

Industrial Remote Communication: Data Communication via GPRS with S7-1200 and CP 1242-7

S7-1200 Set 32

Wireless Signaling and Switching via SMS with S7-1200
Controllers

[Application Description • June 2013](#)

Applications & Tools

Answers for industry.

SIEMENS

Warranty and Liability

Note

The application examples are not binding and do not claim to be complete regarding the circuits shown, equipping and any eventuality. The application examples do not represent customer-specific solutions. They are only intended to provide support for typical applications. You are responsible for ensuring that the described products are correctly used. These application examples do not relieve you of the responsibility of safely and professionally using, installing, operating and servicing equipment. When using these application examples, you recognize that Siemens cannot be made liable for any damage/claims beyond the liability clause described. We reserve the right to make changes to these application examples at any time without prior notice. If there are any deviations between the recommendations provided in these application examples and other Siemens publications – e.g. Catalogs – then the contents of the other documents have priority.

We do not accept any liability for the information contained in this document.

Any claims against us – based on whatever legal reason – resulting from the use of the examples, information, programs, engineering and performance data etc. described in this application example shall be excluded. Such an exclusion shall not apply in the case of mandatory liability, e.g. under the German Product Liability Act (“Produkthaftungsgesetz”), in case of intent, gross negligence, or injury of life, body or health, guarantee for the quality of a product, fraudulent concealment of a deficiency or breach of a condition which goes to the root of the contract (“wesentliche Vertragspflichten”). However, claims arising from a breach of a condition which goes to the root of the contract shall be limited to the foreseeable damage which is intrinsic to the contract, unless caused by intent or gross negligence or based on mandatory liability for injury of life, body or health. The above provisions do not imply a change in the burden of proof to your detriment.

It is not permissible to transfer or copy these application examples or excerpts of them without first having prior authorization from Siemens Industry Sector in writing.

Caution

The functions and solutions described in this article confine themselves to the realization of the automation task predominantly. Please take into account furthermore that corresponding protective measures have to be taken up in the context of Industrial Security when connecting your equipment to other parts of the plant, the enterprise network or the Internet. Further information can be found under the Item-ID 50203404.

<http://support.automation.siemens.com/WW/view/en/50203404>

Table of Contents

Warranty and Liability	2
Table of Contents	3
1 Automation Problem	4
1.1 Overview	4
2 Solution.....	6
2.1 Overview of the overall solution	6
2.2 Description of the core functionality.....	8
2.3 Hardware and software components used	10
2.4 Performance data	11
3 Basics	12
3.1 GMS network.....	12
3.2 Characteristic features of the CP device configuration.....	14
3.3 Definition of the connection-specific characteristics	14
3.4 Establishing the connection	16
3.5 Transmitting process data via a sub-connection	18
4 Functional Mechanisms of this Application.....	20
4.1 Functionality	21
4.1.1 Basic blocks	21
4.1.2 Related blocks.....	23
4.1.3 Interfaces	29
4.1.4 Status codes of the blocks	30
4.2 Extensions and adaptations	32
5 Commissioning of the Application	34
5.1 Installing and wiring the hardware	34
5.2 Configuration instructions	35
5.3 Error handling.....	36
6 Operation of the Application.....	37
6.1 Overview	38
6.1.1 SMS_Send	38
6.1.2 SMS_Broadcast.....	39
6.1.3 SMS_Recv	40
6.1.4 SMS_Escalation	41
6.1.5 Set/Get.....	41
7 References	44
8 History	44

1 Automation Problem

1.1 Overview

Introduction

This application shows how to set up a simple system for switching and signaling by means of SMS messages. The field of application used as an example is a drink vending machine.

Overview of the automation problem

At regular intervals, the automatic machine checks:

- The remaining stock of the different types of beverages
- The coin box level
- The cooling system temperature

In addition, unauthorized opening of the front can be detected using a sensor.

The automatic machine reacts independently to specific events – such as exceeding/falling below limits – and informs the maintenance staff.

