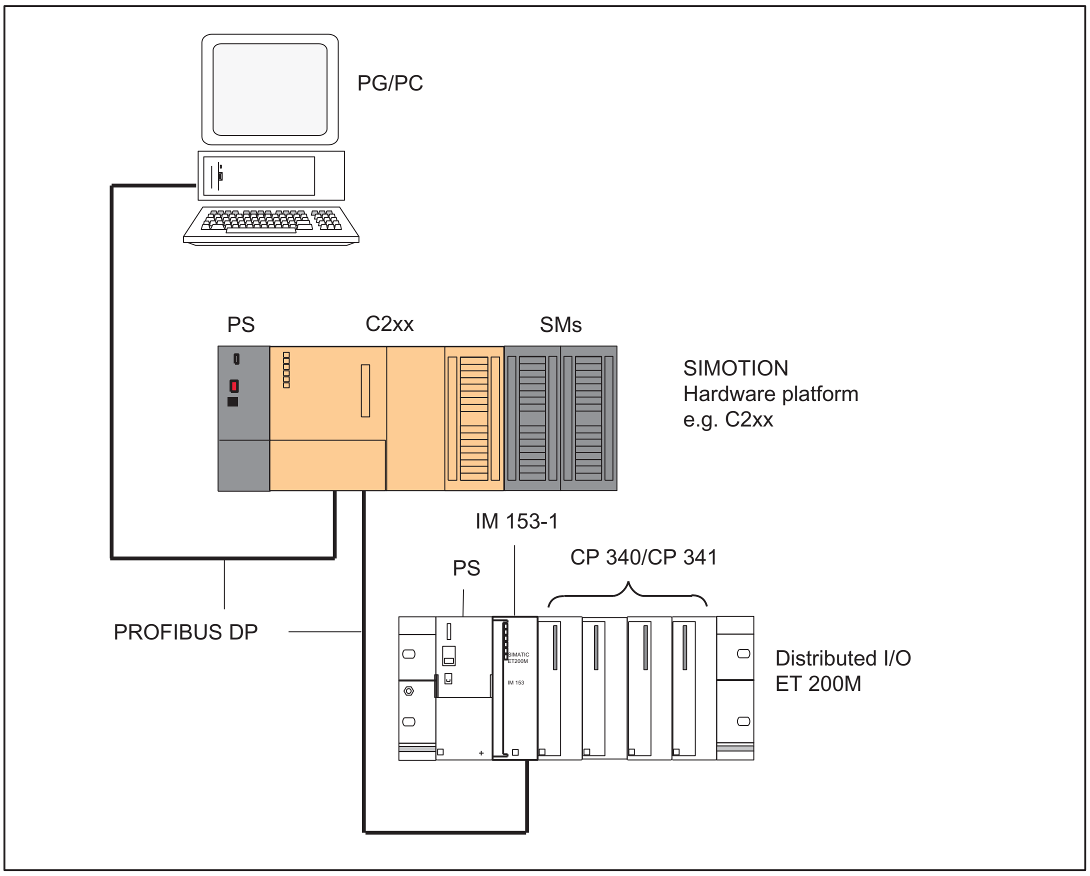


Figure 2-1 Connection of an ET 200M distributed I/O device with IM 153-1 and CP 340 or CP 341 to a SIMOTION C2xx (example of distributed application)

2.3 Setup and connection

Overview

The following sequence of operations is required to commission the CP 340/CP 341 and operate it under the control of the SIMOTION system:

Distributed application (SIMOTION C2xx, SIMOTION P350, and SIMOTION D4xx)

1. Assemble and install the cables for the ET 200M distributed I/O device complete with power supply (PS), interface module (IM) and communications processor (CP).
2. Establish the PROFIBUS connection between the ET 200M and the SIMOTION device.
3. Set the PROFIBUS DP node address on the IM.
4. Switch on the terminating resistor at the first and last bus node.

Note

For steps 1 to 4, refer to the *ET 200M Distributed I/O* manual.

This documentation is included in the SIMOTION SCOUT scope of supply as electronic documentation!

5. For inserting the communications processors CP 340 or CP 341 into the SIMOTION project, see Chapter Integrating the communications processors in the SIMOTION project (Page 15).
6. Assign parameters for the CP 340/CP 341 communications processors.
   The SIMATIC *CP 340 Point-to-Point Connection, Installation and Parameter Assignment and CP 341 Point-to-Point Connection, Installation and Parameter Assignment* manuals contain a description of how to install the parameter assignment interface for the CP 340 and 341 and how to assign parameters for the communications processors.
7. For integrating function blocks into the SIMOTION project, see Chapter Integrating the function blocks in the user project (Page 17).

Centralized application (SIMOTION C2xx only)

1. For information on planning the mechanical installation and preparing and mounting the SIMOTION components, refer to the *SIMOTION C2xx* operating instructions and the *SIMATIC S7-300 Automation System, Software Installation* manual.
   These documents are shipped with SIMOTION SCOUT in electronic form!
2. To continue, refer to steps 5 to 7 for distributed application.

2.4 Integrating the communications processors in the SIMOTION project

Requirement

The following requirements must be met in the case of networking via PROFIBUS:

1. You have created a project in SIMOTION SCOUT and have inserted a rack with a SIMOTION hardware platform in the hardware configuration.
2. You have configured a PROFIBUS subnet (for distributed application only).

Note

For information on creating a project and configuring a PROFIBUS subnet, refer to the online help for *SIMOTION SCOUT*.

The following requirements must be met in the case of networking via PROFINET:

1. You have created a project in SIMOTION SCOUT and have inserted and configured a rack with a PROFINET-compatible SIMOTION device in the hardware configuration.
2. You have configured a PROFINET IO system (for distributed application only).

Note

For information on creating a project and configuring a PROFINET IO system, refer to the online help for *SIMOTION SCOUT*.

Inserting the CP 340/CP 341 (distributed application)

The following description is an example of networking via PROFIBUS.

1. In SIMOTION SCOUT, open the **User Projects** dialog box with the **Project > Open** menu command. In this dialog box, select your project and confirm your choice with **OK**.
2. Open **HW Config**.
3. In the **HW Config** window, open the **Hardware catalog** with the **View > Catalog** menu command.
4. Open the **PROFIBUS DP** folder and the **ET 200M** subfolder in the hardware catalog. There, select e.g. the **IM 153-1 interface module** (MLFB no.: 6ES7 153-1AA03-0XB0 or a replacement module).
5. Use Drag & Drop to place the IM 153-1 I/O device on the PROFIBUS subnet of your project. The **Properties - PROFIBUS IM 153-1 Interface** dialog box is opened. In this dialog box, select the address you set on the IM 153-1 (see *ET 200M Distributed I/O Device* manual) and confirm by pressing **OK**.
   The selected IM 153-1 I/O device is inserted into the project.

6. The inserted I/O device must now be fitted with your project modules. To do this, open the CP 300 subfolder below the selected I/O device in the hardware catalog and select the relevant CP modules.

Note

Diagnostic alarms are not enabled by default. Activate the alarms for each module in the Properties dialog box.

---

7. Save and compile your project.

2.5 Integrating the function blocks in the user project

Creating an instance of the FBs in the user project

The function blocks are part of the program library of the SIMOTION SCOUT engineering system. For working with the blocks, an instance has to be created in the user project for each function block used.

Example:

```
VAR_GLOBAL
...
  myInstCP340Send  : _CP340_send;  // create FB instance
  myInstCP341Send  : _CP341_send;  // create FB instance
...
END_VAR
```

Call (LAD representation)

The LAD representation of the individual function blocks can be found in the respective function block descriptions.

Example of an application

The application example is included on the "SIMOTION Utilities & Applications" CD-ROM and is available for various SIMOTION hardware platforms.

The "SIMOTION Utilities & Applications" CD-ROM is provided free of charge and part of the SIMOTION SCOUT scope of delivery.

2.6 Creating I/O variables

Overview

Communication between the SIMOTION device and the CP 340 and CP 341 takes place by means of direct I/O access and data set transfer. For data set transfer, the module address is transferred to the FB as an input parameter. I/O variables are used to address the direct read/ write access to the I/O.

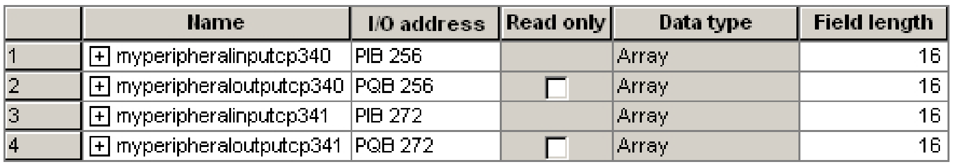
You can freely assign the names of I/O variables in SIMOTION SCOUT. I/O variables must be specified as ARRAY [0..15] of BYTE. You assign the address settings in the hardware configuration to these I/O variables.

The names of the I/O inputs must be transferred to the function blocks as call parameters (**periIn**). The prepared data for the I/O outputs are provided by the FB as in/out parameters (**periOut**). The in/out parameter must be supplied with a variable of type ARRAY [0..15] of BYTE. After the block is called, this variable must be assigned to the I/O variables for the I/O outputs (see Chapter Calling the CP 340 function blocks (Page 48)).

Note

The variable for supplying the in/out parameters must not be created as a temporary variable (VAR_TEMP or local variable of a function).

The following example shows how to assign the module addresses to the I/O variables in SIMOTION SCOUT.



	Name	I/O address	Read only	Data type	Field length
1	⊞ myperipheralinputcp340	PIB 256		Array	16
2	⊞ myperipheraloutputcp340	PQB 256	☐	Array	16
3	⊞ myperipheralinputcp341	PIB 272		Array	16
4	⊞ myperipheraloutputcp341	PQB 272	☐	Array	16

Figure 2-2 Address assignment in SIMOTION SCOUT for two CP modules

Note

For additional information, see the following sources:

- *SIMOTION SCOUT* online help
- Programming Manual of the corresponding programming language, e.g.:
  - *SIMOTION ST, Structured Text* programming manual
  - *SIMOTION MCC, Motion Control Chart* programming manual
  - *SIMOTION LAD/FBD, Ladder Diagram and Function Block Diagram* programming manual

These documents are included in the SIMOTION SCOUT scope of delivery as electronic documentation.

CP 340 function blocks 3

3.1 Overview of the function blocks of the CP 340

This chapter contains a description of all of the function blocks (FBs) and the data structure required for communication between a SIMOTION device and a CP 340.

The function blocks form the software interface between the SIMOTION device and the CPs. They must be called repeatedly (in cycles) from the user program.

The following function blocks are available:

- _CP340_send function block (Page 20)
- _CP340_receive function block (Page 25)
- _CP340_printer function block (Page 29)
- _CP340_getV24Signals function block (Page 45)
- _CP340_setV24Signals function block (Page 47)

Note

The SIMOTION identifiers have changed as of V4.0. A comparison of the SIMOTION and SIMATIC identifiers can be found in the appendix SIMOTION and SIMATIC names (Page 117) in the table "SIMOTION and SIMATIC CP 340 identifiers".

SIMOTION SCOUT contains all of the required FBs and the **Struct_CP340_printData** data structure (for **_CP340_printer** function block only) of the CP 340. The function blocks can be used to control one or more CP 340 modules.

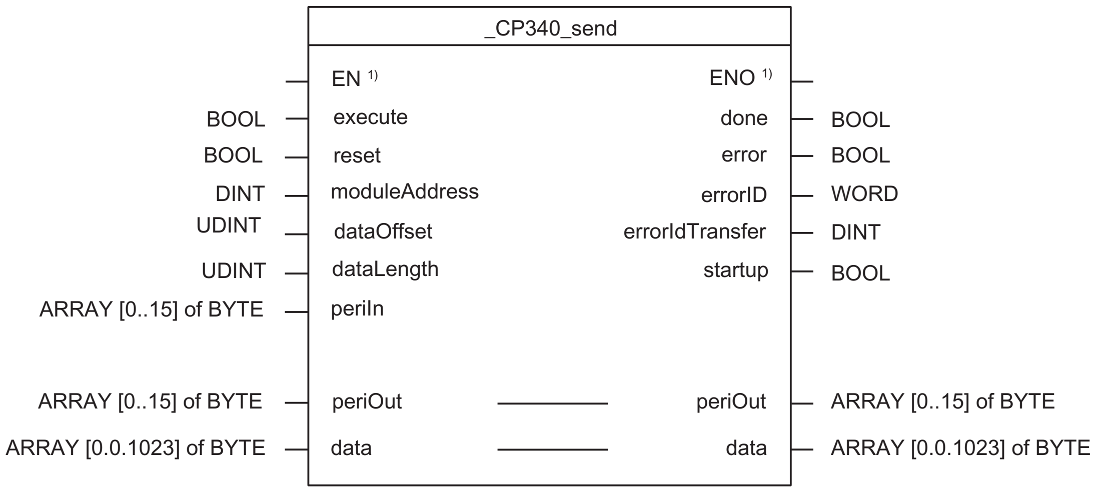
3.2 _CP340_send function block

Function

The **_CP340_send** function block enables you to send data from the **data** send array to a communications partner. You have 1,024 bytes available for this.

For the transfer, you can use the 3964 (R) protocol or ASCII driver.

Call (LAD representation)



1) LAD-specific parameters

Parameter description

Table 3-1 Parameters of the _CP340_send FB

Name	P type 1)	Data type	Meaning	Actions performed by user	Actions performed by block
execute	IN	BOOL	Initiates job on positive edge	Entered	Checked
reset	IN	BOOL	Cancels job	Entered	Checked
moduleAddress	IN	DINT	Module address of the CP for data set transfer (from HW Config)	Entered	Checked
dataOffset	IN	UDINT	Offset of the first element to be sent	Entered	Checked
dataLength	IN	UDINT	Number of elements to be sent	Entered	Checked
periIn	IN	ARRAY[0..15] of BYTE	I/O inputs of the CP transferred to the FB	I/O variable of the I/O inputs of the CP transferred to the FB	Checked
periOut	IN/OUT	ARRAY[0..15] of BYTE	Prepared FB data for the I/O outputs of the CP 2)	Checked and transferred to the I/O variable for the I/O outputs	Entered

Name	P type [1]	Data type	Meaning	Actions performed by user	Actions performed by block
data	IN/OUT	ARRAY[0..1023] of BYTE	Send data array	Entered	Checked
done	OUT	BOOL	Job completed without errors	Checked	Entered
error	OUT	BOOL	Job completed with errors	Checked	Entered
errorID	OUT	WORD	Error specification For **error** = TRUE, the error information (event class and number) is displayed in the **errorID** parameter. [3]	Checked	Entered
errorIdTransfer	OUT	DINT	Error during data transfer between the CP and the SIMOTION device (detailed error diagnostics if 16#1E0F is present in the **errorID** parameter [4])	Checked	Entered
startup	OUT	BOOL	Indicates CP startup	Checked	Entered

[1] Parameter types: IN = input parameters, OUT = output parameters, IN/OUT = in/out parameters

[2] **Note:** The **periOut** parameter must be supplied with a variable of type **ARRAY[0..15] of BYTE**. Create a local or global variable in your program under **VAR** (do not create a temporary variable under VAR_TEMP). After the FB has been called, this variable must be assigned to the I/O variable for the I/O outputs of the module. See call example for CP 340.

[3] For error information, refer to the SIMATIC *CP 340 Point-to-Point Connection, Installation and Parameter Assignment* manual, Chapter "Diagnostics with the CP 340"

[4] For a more detailed description (_readRecord and _writeRecord), see *SIMOTION System Function/Variable Devices* Parameter Manual. This documentation is included in the SIMOTION SCOUT scope of delivery as electronic documentation.

Signal sequence diagram of the _CP340_send FB

The following figure illustrates the behavior of the **done** and **error** parameters according to the input circuit of **execute** and **reset**.

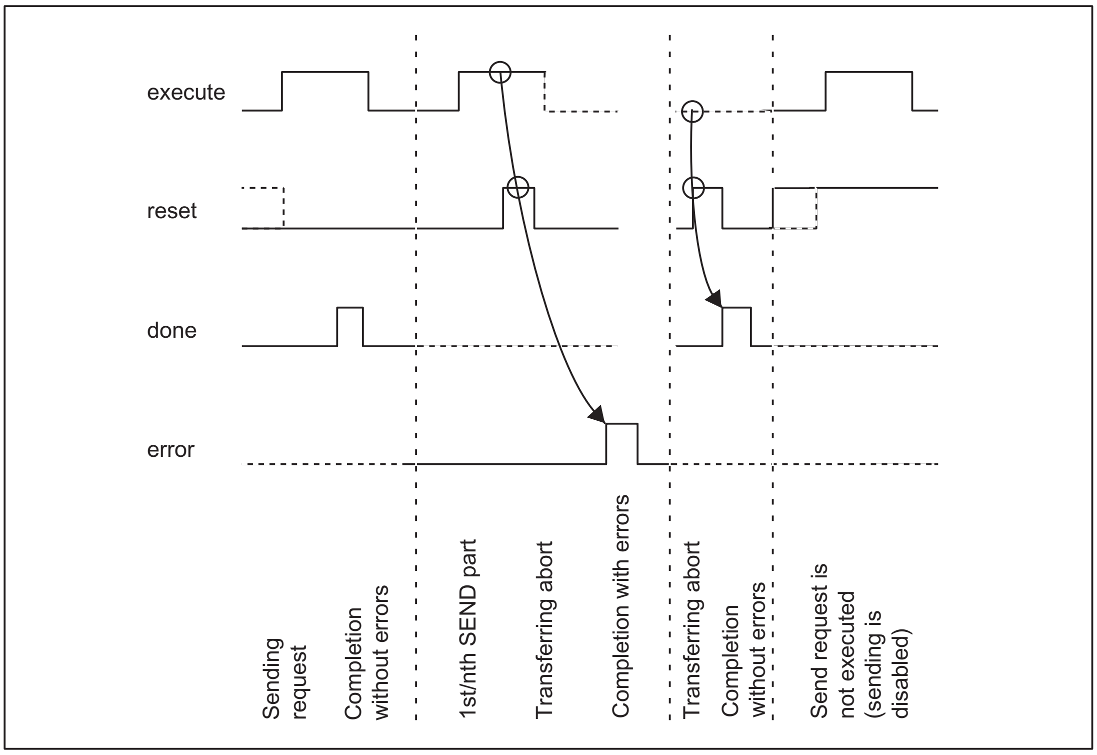


Figure 3-1 Signal sequence diagram of the _CP340_send FB

Note

The **execute** input is edge-triggered. The send job starts when there is a positive edge at the **execute** input.

Task integration (call)

The **_CP340_send** function block must be called cyclically in the **BackgroundTask** or the **TimerInterruptTask**. Calling in the **SystemInterruptTask** is not permitted. Calling the function block in the **IPOSynchronousTask** is not recommended for runtime reasons.

The SIMOTION device sends data to a communications partner

The **_CP340_send** function block transfers a data block that is specified by the following parameters to the CP 340:

- **data** corresponds to the data array containing the send data
- **dataOffset** corresponds to the array index containing the first send byte
- **dataLength** corresponds to the amount of data to be sent in bytes

The **_CP340_send** FB must be called repeatedly by a program. The send job can only be executed by cyclically calling the send FB.

A positive edge at the **execute** input initiates the transfer. A data transfer operation can run over several calls, depending on the amount of data involved.

The active transfer job can be canceled by setting the **reset** parameter to "TRUE". This will reset the **_CP340_send** FB to its initial state. The send operation will remain disabled as long as the signal state at the **reset** parameter is "TRUE".

The **moduleAddress** parameter specifies the module address of the CP 340 being addressed.

Status and error display on the _CP340_send FB

The **done** output indicates that the job has been completed without errors. The **error** output indicates that an error has occurred. If an error occurs, the corresponding event class/number is displayed in the **errorID** output (see "Parameters of the _CP340_send FB" table). If no errors have occurred, **errorID** has a value of 0. **Done** and **error/errorID** are also displayed for **reset** of the **_CP340_send** FB. When 16#1E0F is displayed in the **errorID** parameter, a detailed error description is also output via the **errorIdTransfer** parameter.

The **done**, **error, errorID**, and **errorIdTransfer** parameters are available for one block call only.

Note

There is no parameter check for the **_CP340_send** function block. Incorrect parameterization of this block may cause the SIMOTION device to switch to "STOP" mode.

Before the CP 340 can process an initiated job following a transition of the SIMOTION device from "STOP" to "RUN" mode, the CP-SIMOTION startup mechanism of the **_CP340_send** FB must be complete. Any jobs initiated in the meantime will not be lost. They are transferred to the CP 340 once the startup coordination has finished.

The end of the startup coordination is indicated in output parameter **startup** = FALSE.

Assignment in the data area

The data to be sent is transferred to the **_CP340_send** FB in the **data** parameter (VAR_IN_OUT) as ARRAY of BYTE. At the beginning of the send operation, the data is copied to a local variable of the FB and transferred from there to the CP 340.

Note

Once the data have been entered in the static memory of the send FB, you can modify the variable created in the **data** parameter. This does not affect the data to be sent.

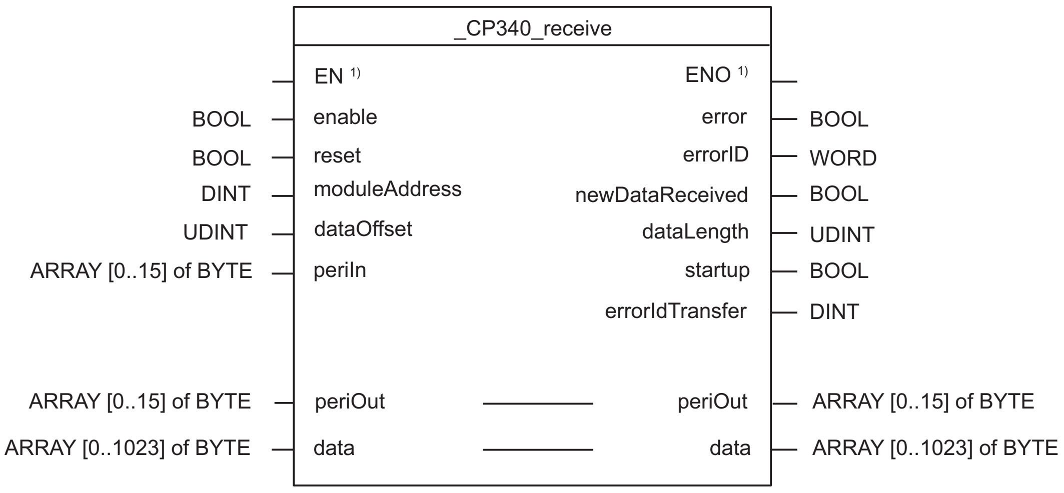
3.3 _CP340_receive function block

Function

The **_CP340_receive** function block enables you to receive data from a communications partner in the **data** receive field. You have 1,024 bytes available for this.

For the transfer, you can use the 3964 (R) protocol or ASCII driver.

Call (LAD representation)



1) LAD-specific parameters

Parameter description

Table 3-2 Parameters of the _CP340_receive FB

Name	P type 1)	Data type	Meaning	Actions performed by user	Actions performed by block
enable	IN	BOOL	Receive enable	Entered	Checked
reset	IN	BOOL	Cancels job	Entered	Checked
moduleAddress	IN	DINT	Module address of the CP for data set transfer (from HW Config)	Entered	Checked
dataOffset	IN	UDINT	Offset of the first element to be received	Entered	Checked
periIn	IN	ARRAY[0..15] of BYTE	I/O inputs of the CP transferred to the FB	I/O variable of the I/O inputs of the CP transferred to the FB	Checked
periOut	IN/OUT	ARRAY[0..15] of BYTE	Prepared FB data for the I/O outputs of the CP 2)	Checked and transferred to the I/O variable for the I/O outputs	Entered

Name	P type [1]	Data type	Meaning	Actions performed by user	Actions performed by block
data	IN/OUT	ARRAY[0..1023] of BYTE	Receive data array	Checked	Entered
error	OUT	BOOL	Job completed with errors	Checked	Entered
errorID	OUT	WORD	Error specification For **error**=TRUE, the error information (event class and number) is displayed in the **errorID** parameter.[3]	Checked	Entered
newDataReceived	OUT	BOOL	Receive new data	Checked	Entered
dataLength	OUT	UDINT	Quantity of data received	Checked	Entered
startup	OUT	BOOL	Indicates CP startup	Checked	Entered
errorIdTransfer	OUT	DINT	Error during data transfer between the CP and the SIMOTION device (precise error diagnostics if 16#1E0F is present in the **errorID** parameter [4])	Checked	Entered

[1] Parameter types: IN = input parameters, OUT = output parameters, IN/OUT = in/out parameters

[2] **Note:** The **periOut** parameter must be supplied with a variable of type **ARRAY[0..15] of BYTE**. Create a local or global variable in your program under **VAR** (do not create a temporary variable under VAR_TEMP). After the FB has been called, this variable must be assigned to the I/O variable for the I/O outputs of the module. See call example for CP 340.

[3] For error information, refer to the SIMATIC *CP 340 Point-to-Point Connection, Installation and Parameter Assignment* manual, Chapter "Diagnostics with the CP 340".

[4] For a more detailed description (_readRecord and _writeRecord), see *SIMOTION System Function/Variable Devices* Parameter Manual. This documentation is included in the SIMOTION SCOUT scope of delivery as electronic documentation.

Signal sequence diagram of the _CP340_receive FB

The following figure illustrates the behavior of the **newDataReceived**, **dataLength**, and **error** parameters according to the input circuit of **enable** and **reset**.