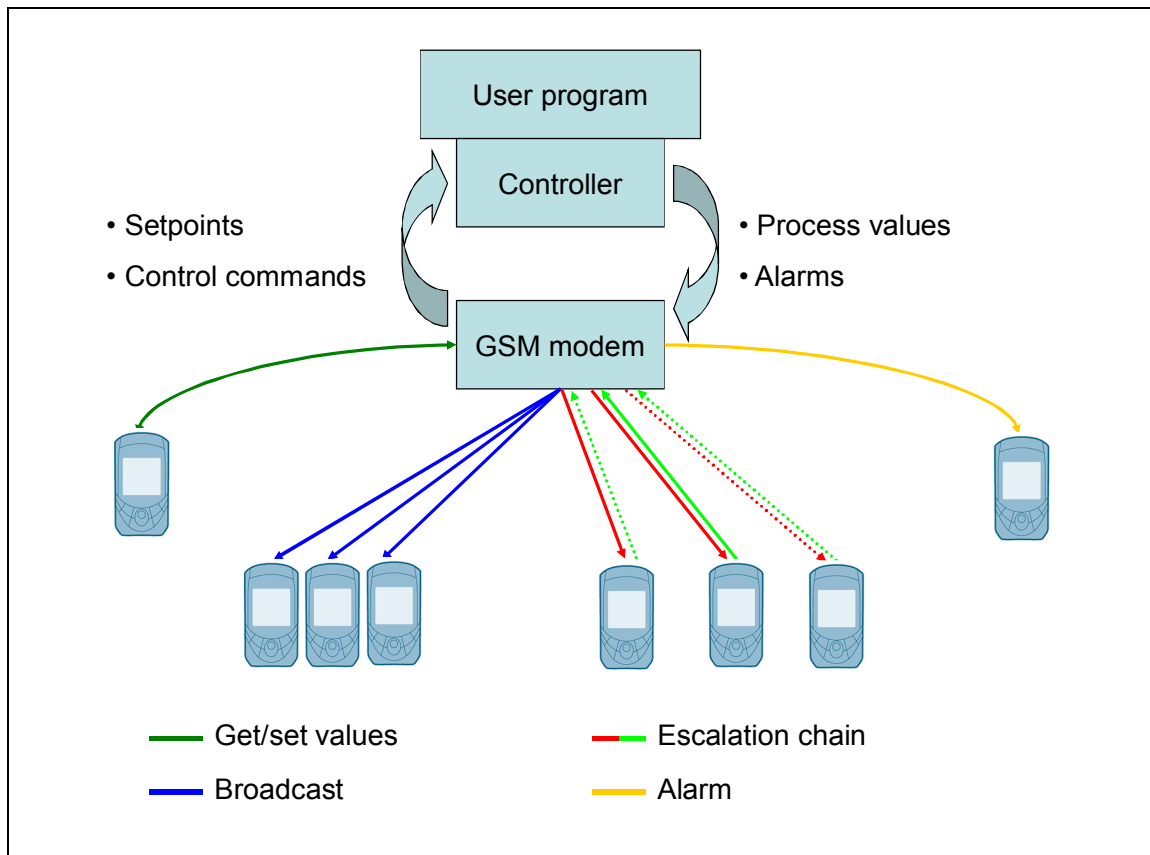
Irrespective of that, the automatic machine answers requests that have been received for process values.

Requirements for the solution

This automation problem results in the following requirements for the functions of the solution:

- Sending an SMS message to a parameterizable phone number
 - Process values can be integrated in the message text
 - The send routine can be started from the user program of the controller using a trigger
- Sending a message to several subscribers (Broadcast)
- Sending a message with subsequent waiting for acknowledgement by one of the configured recipients of the message (Escalation)
- Receiving messages. Depending on the message text,
 - requested values are read out of the controller and sent to a mobile phone
 - received commands are executed in the controller and a confirmation/error message is sent to the mobile phone