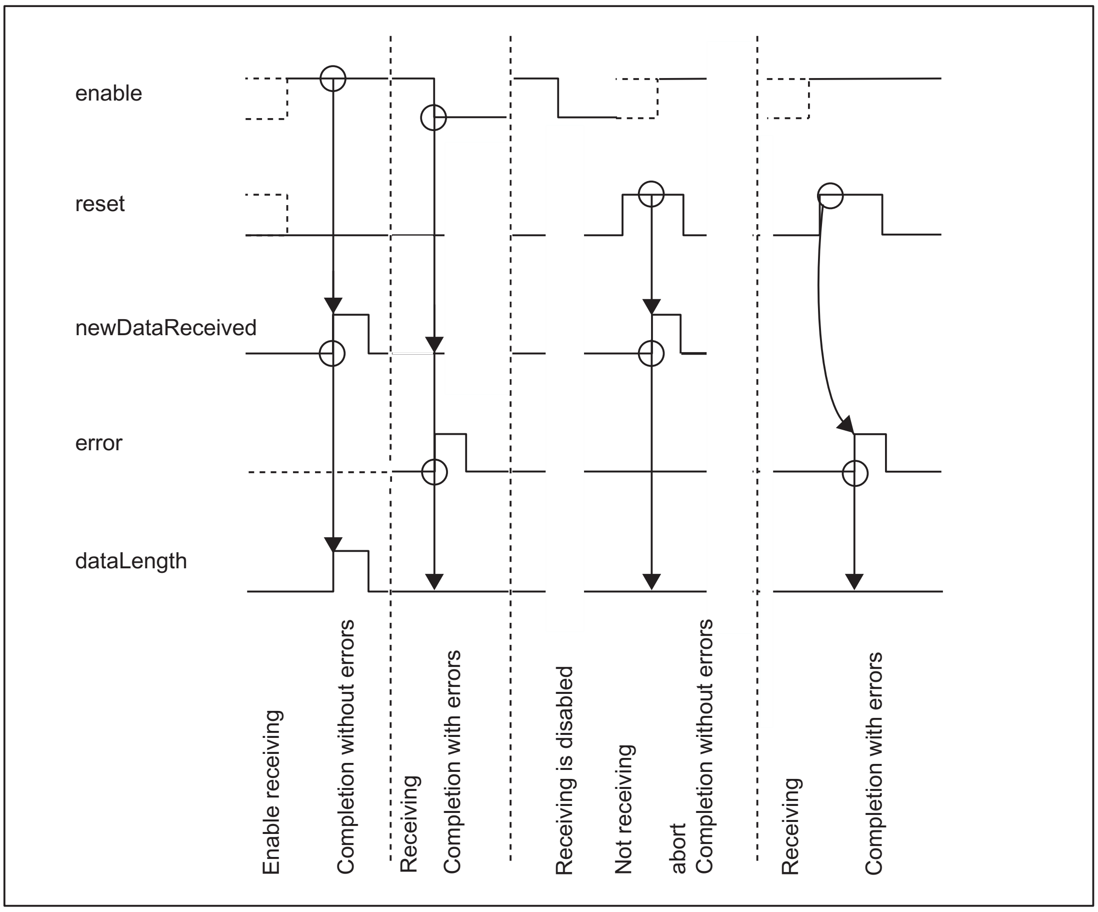


Figure 3-2 Signal sequence diagram of the _CP340_receive FB

Task integration (call)

The **_CP340_receive** function block must be called cyclically in the **BackgroundTask** or the **TimerInterruptTask**. Calling in the **SystemInterruptTask** is not permitted. Calling the function block in the **IPOSynchronousTask** is not recommended for runtime reasons.

SIMOTION device receives data from a communications partner

The **_CP340_receive** FB transfers a data block that is specified by the **data** and **dataOffset** parameters from the CP 340 to a SIMOTION device. The **_CP340_receive** function block must be called repeatedly by a program. The receive job can only be executed by cyclically calling the receive FB.

Receiving of data is enabled with static signal state "TRUE" in the **enable** parameter. An active data transfer can be canceled with signal state "FALSE" in the **enable** parameter. The canceled receive job is terminated with an error message (**errorID** output). The receive operation will remain disabled as long as the signal state at the **enable** parameter is "FALSE". A data transfer operation can run over several calls, depending on the amount of data involved.

The active transfer job can be canceled by setting the **reset** parameter to "TRUE". This will reset the **_CP340_receive** FB to its initial state. The receive operation will remain disabled as long as the signal state at the **reset** parameter is "TRUE".

The **moduleAddress** parameter specifies the module address of the CP 340 being addressed for the data set transfer.

Status and error display on the _CP340_receive FB

The **newDataReceived** output indicates that new data have been received without errors. The amount of data received is indicated in the **dataLength** parameter. The **error** output indicates that an error has occurred. If an error occurs, the corresponding event class/number is displayed in the **errorID** output (see "Parameters of the _CP340_receive FB" table). If no error has occurred, **errorID** has the value "0". **newDataReceived** and **error/errorID** will also be output when the **_CP340_receive** FB is **reset**. When 16#1E0F is displayed in the **errorID** parameter, a detailed error description is also output via the **errorIdTransfer** parameter.

The **newDataReceived**, **dataLength**, **error**, **errorID**, and **errorIdTransfer** parameters are available for one block passage only.

Note

There is no parameter check for the **_CP340_receive** function block. Incorrect parameterization of this block may cause the SIMOTION device to switch to STOP mode.

Before a job from the CP 340 can be received following a transition of the SIMOTION device from "STOP" to "RUN" mode, the CP SIMOTION startup mechanism of the **_CP340_receive**FB must be complete.

The end of the startup coordination is indicated in output parameter **startup** = FALSE.

Assignment in the data area

During the receive operation, the data to be received is stored temporarily in the FB. Once data transmission from the CP 340 to the SIMOTION device has been completed, the data is made available in the **data** parameter (VAR_IN_OUT) of the **_CP340_receive** FB.

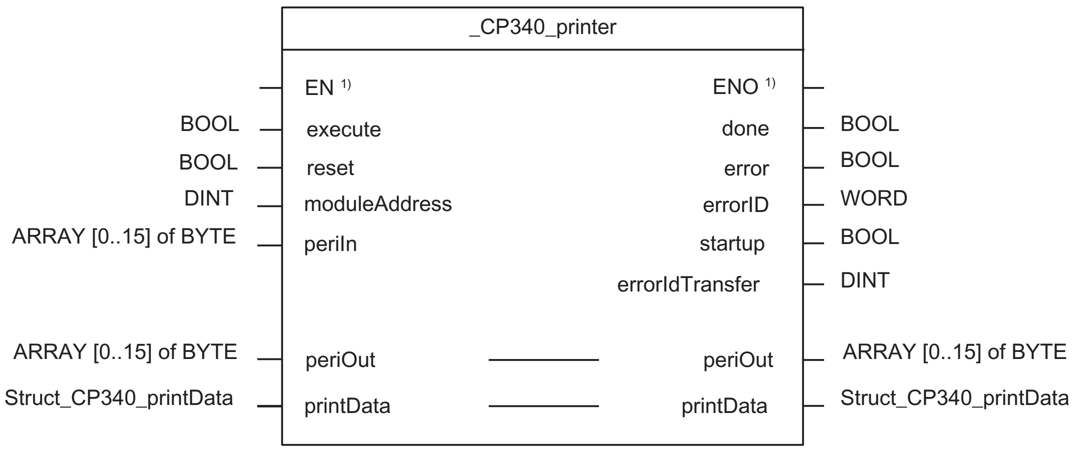
3.4 _CP340_printer function block

3.4.1 Description of the _CP340_printer FB

Function

The **_CP340_printer** function block is used to send data of type **Struct_CP340_printData** from the printer memory area to a serial printer. For example, the **_CP340_printer** function block might send a process message to the CP 340. The CP 340 prints out the process message on the connected printer.

Call (LAD representation)



1) LAD-specific parameters

Parameter description

Table 3-3 Parameters of the _CP340_printer FB

Name	P type 1)	Data type	Comment	Actions performed by user	Actions performed by block
execute	IN	BOOL	Initiates job on positive edge	Entered	Checked
reset	IN	BOOL	Cancels job	Entered	Checked
moduleAddress	IN	DINT	Module address of the CP for data set transfer (from HW Config)	Entered	Checked
periIn	IN	ARRAY[0..15] of BYTE	I/O inputs of the CP transferred to the FB	I/O variable of the I/ O inputs of the CP transferred to the FB	Checked

Name	P type [1]	Data type	Comment	Actions performed by user	Actions performed by block
periOut	IN/OUT	ARRAY[0..15] of BYTE	Prepared FB data for the I/O outputs of the CP [2]	Checked and transferred to the I/O variable for the I/O outputs	Entered
printData	IN/OUT	**Struct_CP340_ printData**	Send data array (data to be printed)	Entered	Checked
Struct_CP340_printData (data structure) [5]					
variable		ARRAY[0..3] of Struct_CP340_ dataRecord	Variable to be printed	Entered	Checked
format		ARRAY[0..150] of BYTE	Format string	Entered	Checked
Struct_CP340_dataRecord (data structure) [5]					
dataLength		UDINT	Quantity of data	Entered	Checked
data		ARRAY[0..31] of BYTE	Print data	Entered	Checked
done	OUT	BOOL	Job completed without errors	Checked	Entered
error	OUT	BOOL	Job completed with errors	Checked	Entered
errorID	OUT	WORD	Error specification For **error**=TRUE, the error information (event class and number) is displayed in the **errorID** parameter.[3]	Checked	Entered
startup	OUT	BOOL	Indicates CP startup	Checked	Entered
errorIdTransfer	OUT	DINT	Error during data transfer to the CP (precise error diagnostics if 16#1E0F is present at the **errorID** parameter [4])	Checked	Entered

[1] Parameter types: IN = input parameters, OUT = output parameters, IN/OUT = in/out parameters

[2] **Note:** The **periOut** parameter must be supplied with a variable of type **ARRAY[0..15] of BYTE**. Create a local or global variable in your program under **VAR** (do not create a temporary variable under VAR_TEMP). After the FB has been called, this variable must be assigned to the I/O variable for the I/O outputs of the module. See call example for CP 340.

[3] For error information, refer to the SIMATIC *CP 340 Point-to-Point Connection, Installation and Parameter Assignment* manual, Chapter "Diagnostics with the CP 340".

[4] For a more detailed description (_readRecord and _writeRecord), see *SIMOTION System Function/Variable Devices* Parameter Manual. This documentation is included in the SIMOTION SCOUT scope of delivery as electronic documentation.

[5] See "Message text structure", print storage area structure example

Signal sequence diagram of the _CP340_printer FB

The following figure illustrates the behavior of the **done** and **error** parameters according to the input circuit of **execute** and **reset.**

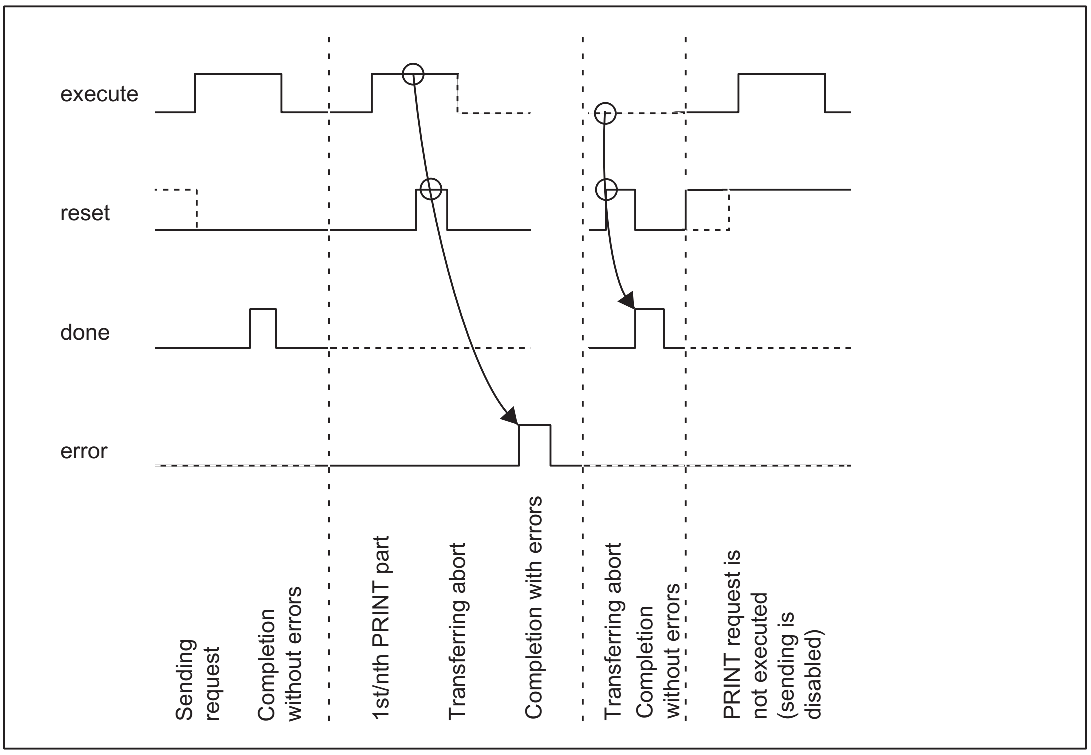


Figure 3-3 Signal sequence diagram of the _CP340_printer FB

Note

The **execute** input is edge-triggered. The send job starts when there is a positive edge at the **execute** input.

Task integration (call)

The **_CP340_printer** function block must be called cyclically in the **BackgroundTask** or the **TimerInterruptTask**. Calling in the **SystemInterruptTask** is not permitted. Calling the function block in the **IPOSynchronousTask** is not recommended for runtime reasons.

The **_CP340_printer** function block is called repeatedly by a program. The print job can only be executed by cyclically calling the print FB.

A positive edge at the **execute** input initiates the transfer of the message text. A data transfer operation can run over several calls (program cycles), depending on the amount of data involved.

The active transfer job can be canceled by setting the **reset** parameter to "TRUE". This will reset the **_CP340_printer** FB to its initial state. The sending of print jobs will remain disabled as long as the signal state at the **reset** parameter is TRUE.

The **moduleAddress** parameter specifies the module address of the CP 340 being addressed for the data set transfer.

For call examples for the **_CP340_printer** FB, see Chapter CP 340 print call examples (Page 40).

Status and error display on the _CP340_printer FB

The **done** output parameter indicates that the job has been completed without errors. The **error** output indicates that an error has occurred. If an error occurs, the corresponding event class/number is displayed in the **errorID** output parameter (see "Parameters of the CP340_printer FB" table). If no errors have occurred, **errorID** has a value of 0. **Done** and **error/ errorID** are also displayed for **reset** of the **_CP340_printer** FB. When 16#1E0F is displayed in the **errorID** parameter, a detailed error description is also output via the **errorIdTransfer** parameter.

The **done**, **error, errorID**, and **errorIdTransfer** parameters are available for one block call only.

Note

There is no parameter check for the **_CP340_printer** function block. Incorrect parameterization of this block may cause the SIMOTION device to switch to "STOP" mode. Before the CP 340 can process an initiated job following a transition of the SIMOTION device from "STOP" to "RUN" mode, the CP-SIMOTION startup mechanism of the **_CP340_printer** FB must be complete. Any jobs initiated in the meantime will not be lost. They are transferred to the CP 340 once the startup coordination has finished.

The end of the startup coordination is indicated in output parameter **startup** = FALSE.

Assignment in the data area

The print data is transferred to the **_CP340_printer** FB as a data structure of type **Struct_CP340_printData** and copied to a local variable of the FB at the beginning of the print operation.

Structure of printer memory of type Struct_CP340_printData

The four variables to be printed and the format string must be entered in a variable with the following data type:

Example:

```
Struct_CP340_dataRecord     : STRUCT
     dataLength : UDINT;                      // Data quantity
     data       : ARRAY [0..31] of BYTE;      // Data field
END_STRUCT


Struct_CP340_printData    : STRUCT
     variable   : ARRAY [0..3] of Struct_CP340_dataRecord;  // 1st to 4th variable
     format     : ARRAY [0..150] of BYTE;                   // Format string
```

```
END_STRUCT
```

The first variable to be printed corresponds to the **variable [0]** element, the second variable to be printed corresponds to the **variable [1]** element, etc. The number of bytes to be printed per variable is limited to 32. The data for variable **i** must be placed in variable [**i-1**].data[0..31]. The number of bytes to be printed must be entered in the **variable [i-1].dataLength** element.

The format string corresponds to the **format** element. The format string must be structured as follows (refer to the SIMATIC *CP 340 Point-to-Point Connection, Installation and Parameter Assignment* manual):

- Specification of string length in **format [0]**
- Specification of individual characters in **format [1 to 150]**

Note

If the maximum length is exceeded, the print job is canceled and event number 16#1E41 is indicated at the **errorID** parameter output of the **_CP340_printer** FB.

Entering variables and message texts in the printer memory area

Before the data transfer to the CP 340 begins, the variable values to be printed must be entered **byte by byte and in the proper format** in the **variable[].data** element of the data structure of type **Struct_CP340_printData** (see item 2 in the example below). The number of bytes for each variable (variable length) must be assigned to the **variable.datalength** element (e.g. WORD type variable - variable.datalength:=2). An entry corresponding to the data type of the value must be made in the **format** element for each value entered in the **variable** element. (e.g. WORD type variable - %I). The total length of the entries in the **format** element must be assigned to the **format[0]** element.

You configure message texts with the CP 340 "point-to-point connection" parameter assignment interface. Once the hardware configuration has been downloaded to the SIMOTION device, the message texts are stored in the CP 340. The message texts that have been saved can be selected with corresponding entries in the **variable** and **format** elements.

Note

You can use supplemental function blocks (see Chapter supplemental function blocks (Page 35)) to enter values into the printer memory area and to select message texts.

Example:

- Print message text no. 3 (stored in CP 340).
  Configured message text: "This is message text no. 3"

```
myPrintData.variable[0].datalength := 1;
myPrintData.variable[0].data[0]    := 3;   // Message text no. 3
myPrintData.format[0]      :=2;      // Format string length
myPrintData.format[1]     :=16#25;  // "%" Format specification for message
text
myPrintData.format[1]     :=16#4E   // "N" Format specification for message
text
```

- Print message text no. 4 with a WORD-type variable.

```
Configured message text   : "This is message text no. %I"
Printed text          : "This is message text no. 4"

myPrintData.variable[0].datalength := 1;
myPrintData.variable[0].data[0]    := 4;  // Message text no.4

myPrintData.variable[1].datalength := 2;  // 2 Byte data type WORD
myPrintData.variable[1].data[0]    := 0   // High - Byte
myPrintData.variable[1].data[1]    := 4   // Low - Byte
myPrintData.format[0]              := 4;  // Format string length
myPrintData.format[1]          := 16#25;  // ASCII code "%" format
specification
myPrintData.format[2]          := 16#4E;  // ASCII code "N" format
specification
                                          // for message text
myPrintData.format[3]          := 16#25;  // ASCII code "%" format
specification
myPrintData.format[4]          := 16#49;  // ASCII code "I" format
specification
                                          // for integer
```

Notes on handling

The format string entry in the "format" field must be hexadecimal.

Example:
% corresponds to 25 hex in the IBM character set,
N (message text output) corresponds to 4E hex in the IBM character set.
(see Hardware configuration > Character set)

Data types DATE, TIME, DATE_AND_TIME_OF_DAY and TIME_OF_DAY are not supported. The date information must be entered as a DWORD or WORD in the printer data structure.

Representation type "A" (German date format):
//datefrg:=4018 (01.01.2001) and 4199 (01.07.2001)
printData.variable[0].datalength:=2;
printData.variable[0].data[0]:=WORD_TO_BYTE(SHR(datefrg,8));
printData.variable[0].data[1]:=WORD_TO_BYTE(SHR(datefrg,0));

Representation type "F":
The value to be printed must be in floating point format (mantissa/exponent) (see call example 2, Chapter CP 340 print call examples (Page 40))

Representation type "C":
If variable[ ].datalength:=1 in the printer data structure, the characters will be printed horizontally.
If variable[ ].datalength:=2 (3,4) in the printer data structure, the characters will be printed vertically.

Representation type "X":
For CP 340 RS232, product version E08 and higher, representation type "X" (binary) outputs the values correctly on a serial printer.

Disconnected printer
The communications link is not monitored for printers even if alarm generation is enabled in HW Config of STEP 7.

Example:

- A break in the connection between the printer and the CP 340 triggers neither an error nor a diagnostic alarm.
- Nor are they triggered if a print job is started but no printer is connected.

Note

The code in examples 1, 2, and 3 (see Chapter CP 340 print call examples (Page 40)) can be transferred to the SIMOTION SCOUT editor with **Copy** and **Paste**.

3.4.2 supplemental function blocks

Function

Supplemental function blocks are provided for entering variables of various data types as well as for entering message texts into data structure **Struct_CP340_printData**. You enter the value of the variables **byte by byte** and **in the proper format** in the **variable** element of the printer memory area and, optionally, in the **format** element. When numbers are entered for message texts, one entry is made in each of the **format** and **variable** elements. The method of representation for the variables in printed text and the method of entry in the format string can be selected by means of parameters.

The following supplemental function blocks are available:

- **_CP340_realToPrintData**
  Entry of a number of data type REAL into data structure **Struct_CP340_printData**
- **_CP340_dwordToPrintData**
  Entry of a number of data type DWORD into data structure **Struct_CP340_printData**
- **_CP340_wordToPrintData**
  Entry of a number of data type WORD into data structure **Struct_CP340_printData**
- **_CP340_byteToPrintData**
  Entry of a number of data type BYTE into data structure **Struct_CP340_printData**
- **_CP340_dintToPrintData**
  Entry of a number of data type DINT into data structure **Struct_CP340_printData**

- **_CP340_intToPrintData**
  Entry of a number of data type INT into data structure **Struct_CP340_printData**
- **_CP340_printMsgText**
  Selection of message texts stored in the CP 340

Note

SINT and USINT data types can be entered into data structure **Struct_CP340_printData** with the **_CP340_intToPrintData** function block. Type conversion is implicit.

Parameter description

Table 3-4 _CP340_byteToPrintData, _CP340_wordToPrintData, _CP340_dwordToPrintData parameters

Name	P type [1)]	Data type	Comment	Actions performed by user	Actions performed by block
execute	IN	BOOL	Edge-triggered job initiation	Entered	Checked
data	IN	BYTE/WORD/ DWORD	Variable/value to be entered in the data structure.	Entered	Checked
numVariable	IN	INT	Number of the variable in which the entry is to be made. 1≤ **numVariable** ≤ 4	Entered	Checked
entryFormatString	IN	ENUM	Method of entry in the format string and method of representation of the value in the **data** parameter.	Entered	Checked
		CP_REPLACE_WITH_SIGN	Entry starting at byte 1 in the substructure of the format string; existing entries are overwritten. Method of representation: signed integer		
		CP_REPLACE_WITHOUT_SIGN	Entry starting at byte 1 in the substructure of the format string; existing entries are overwritten. Method of representation: unsigned integer		
		CP_REPLACE_BINARY	Entry starting at byte 1 in the substructure of the format string; existing entries are overwritten. Method of representation: binary		
		CP_ADD_WITHOUT_SIGN	Entry is added to the substructure of the format string, **entryAtByteNumber** parameter is evaluated. Method of representation: unsigned integer		
		CP_ADD_WITH_SIGN	Entry is added to the substructure of the format string, **entryAtByteNumber** parameter is evaluated. Method of representation: signed integer		

Name	P type [1)]	Data type	Comment	Actions performed by user	Actions performed by block
		CP_NO_ENTRY	No entry in the substructure of the format string		
entryAtByteNumber	IN	USINT	Specifies the byte at which the entry is to begin in the substructure of the format string. **entryAtByteNumber** = 0 Entry is made after the last entry found	Entered	Checked
printData	IN/OUT	Struct_CP340_printData	Data structure for the printer data	No actions	Enters values
done	OUT	BOOL	Job completed without errors	Checked	Entered
error	OUT	BOOL	Job completed with errors (permissible value range exceeded)	Checked	Entered

1) Parameter types: IN = input parameters, OUT = output parameters, IN/OUT = in/out parameters

Table 3-5 _CP340_realToPrintData

Name	P type [1)]	Data type	Comment	Actions performed by user	Actions performed by block
execute	IN	BOOL	Edge-triggered job initiation	Entered	Checked
data	IN	REAL	Variable/value to be entered in the data structure.	Entered	Checked
numVariable	IN	INT	Number of the variable in which the entry is to be made. 1 ≤ **numVariable** ≤ 4	Entered	Checked
entryFormatString	IN	ENUM	Method of entry in the format string and method of representation of the value in the **data** parameter.	Entered	Checked
		CP_REPLACE_WITHOUT_EXPONENT	Entry starting at byte 1 in the substructure of the format string; existing entries are overwritten. Method of representation: floating-point		
		CP_REPLACE_WITH_EXPONENT	Entry starting at byte 1 in the substructure of the format string; existing entries are overwritten. Method of representation: with exponent		
		CP_ADD_WITHOUT_EXPONENT	Entry is added in the substructure of the format string, **entryAtByteNumber** parameter is evaluated. Method of representation: floating-point		
		CP_ADD_WITH_EXPONENT	Entry is added in the substructure of the format string, **entryAtByteNumber** parameter is evaluated. Method of representation: with exponent		

Name	P type [1)]	Data type	Comment	Actions performed by user	Actions performed by block
		CP_NO_ENTRY	No entry in the substructure of the format string		
entryAtByteNumber	IN	USINT	Specifies the byte at which the entry is to begin in the substructure of the format string. **entryAtByteNumber** = 0 Entry is made after the last entry found	Entered	Checked
printData	IN/OUT	Struct_CP340_printData	Data structure for the printer data	No actions	Enters values
done	OUT	BOOL	Job completed without errors	Checked	Entered
error	OUT	BOOL	Job completed with errors (permissible value range exceeded)	Checked	Entered

1) Parameter types: IN = input parameters, OUT = output parameters, IN/OUT = in/out parameters

Table 3-6 _CP340_dintToPrintData, _CP340_intToPrintData

Name	P type [1)]	Data type	Comment	Actions performed by user	Actions performed by block
execute	IN	BOOL	Edge-triggered job initiation	Entered	Checked
data	IN	INT/DINT	Variable/value to be entered in the data structure.	Entered	Checked
numVariable	IN	INT	Number of the variable in which the entry is to be made. 1 ≤ **numVariable** ≤ 4	Entered	Checked
entryFormatString	IN	ENUM	Method of entry in the format string and method of representation of the value in the **data** parameter.	Entered	Checked
		CP_DEFAULT	Entry starting at byte 1 in the substructure of the format string; existing entries are overwritten. Method of representation: signed integer		
		CP_ADD_TO_STRING	Entry is added to the substructure of the format string, **entryAtByteNumber** parameter is evaluated. Method of representation: signed integer		
		CP_NO_ENTRY	No entry in the substructure of the format string		
entryAtByteNumber	IN	USINT	Specifies the byte at which the entry is to begin in the substructure of the format string. **entryAtByteNumber** = 0 Entry is made after the last entry found	Entered	Checked

Name	P type [1)]	Data type	Comment	Actions performed by user	Actions performed by block
printData	IN/OUT	Struct_CP340_printData	Data structure for the printer data	No actions	Enters values
done	OUT	BOOL	Job completed without errors	Checked	Entered
error	OUT	BOOL	Job completed with errors (permissible value range exceeded)	Checked	Entered

[1)] Parameter types: IN = input parameters, OUT = output parameters, IN/OUT = in/out parameters

Table 3-7 _CP340_printMsgText

Name	P type [1)]	Data type	Comment	Actions performed by user	Actions performed by block
execute	IN	BOOL	Edge-triggered job initiation	Entered	Checked
numMsgText	IN	USINT	Number of the message text (stored in the CP 340)	Entered	Checked
numVariable	IN	INT	Number of the variable in which the entry is to be made. 1 ≤ **numVariable** ≤ 4	Entered	Checked
entryFormatString	IN	ENUM	Method of entry in the format string and method of representation of the value in the **data** parameter.	Entered	Checked
		CP_DEFAULT	Entry starting at byte 1 in the substructure of the format string; existing entries are overwritten. Method of representation: signed integer		
		CP_ADD_TO_STRING	Entry is added to the substructure of the format string, **entryAtByteNumber** parameter is evaluated. Method of representation: signed integer		
		CP_NO_ENTRY	No entry in the substructure of the format string		
entryAtByteNumber	IN	USINT	Specifies the byte at which the entry is to begin in the substructure of the format string **entryAtByteNumber** = 0 Entry is made after the last entry found	Entered	Checked
printData	IN/OUT	Struct_CP340_printData	Data structure for the printer data	No actions	Enters values
done	OUT	BOOL	Job completed without errors	Checked	Entered
error	OUT	BOOL	Job completed with errors (permissible value range exceeded)	Checked	Entered

[1)] Parameter types: IN = input parameters, OUT = output parameters, IN/OUT = in/out parameters

Task integration (call)

The supplemental blocks must be called in the **BackgroundTask** or the **TimerInterruptTask**.

For call examples for the **_CP340_printer** FB, see Chapter CP 340 print call examples (Page 40).

Status and error indicators

The **done** output indicates that the job has been completed without errors. The **error** output indicates that an error has occurred.

The **done** and **error** parameters are available for one block call only.

3.4.3 CP 340 print call examples

Call example 1

```
UNIT E1CP340p;

INTERFACE
VAR_GLOBAL
  myPreparePrintData          : BOOL;              // Initiate prepare print request
  myRequestPrint              : BOOL;              // Initiate transfer to printer
  myReset                     : BOOL;              // Abort print
  myActualLevel               : REAL := 5.67;      // actual value "level"
  myModuleAddress_1           : DINT:= 256;        // address of 1st CP340 module
  myPrintData                 : Struct_CP340_printData; // instance of datastruct
  myFB_CP340_printMessageText : _CP340_printMsgText;    // instances of function blocks
  myFB_CP340_realToPrintData  : _CP340_realToPrintData; // instances of function blocks
  myFB_CP340_print            : _CP340_printer;         // instance of function block
  myOutputArrayCP340_1        : ARRAY[0..15] of BYTE;   // field for CP340 output data
END_VAR
PROGRAM Example_print_1;                           // program for BackgroundTask
END_INTERFACE

IMPLEMENTATION
PROGRAM Example_print_1
// BackgroundTask program

// Example to print out messagetext 3 with one variable
// ("actualLevel") of type REAL: "The actual level (l) is: <actualLevel>"
// The message-text 3 must be specified in the hardwareconfiguration of CP340
// like this: "The actual level (l) is:"
// The following I/O-variable for CP340 module are required:
// peripheralInputCP340_1: input address of CP340 module; type Array; length 16
// peripheralOutputCP340_1: output address of CP340 module; type Array; length 16

// entry to printDatastruct myprintData
myFB_CP340_printMessageText ( execute           := myPreparePrintData,
                              printData         := myPrintData,
                              numMsgText        := 3,        // number of message text
                              numVariable       := 1,        // number of variable
                              entryFormatString := CP_DEFAULT);

myFB_CP340_realToprintData  ( execute           := myPreparePrintData,
                              printData         := myPrintData,
                              data              := myActualLevel,
                              numVariable       := 2,
                              entryFormatString := CP_ADD_WITH_EXPONENT );
// call instance of _CP340_printer
// use requestPrint to start datatransfer to serial printer

myFB_CP340_print      ( execute       := myRequestPrint,           // initiate request
                        reset         := myReset,                  // abort request
                        moduleAddress := myModuleAddress_1,        // module address
                        periIn        := myPeripheralinputcp340_1, // peripheral input
                        periOut       := myOutputArrayCP340_1,     // output data field
                        printData     := myPrintData);             // data to print out

// transfer output data field to peripheral output
myPeripheralOutputCP340_1 := myOutputArrayCP340_1;

END_PROGRAM //Example_print_1
END_IMPLEMENTATION
```

Call example 2

```
UNIT E2CP340p;

INTERFACE
VAR_GLOBAL
  myRequestPrint        : BOOL;                      // Initiate transfer to printer
  myReset               : BOOL;                      // abort print
  myPrintDword          : DWORD;                     // value to print out
  myModuleAddress_1     : DINT := 256;               // address of 1st CP340 module
  myPrintData           : Struct_CP340_printData;    // instance of datastruct
  myFB_CP340_print      : _CP340_printer;            // instance of function block
  myOutputArrayCP340_1  : ARRAY[0..15] OF BYTE;      // field for CP340 output data
END_VAR
PROGRAM Example_print_2;                             // program for BackgroundTask
END_INTERFACE

IMPLEMENTATION
PROGRAM Example_print_2                              // BackgroundTask program
// example with notation "F" and variable of type DWORD
// (containing mantissa and exponent)

// The following I/O-variable for CP340 module are required:
// peripheralInputCP340_1: input address of CP340 module; type Array; length 16
// peripheralOutputCP340_1: output address of CP340 module; type Array; length 16

// formatstring (length 2 Byte, notation "F"):
myPrintData.format[0]  := 2 ;
myPrintData.format[1]  := 16#25 ;            // "%"
myPrintData.format[2]  := 16#46 ;            // "F"

// assignment for variable (type DWORD), to print out with notation "F"
myPrintDword := REAL_TO_DWORD(10000.0);  // variable (DWORD) with mantissa and exponent

// !!!
// ATTENTION! wrong example for this case is an assingnment with an integer value e.g.:
// myprintDword := 10000;        // because the format is WITHOUT mantissa and exponent
// !!!

// fill out printData interface manually with DWORD-variable
myPrintData.variable[0].dataLength := 4 ;    // 1st variable, lenth 4 Byte
myPrintData.variable[0].data[0] := WORD_TO_BYTE(DWORD_TO_WORD(SHR(printDword,24)));
myPrintData.variable[0].data[1] := WORD_TO_BYTE(DWORD_TO_WORD(SHR(printDword,16)));
myPrintData.variable[0].data[2] := WORD_TO_BYTE(DWORD_TO_WORD(SHR(printDword,8)));
myPrintData.variable[0].data[3] := WORD_TO_BYTE(DWORD_TO_WORD(SHR(printDword,0)));

// call instance of _CP340_printer
// use requestPrint to start datatransfer to serial printer
myFB_CP340_print ( execute        := myRequestPrint,            // initiate request
                   reset          := myReset,                   // abort request
                   moduleAddress  := myModuleAddress_1,         // module address
                   periIn         := myPeripheralinputcp340_1,  // peripheral input
                   periOut        := myOutputArrayCP340_1,      // output data field
                   printData      := myPrintData);              // data to print out

// transfer output data field to peripheral output
myPeripheralOutputCP340_1 := myOutputArrayCP340_1;

END_PROGRAM //Example_print_2
END_IMPLEMENTATION
```

Call example 3

```
UNIT E3CP340p;

INTERFACE
VAR_GLOBAL
  myRequestPrint              : BOOL;                    // Initiate transfer to printer
  myReset                     : BOOL;                    // Abort print
  myPrintReal1                : REAL;                    // 1st value
  myPrintReal2                : REAL;                    // 2nd value
  myPrintReal3                : REAL;                    // 3rd value
  myModuleAddress_1           : DINT := 256;             // address of 1st CP340 module
  myPrintData                 : Struct_CP340_printData;  // instance of datastruct
  myFB_CP340_print            : _CP340_printer;          // instance of function block
  myFB_CP340_realToprintData1 : _CP340_realToPrintData;  // instances of function blocks
  myFB_CP340_realToprintData2 : _CP340_realToPrintData;  // instances of function blocks
  myFB_CP340_realToprintData3 : _CP340_realToPrintData;  // instances of function blocks
  myOutputArrayCP340_1        : ARRAY[0..15] OF BYTE;
END_VAR
PROGRAM Example_print_3;                                 // program for BackgroundTask
END_INTERFACE

IMPLEMENTATION
PROGRAM Example_print_3                                  // BackgroundTask program
// The following example program demonstrates usage of _CP340_realToPrintData()

// The following I/O-variable for CP340 module are required:
// peripheralInputCP340_1: input address of CP340 module; type Array; length 16
// peripheralOutputCP340_1: output address of CP340 module; type Array; length 16

// preset variables with user-values
myPrintReal1 := 1.11; myPrintReal2 := 2.22; myPrintReal3 := 3.33;

// write variables (type REAL) to printDatastruct with _CP340_realToPrintData
```

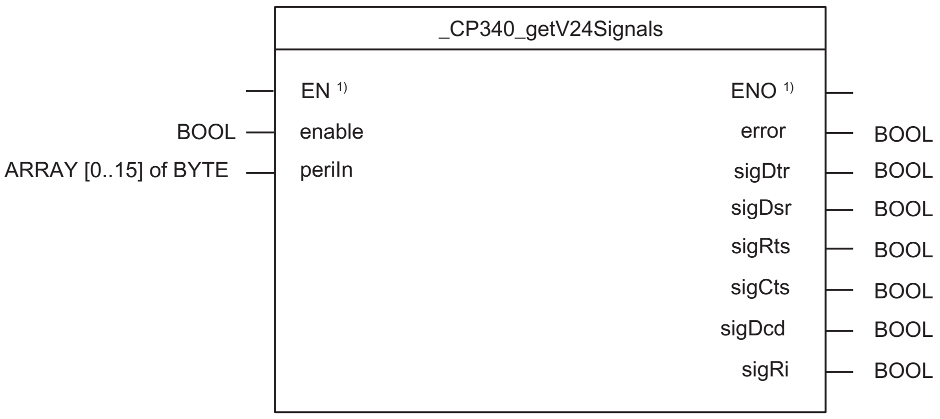
```
myFB_CP340_realToprintData1 (execute           := TRUE,
                             data              := myPrintReal1,        // 1st variable
                             numVariable       := 1,
                             entryFormatString := CP_ADD_WITHOUT_EXPONENT,
                             entryAtByteNumber := 0,
                             printData         := myPrintData);
myFB_CP340_realToprintData2 (execute           := TRUE,
                             data              := myPrintReal2;        // 2nd variable
                             numVariable       := 2,
                             entryFormatString := CP_ADD_WITHOUT_EXPONENT,
                             entryAtByteNumber := 0,
                             printData         := myPrintData);
myFB_CP340_realToprintData3 (execute           := TRUE,
                             data              := myPrintReal3,        // 3rd variable
                             numVariable       := 3,
                             entryFormatString := CP_ADD_WITHOUT_EXPONENT,
                             entryAtByteNumber := 0,
                             printData         := myPrintData);

// call instance of _CP340_printer, "requestPrint" starts data transfer
// to serial printer
myFB_CP340_print (  execute        := myRequestPrint,            // initiate request
                    reset          := myReset,                   // abort request
                    moduleAddress  := myModuleAddress_1,         // module address
                    periIn         := myPeripheralinputcp340_1,  // peripheral input
                    periOut        := myOutputArrayCP340_1,      // output data field
                    printData      := myPrintData);              // data to print out

// transfer output data field to peripheral output
myPeripheralOutputCP340_1 := myOutputArrayCP340_1;

END_PROGRAM                       //Example_print_3
END_IMPLEMENTATION
```

3.5 _CP340_getV24Signals function block

Function

The **_CP340_getV24Signals** function block reads the RS-232-C accompanying signals from the CP 340 and makes them available to the user in the block parameters. The functionality of the **_CP340_getV24Signals** FB can only be used if a parameterized ASCII driver is specified.

Call (LAD representation)



1) LAD-specific parameters

Parameter description

Table 3-8 Parameters of the _CP340_getV24Signals FB

Name	P type 1)	Data type	Meaning	Actions performed by user	Actions performed by block
enable	IN	BOOL	Block enable	Entered	Checked
periIn	IN	ARRAY[0..15] of BYTE	I/O inputs of the CP transferred to the FB	I/O variable of the I/O inputs of the CP transferred to the FB	Checked
error	OUT	BOOL	Job completed with errors	Checked	Entered
sigDtr	OUT	BOOL	Data terminal ready	Checked	Entered
sigDsr	OUT	BOOL	Data set ready	Checked	Entered
sigRts	OUT	BOOL	Request to send	Checked	Entered
sigCts	OUT	BOOL	Clear to send	Checked	Entered
sigDcd	OUT	BOOL	Data carrier detected	Checked	Entered
sigRi	OUT	BOOL	Ring Indicator	Checked	Entered

1) Parameter types: IN = input parameters, OUT = output parameters, IN/OUT = in/out parameters

Task integration (call)

The **_CP340_getV24Signals** function block must be called cyclically in the **BackgroundTask** or in the **TimerInterruptTask**. Calling in the **SystemInterruptTask** is not permitted. For runtime reasons, calling the FB in the **IPOSynchronousTask** is not recommended.
The RS 232C accompanying signals will be updated with each FB call (cyclical polling). The CP 340 updates the status of the inputs/outputs in a time base of 20 ms.

Note

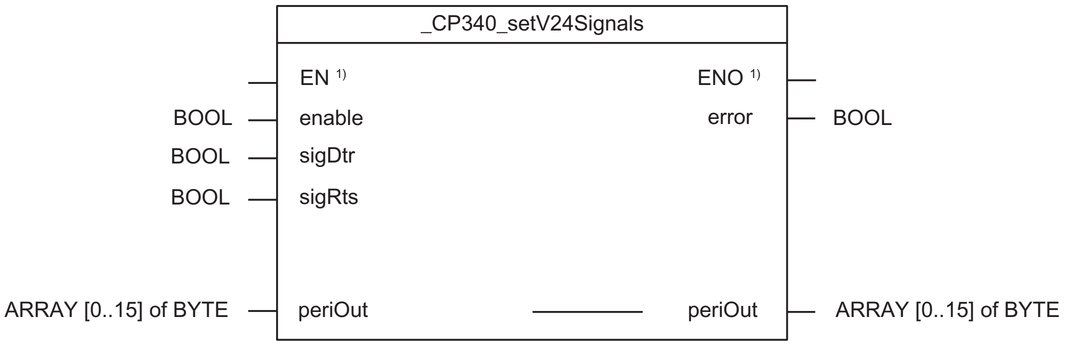
A minimum pulse duration is necessary to detect a signal change. Determining factors are the cycle time (SIMOTION device), the update time on the CP 340, and the response time of the communications partner.

3.6 _CP340_setV24Signals function block

Function

The **_CP340_setV24Signals** function block can be used to set or reset RS 232C accompanying signals. The functionality of the **_CP340_setV24Signals** FB can only be used if a parameterized ASCII driver is specified.

Call (LAD representation)



1) LAD-specific parameters

Parameter description

Table 3-9 Parameters of the _CP340_setV24Signals FB

Name	P type 1)	Data type	Meaning	Actions performed by user	Actions performed by block
enable	IN	BOOL	Block enable	Entered	Checked
sigDtr	IN	BOOL	Data terminal ready	Entered	Checked
sigRts	IN	BOOL	Request to send	Entered	Checked
periOut	IN/OUT	ARRAY[0..15] of BYTE	Prepared FB data for the I/O outputs of the CP 2)	Checked and transferred to the I/O variable for the I/O outputs	Entered
error	OUT	BOOL	Job completed with errors	Checked	Entered

1) Parameter types: IN = input parameters, OUT = output parameters, IN/OUT = in/out parameters

2) **Note:** The **periOut** parameter must be supplied with a variable of type **ARRAY[0..15] of BYTE**. Create a local or global variable in your program under **VAR** (do not create a temporary variable under VAR_TEMP). After the FB has been called, this variable must be assigned to the I/O variable for the I/O outputs of the module. See call example for CP 340.

Task integration (call)

The **_CP340_setV24Signals** function block must be called cyclically in the **BackgroundTask** or in the **TimerInterruptTask**. Calling in the **SystemInterruptTask** is not permitted. Calling the function block in the **IPOSynchronousTask** is not recommended for runtime reasons.

3.7 Calling the CP 340 function blocks

In order to be able to work with the function blocks in your user project, proceed as follows (the numbers shown in the following program segment correspond to the steps below):

1. Create the function block instance (see the following program segment, e.g. create instance for the **_CP340_send** FB).
2. Create a variable for the data structure (for FB **_CP340_printer** only).
3. Create an array for the in/out parameters of the FB.
4. Call instance of the function block.
5. Transfer input parameters.
6. The output parameters of the FB are accessed with <instance name of FB>. <name of the output parameter>.
7. Data prepared by the FB for the I/O outputs are assigned to the array of the I/O variable created in step 3.

Note

The CP 340 call example is an extract from the supplied E_CP340 application example, which is included on the "SIMOTION Utilities & Applications" CD-ROM.

If you wish to control multiple CP 340 devices, you must create a new variable for the data structure and FB instances with a new name for each CP 340 you implement.

Call example for CP 340

```
UNIT E_CP340;

INTERFACE
  VAR_GLOBAL
   myExecSendCP340      : BOOL;                    // Trigger send task
   myModuleAddrCP340    : DINT:=256;               // module address
   mySendDataArrayCP340 : ARRAY [0..1023] OF BYTE; // Send data array 1024 bytes
   myInstCP340Send      : _CP340_send;             // Create instance of FB                   (1)
  END_VAR

  PROGRAM ExampleCP340;          // Program in BackgroundTask

END_INTERFACE

IMPLEMENTATION
  VAR_GLOBAL
   MyResetSend          : BOOL;                    // Cancel send order
  END_VAR

  VAR
```

```
   MyCPOutputArray      : ARRAY [0..15] OF BYTE;   // Array for CP output data              (3)
  END_VAR

  VAR_TEMP
   MyDataLengthSend     : UDINT;             // Length of data to be sent
   MyDataOffsetSend     : UDINT;             // Offset of first byte to be sent
  END_VAR

   // CALL FB INSTANCE TO SEND                                                               (4)
   myInstCP340Send (                                                                         (5)
          execute       := myExecSendCP340,         // Trigger order
          reset         := myResetSend,             // Order cancellation
          moduleAddress := myModuleAddrCP340,       // Module address
          dataOffset    := myDataOffsetSend,        // Data offset
          dataLength    := myDataLengthSend,        // Number of data to be sent
          periIn        := myPeripheralInputCP340,  // I/O variable of I/O inputs
          periOut       := myCPOutputArray,         // Output data array
          data          := mySendDataArrayCP340     // Send data array
          );

   myStateStartUpCP340  := myInstCP340Send.startUp; // Module start-up status               (6)

   // TRANSFER DATA TO CP340
   myPeripheralOutputCP340 := myCPOutputArray;  // Assign array for CP output data

END_PROGRAM                                 // ExampleCP340                                  (7)

END_IMPLEMENTATION
```

Note

The PROGRAM "ExampleCP340" must be assigned in the execution system.

3.8 Data consistency

When sending data

Once a job is initiated by a positive edge at the **execute** input, the data to be sent is copied to the static memory area of the send FB. This means that once the FB call has ended, the send data array can be written to again for the next send request on a positive edge. The data are retained as consistent data within the send FB.

Note

During the copy operation to the static memory area of the FB, data consistency cannot be guaranteed if the send/receive data areas are accessed in a higher priority task.

When receiving data

Once the receive request is complete, the data are copied over to the receive buffer in a block from the static memory area of the receive FB. This means that once the FB call has ended, either all data are entered in the receive buffer (**newDataReceived** = TRUE) or no data are entered in the receive buffer (**newDataReceived** = FALSE).

Note

During the copy operation from the static memory area of the receive FB, data consistency cannot be guaranteed if the send/receive data areas are accessed in a higher priority task.

When printing

The data in the **printData** parameter in the static memory area of the FB are copied when a positive edge occurs at the **execute** input of the **_CP340_printer** FB. This means that once the FB call has ended, you can write to the variable and the format string for the next print request.

3.9 Application Examples

3.9.1 sending and receiving with CP 340

Function

This example shows how to:

- use the **_CP340_send** function block to send data from the Send data array to a communications partner.
- use the **_CP340_receive** function block to receive data in the receive data array.

In the example program, the CP 340 is used both as sender and as receiver. This requires the jumpering of the send and receive lines (PIN 2 and PIN 3 on the RS232 interface) and the "ASCII" setting in the parameter assignment tool. The **_CP340_send** FB is used to transfer the send data to the CP module. This sends the data using the RS232 interface. The jumpered send and receive line means that the data to be sent is read immediately by the CP module. The **_CP340_receive** FB reads the received data from the CP module and copies these data to the receive data array.

This example requires proper installation of the parameter assignment tool, as described in the SIMATIC *CP 340 Point-to-Point Connection, Installation and Parameter Assignment* manual.

Hardware platform

The application example is available for various SIMOTION hardware platforms. You must adapt the example for centralized applications with SIMOTION C.

Note

If the application example is not available for your hardware platform, you must adapt the hardware configuration.

Assigning module parameters

Proceed as follows:

1. Open your project in SIMOTION SCOUT.
2. Open the hardware configuration in SIMOTION SCOUT.
3. Configure your hardware station with a CP 340 module.
4. Double-click the **CP 340** to open the **Properties** dialog box for this module. Click the **Parameter** button to launch the parameter assignment tool of the CP 340 module.
5. The "ASCII" protocol must be selected in the protocol selection box.
   The standard settings of the ASCII protocol suffices for the application example.
6. Jumper the send and receive line (PIN 2/PIN 3) of the RS232 interface.

7. Apply your settings in the parameter assignment interface with the **File > Save** menu command, and close the interface with the **File > Exit** menu command. Close the Properties window for the CP 340 module by clicking the **OK** button.
8. Save the hardware configuration with the **Station > Save and compile** menu command.
9. Download the hardware configuration with the **Target system > Download to module** menu command.
   The red "SF" LED on the IM 153 turns on and then off if the assigned module parameters have been downloaded without errors.

Adapting the application example

The configuration in the example and its available hardware must be adapted.

The following options are available:

1. You can adapt the configuration in the example to the available hardware (e.g. PROFIBUS DP address).
2. You can adapt the configuration of the hardware to the example (e.g. PROFIBUS DP address).

Calling the application example

The application example can be found on the "SIMOTION Utilities & Applications" CD-ROM. The "SIMOTION Utilities & Applications" CD-ROM is provided free of charge and is included in the SIMOTION SCOUT scope of delivery.

1. Dearchive and open the project containing the application example.
2. Check the axis configuration: PROFIBUS DP addresses.
3. Check the module addresses (hardware configuration) against the I/O addresses of the controller in SIMOTION SCOUT and module address in the program (myModuleAddrCP340).
4. **Save** and **compile** the example project. Then, you can download the example to the SIMOTION device and switch to **RUN** mode.

Sequence of the application example

Table 3-10 Input icons used

Symbol	Data type	Designation
mySelectPointToPointCP340	BOOL	Select point-to-point connection
myExecSendCP340	BOOL	Start send job
mySendOrder1	BOOL	Select send: Job 1
mySendOrder2	BOOL	Select send: Job 2
mySendOrder3	BOOL	Select send: Job 3
myResetSend	BOOL	Cancel send job
myEnableToReceive	BOOL	Receive enable
myReceiveOrder1	BOOL	Select receive: Job 1

Symbol	Data type	Designation
myReceiveOrder2	BOOL	Select receive: Job 2
myReceiveOrder3	BOOL	Select receive: Job 3
myResetReceive	BOOL	Cancel receive job
myModuleAddrCP340	DINT	CP 340 module address, default 256
mySendDataArrayCP340	ARRAY[0..1023] of BYTE	Send data array
myReceiveDataArrayCP340	ARRAY[0..1023] of BYTE	Receive data array

Table 3-11 Output symbols used

Symbol	Data type	Designation
mySendDone	BOOL	Send: Completed
mySendError	BOOL	Send: Error display
mySendErrorNumber	WORD	Send: Error status
mySendTransErrorNumber	DINT	Send: Error status transfer
myNewDataReceived	BOOL	Receive: New data have been received
myReceiveError	BOOL	Receive: Error display
myReceiveErrorNumber	WORD	Receive: Error status
myReceiveTransErrorNumber	DINT	Receive: Error status transfer
myStateStartupCP340	BOOL	CP 340 startup status FALSE = startup completed
myDiagnosticAlarm	BOOL	TRUE = diagnostic alarm present on the CP 340
myProcessAlarm	BOOL	TRUE = process alarm present on the CP340
myAlarmInterrupt	UDINT	Type of the alarm (process, diagnostic alarm)
myLogBaseAddrIn	DINT	Module address
myLogBaseAddrOut	DINT	
myLogAddress	DINT	Diagnostic address
myAlarmDetails	DWORD	Alarm information

Note

You can either monitor and control the input and output variables used in the programming example in the INTERFACE area of the unit (under VAR_GLOBAL) using the symbol browser, or you can assign real inputs and outputs to the input and output variables in your user program.

For the "point-to-point connection" application example, set the "mySelectPointToPointCP340" input to "TRUE". This requires that the "E_CP340p" element be selected in the "PROGRAMS" folder in the project navigator. This will call function blocks contained in the application example.

Receiving data:

To receive data, you must set the "myEnableToReceive" variable to "TRUE" (static signal). If receive jobs 1 and 3 are enabled ("myReceiveOrder1" = TRUE and "myReceiveOrder3" = TRUE), the data is stored in the "myReceiveDataArrayCP340" data array starting with the "myReceiveDataArrayCP340[0]" array element (data offset is 0). If job 2 is enabled ("myReceiveOrder2" = TRUE), the data is stored in the "receiveDataArray" data

array starting with the "myReceiveDataArrayCP340[20]" array element (data offset is 20).
If "myNewDataReceived" = TRUE, this indicates that new data has been received. This signal is present for one cycle only.
If an error occurred during the transfer ("myReceiveError" = TRUE), the error code is stored in the "myReceiveErrorNumber" variable. If error code 16#1E0F is present in "myReceiveErrorNumber", an error occurred during the data transfer. The transfer error code is stored in the "myReceiveTransErrorNumber" variable. The error signals are deleted when you set input "myResetReceive" = TRUE.

Sending data:

You can use the "mySendOrder1", "mySendOrder2" and "mySendOrder3" inputs to select between three send jobs:

- Job 1 sends 10 bytes of data from the "mySendDataArrayCP340" data array from array element "mySendDataArrayCP340[0]" to "mySendDataArrayCP340[9]"
- Job 2 sends 20 bytes of data from the "mySendDataArrayCP340" data array from array element "mySendDataArrayCP340[20]" to "mySendDataArrayCP340[39]"
- Job 3 sends 1024 bytes of data from the "mySendDataArrayCP340" data array.

The data is sent to the communications partner if the "myExecSendCP340" input detects a signal change from "FALSE" to "TRUE" (positive edge).
If output signal "mySendDone" = TRUE, the send job has been completed. A new job can be sent if the "myExecSendCP340" input signal detects another signal change from FALSE to TRUE. If an error occurred during the transfer ("mySendError" = TRUE), the error code is stored in the "mySendErrorNumber" variable. If error code 16#1E0F is present in "mySendErrorNumber" , an error occurred during the data transfer. The transfer error code is stored in the "mySendTransErrorNumber" variable. The error signals are deleted when you set input "myExecSendCP340" = FALSE.

When the signal state at the "myResetSend" or "myResetReceive" input is set to "TRUE", the send job or receive job is canceled, respectively. If the signal state remains "TRUE", sending and receiving of data is disabled.

Note

Proper data transfer can be observed as follows:

- The "TxD" and "RxD" LEDs on the CP module illuminate.
- Output parameter (_CP_send FBs) done = TRUE or NewDataReceived = TRUE

3.9.2 Printing with CP 340

Function

The example shows how to use the _CP340_printer function block to send Struct_CP340_printData type data from the print memory area to a serial printer. For more information, see Chapter _CP340_printer function block (Page 29).

In the example program, text examples with and without variables, date specifications, and line and page feeds are output to the printer by means of the CP 340.

This example requires proper installation of the parameter assignment tool, as described in the SIMATIC *CP 340 Point-to-Point Connection, Installation and Parameter Assignment* manual.

Hardware platform

The application example is available for various SIMOTION hardware platforms.

Note

If the application example is not available for your hardware platform, you must adapt the hardware configuration.

Assigning module parameters

Proceed as follows:

1. Open your project in SIMOTION SCOUT.
2. Open the hardware configuration in SIMOTION SCOUT.
3. Configure your hardware station with a CP 340 module.
4. Double-click the **CP 340** to open the **Properties** dialog box for this module. Click the **Parameter** button to launch the parameter assignment tool of the CP 340 module.
5. The "PRINTER" protocol must be selected in the protocol selection box.
6. Open the protocol properties by selecting the **Edit > Go to protocol** menu command and the **Edit > Open object** menu command.
7. Select the following setting in the **Protocol** dialog box (all other settings remain unchanged):
   - **Baud rate: 9600 bits/s**
   - **Data flow control: none**
   - **Data bits: 8**
   - **Stop bits: 1**
   - **Parity: even**

   Confirm your selection with **OK**.
8. To edit the message texts, double-click the **Messages** icon to open the relevant dialog box. In the next dialog box, click the **SDB** button.
9. Under "Edit Message", enter a number on the left and the text on the right, as follows:
   - 1 System level log
   - 2 Date:
   - 3 Level %I I reached at %Z (time)

   Confirm your selection with **OK**.
10. To edit the page layout, double-click the **Page Layout** icon to open the relevant dialog box.

11. Select the following settings in this dialog box:
    - Left-hand margin: 3
    - Lines per page: 20
    - Separators / end of line: CR LF
    - Header lines: Example: Printing with the CP 340 for SIMOTION
    - Page footers: page %P

    Confirm your selection with OK.
12. The default settings for the "IBM" character set remain selected under "Font" and "Control Characters".
13. Apply your settings in the parameter assignment interface with the File > Save menu command, and close the interface with the File > Exit menu command. Click OK to close the Properties window for the CP 340 module.
14. Save the hardware configuration with the Station > Save and compile menu command.
15. Download the hardware configuration with the Target system > Download to module menu command.
    The red "SF" LED on the IM 153 turns on and then off if the assigned module parameters have been downloaded without errors.

Adapting the application example

The configuration in the example and its available hardware must be adapted.

The following options are available:

1. You can adapt the configuration in the example to the available hardware (e.g. PROFIBUS DP address).
2. You can adapt the configuration of the hardware to the example (e.g. PROFIBUS DP address).

Calling the application example

The application example can be found on the "SIMOTION Utilities & Applications" CD-ROM. The "SIMOTION Utilities & Applications" CD-ROM is provided free of charge and part of the SIMOTION SCOUT scope of delivery.

1. Dearchive and open the project containing the application example
2. Check the axis configuration: PROFIBUS DP addresses
3. Check the module addresses (hardware configuration) against the I/O addresses of the controller in SIMOTION SCOUT and the module addresses in the program (myModuleAddr_1)
4. Save and compile the example project. Then, you can download the example to the SIMOTION device and switch to RUN mode.

Sequence of the application example

Further steps for executing the example are performed in the symbol browser using the "myPrintSelect" variable. This requires that the "E_CP340p" element be selected in the "PROGRAMS" folder in the project navigator.

The program resets the "myPrintSelect" variable autonomously; this can be seen in the "Status Value" column in the symbol browser.

This application example shows you how to use the _FB_CP340.print function block to send data from the send data array to a serial printer.

The following texts can be printed with the example program:

1. Print new page with header:
   "Example: Printing with the CP 340 for SIMOTION"
2. Print new line and five line feeds
3. Print message text without variables and two line feeds:
   "System level log"
4. Message text, two tabs and WORD variable for German date format.
   "Date:"
   "Date: 01.01.2001"
5. Message text with time of day and quantity (DWORD variable and DINT variable) and six line feeds
   "Level %I l was reached at %Z (time)"
   "Level 3000 l was reached at 08:45:04.018 (time)"
6. Print footer with conversion instruction for page number
   "Page: %P"
   "Page: 1"

To print out all the lines from the example, select the check box for "myPrintSelect" and set ALL_PRINT in the "Control Value" column in the symbol browser.

Click "Immediate control" to assign the value ALL_PRINT to the variable and to print the example.

Other settings can be assigned to the "myPrintSelect" variable:

"DATE_PRINT"	Output "Date: 01.01.2001"
"REPORT_TEXT_PRINT"	Output "Level 3000 reached at 08:45:04.018"
"NO_PRINT"	No output made

The date, time, and level quantity can be changed using the following variables:

- myDateGermany
- myTimeValue,
- myQuantity

Utilized control variables

Table 3-12 Utilized control variables

Symbol	Data type	Initial value	Designation
myPrintSelect	EnumPrintSelect	NO_PRINT	Selection of printer output
myDateGermany	WORD	4018	Date: 01.01.2001
myTimeValue	DWORD	31504018	Time: 08:45:04.018
myQuantity	DINT	3000	Quantity: 3000
myModuleAddr_1	DINT	256	Module address

Pending errors are indicated by the following variables:

Table 3-13 Variables used for displaying errors

Symbol	Data type	Initial value	Designation
myDiagnosticAlarm	BOOL		Diagnostic interrupt
myProcessAlarm	BOOL		Process interrupt
myErrorNumberprinter	WORD		Error code
myTransErrorNumberprinter	DINT		Transfer error number
myStopPrinter	BOOL	FALSE	Cancel print

Note

The active transfer job can be canceled by setting the "myStopPrinter" variable to "TRUE". The **_CP340_printer** FB is reset to its initial state.

The sending of print jobs will remain disabled as long as the signal state of the "myStopPrinter" variable is "TRUE".

CP 341 function blocks 4

4.1 Overview of the function blocks of the CP 341

This chapter contains a description of all of the function blocks (FBs) and the data structure required for communication between a SIMOTION device and a CP 341.

The function blocks form the software interface between the SIMOTION device and the CPs. They must be called repeatedly (in cycles) from the user program.

The following function blocks are available:

- _CP341_send function block (Page 60)
- _CP341_receive function block (Page 72)
- _CP341_printer function block (Page 84)
- _CP341_getV24Signals function block (Page 100)
- _CP341_setV24Signals function block (Page 102)

Note

The SIMOTION identifiers have changed as of V4.0. A comparison of the SIMOTION and SIMATIC identifiers can be found in the appendix SIMOTION and SIMATIC names (Page 117) in the table "SIMOTION and SIMATIC identifiers CP 341".

SIMOTION SCOUT contains all of the required function blocks and the **Struct_CP341_Cl512Data** data structure (for RK 512 computer link only) of the CP 341. The function blocks can be used to control one or more CP 341 modules.

4.2 _CP341_send function block

4.2.1 Description of the _CP341_send FB

Function

The _CP341_send function block enables you to send data from the send data array to a communications partner (3964(R) protocol, RK 512 protocol, ASCII driver) or to fetch data from a communications partner and store it there (RK 512). You have 4,096 bytes available for this.

Task integration (call)

The _CP341_send function block must be called cyclically in the BackgroundTask or in the TimerInterruptTask. Calling in the SystemInterruptTask is not permitted. Calling the function block in the IPOSynchronousTask is not recommended for runtime reasons.

4.2.2 Application with 3964(R) protocol or ASCII driver

The SIMOTION device sends data to a communications partner

The _CP341_send function block transfers a data block that is specified by the following parameters to the CP 341:

- data corresponds to the data array containing the send data
- dataOffset corresponds to the array index containing the first send byte
- dataLength corresponds to the amount of data to be sent in bytes

The _CP341_send function block is called repeatedly by a program. The send job can only be executed by cyclically calling the send FB.

A positive edge at the execute input initiates the transfer. A data transfer operation can run over several calls, depending on the amount of data involved.

The active transfer job can be canceled by setting the reset parameter to "TRUE". This will reset the _CP341_send FB to its initial state. The send operation will remain disabled as long as the signal state at the reset parameter is "TRUE".

The moduleAddress parameter specifies the module address of the CP 341 being addressed.

Status and error display on the _CP341_send FB

The done output indicates that the job has been completed without errors. The error output indicates that an error has occurred. If an error occurs, the corresponding event class/number is displayed in the errorID output (see "Parameters of the _CP341_send FB" table (3964 (R) procedure or ASCII driver application)"). If no errors have occurred, errorID has a value of 0. Done and error/errorID are also displayed on reset of the _CP341_send function block. When

16#1E0F is displayed in the **errorID** parameter, a detailed error description is also output via the **errorIdTransfer** parameter.

Parameters **done**, **error,errorID**, and **errorIdTransfer** are present for one block call only.

Note

There is no parameter check for the **_CP341_send** function block. Incorrect parameterization of this block may cause the SIMOTION device to switch to "STOP" mode.

Before the CP 341 can process an initiated job following a transition of the SIMOTION device from "STOP" to "RUN" mode, the CP-SIMOTION startup mechanism of the **_CP341_send** FB must be complete. Any jobs initiated in the meantime will not be lost. They are transferred to the CP 341 once the startup coordination has finished.

The end of the startup coordination is indicated in output parameter **startup** = FALSE.

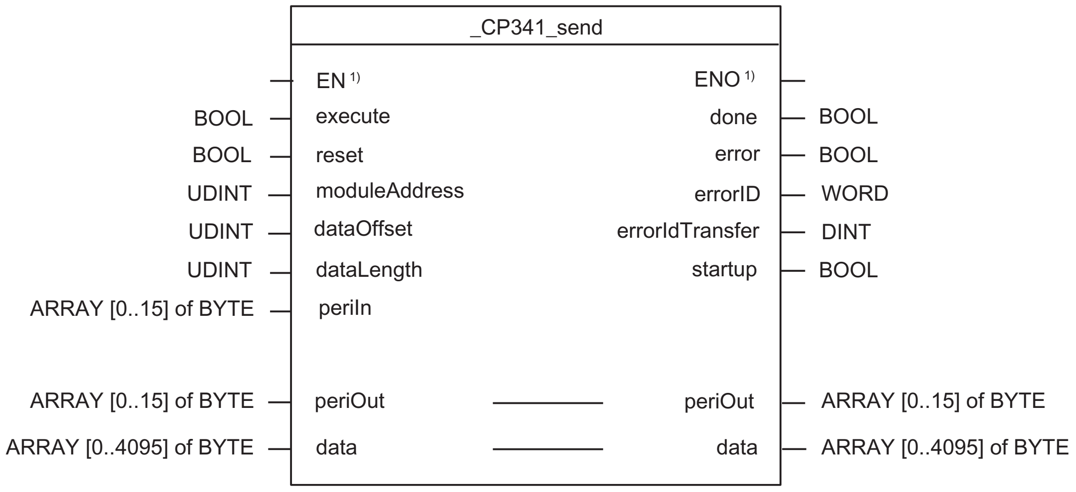
Assignment in the data area

The data to be sent are transferred to the **_CP341_send** FB in the **data** parameter (VAR_IN_OUT) as ARRAY of BYTE. At the beginning of the send operation, the data is copied to the FB and transferred from there to the CP 341.

Note

Once the data have been entered in the static memory of the send FB, you can modify the variable created in the **data** parameter. This does not affect the data to be sent.

Call (LAD representation)



1) LAD-specific parameters

Parameter description

Table 4-1 Parameters of the _CP341_send FB (application with 3964(R) protocol or ASCII driver)

Name	P type [1]	Data type	Meaning	Actions performed by user	Actions performed by block
execute	IN	BOOL	Initiates job on positive edge	Entered	Checked
reset	IN	BOOL	Cancels job	Entered	Checked
moduleAddress	IN	DINT	Module address of the CP for data set transfer (from HW Config)	Entered	Checked
dataOffset	IN	UDINT	First element to be sent	Entered	Checked
dataLength [2]	IN	UDINT	Number of elements to be sent	Entered	Checked
periIn	IN	ARRAY [0..15] of BYTE	I/O inputs of the CP transferred to the FB	I/O variable of the I/O inputs of the CP transferred to the FB	Checked
periOut	IN/OUT	ARRAY [0..15] of BYTE	Prepared FB data for the I/O outputs of the CP [3]	Checked and transferred to the I/O variable for the I/O outputs	Entered
data	IN/OUT	ARRAY[0..4095] of BYTE	Send data array	Entered and checked	Entered
done	OUT	BOOL	Job completed without errors	Checked	Entered
error	OUT	BOOL	Job completed with errors	Checked	Entered
errorID	OUT	WORD	Error specification For **error** = TRUE, the error information (event class and number) is displayed in the **errorID** parameter. [4]	Checked	Entered
errorIdTransfer	OUT	DINT	Error during data transfer to the CP (precise error diagnostics if 16#1E0F is present at the **errorID** parameter [5])	Checked	Entered
startup	OUT	BOOL	Indicates CP startup	Checked	Entered

[1] Parameter types: IN = input parameters, OUT = output parameters, IN/OUT = in/out parameters

[2] If an application uses earlier CP 341 versions with a maximum telegram length of 1 kB, only the maximum value of 1,024 can be set at the **dataLength** input parameter.

[3] **Note:** The **periOut** parameter must be supplied with a variable of type **ARRAY [0..15] of BYTE**. Create a local or global variable in your program under **VAR** (do not create a temporary variable under VAR_TEMP). After the FB has been called, this variable must be assigned to the I/O variable for the I/O outputs of the module. See call example for CP 341.

[4] For error information, refer to the SIMATIC *CP 341 Point-to-Point Connection, Installation and Parameter Assignment* manual, Chapter "Diagnostics with the CP 341".

[5] For a more detailed description (_readRecord and _writeRecord), see *SIMOTION System Function/Variable Devices* Parameter Manual. This documentation is included in the SIMOTION SCOUT scope of delivery as electronic documentation.

Signal sequence diagram of the _CP341_send FB

The following figure illustrates the behavior of the **done** and **error** parameters according to the input circuit of **execute** and **reset**.

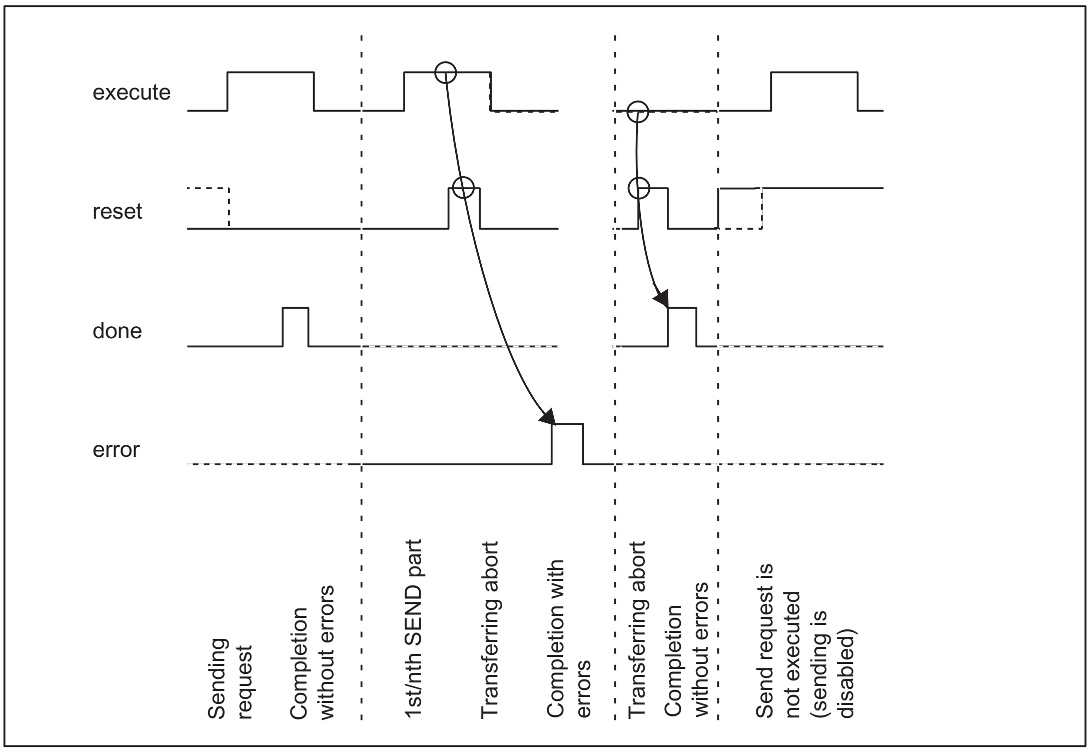


Figure 4-1 Signal sequence diagram of the _CP341_send FB

Note

The **execute** input is edge-triggered. The send job starts when there is a positive edge at the **execute** input.

4.2.3 Application with RK 512 computer interfacing

Sending data with the _CP341_send FB: SIMOTION with SIMOTION

Note

The handling of the **_CP341_send** FB differs according to whether data is to be exchanged with a **SIMOTION System** or a **SIMATIC System**.

The parameter assignment is described below for a function block for **SIMOTION with SIMOTION** communication.

You can send data to a communications partner by specifying 'SEND_CP' in the **mode** parameter.

A positive edge at the **execute** input initiates the data transfer. A data transfer operation can run over several calls (program cycles), depending on the amount of data involved.

The **moduleAddress** parameter specifies the module address of the CP that is to be used for the data transfer.

The data to be sent is specified in the following parameters:

- **data** indicates the data array containing the send data.
- **dataOffset** corresponds to the array index containing the first send byte.
- **dataLength** corresponds to the amount of data to be sent in bytes.

You must also indicate where the data is to be entered on the receiver.

- The **remoteCpuId** parameter specifies the number of the receiver (relevant for multiprocessor communication only).
- **remoteDataType** specifies the memory area where the data will be entered on the receiver (if the communications partner is a SIMOTION device, 68 must always be entered).

- **remoteMemIndex** specifies the memory index where the receive data will be entered. **RemoteMemIndex** = 0 is not permissible. An error message will be generated.
- **remoteDataOffset** specifies the array index in which the first receive byte is to be entered.

Note

The above parameters relate to a variable addressed specifically by the receiver. The receiver specifies the variable in which the data will be entered.

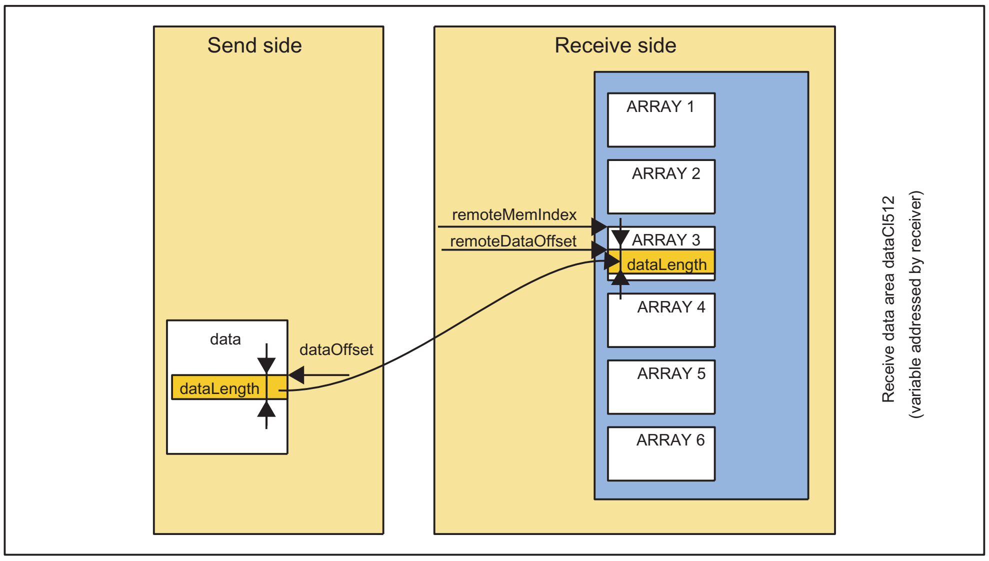


Figure 4-2 Send data _FB_P_Send: SIMOTION with SIMOTION

The communication flag bit (see Chapter Special features related to data transfer (Page 106)), which is checked in the receiver before the send job is initiated and set once the send job is complete, is specified with parameters **remoteComFlagByte** (array index containing the communication flag bit) and **remoteComFlagBit** (bit position of the communication flag bit).

Special features for sending data

Assigning 255 to the **remoteComFlagByte** parameter will disable the communication flag functionality.

Sending data with the _CP341_send FB: SIMOTION with SIMATIC

Note

The handling of the **_CP341_send** FB differs according to whether data is to be exchanged with a **SIMOTION System** or a **SIMATIC System**.

The parameter assignment is described below for a function block for **SIMOTION with SIMATIC** communication.

You can send data to a communications partner by specifying 'SEND_CP' in the **mode** parameter.

A positive edge at the **execute** input initiates the data transfer. A data transfer operation can run over several calls (program cycles), depending on the amount of data involved.

The **moduleAddress** parameter specifies the module address of the CP that is to be used for the data transfer.

The data to be sent is specified in the following parameters:

- **data** indicates the data array containing the send data.
- **dataOffset** corresponds to the array index containing the first send byte.
- **dataLength** corresponds to the amount of data to be sent in bytes.

You must also indicate where the data is to be entered on the receiver.

- The **remoteCpuId** parameter specifies the number of the receiver (relevant for multiprocessor communication only).
- **remoteDataType** indicates the memory area where data will be entered on the receiver.
  – Data block = 68
  – Extended data block = 88
  – Flag = 77
  – Input = 69
  – Output = 65
  – Counter = 90
  – Timer = 84
- **remoteMemIndex** specifies the number of the data block or extended data block where the receive data will be entered. It has no relevance for all other receive areas. **RemoteMemIndex** = 0 is not permissible. An error message will be generated.
- **remoteDataOffset** specifies the array index in which the first receive byte is to be entered.

The communication flag bit (see Chapter Special features related to data transfer (Page 106)), which is checked in the receiver before the send job is initiated and set once the send job is complete, is specified with parameters **remoteComFlagByte** (array index containing the communication flag bit) and **remoteComFlagBit** (bit position of the communication flag bit) in the SIMATIC flag area.

Special features for sending data

Assigning 255 to the **remoteComFlagByte** parameter will disable the communication flag functionality.

The signal sequence diagram of the **_CP341_send** FB can be found in Chapter Description of the _CP341_send FB (Page 60).

Fetching data with the _CP341_send FB: SIMOTION with SIMOTION

Note

The handling of the **_CP341_send** FB differs according to whether data is to be exchanged with a **SIMOTION System** or a **SIMATIC System**.

The parameter assignment is described below for a function block for **SIMOTION with SIMOTION** communication.

You can fetch data from a communications partner by specifying "FETCH_CP" in the **mode** parameter.

A positive edge at the **execute** input initiates the data transfer. A data transfer operation can run over several calls (program cycles), depending on the amount of data involved.

The **moduleAddress** parameter specifies the module address of the CP that is to be used for the data transfer.

You must also indicate where the data are to be fetched on the receiver.

- The **remoteCpuId** parameter specifies the number of the receiver (relevant for multiprocessor communication only).
- **remoteDataType** specifies the memory area where the data will be fetched on the receiver (if the communications partner is a SIMOTION device, 68 must always be entered).
- **remoteMemIndex** specifies the memory index where the data will be fetched. **RemoteMemIndex** = 0 is not permissible. An error message will be generated.
- **remoteDataOffset** specifies the array index where the first byte is to be fetched.

Note

The above parameters relate to a variable addressed specifically by the communications partner. They specify the variable from which the data will be fetched.

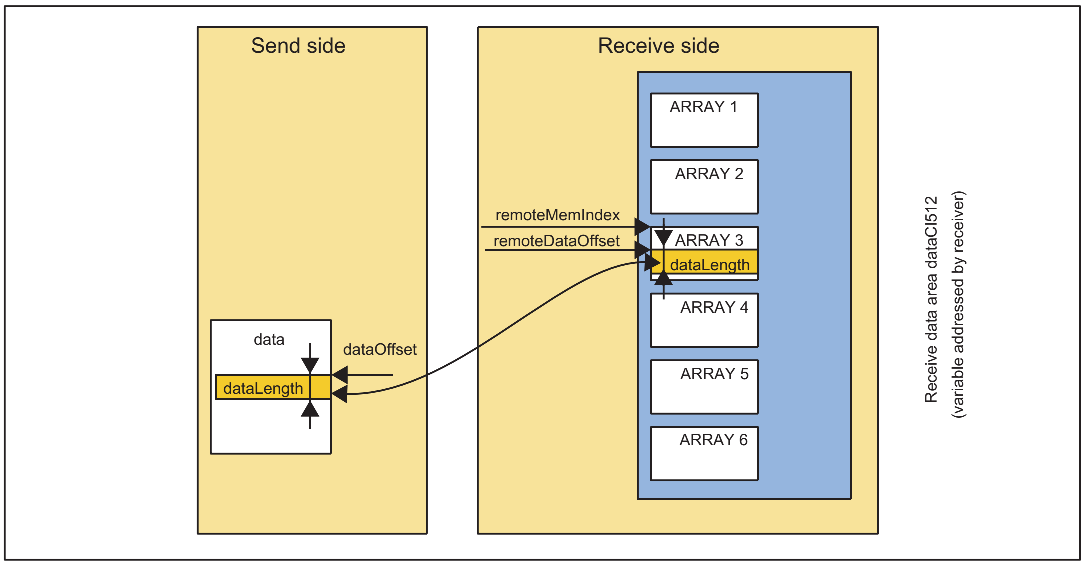


Figure 4-3 Fetch data _P_Send FB: SIMOTION with SIMOTION

The following parameters are used to specify where the fetched data will be saved:

- **data** corresponds to the data array in which the requested data will be entered.
- **dataOffset** corresponds to the array index in which the first fetched data byte will be entered.
- **dataLength** corresponds to the amount of data to be fetched in bytes.

The communication flag bit (see Chapter Special features related to data transfer (Page 106)), which is checked in the receiver before the send job is initiated and set once the fetch job is complete, is specified with parameters **remoteComFlagByte** (array index containing the communication flag bit) and **remoteComFlagBit** (bit position of the communication flag bit).

Special features for fetching data

Assigning 255 to the **remoteComFlagByte** parameter will disable the communication flag functionality.

Fetching data with the _CP341_send FB: SIMOTION with SIMATIC

Note

The handling of the **_CP341_send** FB differs according to whether data is to be exchanged with a **SIMOTION System** or a **SIMATIC System**.

The parameter assignment is described below for a function block for **SIMOTION with SIMATIC** communication.

You can fetch data from a communications partner by specifying "FETCH_CP" in the **mode** parameter.

A positive edge at the **execute** input initiates the data transfer. A data transfer operation can run over several calls (program cycles), depending on the amount of data involved.

The **moduleAddress** parameter specifies the module address of the CP that is to be used for the data transfer.

You must also indicate where the data is to be fetched on the communications partner.

- **remoteCpuId** specifies the number of the communications partner (relevant for multiprocessor communication only).
- **remoteDataType** specifies the memory area where the data will be fetched on the communications partner.
  - Data block = 68
  - Extended data block = 88
  - Flag = 77
  - Input = 69
  - Output = 65
  - Counter = 90
  - Timer = 84

- **remoteMemIndex** specifies the number of the data block or extended data block where the receive data will be fetched.
  RemoteMemIndex = 0 is not permissible. An error message will be generated.
- **remoteDataOffset** specifies the array index where the first byte is to be fetched.

The following parameters are used to specify where the fetched data will be saved:

- **data** corresponds to the data array in which the requested data will be entered.
- **dataOffset** corresponds to the array index in which the first fetched data byte will be entered.
- **dataLength** corresponds to the amount of data to be fetched in bytes.

The communication flag bit (see Chapter Special features related to data transfer (Page 106)), which is checked in the communications partner before the send job is initiated and set once the send job is complete, is specified with parameters **remoteComFlagByte** (array index containing the communication flag bit) and **remoteComFlagBit** (bit position of the communication flag bit) in the SIMATIC communication flag area.

Special features for fetching data

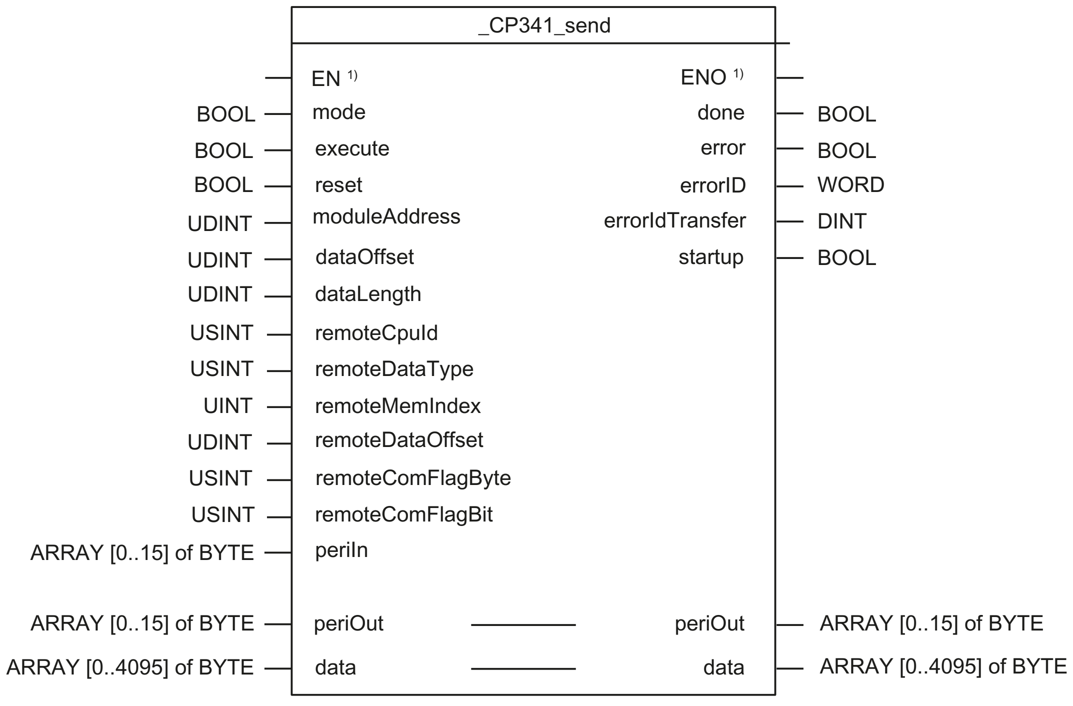
Assigning 255 to the **remoteComFlagByte** parameter will disable the communication flag functionality.

The signal sequence diagram of the **_CP341_send** FB can be found in Chapter Description of the _CP341_send FB (Page 60).

Assignment in the data area

During the receive operation, the data to be received are stored temporarily in a local ARRAY. Once the data transfer from the CP 341 to the SIMOTION device is complete, the data is made available in the **data** (VAR_IN_OUT) parameter of the **_CP341_send** FB.

Call (LAD representation)



1) LAD-specific parameters

Parameter description

Table 4-2 Parameters of the _CP341_send FB (application with RK 512 computer link)

Name	P type 1)	Data type	Comment	Actions performed by user	Actions performed by block
mode	IN	EnumSendFetch	SEND_CP: Sends data FETCH_CP: Fetches data from the communications partner	Entered	Checked
execute	IN	BOOL	Initiates job on positive edge	Entered	Checked
reset	IN	BOOL	Cancels job	Entered	Checked
moduleAddress	IN	DINT	Module address of the CP for data set transfer (found in **HW Config**)	Entered	Checked
dataOffset	IN	UDINT	First element to be sent	Entered	Checked
dataLength	IN	UDINT	Number of elements to be sent	Entered	Checked
remoteCpuId	IN	USINT	Number of the remote communications partner	Entered	Checked
remoteDataType	IN	USINT	Area type in the remote communications partner	Entered	Checked

Name	P type [1]	Data type	Comment	Actions performed by user	Actions performed by block
remoteMemIndex	IN	UINT	Memory index in the remote communications partner (1 ≤ remoteMemIndex ≤ 6) [2]	Entered	Checked
remoteDataOffset	IN	UDINT	First element in the remote communications partner	Entered	Checked
remoteComFlagByte	IN	USINT	Communication flag index in the local communications partner	Entered	Checked
remoteComFlagBit	IN	USINT	Communication flag bit no. in the local communications partner	Entered	Checked
periIn	IN	ARRAY[0..15] of BYTE	I/O inputs of the CP transferred to the FB	I/O variable of the I/O inputs of the CP transferred to the FB	Checked
periOut	IN/OUT	ARRAY[0..15] of BYTE	Prepared FB data for the I/O outputs of the CP [3]	Checked and transferred to the I/O variable for the I/O outputs	Entered
data	IN/OUT	ARRAY[0..4095] of BYTE	Send data array Receive data for parameter **mode** = FETCH_CP	Entered and checked	Entered
done	OUT	BOOL	Job completed without errors	Checked	Entered
error	OUT	BOOL	Job completed with errors	Checked	Entered
errorID	OUT	WORD	Error specification For **error**=TRUE, the error information (event class and number) is displayed in the **errorID** parameter.[4]	Checked	Entered
errorIdTransfer	OUT	DINT	Error during data transfer to the CP (precise error diagnostics if 16#1E0F is present at the **errorID** parameter [5])	Checked	Entered
startup	OUT	BOOL	Indicates CP startup	Checked	Entered

1) Parameter types: IN = input parameters, OUT = output parameters, IN/OUT = in/out parameters

2) **remoteMemIndex** = 0 is not permissible. An error message will be generated.

3) **Note:** The **periOut** parameter must be supplied with a variable of type **ARRAY[0..15] of BYTE**. Create a local or global variable in your program under **VAR** (do not create a temporary variable under VAR_TEMP). After the FB has been called, this variable must be assigned to the I/O variable for the I/O outputs of the module. See call example for CP 341.

4) For error information, refer to the SIMATIC *CP 341 Point-to-Point Connection, Installation and Parameter Assignment* manual, Chapter "Diagnostics with the CP 341"

5) For a more detailed description (_readRecord and _writeRecord), see *SIMOTION System Function/Variable Devices* Parameter Manual. This documentation is included in the SIMOTION SCOUT scope of delivery as electronic documentation.

4.3 _CP341_receive function block

4.3.1 Description of the _CP341_receive FB

Function

The **_CP341_receive** function block enables you to receive data from a communications partner in the receive data array (3964(R) protocol, ASCII driver, RK 512) or to make data available for the communications partner (RK 512). Each data array depends on the type of protocol used. If you are using the 3964(R) protocol or ASCII driver, the receive data array is **dataCl3964**. For an RK 512 computer link, the data array is **dataCl512**. In both cases, 4,096 bytes are available. From the **dataCl512** data array, data is also made available for the communications partner.

Task integration (call)

The **_CP341_receive** function block must be called cyclically in the **BackgroundTask** or in the **TimerInterruptTask**. Calling in the **SystemInterruptTask** is not permitted. Calling the function block in the **IPOSynchronousTask** is not recommended for runtime reasons.

4.3.2 Application with 3964(R) protocol or ASCII driver

SIMOTION device receives data from a communications partner

The **_CP341_receive** function block transfers a data block that is specified by the **dataCl3964** and **dataOffset** parameters from the CP 341 to a SIMOTION device. The **_CP341_receive** function block is called repeatedly by a program. The receive job can only be executed by cyclically calling the receive FB.

Receiving of data is enabled with static signal state "TRUE" in the **enable** parameter. An active data transfer can be canceled with signal state "FALSE" in the **enable** parameter. The canceled receive job is terminated with an error message at the **errorID** output. The receive operation will remain disabled as long as the signal state at the **enable** parameter is "FALSE". A data transfer operation can run over several calls, depending on the amount of data involved.

The active transfer job can be canceled by setting the **reset** parameter to "TRUE". This will reset the **_CP341_receive** FB to its initial state. The receive operation will remain disabled as long as the signal state at the **reset** parameter is "TRUE".

The **moduleAddress** parameter specifies the module address of the CP 341 being addressed.

Status and error display on the _CP341_receive FB

The **newDataReceived** output indicates that the job has been completed without errors. The amount of data received is indicated in the **dataLength** parameter. The **error** output indicates that an error has occurred. If an error occurs, the corresponding event class/number is displayed in the **errorID** output (see "Parameters of the _CP341_receive FB" table (3964 (R) procedure or ASCII driver)"). If no error has occurred, **errorID** has the value "0". **newDataReceived** and **error**/**errorID** will also be output when the **_CP341_receive** FB is **reset**. When 16#1E0F is displayed in the **errorID** parameter, a detailed error description is also output via the **errorIdTransfer** parameter.

The **newDataReceived**, **dataLength**, **error**, **errorID**, and **errorIdTransfer** parameters are available for one block call only.

Note

There is no parameter check for the **_CP341_receive** FB. Incorrect parameterization of this block may cause the SIMOTION device to switch to "STOP" mode.

Before a job from the CP 341 can be received following a transition of the SIMOTION device from "STOP" to "RUN" mode, the CP-SIMOTION startup mechanism of the **_CP341_receive** FB must be complete.

The end of the startup coordination is indicated in output parameter **startup** = FALSE.

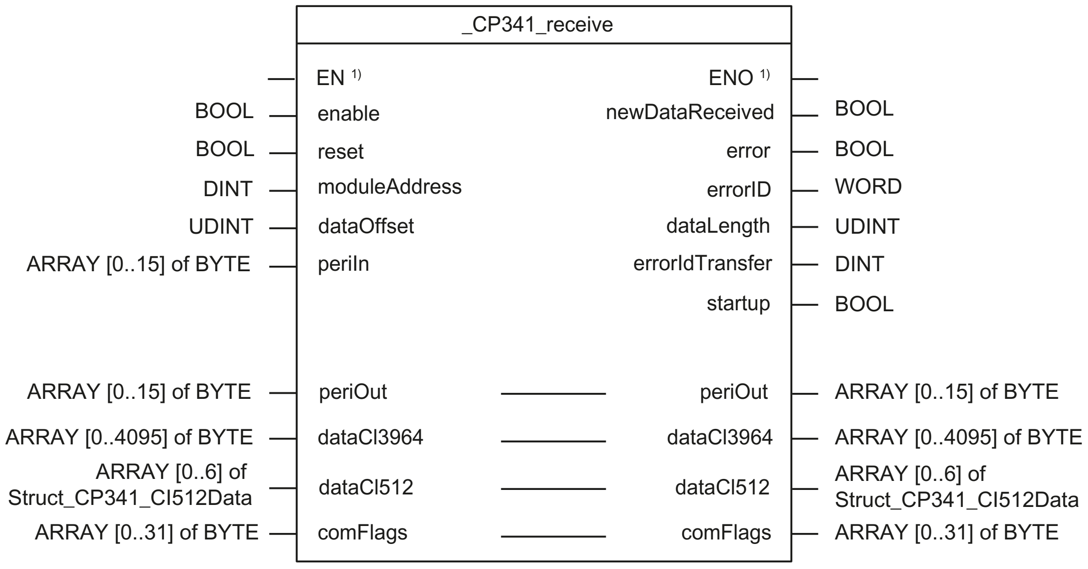
Assignment in the data area

During the receive operation, the data to be received are stored temporarily in a local ARRAY. Once the data transfer from the CP 341 to the SIMOTION device is complete, the data are made available in the **dataCl3964** (VAR_IN_OUT) parameter of the **_CP341_receive** FB.

Note

To ensure data consistency, do not access the receive data array until all the data has been received (**newDataReceived** = TRUE).

Call (LAD representation)



1) LAD-specific parameters

Parameters of the _CP341_receive FB (3964(R) protocol or ASCII driver)

Table 4-3 Parameters of the _CP341_receive FB (3964(R) protocol or ASCII driver)

Name	P type 1)	Data type	Meaning	Actions performed by user	Actions performed by block
enable	IN	BOOL	Receive enable	Entered	Checked
reset	IN	BOOL	Cancels job	Entered	Checked
moduleAddress	IN	DINT	Module address of the CP for data set transfer (from HW Config)	Entered	Checked
dataOffset	IN	UDINT	First element to be received	Entered	Checked
periIn	IN	ARRAY[0..15] of BYTE	I/O inputs of the CP transferred to the FB	I/O variable of the I/O inputs of the CP transferred to the FB	Checked
periOut	IN/OUT	ARRAY[0..15] of BYTE	Prepared FB data for the I/O outputs of the CP 2)	Checked and transferred to the I/O variable for the I/O outputs	Entered
dataCl3964	IN/OUT	ARRAY[0..4095] of BYTE	Receive data array	Entered and checked	Entered
dataCl512	IN/OUT	ARRAY[0..6] of 'Struct_CP341_ Cl512Data'	Data area for RK 512 two-dimensional array 3)	Entered and checked	Checked
comFlags	IN/OUT	ARRAY[0..31] of BYTE	Communication flag area for RK 512	Entered and checked	Entered
newDataReceived	OUT	BOOL	New data have been received	Checked	Entered

Name	P type [1)]	Data type	Meaning	Actions performed by user	Actions performed by block
error	OUT	BOOL	Job completed with errors	Checked	Entered
dataLength	OUT	UDINT	Quantity of data received	Checked	Entered
errorID	OUT	WORD	Error specification For **error**=TRUE, the error information (event class and number) is displayed in the **status** parameter.[4)]	Checked	Entered
errorIdTransfer	OUT	DINT	Error during data transfer to the CP (precise error diagnostics if 16#1E0F is present at the **errorID** parameter [5)])	Checked	Entered
startup	OUT	BOOL	Indicates CP startup	Checked	Entered

1) Parameter types: IN = input parameters, OUT = output parameters, IN/OUT = in/out parameters

2) **Note:** The **periOut** parameter must be supplied with a variable of type **ARRAY[0..15] of BYTE**. Create a local or global variable in your program under **VAR** (do not create a temporary variable under VAR_TEMP). After the FB has been called, this variable must be assigned to the I/O variable for the I/O outputs of the module. See call example for CP 341.

3) No data can be exchanged in array index **dataCl512[0]**.

4) For error information, refer to the SIMATIC *CP 341 Point-to-Point Connection, Installation and Parameter Assignment* manual, Chapter "Diagnostics with the CP 341"

5) For a more detailed description (_readRecord and _writeRecord), see *SIMOTION System Function/Variable Devices* Parameter Manual. This documentation is included in the SIMOTION SCOUT scope of delivery as electronic documentation.

Signal sequence diagram of the _CP341_receive FB

The following figure illustrates the behavior of the **newDataReceived**, **dataLength**, and **error** parameters according to the input circuit of **enable** and **reset.**

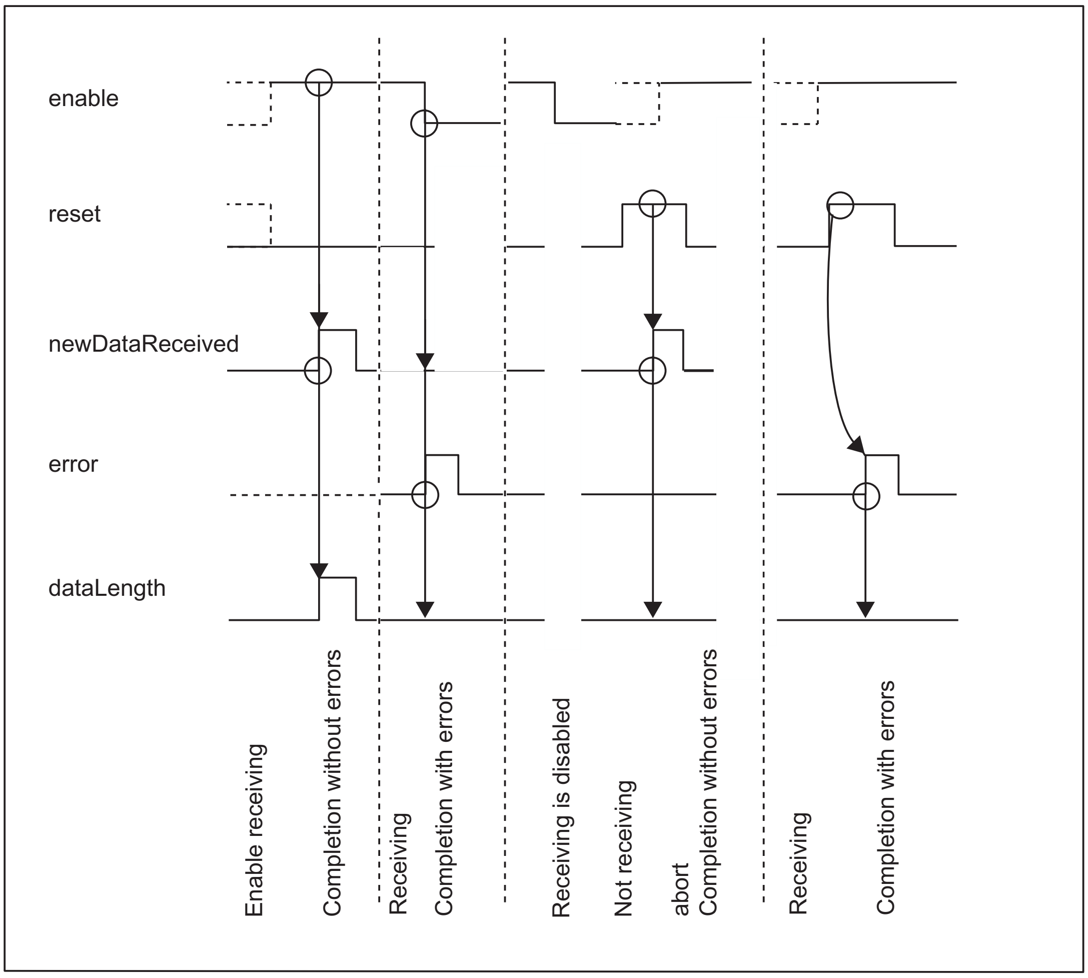


Figure 4-4 Signal sequence diagram of the _CP341_receive FB

4.3.3 Application with RK 512 computer interfacing

Receiving data with the _CP341_receive FB: SIMOTION with SIMOTION

Note

The handling of the **_CP341_receive** FB differs according to whether data is to be exchanged with a **SIMOTION System** or a **SIMATIC System.**

The parameter assignment is described below for a function block for **SIMOTION with SIMOTION** communication.

The data to be received will be entered in a variable created in the **dataCl512** parameter. The communications partner specifies where the receive data will be entered in the variable. The information about the entry location is output at the following output parameters.

- **localCpuId** specifies the number of the receiver (relevant for multiprocessor communication only).
- **localDataType** specifies the memory area where the receive data have been entered (irrelevant for SIMOTION with SIMOTION communication).
- **localMemIndex** specifies the memory index where the receive data have been entered.
- **localDataOffset** specifies the array index where the first receive byte has been entered.
- **dataLength** corresponds to the amount of data received in bytes.

A static "TRUE" signal state in the **enable** parameter enables a check to determine whether data are to be read from the CP 341. An active data transfer can be canceled with signal state "FALSE" in the **enable** parameter. The canceled receive job is terminated with an error message at the **errorID** output. The receive operation will remain disabled as long as the signal state at the **enable** parameter is "FALSE". A data transfer operation can run over several calls, depending on the amount of data involved.

The **moduleAddress** parameter specifies the module address of the CP that is to be used for the data transfer.

The communication flag area (see Chapter Special features related to data transfer (Page 106)) is specified by a variable created in the **comFlags** parameter. The communication flag bit, which was checked in the receiver before the send job was initiated and was set after the send job was finished, is indicated at output parameters **localComFlagByte** (array index containing the communication flag bit) and **localComFlagBit** (bit position of communication flag bit).

Special features for receiving data

Assigning 255 to the **localComFlagByte** parameter will disable the communication flag functionality.

Receiving data with the _CP341_receive FB: SIMOTION with SIMATIC

Note

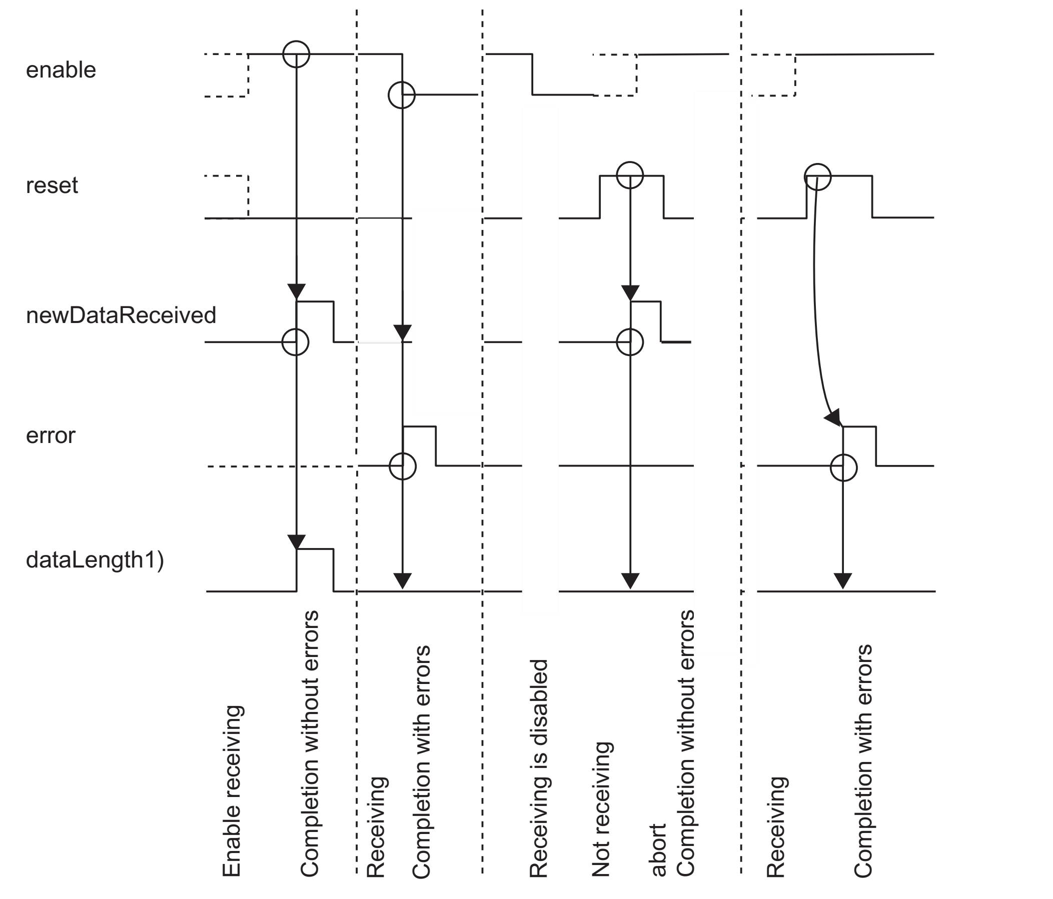
The handling of the **_CP341_receive** FB differs according to whether data is to be exchanged with a **SIMOTION System** or a **SIMATIC System.**

The parameter assignment is described below for a function block for **SIMOTION with SIMATIC** communication.

The communications partner on the SIMATIC side must set the input parameters according to the SIMOTION description.

Signal sequence diagram of the _CP341_receive FB

The following figure illustrates the behavior of the **newDataReceived, dataLength**, and **error** parameters according to the input circuit of **enable** and **reset.**



1) For RK 512 the parameters of the current request are also indicated at the local outputs ... until the next time the FB is called.

Figure 4-5 Signal sequence diagram of the _CP341_receive - receive data FB

Providing data with the _CP341_receive FB: SIMOTION with SIMOTION

Note

The handling of the **_CP341_receive** FB differs according to whether data is to be exchanged with a **SIMOTION System** or a **SIMATIC System.**

The parameter assignment is described below for a function block for **SIMOTION with SIMOTION** communication.

Function block processing is enabled with static signal state "TRUE" in the **enable** parameter. An active data transfer can be canceled with signal state "FALSE" in the **enable** parameter. The canceled fetch job is terminated with an error message at the **errorID** output. Data cannot be provided as long as the signal state at the **enable** parameter is "FALSE". A data transfer operation can run over several calls, depending on the amount of data involved.

The **moduleAddress** parameter specifies the module address of the CP that is to be used for the data transfer.

The data to be provided are entered in a variable created in the **dataCl512** parameter. The communications partner specifies which data is to be fetched from the variable. The information about the fetched data is output in the following output parameters.

- **localCpuId** specifies the number of the communications partner providing the data (relevant for multiprocessor communication only).
- **localDataType** specifies the memory area where the data was fetched (irrelevant for SIMOTION).
- **localMemIndex** specifies the memory index where the data were fetched.
- **localDataOffset** specifies the array index where the first data byte was fetched.
- **dataLength** corresponds to the amount of data sent in bytes.

Special features for providing data

Assigning 255 to the **localComFlagByte** parameter will disable the communication flag functionality.

Providing data with the _CP341_receive FB: SIMOTION with SIMATIC

Note

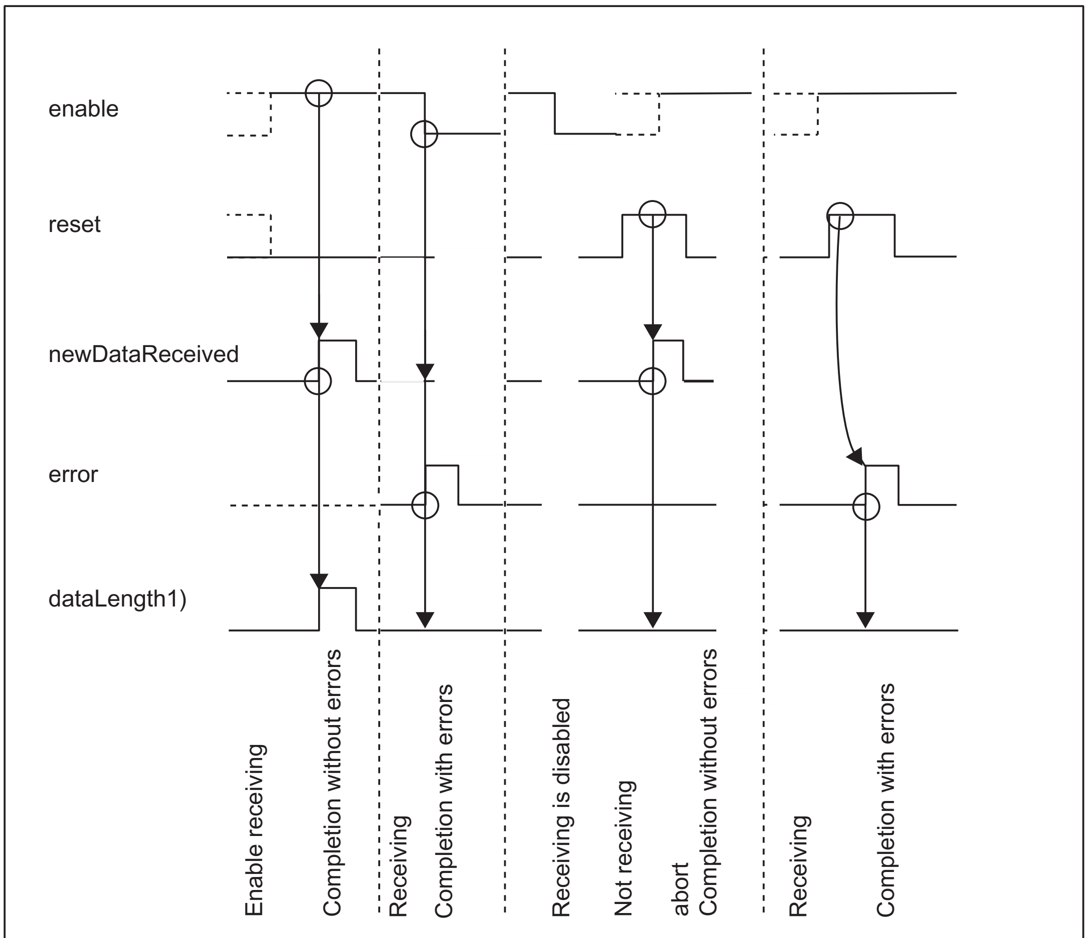
The handling of the **_CP341_receive** FB differs according to whether data is to be exchanged with a **SIMOTION System** or a **SIMATIC System**.

The parameter assignment is described below for a function block for **SIMOTION with SIMATIC** communication.

The communications partner on the SIMATIC side must specify the input parameters according to the SIMOTION description.

Signal sequence diagram of the _CP341_receive FB

The following figure illustrates the behavior of the **newDataReceived, dataLength**, and **error** parameters according to the input circuit of **enable** and **reset.**



1) For RK 512 the parameters of the current request are also indicated at the local outputs ... until the next time the FB is called.

Figure 4-6 Signal sequence diagram of the _CP341_receive - providing data FB

Error display on the _CP341_receive FB

The **newDataReceived** output parameter indicates that the job has been completed without errors. The amount of data received is indicated in the **dataLength** parameter. The **error** output indicates that an error has occurred. If an error occurs, the corresponding event class/number is displayed in the **errorID** output parameter (see "Parameters of the _CP341_receive FB" table (application with RK 512 computer link)"). If no errors have occurred, **errorID** has a value of 0. The **newDataReceived** and **error/errorID** parameters are also displayed on **reset** of the **_CP341_receive** FB. When 16#1E0F is displayed in the **errorID** parameter, a detailed error description is also output via the **errorIdTransfer** parameter.

The **newDataReceived**, **dataLength, error, errorID**, and **errorIdTransfer** parameters are available for one block passage only.

Note

There is no parameter check for the **_CP341_receive** FB. Incorrect parameterization of this block may cause the SIMOTION device to switch to "STOP" mode.

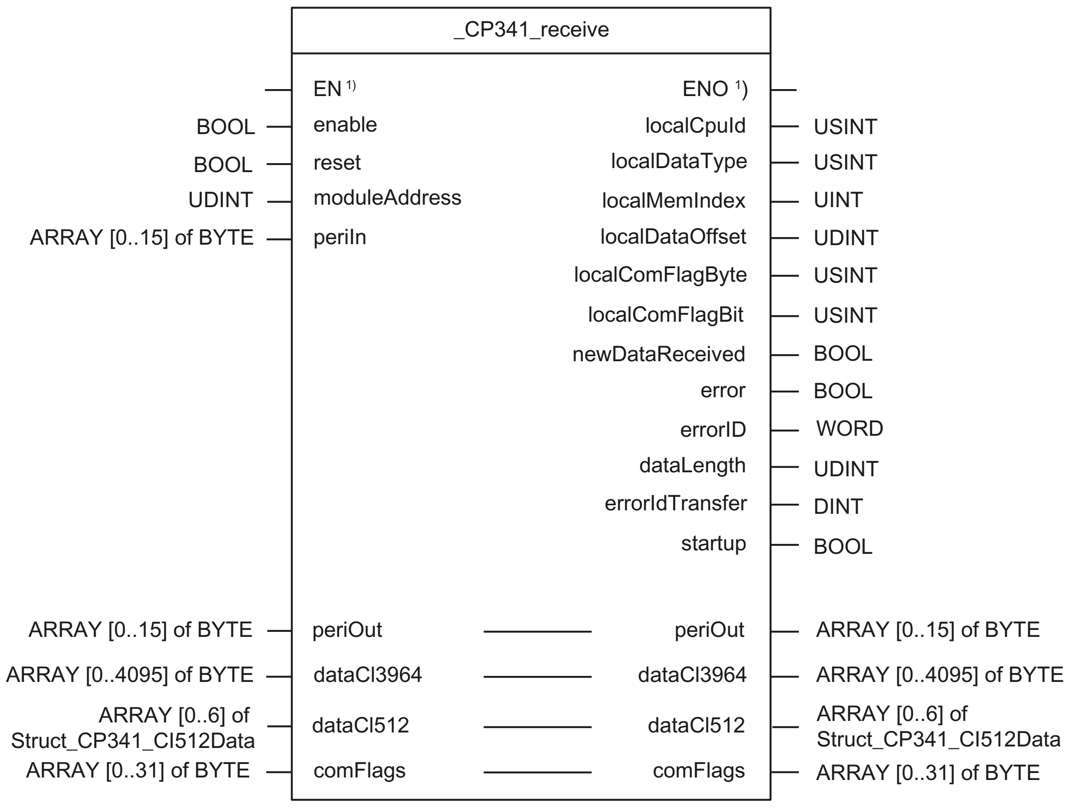
Before a job from the CP 341 can be received following a transition of the SIMOTION device from "STOP" to "RUN" mode, the CP-SIMOTION startup mechanism of the **_CP341_receive** FB must be complete.

The end of the startup coordination is indicated in output parameter **startup** = FALSE.

Assignment in the data area

During the receive operation, the data to be received is stored temporarily in the **_CP341_receive** FB. Once the data transfer from the CP 341 to the SIMOTION device is complete, the data are made available in the **dataCl512** (VAR_IN_OUT) parameter of the **_CP341_receive** FB.

Call (LAD representation)


_CP341_receive
EN 1)
ENO 1)
BOOL enable
BOOL reset
UDINT moduleAddress
ARRAY [0..15] of BYTE periIn
localCpuId USINT
localDataType USINT
localMemIndex UINT
localDataOffset UDINT
localComFlagByte USINT
localComFlagBit USINT
newDataReceived BOOL
error BOOL
errorID WORD
dataLength UDINT
errorIdTransfer DINT
startup BOOL
ARRAY [0..15] of BYTE periOut — periOut ARRAY [0..15] of BYTE
ARRAY [0..4095] of BYTE dataCl3964 — dataCl3964 ARRAY [0..4095] of BYTE
ARRAY [0..6] of Struct_CP341_Cl512Data dataCl512 — dataCl512 ARRAY [0..6] of Struct_CP341_Cl512Data
ARRAY [0..31] of BYTE comFlags — comFlags ARRAY [0..31] of BYTE

1) LAD-specific parameters

Parameter description

Table 4-4 Parameters of the _CP341_receive FB (application with RK 512 computer link)

Name	P type [1)]	Data type	Meaning	Actions performed by user	Actions performed by block
enable	IN	BOOL	Receive enable	Entered	Checked
reset	IN	BOOL	Cancels job	Entered	Checked
moduleAddress	IN	UDINT	Module address of the CP for data set transfer (from HW Config)	Entered	Checked
periIn	IN	ARRAY[0..15] of BYTE	I/O inputs of the CP transferred to the FB	I/O variable of the I/O inputs of the CP transferred to the FB	Checked
periOut	IN/OUT	ARRAY[0..15] of BYTE	Prepared FB data for the I/O outputs of the CP [3)]	Checked and transferred to the I/O variable for the I/O outputs	Entered
dataCl3964	IN/OUT	ARRAY[0..4095] of BYTE	Receive data array	Entered and checked	Entered
dataCl512	IN/OUT	ARRAY[0..6] of 'Struct_CP341_Cl512Data'	Data area for RK 512 two-dimensional array [2)]	Entered and checked	Checked
Struct_CP341_Cl512Data (data structure)					
data		ARRAY[0..4095] of BYTE	Data array for RK 512 computer link (send and fetch job)	SEND_CP: checked FETCH_CP: Entered	SEND_CP: entered FETCH_CP: Checked
comFlags	IN/OUT	ARRAY[0..31] of BYTE	Communication flag area for RK 512	Entered and checked	Entered
localCpuId	OUT	USINT	Number of the local communications partner	Checked	Entered
localDataType	OUT	USINT	Area type in the local communications partner	Checked	Entered
localMemIndex	OUT	UINT	Memory index in the local communications partner	Checked	Entered
localDataOffset	OUT	UDINT	First element in the local communications partner	Checked	Entered
localComFlagByte	OUT	USINT	Communication flag index in the local communications partner	Checked	Entered
localComFlagBit	OUT	USINT	Communication flag bit no. in the local communications partner	Checked	Entered
newDataReceived	OUT	BOOL	Job completed without errors	Checked	Entered
error	OUT	BOOL	Job completed with errors	Checked	Entered

Name	P type [1]	Data type	Meaning	Actions performed by user	Actions performed by block
errorID	OUT	WORD	Error specification For **error**=TRUE, the error information (event class and number) is displayed in the **errorID** parameter.[4]	Checked	Entered
dataLength	OUT	UDINT	Quantity of data received	Checked	Entered
errorIdTransfer	OUT	DINT	Error during data transfer to the CP (precise error diagnostics if 16#1E0F is present at the **errorID** parameter [5])	Checked	Entered
startup	OUT	BOOL	Indicates CP startup	Checked	Entered

[1] Parameter types: IN = input parameters, OUT = output parameters, IN/OUT = in/out parameters

[2] No data can be exchanged in array index **dataCl512[0]**.

[3] **Note:** The **periOut** parameter must be supplied with a variable of type **ARRAY[0..15] of BYTE**. Create a local or global variable in your program under **VAR** (do not create a temporary variable under VAR_TEMP). After the FB has been called, this variable must be assigned to the I/O variable for the I/O outputs of the module. See call example for CP 341.

[4] For error information, refer to the SIMATIC *CP 341 Point-to-Point Connection, Installation and Parameter Assignment* manual, Chapter "Diagnostics with the CP 341".

[5] For a more detailed description (_readRecord and _writeRecord), see *SIMOTION System Function/Variable Devices* Parameter Manual. This documentation is included in the SIMOTION SCOUT scope of delivery as electronic documentation.

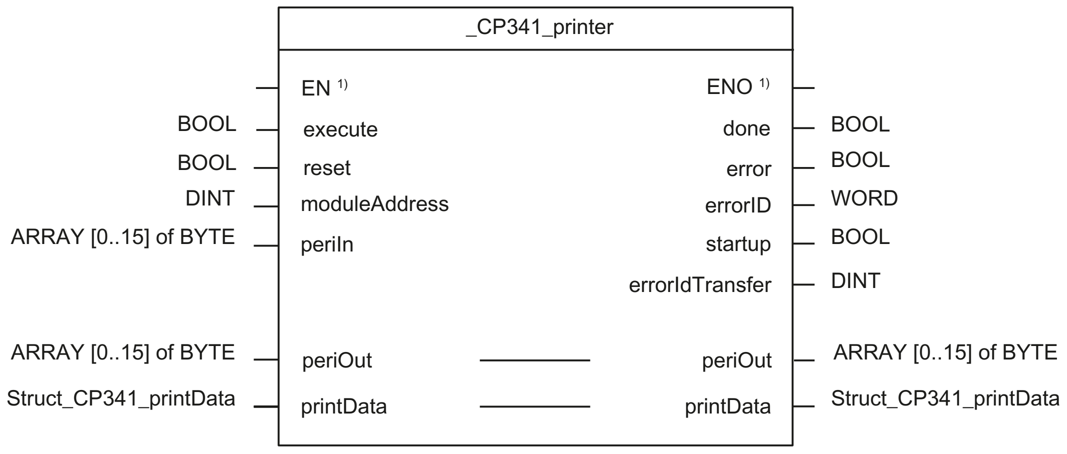
4.4 _CP341_printer function block

4.4.1 Description of the _CP341_printer FB

Function

The **_CP341_printer** function block is used to send data of type **Struct_CP341_printData** from the printer memory area to a serial printer. For example, the **_CP341_printer** function block might send a process message to the CP 341. The CP 341 prints out the process message on the connected printer.

Call (LAD representation)



1) LAD-specific parameters

Parameter description

Table 4-5 Parameters of the _CP341_printer FB

Name	P type 1)	Data type	Comment	Actions performed by user	Actions performed by block
execute	IN	BOOL	Initiates job on positive edge	Entered	Checked
reset	IN	BOOL	Cancels job	Entered	Checked
moduleAddress	IN	DINT	Module address of the CP for data set transfer (from HW Config)	Entered	Checked
periIn	IN	ARRAY[0..15] of BYTE	I/O inputs of the CP transferred to the FB	I/O variable of the I/O inputs of the CP transferred to the FB	Checked

Name	P type [1)]	Data type	Comment	Actions performed by user	Actions performed by block
periOut	IN/OUT	ARRAY[0..15] of BYTE	Prepared FB data for the I/O outputs of the CP [2)]	Checked and transferred to the I/O variable for the I/O outputs	Entered
printData	IN/OUT	**Struct_CP341_ printData**	Send data array (data to be printed)	Entered	Checked
Struct_CP341_printData (data structure) [5)]					
variable		ARRAY[0..3] of Struct_CP341_ dataRecord	Variable to be printed	Entered	Checked
format		ARRAY[0..150] of BYTE	Format string	Entered	Checked
Struct_CP341_dataRecord (data structure) [5)]					
dataLength		UDINT	Quantity of data	Entered	Checked
data		ARRAY[0..31] of BYTE	Print data	Entered	Checked
done	OUT	BOOL	Job completed without errors	Checked	Entered
error	OUT	BOOL	Job completed with errors	Checked	Entered
errorID	OUT	WORD	Error specification For **error**=TRUE, the error information (event class and number) is displayed in the **errorID** parameter.[3)]	Checked	Entered
startup	OUT	BOOL	Indicates CP startup	Checked	Entered
errorIdTransfer	OUT	DINT	Error during data transfer to the CP (precise error diagnostics if 16#1E0F is present at the **errorID** parameter [4)])	Checked	Entered

1) Parameter types: IN = input parameters, OUT = output parameters, IN/OUT = in/out parameters

2) **Note:** The **periOut** parameter must be supplied with a variable of type **ARRAY[0..15] of BYTE**. Create a local or global variable in your program under **VAR** (do not create a temporary variable under VAR_TEMP). After the FB has been called, this variable must be assigned to the I/O variable for the I/O outputs of the module. See call example for CP 341.

3) For error information, refer to the SIMATIC *CP 341 Point-to-Point Connection, Installation and Parameter Assignment* manual, Chapter "Diagnostics with the CP 341".

4) For a more detailed description (_readRecord and _writeRecord), see *SIMOTION System Function/Variable Devices* Parameter Manual. This documentation is included in the SIMOTION SCOUT scope of delivery as electronic documentation.

5) See "Message text structure", print storage area structure example

Signal sequence diagram of the _CP341_printer FB

The following figure illustrates the behavior of the **done** and **error** parameters according to the input circuit of **execute** and **reset**.

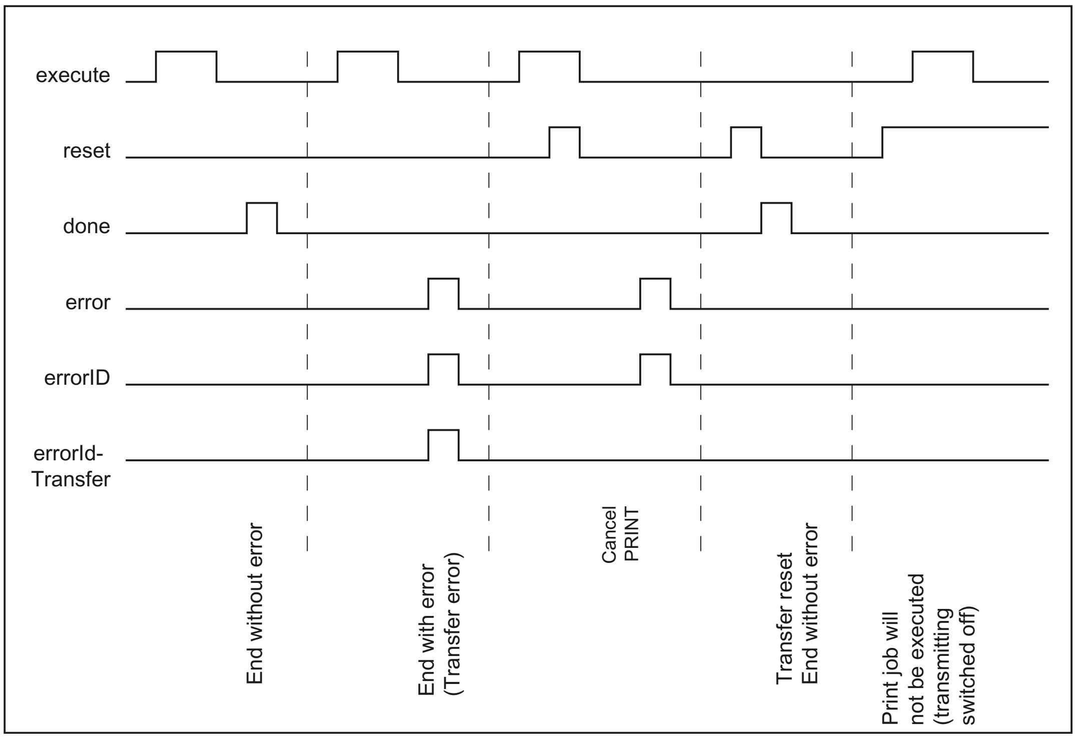

execute
reset
done
error
errorID
errorId-Transfer
End without error
End with error (Transfer error)
Cancel PRINT
Transfer reset End without error
Print job will not be executed (transmitting switched off)

Figure 4-7 Signal sequence diagram of the _CP341_printer FB

Note

The **execute** input is edge-triggered. The send job starts when there is a positive edge at the **execute** input.

Task integration (call)

The **_CP341_printer** function block must be called cyclically in the **BackgroundTask** or the **TimerInterruptTask**. Calling in the **SystemInterruptTask** is not permitted. Calling the function block in the **IPOSynchronousTask** is not recommended for runtime reasons.

The **_CP341_printer** function block is called repeatedly by a program. The print job can only be executed by cyclically calling the print FB.

A positive edge at the **execute** input initiates the transfer of the message text. A data transfer operation can run over several calls (program cycles), depending on the amount of data involved.

The active transfer job can be canceled by setting the **reset** parameter to "TRUE". This will reset the **_CP341_printer** FB to its initial state. The sending of print jobs will remain disabled as long as the signal state at the **reset** parameter is TRUE.

The **moduleAddress** parameter specifies the module address of the CP 341 being addressed for the data set transfer.

For call examples for the **_CP341_printer** FB, see Chapter CP 341 print call examples (Page 95).

Status and error display on the _CP341_printer FB

The **done** output parameter indicates that the job has been completed without errors. The **error** output indicates that an error has occurred. If an error occurs, the corresponding event class/number is displayed in the **errorID** output parameter (see "Parameters of the CP341_printer FB" table). If no errors have occurred, **errorID** has a value of 0. **Done** and **error/ errorID** are also output for **reset** of the **_CP341_printer** FB. When 16#1E0F is displayed in the **errorID** parameter, a detailed error description is also output via the **errorIdTransfer** parameter.

The **done**, **error, errorID**, and **errorIdTransfer** parameters are available for one block call only.

Note

There is no parameter check for the **_CP341_printer** function block. Incorrect parameterization of this block may cause the SIMOTION device to switch to "STOP" mode. Before the CP 341 can process an initiated job following a transition of the SIMOTION device from "STOP" to "RUN" mode, the CP-SIMOTION startup mechanism of the **_CP341_printer** FB must be complete. Any jobs initiated in the meantime will not be lost. They are transferred to the CP 341 once the startup coordination has finished.

The end of the startup coordination is indicated in output parameter **startup** = FALSE.

Assignment in the data area

The print data is transferred to the **_CP341_printer** FB as a data structure of type **Struct_CP341_printData** and copied to a local variable of the FB at the beginning of the print operation.

Structure of printer memory of type Struct_CP341_printData

The four variables to be printed and the format string must be entered in a variable with the following data type:

Example:

```
Struct_CP341_dataRecord     : STRUCT
     dataLength : UDINT;                        // Data quantity
     data       : ARRAY [0..31] of BYTE;        // Data field
END_STRUCT


Struct_CP341_printData     : STRUCT
     variable   : ARRAY [0..3] of Struct_CP341_dataRecord;  // 1st to 4th variable
     format     : ARRAY [0..150] of BYTE;                   // Format string
END_STRUCT
```

The first variable to be printed corresponds to the **variable [0]** element, the second variable to be printed corresponds to the **variable [1]** element, etc. The number of bytes to be printed per variable is limited to 32. The data for variable **i** must be placed in **variable [i-1].data[0..31].** The number of bytes to be printed must be entered in the **variable [i-1].dataLength** element.

The format string corresponds to the format element. The format string must be structured as follows (refer to the SIMATIC *CP 341 Point-to-Point Connection, Installation and Parameter Assignment* Manual):

- Specification of string length in **format [0]**
- Specification of individual characters in **format [1 to 150]**

Note

If the maximum length is exceeded, the print job is canceled and event number 16#1E41 is indicated at the **errorID** parameter output of the **_CP341_printer** FB.

Entering variables and message texts in the printer memory area

Before the data transfer to the CP 341 begins, the variable values to be printed must be entered **byte by byte and in the proper format** in the **variable[].data** element of the data structure of type **Struct_CP341_printData** (see item 2 in the example below). The number of bytes for each variable (variable length) must be assigned to the **variable.datalength** element (e.g. WORD type variable - variable.datalength:=2). An entry corresponding to the data type of the value must be made in the **format** element for each value entered in the **variable** element. (e.g. WORD type variable - %I). The total length of the entries in the **format** element must be assigned to the **format[0]** element.

You configure message texts with the CP 341 "point-to-point connection" parameter assignment interface. Once the hardware configuration has been downloaded to the SIMOTION device, the message texts are stored in the CP 341. The message texts that have been saved can be selected with corresponding entries in the **variable** and **format** elements.

Note

You can use supplemental function blocks (see Chapter supplemental function blocks (Page 90)) to enter values into the printer memory area and to select message texts.

Example:

- Print message text no. 3 (stored in CP 341).
  Configured message text: "This is message text no. 3"

```
myPrintData.variable[0].datalength := 2;   // Data type WORD
myPrintData.variable[0].data[0]    := 0;
myPrintData.variable[1].data[0]    := 3;   // Message text no. 3


myPrintData.format[0]     :=2;      // Format string length
myPrintData.format[1]     :=16#25;  // "%" Format specification for message
text
myPrintData.format[1]     :=16#4E   // "N" Format specification for message
text
```

- Print message text no. 4 with a WORD-type variable.

```
Configured message text   : "This is message text no. %I"
Printed text             : "This is message text no. 4"

myPrintData.variable[0].datalength := 2;  // Data type WORD
myPrintData.variable[0].data[0]    := 0;
myPrintData.variable[0].data[0]    := 4;  // Message text no.4

myPrintData.variable[1].datalength := 2;  // 2 Byte data type WORD
myPrintData.variable[1].data[0]    := 0   // High - Byte
myPrintData.variable[1].data[1]    := 4   // Low - Byte
myPrintData.format[0]              := 4;  // Format string length
myPrintData.format[1]          := 16#25;  // ASCII code "%" format
specification
myPrintData.format[2]          := 16#4E;  // ASCII code "N" format
specification
                                          // for message text
myPrintData.format[3]          := 16#25;  // ASCII code "%" format
specification
myPrintData.format[4]          := 16#49;  // ASCII code "I" format
specification
                                          // for integer
```

Notes on handling

The format string entry in the "format" field must be hexadecimal.

Example:
% corresponds to 25 hex in the IBM character set,
N (message text output) corresponds to 4E hex in the IBM character set.
(see Hardware configuration > Character set)

Data types DATE, TIME, DATE_AND_TIME_OF_DAY and TIME_OF_DAY are not supported. The date information must be entered as a DWORD or WORD in the printer data structure.

Representation type "A" (German date format):
//datefrg:=4018 (01.01.2001) and 4199 (01.07.2001)
printData.variable[0].datalength:=2;
printData.variable[0].data[0]:=WORD_TO_BYTE(SHR(datefrg,8));
printData.variable[0].data[1]:=WORD_TO_BYTE(SHR(datefrg,0));

Representation type "F":
The value to be printed must be in floating point format (mantissa/exponent) (see call example 2, Chapter CP 341 print call examples (Page 95))

Representation type "C":
If variable[ ].datalength:=1 in the printer data structure, the characters will be printed horizontally.
If variable[ ].datalength:=2 (3,4) in the printer data structure, the characters will be printed vertically.

Representation type "X":
For CP 341 RS232, product version E08 and higher, representation type "X" (binary) outputs the values correctly on a serial printer.

Disconnected printer
The communications link is not monitored for printers even if alarm generation is enabled in HW Config of STEP 7.

Example:

- A break in the connection between the printer and the CP 341 triggers neither an error nor a diagnostic alarm.
- Nor are they triggered if a print job is started but no printer is connected.

Note

The code in examples 1, 2, and 3 (see Chapter CP 341 print call examples (Page 95)) can be transferred to the SIMOTION SCOUT editor with Copy and Paste.

4.4.2 supplemental function blocks

Function

Supplemental function blocks are provided for entering variables of various data types as well as for entering message texts into data structure Struct_CP341_printData. You enter the value of the variables byte by byte and in the proper format in the variable element of the printer memory area and, optionally, in the format element. When numbers are entered for message texts, one entry is made in each of the format and variable elements. The method of representation for the variables in printed text and the method of entry in the format string can be selected by means of parameters.

The following supplemental function blocks are available:

- _CP341_realToPrintData
  Entry of a number of data type REAL into data structure Struct_CP341_printData
- _CP341_dwordToPrintData
  Entry of a number of data type DWORD into data structure Struct_CP341_printData
- _CP341_wordToPrintData
  Entry of a number of data type WORD into data structure Struct_CP341_printData
- _CP341_byteToPrintData
  Entry of a number of data type BYTE into data structure Struct_CP341_printData
- _CP341_dintToPrintData
  Entry of a number of data type DINT into data structure Struct_CP341_printData

- **_CP341_intToPrintData**
  Entry of a number of data type INT into data structure **Struct_CP341_printData**
- **_CP341_printMsgText**
  Selection of message texts stored in CP 341

Note

SINT and USINT data types can be entered into data structure **Struct_CP341_printData** with the **_CP341_intToPrintData** function block. Type conversion is implicit.

Parameter description

Table 4-6 _CP341_byteToPrintData, _CP341_wordToPrintData, _CP341_dwordToPrintData parameters

Name	P type [1)]	Data type	Comment	Actions performed by user	Actions performed by block
execute	IN	BOOL	Edge-triggered job initiation	Entered	Checked
data	IN	BYTE/WORD/ DWORD	Variable/value to be entered in the data structure.	Entered	Checked
numVariable	IN	INT	Number of the variable in which the entry is to be made. 1≤ **numVariable** ≤ 4	Entered	Checked
entryFormatString	IN	ENUM	Method of entry in the format string and method of representation of the value in the **data** parameter.	Entered	Checked
		CP_REPLACE_WITH_SIGN	Entry starting at byte 1 in the substructure of the format string; existing entries are overwritten. Method of representation: signed integer		
		CP_REPLACE_WITHOUT_SIGN	Entry starting at byte 1 in the substructure of the format string; existing entries are overwritten. Method of representation: unsigned integer		
		CP_REPLACE_BINARY	Entry starting at byte 1 in the substructure of the format string; existing entries are overwritten. Method of representation: binary		
		CP_ADD_WITHOUT_SIGN	Entry is added to the substructure of the format string, **entryAtByteNumber** parameter is evaluated. Method of representation: unsigned integer		
		CP_ADD_WITH_SIGN	Entry is added to the substructure of the format string, **entryAtByteNumber** parameter is evaluated. Method of representation: signed integer		

Name	P type [1)]	Data type	Comment	Actions performed by user	Actions performed by block
		CP_NO_ENTRY	No entry in the substructure of the format string		
entryAtByteNumber	IN	USINT	Specifies the byte at which the entry is to begin in the substructure of the format string. **entryAtByteNumber** = 0 Entry is made after the last entry found	Entered	Checked
printData	IN/OUT	Struct_CP341_printData	Data structure for the printer data	No actions	Enters values
done	OUT	BOOL	Job completed without errors	Checked	Entered
error	OUT	BOOL	Job completed with errors (permissible value range exceeded)	Checked	Entered

1) Parameter types: IN = input parameters, OUT = output parameters, IN/OUT = in/out parameters

Table 4-7 _CP341_realToPrintData

Name	P type [1)]	Data type	Comment	Actions performed by user	Actions performed by block
execute	IN	BOOL	Edge-triggered job initiation	Entered	Checked
data	IN	REAL	Variable/value to be entered in the data structure.	Entered	Checked
numVariable	IN	INT	Number of the variable in which the entry is to be made. 1 ≤ **numVariable** ≤ 4	Entered	Checked
entryFormatString	IN	ENUM	Method of entry in the format string and method of representation of the value in the **data** parameter.	Entered	Checked
		CP_REPLACE_WITHOUT_EXPONENT	Entry starting at byte 1 in the substructure of the format string; existing entries are overwritten. Method of representation: floating-point		
		CP_REPLACE_WITH_EXPONENT	Entry starting at byte 1 in the substructure of the format string; existing entries are overwritten. Method of representation: with exponent		
		CP_ADD_WITHOUT_EXPONENT	Entry is added in the substructure of the format string, **entryAtByteNumber** parameter is evaluated. Method of representation: floating-point		
		CP_ADD_WITH_EXPONENT	Entry is added in the substructure of the format string, **entryAtByteNumber** parameter is evaluated. Method of representation: with exponent		

Name	P type [1)]	Data type	Comment	Actions performed by user	Actions performed by block
		CP_NO_ENTRY	No entry in the substructure of the format string		
entryAtByteNumber	IN	USINT	Specifies the byte at which the entry is to begin in the substructure of the format string. **entryAtByteNumber** = 0 Entry is made after the last entry found	Entered	Checked
printData	IN/OUT	Struct_CP341_printData	Data structure for the printer data	No actions	Enters values
done	OUT	BOOL	Job completed without errors	Checked	Entered
error	OUT	BOOL	Job completed with errors (permissible value range exceeded)	Checked	Entered

1) Parameter types: IN = input parameters, OUT = output parameters, IN/OUT = in/out parameters

Table 4-8 _CP341_dintToPrintData, _CP341_intToPrintData

Name	P type [1)]	Data type	Comment	Actions performed by user	Actions performed by block
execute	IN	BOOL	Edge-triggered job initiation	Entered	Checked
data	IN	INT/DINT	Variable/value to be entered in the data structure.	Entered	Checked
numVariable	IN	INT	Number of the variable in which the entry is to be made. 1 ≤ **numVariable** ≤ 4	Entered	Checked
entryFormatString	IN	ENUM	Method of entry in the format string and method of representation of the value in the **data** parameter.	Entered	Checked
		CP_DEFAULT	Entry starting at byte 1 in the substructure of the format string; existing entries are overwritten. Method of representation: signed integer		
		CP_ADD_TO_STRING	Entry is added to the substructure of the format string, **entryAtByteNumber** parameter is evaluated. Method of representation: signed integer		
		CP_NO_ENTRY	No entry in the substructure of the format string		
entryAtByteNumber	IN	USINT	Specifies the byte at which the entry is to begin in the substructure of the format string. **entryAtByteNumber** = 0 Entry is made after the last entry found	Entered	Checked

Name	P type [1)]	Data type	Comment	Actions performed by user	Actions performed by block
printData	IN/OUT	Struct_CP341_printData	Data structure for the printer data	No actions	Enters values
done	OUT	BOOL	Job completed without errors	Checked	Entered
error	OUT	BOOL	Job completed with errors (permissible value range exceeded)	Checked	Entered

1) Parameter types: IN = input parameters, OUT = output parameters, IN/OUT = in/out parameters

Table 4-9 _CP341_printMsgText

Name	P type [1)]	Data type	Comment	Actions performed by user	Actions performed by block
execute	IN	BOOL	Edge-triggered job initiation	Entered	Checked
numMsgText	IN	USINT	Number of the message text (stored in the CP 341)	Entered	Checked
numVariable	IN	INT	Number of the variable in which the entry is to be made. 1 ≤ **numVariable** ≤ 4	Entered	Checked
entryFormatString	IN	ENUM	Method of entry in the format string and method of representation of the value in the **data** parameter.	Entered	Checked
		CP_DEFAULT	Entry starting at byte 1 in the substructure of the format string; existing entries are overwritten. Method of representation: signed integer		
		CP_ADD_TO_STRING	Entry is added to the substructure of the format string, **entryAtByteNumber** parameter is evaluated. Method of representation: signed integer		
		CP_NO_ENTRY	No entry in the substructure of the format string		
entryAtByteNumber	IN	USINT	Specifies the byte at which the entry is to begin in the substructure of the format string **entryAtByteNumber** = 0 Entry is made after the last entry found	Entered	Checked
printData	IN/OUT	Struct_CP341_printData	Data structure for the printer data	No actions	Enters values
done	OUT	BOOL	Job completed without errors	Checked	Entered
error	OUT	BOOL	Job completed with errors (permissible value range exceeded)	Checked	Entered

1) Parameter types: IN = input parameters, OUT = output parameters, IN/OUT = in/out parameters

Task integration (call)

The supplemental blocks must be called in the **BackgroundTask** or the **TimerInterruptTask**.

For call examples for the **_CP341_printer** FB, see Chapter CP 341 print call examples (Page 95).

Status and error indicators

The **done** output indicates that the job has been completed without errors. The **error** output indicates that an error has occurred.

The **done** and **error** parameters are available for one block call only.

4.4.3 CP 341 print call examples

Call example 1

```
UNIT E1CP341p;

INTERFACE
VAR_GLOBAL
  myPreparePrintData          : BOOL;              // Initiate prepare print request
  myRequestPrint              : BOOL;              // Initiate transfer to printer
  myReset                     : BOOL;              // Abort print
  myActualLevel               : REAL := 5.67;      // actual value "level"
  myModuleAddress_1           : DINT:= 256;        // address of 1st CP341 module
  myPrintData                 : Struct_CP341_printData; // instance of datastruct
  myFB_CP341_printMessageText : _CP341_printMsgText;    // instances of function blocks
  myFB_CP341_realToPrintData  : _CP341_realToPrintData; // instances of function blocks
  myFB_CP341_print            : _CP341_printer;         // instance of function block
  myOutputArrayCP341_1        : ARRAY[0..15] of BYTE;   // field for CP341 output data
END_VAR
PROGRAM Example_print_1;                           // program for BackgroundTask
END_INTERFACE

IMPLEMENTATION
PROGRAM Example_print_1
// BackgroundTask program

// Example to print out messagetext 3 with one variable
// ("actualLevel") of type REAL: "The actual level (l) is: <actualLevel>"
// The message-text 3 must be specified in the hardwareconfiguration of CP341
// like this: "The actual level (l) is:"
// The following I/O-variable for CP341 module are required:
// peripheralInputCP341_1: input address of CP341 module; type Array; length 16
// peripheralOutputCP341_1: output address of CP341 module; type Array; length 16

// entry to printDatastruct myprintData
myFB_CP341_printMessageText ( execute           := myPreparePrintData,
                              printData         := myPrintData,
                              numMsgText        := 3,        // number of message text
                              numVariable       := 1,        // number of variable
                              entryFormatString := CP_DEFAULT);

myFB_CP341_realToprintData  ( execute           := myPreparePrintData,
                              printData         := myPrintData,
                              data              := myActualLevel,
                              numVariable       := 2,
                              entryFormatString := CP_ADD_WITH_EXPONENT );
// call instance of _CP341_printer
// use requestPrint to start datatransfer to serial printer

myFB_CP341_print      ( execute       := myRequestPrint,           // initiate request
                        reset         := myReset,                  // abort request
                        moduleAddress := myModuleAddress_1,        // module address
                        periIn        := myPeripheralinputcp341_1, // peripheral input
                        periOut       := myOutputArrayCP341_1,     // output data field
                        printData     := myPrintData);             // data to print out

// transfer output data field to peripheral output
myPeripheralOutputCP341_1 := myOutputArrayCP341_1;

END_PROGRAM //Example_print_1
END_IMPLEMENTATION
```

Call example 2

```
UNIT E2CP341p;

INTERFACE
VAR_GLOBAL
  myRequestPrint        : BOOL;                     // Initiate transfer to printer
  myReset               : BOOL;                     // abort print
  myPrintDword          : DWORD;                    // value to print out
  myModuleAddress_1     : DINT := 256;              // address of 1st CP341 module
  myPrintData           : Struct_CP341_printData;   // instance of datastruct
  myFB_CP341_print      : _CP341_printer;           // instance of function block
  myOutputArrayCP341_1  : ARRAY[0..15] OF BYTE;     // field for CP341 output data
END_VAR
PROGRAM Example_print_2;                            // program for BackgroundTask
END_INTERFACE

IMPLEMENTATION
PROGRAM Example_print_2                             // BackgroundTask program
// example with notation "F" and variable of type DWORD
// (containing mantissa and exponent)

// The following I/O-variable for CP341 module are required:
// peripheralInputCP341_1: input address of CP341 module; type Array; length 16
// peripheralOutputCP341_1: output address of CP341 module; type Array; length 16

// formatstring (length 2 Byte, notation "F"):
myPrintData.format[0]  := 2 ;
myPrintData.format[1]  := 16#25 ;           // "%"
myPrintData.format[2]  := 16#46 ;           // "F"

// assignment for variable (type DWORD), to print out with notation "F"
myPrintDword := REAL_TO_DWORD(10000.0);  // variable (DWORD) with mantissa and exponent

// !!!
// ATTENTION! wrong example for this case is an assingnment with an integer value e.g.:
// myprintDword := 10000;        // because the format is WITHOUT mantissa and exponent
// !!!

// fill out printData interface manually with DWORD-variable
myPrintData.variable[0].dataLength := 4 ;    // 1st variable, lenth 4 Byte
myPrintData.variable[0].data[0] := WORD_TO_BYTE(DWORD_TO_WORD(SHR(printDword,24)));
myPrintData.variable[0].data[1] := WORD_TO_BYTE(DWORD_TO_WORD(SHR(printDword,16)));
myPrintData.variable[0].data[2] := WORD_TO_BYTE(DWORD_TO_WORD(SHR(printDword,8)));
myPrintData.variable[0].data[3] := WORD_TO_BYTE(DWORD_TO_WORD(SHR(printDword,0)));

// call instance of _CP341_printer
// use requestPrint to start datatransfer to serial printer
myFB_CP341_print ( execute        := myRequestPrint,            // initiate request
                   reset          := myReset,                   // abort request
                   moduleAddress  := myModuleAddress_1,         // module address
                   periIn         := myPeripheralinputcp341_1,  // peripheral input
                   periOut        := myOutputArrayCP341_1,      // output data field
                   printData      := myPrintData);              // data to print out

// transfer output data field to peripheral output
myPeripheralOutputCP341_1 := myOutputArrayCP341_1;

END_PROGRAM //Example_print_2
END_IMPLEMENTATION
```

Call example 3

```
UNIT E3CP341p;

INTERFACE
VAR_GLOBAL
  myRequestPrint              : BOOL;                    // Initiate transfer to printer
  myReset                     : BOOL;                    // Abort print
  myPrintReal1                : REAL;                    // 1st value
  myPrintReal2                : REAL;                    // 2nd value
  myPrintReal3                : REAL;                    // 3rd value
  myModuleAddress_1           : DINT := 256;             // address of 1st CP341 module
  myPrintData                 : Struct_CP341_printData;  // instance of datastruct
  myFB_CP341_print            : _CP341_printer;          // instance of function block
  myFB_CP341_realToprintData1 : _CP341_realToPrintData;  // instances of function blocks
  myFB_CP341_realToprintData2 : _CP341_realToPrintData;  // instances of function blocks
  myFB_CP341_realToprintData3 : _CP341_realToPrintData;  // instances of function blocks
  myOutputArrayCP341_1        : ARRAY[0..15] OF BYTE;
END_VAR
PROGRAM Example_print_3;                                 // program for BackgroundTask
END_INTERFACE

IMPLEMENTATION
PROGRAM Example_print_3                                  // BackgroundTask program
// The following example program demonstrates usage of _CP341_realToPrintData()

// The following I/O-variable for CP341 module are required:
// peripheralInputCP341_1: input address of CP341 module; type Array; length 16
// peripheralOutputCP341_1: output address of CP341 module; type Array; length 16

// preset variables with user-values
myPrintReal1 := 1.11; myPrintReal2 := 2.22; myPrintReal3 := 3.33;

// write variables (type REAL) to printDatastruct with _CP341_realToPrintData
```

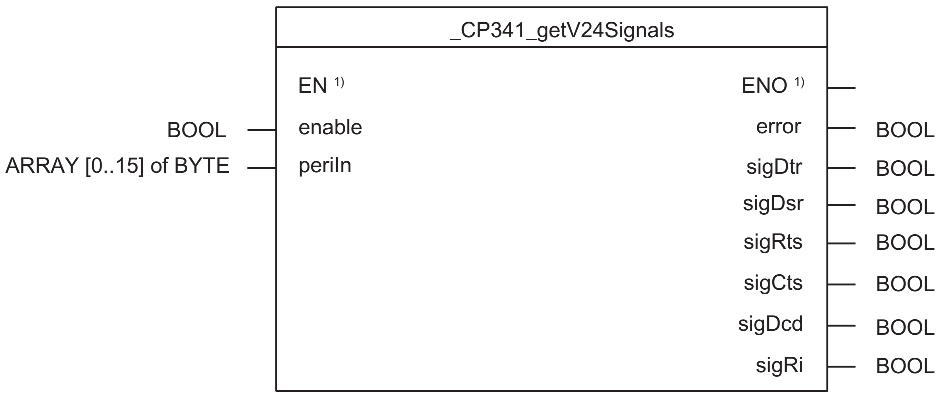
```
myFB_CP341_realToprintData1 (execute           := TRUE,
                             data              := myPrintReal1,        // 1st variable
                             numVariable       := 1,
                             entryFormatString := CP_ADD_WITHOUT_EXPONENT,
                             entryAtByteNumber := 0,
                             printData         := myPrintData);
myFB_CP341_realToprintData2 (execute           := TRUE,
                             data              := myPrintReal2;        // 2nd variable
                             numVariable       := 2,
                             entryFormatString := CP_ADD_WITHOUT_EXPONENT,
                             entryAtByteNumber := 0,
                             printData         := myPrintData);
myFB_CP341_realToprintData3 (execute           := TRUE,
                             data              := myPrintReal3,        // 3rd variable
                             numVariable       := 3,
                             entryFormatString := CP_ADD_WITHOUT_EXPONENT,
                             entryAtByteNumber := 0,
                             printData         := myPrintData);

// call instance of _CP341_printer, "requestPrint" starts data transfer
// to serial printer
myFB_CP341_print (  execute        := myRequestPrint,            // initiate request
                    reset          := myReset,                   // abort request
                    moduleAddress  := myModuleAddress_1,         // module address
                    periIn         := myPeripheralinputcp341_1,  // peripheral input
                    periOut        := myOutputArrayCP341_1,      // output data field
                    printData      := myPrintData);              // data to print out

// transfer output data field to peripheral output
myPeripheralOutputCP341_1 := myOutputArrayCP341_1;

END_PROGRAM                       //Example_print_3
END_IMPLEMENTATION
```

4.5 _CP341_getV24Signals function block

Function

The **_CP341_getV24Signals** function block reads the RS-232-C accompanying signals from the CP 341 and makes them available to the user in the block parameters. The functionality of the **_CP341_getV24Signals** FB can only be used if a parameterized ASCII driver is specified.

Call (LAD representation)



1) LAD-specific parameters

Parameter description

Table 4-10 Parameters of the _CP341_getV24Signals FB

Name	P type 1)	Data type	Meaning	Actions performed by user	Actions performed by block
enable	IN	BOOL	Block enable	Entered	Checked
periIn	IN	ARRAY[0..15] of BYTE	I/O inputs of the CP transferred to the FB	I/O variable of the I/O inputs of the CP transferred to the FB	Checked
error	OUT	BOOL	Job completed with errors	Checked	Entered
sigDtr	OUT	BOOL	Data terminal ready	Checked	Entered
sigDsr	OUT	BOOL	Data set ready	Checked	Entered
sigRts	OUT	BOOL	Request to send	Checked	Entered
sigCts	OUT	BOOL	Clear to send	Checked	Entered
sigCcd	OUT	BOOL	Data carrier detected	Checked	Entered
sigRi	OUT	BOOL	Ring Indicator	Checked	Entered

1) Parameter types: IN = input parameters, OUT = output parameters, IN/OUT = in/out parameters

Task integration (call)

The _CP341_getV24Signals function block must be called cyclically in the BackgroundTask or the TimerInterruptTask. Calling in the SystemInterruptTask is not permitted. Calling the function block in the IPOSynchronousTask is not recommended for runtime reasons.

The RS 232C accompanying signals are updated each time the function is called (cyclic polling). The CP 341 updates the status of the inputs/outputs in a time base of 20 ms.

Note

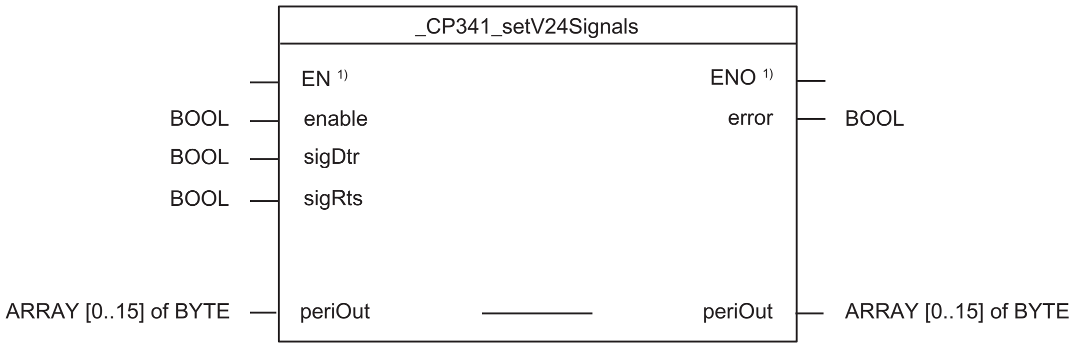
A minimum pulse duration is necessary to detect a signal change. Determining factors are the cycle time (SIMOTION device), the update time on the CP 341, and the response time of the communications partner.

4.6 _CP341_setV24Signals function block

Function

The **_CP341_setV24Signals** function block can be used to set or reset RS-232-C accompanying signals. The functionality of the **_CP341_setV24Signals** FB can only be used if a parameterized ASCII driver is specified.

Call (LAD representation)



1) LAD-specific parameters

Parameter description

Table 4-11 Parameters of the _CP341_setV24Signals FB

Name	P type 1)	Data type	Meaning	Actions performed by user	Actions performed by block
enable	IN	BOOL	Block enable	Entered	Checked
sigDtr	IN	BOOL	Data terminal ready	Entered	Checked
sigRts	IN	BOOL	Request to send	Entered	Checked
periOut	IN/OUT	ARRAY [0..15] of BYTE	Prepared FB data for the I/O outputs of the CP 2)	Checked and transferred to the I/O variables for the I/O outputs	Entered
error	OUT	BOOL	Job completed with errors	Checked	Entered

1) Parameter types: IN = input parameters, OUT = output parameters, IN/OUT = in/out parameters

2) **Note:** The **periOut** parameter must be supplied with a variable of type **ARRAY[0..15] of BYTE**. Create a local or global variable in your program under **VAR** (do not create a temporary variable under VAR_TEMP). After the FB has been called, this variable must be assigned to the I/O variable for the I/O outputs of the module. See call example for CP 341.

Task integration (call)

The **_CP341_getV24Signals** function block must be called cyclically in the **BackgroundTask** or the **TimerInterruptTask**. Calling in the **SystemInterruptTask** is not permitted. Calling the function block in the **IPOSynchronousTask** is not recommended for runtime reasons.

4.7 Calling the CP 341 function blocks

In order to be able to work with the function blocks in your user project, proceed as follows (the numbers shown in the following program segment correspond to the steps below):

1. Create the function block instance (see the following program segment, e.g. create instance for the **_CP341_send** FB).
2. Create a variable for the data structure (for RK 512 computer interfacing only).
3. Create an array for the in/out parameters of the FB.
4. Call instance of the function block.
5. Transfer input parameters.
6. The output parameters of the FB are accessed with <instance name of FB>.
   <name of output parameter>.
7. Data prepared by the FB for the I/O outputs are assigned to the array of the I/O variables created in step 3.

Note

The CP 341 call example is an extract from the supplied E_CP341 application example, which is included on the "SIMOTION Utilities & Applications" CD-ROM.

If you wish to control multiple CP 341 devices, you must create a new variable for the data structure and FB instances with a new name for each CP 341 you implement.

Call example for CP 341

```
UNIT E_CP341;

INTERFACE

VAR_GLOBAL
      myExecSendCP341      : BOOL;                    // Trigger send task
      myModuleAddrCP341    : DINT:=256;               // module address
      mySendDataArrayCP341 : ARRAY [0..4095] OF BYTE; // Send data array 4,096 bytes
      myInstCP341Send      : _CP341_send;             // Create instance of FB             (1)
END_VAR

PROGRAM ExampleCP341;                                // Program in BackgroundTask

END_INTERFACE

IMPLEMENTATION
    VAR_GLOBAL
      MyResetSend          : BOOL;                    // Cancel send order
    END_VAR
```

```
PROGRAM ExampleCP341                                 // Program in BackgroundTask

// Variables used: see interface area under VAR_GLOBAL
VAR
    MyCPOutputArray       : ARRAY [0..15] OF BYTE;   // Array for CP output data                (3)
END_VAR

VAR_TEMP
    MyDataLengthSend      : UDINT;            // Length of data to be sent
    MyDataOffsetSend      : UDINT;            // Offset of first byte to be sent
END_VAR

// CALL FB INSTANCE TO SEND
    myInstCP341Send (                                                                         (4)
        mode            := SEND_CP,                // Send data
        execute         := myExecSendCP341,        // Trigger order                          (5)
        reset           := myResetSend,            // Order book
        moduleAddress   := myModuleAddrCP341,      // Module address
        dataOffset      := myDataOffsetSend,       // Offset
        dataLength      := myDataLengthSend,       // Number of data to be sent
        periIn          := myPeripheralInputCP341, // I/O variable of I/O inputs
        periOut         := myCPOutputArray,        // Data for I/O outputs
        data            := mySendDataArrayCP341    // Send data array
        );

    myStateStartUpCP341 := myInstCP341Send.startUp;  // Start-up status                     (6)

// TRANSFER DATA TO CP341
    myPeripheralOutputCP341  := myCPOutputArray;    // Array for output variables            (7)
                                                    // of I/O variables

END_PROGRAM               // ExampleCP341

END_IMPLEMENTATION
```

Note

The "ExampleCP341" program must be assigned in the execution system.

4.8 Data consistency

When sending data

Once a job is initiated by a positive edge at the **execute** input, the data to be sent is copied to the static memory area of the send FB. This means that once the FB call has ended, the send data array can be written to again for the next send request on a positive edge. The data are retained as consistent data within the send FB.

Note

During the copy operation to the static memory area of the FB, data consistency cannot be guaranteed if the send/receive data areas are accessed in a higher priority task.

When receiving data

Once the receive request is complete, the data are copied over to the receive buffer in a block from the static memory area of the receive FB. This means that when the FB call is complete, either all data (**newDataReceived** = TRUE) are entered in the receive buffer or no data (**newDataReceived** = FALSE) have been entered.

Note

During the copy operation from the static memory area of the receive FB, data consistency cannot be guaranteed if the send/receive data areas are accessed in a higher priority task.

4.9 Special features related to data transfer

4.9.1 Communication flag function with the CP 341

The communication flag functionality with respect to RK 512 computer interfacing is similar to that in the SIMATIC. Once data has been sent or fetched, a bit is set in the communications partner. This bit is checked when a new job is initiated. If it is set (= TRUE), the job is not processed. The communications partner must first reset this bit (= FALSE). The bit to be set or checked is specified by the requester FB (_CP341_send) in the remoteComFlagByte and remoteComFlagBit parameters.

SIMOTION ↔ SIMOTION

In the variable (ARRAY [0..31] of BYTE) created in the comFlags parameter, the bit position of the communication flag bit to be checked or set is specified as follows:

- remoteComFlagByte corresponds to the array index of the byte in which the bit is to be set/ checked.
- remoteComFlagBit corresponds to the bit position in the byte specified by the remoteComFlagByte parameter.

Note

The same variable must always be transferred at the comFlags parameter during the block call to ensure consistency of the communication flag bit.

SIMOTION ↔ SIMATIC

Communication flags are always stored in the flag area MB0 to MB254 in SIMATIC. The bit position of the communication flag bit to be checked or set is specified as follows:

- remoteComFlagByte corresponds to the flag byte in which the bit is to be set or checked (e.g. remoteComFlagByte=1 corresponds to MB1).
- remoteComFlagBit corresponds to the bit position in the byte specified by the remoteComFlagByte parameter.

4.9.2 Requests that can be processed simultaneously with the CP 341

The following function blocks may only be programmed once in your user program for each CP 341 communication processor used:

- _CP341_send FB
- _CP341_receive FB

4.9.3 Data transfer with the RK 512 computer interfacing

Data transfer options

Active jobs:

The **_CP341_send** function block enables you to issue active jobs to the CP 341 in the user program of the SIMOTION system. You can

- Send data from your automation system to a remote communications partner.
- Fetch data from a remote communications partner and store it in the send data array.

Note

If you fetch data from a CP 341, a **_CP341_receive** function block must be programmed for the communications partner.

Passive jobs:

The **_CP341_receive** function block enables you to use passive jobs to coordinate the reading of data on the CP 341 and make the data available. The communications partner is active. You can

- Enter data sent by the communications partner in the **dataCl512** receive data array.
- Make data available from **dataCl512** for a remote communications partner.

Special feature related to sending data

Note the following special features related to "sending" data:

- With RK 512, the amount of sent data must be an even number. If an odd value is specified for the length in the **dataLength** parameter, an additional filler byte with a value of "FALSE" will be transferred at the end of the data.
- With RK 512, any specified offset must be an even number. If an odd offset is specified, the data will be stored for the partner starting with the next smallest even offset.

4.10 Application example of the CP 341

Function

This example shows how to:

- use the **_CP341_send** function block to send data from the Send data array to a communications partner.
- use the **_CP341_receive** function block to receive data in the receive data array.

In the example program, the CP 341 is used both as the sender and receiver. This requires the jumpering of the send and receive lines (PIN 2 and PIN 3 on the RS232 interface) and the "ASCII" setting in the parameter assignment tool. The **_CP341_send** FB is used to transfer the send data to the CP module. This sends the data using the RS232 interface. The jumpered send and receive line means that the data to be sent is read immediately by the CP module. The **_CP341_receive** FB reads the received data from the CP module and copies these data to the receive data array.

This example requires proper installation of the parameter assignment tool, as described in the SIMATIC *CP 341 Point-to-Point Connection, Installation and Parameter Assignment* manual.

Hardware platform

The application example is available for various SIMOTION hardware platforms. You must adapt the example for centralized applications with SIMOTION C.

Note

If the application example is not available for your hardware platform, you must adapt the hardware configuration.

Assigning module parameters

Proceed as follows:

1. Open your project in SIMOTION SCOUT.
2. Open the hardware configuration in SIMOTION SCOUT.
3. Configure your hardware station with a CP 341 module.
4. Double-click the **CP 341** to open the "Properties" dialog box for this module. Click the "Parameters" button to launch the parameter assignment tool of the CP 341 module.
5. The "ASCII" protocol must be selected in the protocol selection box.
   The standard settings of the ASCII protocol suffices for the application example.
6. Jumper the send and receive line (PIN 2/PIN 3) of the RS232 interface.
7. Apply your settings in the parameter assignment interface with the "File" > "Save" menu command, and close the interface with the "File" > "Exit" menu command. Click "OK" to close the Properties window for the CP 341 module.

8. Save the hardware configuration with the "Station" > "Save and compile" menu command.
9. Download the hardware configuration with the "Target system" > "Download to module" menu command.
   The red "SF" LED on the IM 153 turns on and then off if the assigned module parameters have been downloaded without errors.

Adapting the application example

The configuration in the example and its available hardware must be adapted.

The following options are available:

1. You can adapt the configuration in the example to the available hardware (e.g. PROFIBUS DP address).
2. You can adapt the configuration of the hardware to the example (e.g. PROFIBUS DP address).

Calling the application example

The application example can be found on the "SIMOTION Utilities & Applications" CD-ROM. The "SIMOTION Utilities & Applications" CD-ROM is provided free of charge and part of the SIMOTION SCOUT scope of delivery.

1. Dearchive and open the project containing the application example.
2. Check the axis configuration: PROFIBUS DP addresses.
3. Check the module addresses (hardware configuration) against the I/O addresses of the controller in SIMOTION SCOUT and module address in the program (myModuleAddrCP341).
4. Save and compile the example project. Then, you can download the example to the SIMOTION device and switch to **RUN** mode.

Sequence of the application example

Table 4-12 Input icons used

Symbol	Data type	Designation
mySelectPointToPointCP341	BOOL	Select point-to-point connection
myExecSendCP341	BOOL	Initiate send job
mySendOrder1	BOOL	Select send: Job 1
mySendOrder2	BOOL	Select send: Job 2
mySendOrder3	BOOL	Select send: Job 3
myResetSend	BOOL	Cancel send job
myEnableToReceive	BOOL	Receive enable
myReceiveOrder1	BOOL	Select receive: Job 1
myReceiveOrder2	BOOL	Select receive: Job 2
myReceiveOrder3	BOOL	Select receive: Job 3
myResetReceive	BOOL	Cancel receive job

Symbol	Data type	Designation
myModuleAddrCP341	DINT	CP 341 module address, default 256
mySendDataArrayCP341	ARRAY[0..4095] of BYTE	Send data array
myReceiveDataArrayCP341	ARRAY[0..4095] of BYTE	Receive data array

Table 4-13 Output symbols used

Symbol	Data type	Designation
mySendDone	BOOL	Send: Completed
mySendError	BOOL	Send: Error display
mySendErrorNumber	WORD	Send: Error status
mySendTransErrorNumber	DINT	Send: Error status transfer
myNewDataReceived	BOOL	Receive: New data have been received
myReceiveError	BOOL	Receive: Error display
myReceiveErrorNumber	WORD	Receive: Error status
myReceiveTransErrorNumber	DINT	Receive: Error status transfer
myStateStartupCP341	BOOL	CP 341 startup status FALSE = startup completed
myDiagnosticAlarm	BOOL	TRUE = diagnostic alarm present on the CP 341
myProcessAlarm	BOOL	TRUE = process alarm present on the CP341
myAlarmInterrupt	UDINT	Type of the alarm (process, diagnostic alarm)
myLogBaseAddrIn	DINT	Module address
myLogBaseAddrOut	DINT	
myLogAddress	DINT	Diagnostic address
myAlarmDetails	DWORD	Alarm information

Note

You can monitor and control the input and output variables used in the programming example in the INTERFACE area of the unit (under VAR_GLOBAL); alternatively, you can assign real inputs and outputs to the input and output variables in your user program.

For the "point-to-point connection" application example, set the "myselectPointToPointCP341" input to "TRUE". This will call function blocks contained in the application example.

Receiving data:

To receive data, you must set the "myEnableToReceive" variable to "TRUE" (static signal). If receive jobs 1 and 3 are enabled (myReceiveOrder1 = TRUE and "myReceiveOrder3" = TRUE), the data is stored in the "myReceiveDataArrayCP341" data array starting with the "myReceiveDataArrayCP341[0]" array element (data offset is 0). If job 2 is enabled ("myReceiveOrder2" = TRUE), the data is stored in the "myReceiveDataArrayCP341" data array starting with the "myReceiveDataArrayCP341[20]" array element (data offset is 20).

If "myNewDataReceived" = TRUE, this indicates that new data has been received. This signal is present for one cycle only.

If an error occurred during the transfer ("myReceiveError" = TRUE) the error code is stored in the "mySendErrorNumber" variable. If error code 16#1E0F is present in "myReceiveErrorNumber", an error occurred during the data transfer. The transfer error code is stored in the "myReceiveTransErrorNumber" variable. The error signals are reset when you set input "myResetReceive" = TRUE.

Sending data:

You can use the "mySendOrder1", "mySendOrder2" and "mySendOrder3" inputs to select between three send jobs:

- Job 1 sends 10 bytes of data from the "mySendDataArrayCP341" data array from array element "mySendDataArrayCP341[0]" to "mySendDataArrayCP341[9]"
- Job 2 sends 20 bytes of data from the "mySendDataArrayCP341" data array from array element "mySendDataArrayCP341[20]" to "mySendDataArrayCP341[39]"
- Job 3 sends 4,096 bytes of data from the "mySendDataArrayCP341" data array.

The data is sent to the communications partner if the "myExecSendCP341" input detects a signal change from "FALSE" to "TRUE" (positive edge).
If output signal "mySendDone" = TRUE, the send job has been completed. A new job can be sent if the "myExecSendCP341" input signal detects another signal change from FALSE to TRUE.

If an error occurred during the transfer ("mySendError" = TRUE), the error code is stored in the "mySendErrorNumber" variable. If error code 16#1E0F is present in "mySendErrorNumber", an error occurred during the data transfer. The transfer error code is stored in the "mySendTransErrorNumber" variable. The error signals are reset when you set input "myExecSendCP341" = FALSE.

When the signal state at the "myResetSend" or "myResetReceive" input is set to "TRUE", the send job or receive job is canceled, respectively. If the signal state remains "TRUE", sending and receiving of data is disabled.

Note

Proper data transfer can be observed as follows:

- The "TxD" and "RxD" LEDs on the CP module illuminate.
- Output parameter (_CP341_send FB) done = TRUE or NewDataReceived = TRUE

Alarm processing 5

Pending error messages are processed and evaluated differently in a SIMOTION system than in a SIMATIC system. Diagnostic alarms are not enabled by default. Enable the alarms for each module in the hardware configuration, see Chapter Integrating the communications processors in the SIMOTION project (Page 15).

If you have parameterized diagnostic alarms, then you should program the alarm processing sequence according to the principle presented below.

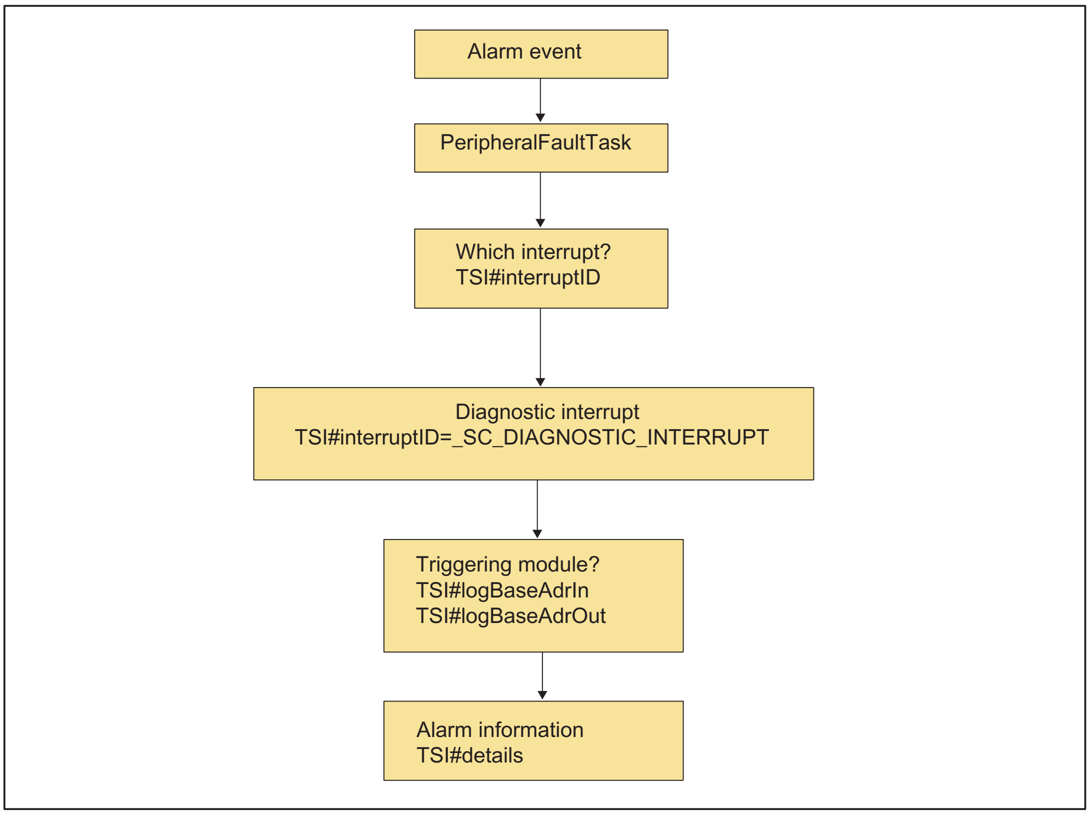

Alarm event
PeripheralFaultTask
Which interrupt? TSI#interruptID
Diagnostic interrupt TSI#interruptID=_SC_DIAGNOSTIC_INTERRUPT
Triggering module? TSI#logBaseAdrIn TSI#logBaseAdrOut
Alarm information TSI#details

Figure 5-1 Alarm processing for CP 340 or CP 341

Alarm evaluation

Alarms originating from the I/O are evaluated in the **PeripheralFaultTask**. When the **PeripheralFaultTask** is started, the **Taskstartinfo** is made available, which you can evaluate in the user program.

The **Taskstartinfo** of **PeripheralFaultTask** is comparable to the local data of OB82 in the SIMATIC system.

Table 5-1 Meaning of the Taskstartinfo

Task	TSI		Remarks
PeripheralFaultTask	DT	TSI#startTime	Start time of the task
	UDINT	TSI#interruptID	Identifies the triggering event: • _SC_PROCESS_INTERRUPT • _SC_DIAGNOSTIC_INTERRUPT • _SC_STATION_DISCONNECTED • _SC_STATION_RECONNECTED
	DINT	TSI#logBaseAdrIn	Logical base address if a process alarm (PRAL) or a diagnostic alarm (DAL) was caused by an input area on the module, otherwise _SC_INVALID_ADDRESS
	DINT	TSI#logBaseAdrOut	Logical base address if a process alarm (PRAL) or a diagnostic alarm (DAL) was caused by an output area on the module, otherwise _SC_INVALID_ADDRESS
	DINT	TSI#logDiagAdr	Diagnostic address of a DP slave if the alarm was caused by a station failure or station recovery of an associated DP slave, otherwise _SC_INVALID_ADDRESS
	DWORD	TSI#details	Detail information (bit fields)

Definition of a diagnostic alarm

If the user program is to respond to an internal or external error, you can set the parameters for a diagnostic alarm that will interrupt the cyclical program of the SIMOTION device.

Events triggering a diagnostic alarm

The criteria (events) that trigger diagnostic alarms in a SIMOTION system are the same as in a SIMATIC system.

More detailed information is available in the following SIMATIC manuals: *CP 340 Point-to-Point Connection, Installation and Parameter Assignment and CP 341 Point-to-Point Connection, Installation and Parameter Assignment*.

Responses to a diagnostic alarm

If a diagnostic alarm occurs, the following take place:

- Diagnostic data are written to **TSI#details** variable in the Taskstartinfo of **PeripheralFaultTask**.
- In the **PeripheralFaultTask**, you can read out and save the 4 bytes of diagnostic data.
- The group error LED (SF) of the CP module lights up. The group error LED (SF) is extinguished as soon as the error has been remedied.

Bit assignment

The **TSI#details** variable is assigned in the same way as in a SIMATIC system.

Note

More information is available in the following SIMATIC manuals:

- *CP 340 Point-to-Point Connection, Installation and Parameter Assignment*
- *CP 341 Point-to-Point Connection, Installation and Parameter Assignment*

Appendices A

A.1 SIMOTION and SIMATIC names

The tables below contain a comparison of SIMOTION and SIMATIC names.

Table A-1 SIMOTION and SIMATIC names for CP 340

Name in the SIMOTION system as of V4.0 (command library in SCOUT)	Name in the SIMATIC system	Name in the SIMOTION system up to V3.2 (SIMOTION Function Library)
Function block parameters		
_CP340_send	FB P_SEND (FB 3)	_FB_CP340_send
execute	REQ	request
reset	R	abort
moduleAddress	LADDR	moduleAddress
dataOffset	DBB_NO	dataOffset
dataLength	LEN	dataLength
periIn	-	inputInterface
periOut	-	outputInterface
data	DB_NO	data
done	DONE	done
error	ERROR	error
errorID	STATUS	errorNumber
errorIdTransfer	-	transferErrorNumber
startup	-	startup
_CP340_receive	FB P_RCV (FB 2)	_FB_CP340_receive
enable	EN_R	enable
reset	R	abort
moduleAddress	LADDR	moduleAddress
dataOffset	DBB_NO	dataOffset
periIn	-	inputInterface
periOut	-	outputInterface
data	DB_NO	data
newDataReceived	NDR	newDataReceived
error	ERROR	error
dataLength	LEN	dataLength
errorID	STATUS	errorNumber
errorIdTransfer	-	transferErrorNumber
startup	-	startup
_CP340_printer	FB P_PRINT (FB 4)	_FB_CP340_print

Name in the SIMOTION system as of V4.0 (command library in SCOUT)	Name in the SIMATIC system	Name in the SIMOTION system up to V3.2 (SIMOTION Function Library)
execute	REQ	request
reset	R	abort
moduleAddress	LADDR	moduleAddress
periIn	-	inputInterface
periOut	-	outputInterface
printData	(DB_NO/DBB_NO)	printerData
done	DONE	done
error	ERROR	error
errorID	STATUS	errorNumber
errorIdTransfer	-	transferErrorNumber
startup	-	startup
_CP340_getV24Signals	FC V24_STAT (FC 5)	_FB_CP340_getV24State
enable	-	-
periIn	-	inputInterface
error	-	-
sigDtr	DTR_OUT	signalDTR
sigDsr	DSR_IN	signalDSR
sigRts	RTS_OUT	signalRTS
sigCts	CTS_IN	signalCTS
sigDcd	DCD_IN	signalDCD
sigRi	RI_IN	signalRI
_CP340_setV24Signals	FC V24_SET (FC 6)	_FB_CP340_setV24Signals
enable	-	-
sigDtr	DTR	signalDTR
sigRts	RTS	signalRTS
periOut	-	outputInterface
error	-	-
Data structure elements	Data structure elements	
Struct_CP340_printData	No direct reference to SIMATIC	Struct_CP340_printerData
variable	-	variable
format	-	format
Struct_CP340_dataRecord	No direct reference to SIMATIC	Struct_CP340_dataRecord
dataLength	-	dataLength
data	-	data
Module parameters (supplemental function blocks)	Module parameters (supplemental function blocks)	

Name in the SIMOTION system as of V4.0 (command library in SCOUT)	Name in the SIMATIC system	Name in the SIMOTION system up to V3.2 (SIMOTION Function Library)
_CP340_byteToPrintData, _CP340_wordToPrintData, _CP340_dwordToPrintData,	-	_FB_CP340_byteToPrinterdata, _FB_CP340_wordToPrinterdata, _FB_CP340_dwordToPrinterdata,
execute	-	execute
data	-	data
numVariable	-	numberVariable
entryFormatString	-	entryFormatstring
entryAtByteNumber	-	entryAtByteNumber
printData	-	printerData
done	-	done
error	-	error
_CP340_realToPrintData	-	_FB_CP340_realToPrinterdata
execute	-	execute
data	-	data
numVariable	-	numberVariable
entryFormatString	-	entryFormatstring
entryAtByteNumber	-	entryAtByteNumber
printData	-	printerData
done	-	done
error	-	error
_CP340_dintToPrintData, _CP340_intToPrintData	-	_FB_CP340_dintToPrinterdata, _FB_CP340_intToPrinterdata
execute	-	execute
data	-	data
numVariable	-	numberVariable
entryFormatString	-	entryFormatstring
entryAtByteNumber	-	entryAtByteNumber
printData	-	printerData
done	-	done
error	-	error
_CP340_printMsgText	-	_FB_CP340_printMessageText
execute	-	execute
numMsgText	-	numberMsgText
numVariable		numberVariable
entryFormatString	-	entryFormatstring
entryAtByteNumber	-	entryAtByteNumber
printData	-	printerData
done	-	done
error	-	error

Table A-2 SIMOTION and SIMATIC names for CP 341

Name in the SIMOTION system as of V4.0 (command library in SCOUT)	Name in the SIMATIC system	Name in the SIMOTION system up to V3.2 (SIMOTION function library)
Function block parameters		
_CP341_send	FB P_SND_RK (FB 8)	_FB_CP341_send
mode	SF	operatingMode
execute	REQ	request
reset	R	abort
moduleAddress	LADDR	moduleAddress
dataOffset	DBB_NO	dataOffset
dataLength	LEN	dataLength
periIn	-	inputInterface
periOut	-	outputInterface
data	DB_NO	data
done	DONE	done
error	ERROR	error
errorID	STATUS	errorNumber
errorIdTransfer	-	transferErrorNumber
remoteCpuId	R_CPU_NO (computer interfacing)	remoteCPUNumber
remoteDataType	R_Typ (computer interfacing)	remoteDataType
remoteMemIndex	R_NO (computer interfacing)	remoteMemoryIndex
remoteDataOffset	R_OFFSET (computer interfacing)	remoteDataOffset
remoteComFlagByte	R_CF_BYT (computer interfacing)	remoteComFlagByte
remoteComFlagBit	R_CF_BIT (computer interfacing)	remoteComFlagBit
startup	_	startup
_CP341_receive	FB P_RCV_RK (FB 7)	_FB_CP341_receive
enable	EN_R	enable
reset	R	abort
moduleAddress	LADDR	moduleAddress
dataOffset	DBB_NO	dataOffset
periIn	-	inputInterface
periOut	-	outputInterface
dataCl3964	DB_NO (procedure/ASCII driver)	dataCL3964
newDataReceived	NDR	newDataReceived
error	ERROR	error
dataLength	LEN	dataLength
errorID	STATUS	errorNumber
errorIdTransfer	-	transferErrorNumber
dataCl512	-	dataCL512
comFlags	-	communicationFlags
localCpuId	-	localCPUNumber
localDataType	L_TYP (computer interfacing)	localDataType

Name in the SIMOTION system as of V4.0 (command library in SCOUT)	Name in the SIMATIC system	Name in the SIMOTION system up to V3.2 (SIMOTION function library)
localMemIndex	L_NO (computer interfacing)	localMemoryIndex
localDataOffset	L_OFFSET (computer interfacing)	localDataOffset
localComFlagByte	L_CF_BYT (computer interfacing)	localComFlagByte
localComFlagBit	L_CF_BIT (computer interfacing)	localComFlagBit
startup	_	startup
_CP341_printer	FB P_PRINT_RK (FB13)	No equivalent
execute	REQ	-
reset	R	-
moduleAddress	LADDR	-
periIn	-	-
periOut	-	-
printData	DB_NO/DBB_NO	-
done	DONE	-
error	ERROR	-
errorID	STATUS	-
errorIdTransfer	-	-
startup	-	-
_CP341_byteToPrintData, _CP341_wordToPrintData, _CP341_dwordToPrintData, _CP341_realToPrintData, _CP341_dintToPrintData, _CP341_intToPrintData, _CP341_printMsgText	-	No equivalent
_CP341_getV24Signals	FC V24_STAT (FC 5)	_FB_CP341_getV24State
enable	-	-
periIn	-	inputInterface
error		-
sigDtr	DTR_OUT	signalDTR
sigDsr	DSR_IN	signalDSR
sigRts	RTS_OUT	signalRTS
sigCts	CTS_IN	signalCTS
sigDcd	DCD_IN	signalDCD
sigRi	RI_IN	signalRI
_CP341_setV24Signals	FC V24_SET (FC 6)	_FB_CP341_setV24Signals
enable		-
periIn	-	inputInterface
error		-

Name in the SIMOTION system as of V4.0 (command library in SCOUT)	Name in the SIMATIC system	Name in the SIMOTION system up to V3.2 (SIMOTION function library)
sigDtr	DTR	signalDTR
sigRts	RTS	signalRTS
periOut	-	outputInterface
Data structure elements	**Data structure elements**	
Struct_CP341_Cl512CpData	No direct reference to SIMATIC	**Struct_CP341_CL512Data**
data	-	data

A.2 List of abbreviations

Table A-3 Abbreviations

Abbreviation	Meaning
CP	Communications processor
DP	Distributed I/O
FB	Function block
HW	Hardware
IM	Interface Modul (SIMATIC S7-300 interface module)
IN	Input parameters
IN/OUT	In/out parameters
I/O	Input/Output
LAD	Ladder Logic
LED	Light Emitting Diode (Light emitting diodes)
OUT	Output parameter
RK	Computer link

Index