Figure 1-1 Overview



2 Solution

2.1 Overview of the overall solution

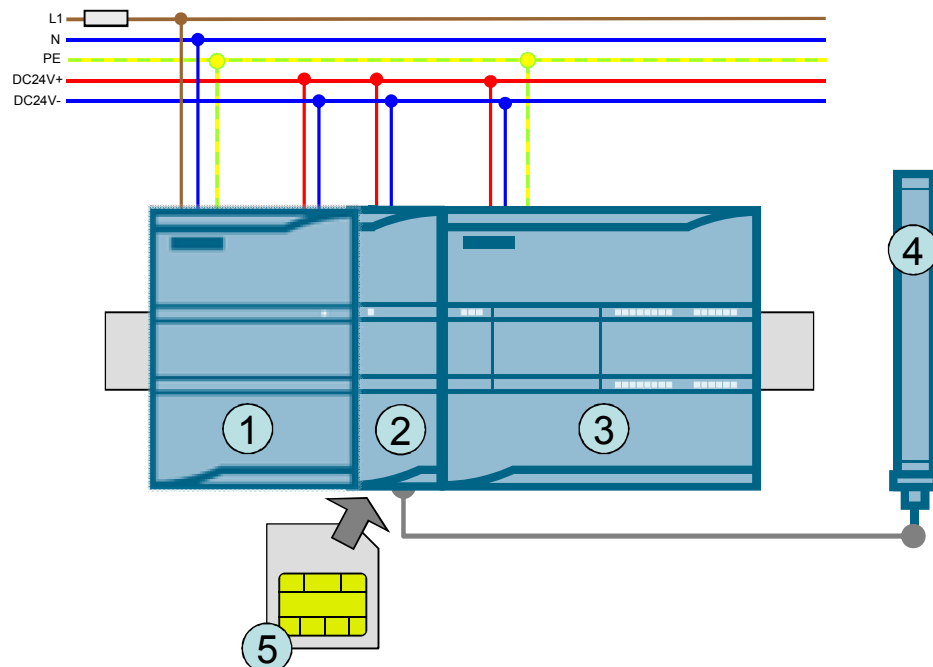
Station configuration

A **SIMATIC CP 1242-7 (2)** is connected to a **SIMATIC S7-1200 controller 1211C (3)** via the bus interface.

A **SIM card (5)** is inserted in the GSM/GPRS modem and an **ANT 794-4MR quad-band GSM/GPRS antenna (4)** is used for the connection to the air interface.

The power supply of all components is provided via a **SIMATIC PM 1207 power module (1)**.

Figure 2-1 Configuration display of the station



Communication partners

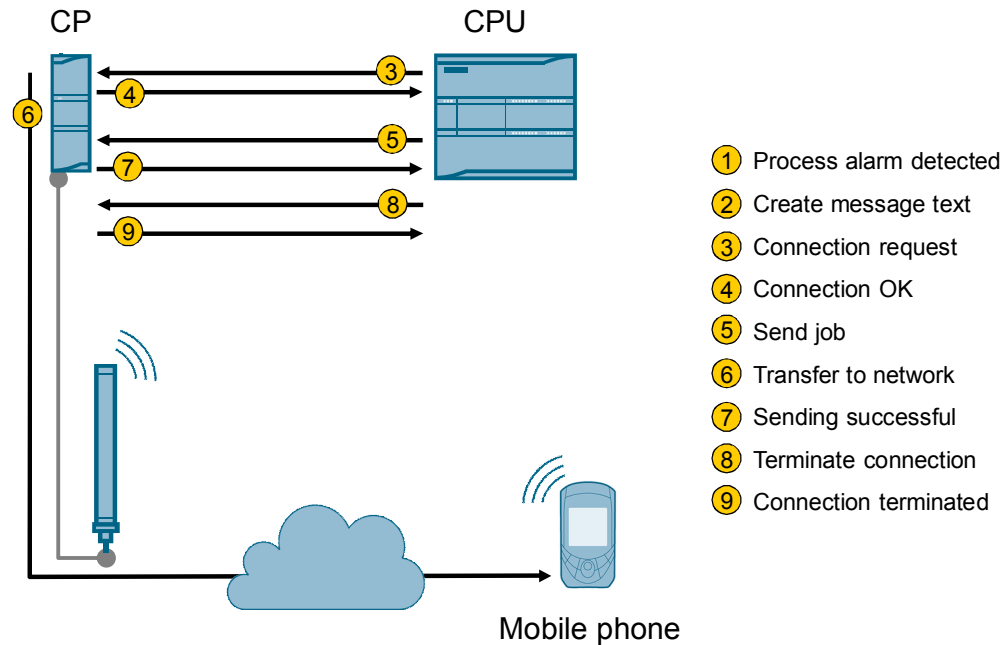
All terminal units that support sending SMS messages in the GSM network are possible for the communication. In most cases, mobile phones will be used. Data exchange with other stations with integrated GSM modem is possible as well.

Using the SMStoFAX / SMStoMail services, the messages can also be forwarded to fax machines or e-mail addresses – if the SIM card provider offers these functions.

Diagrammatic representation

The diagrammatic representation below shows the most important components of the solution:

Figure 2-2



Scope

This application does not include a description of the

- operation of STEP7 V11
- programming in SCL and FBD

Basic knowledge of these topics is required.

2.2 Description of the core functionality

Library blocks

The TC_ communication blocks of the CP form the basis. They represent the basic functions for communication in the GSM network and have already been integrated in STEP 7 V11:

- TC_CON [FB165] → Connect
- TC_SEND [FB163] → Send data
- TC_RECV [FB164] → Receive data
- TC_DISCON [FB166] → Disconnect

SIMATIC CP 1242-7 GPRS – initial situation

The following table describes the system performance of the CP that has to be considered when creating the solution:

Table 2-1

System performance	Solution
Receiving SMS messages: The CP does not transfer the phone number of the sender to the user program (TC blocks).	The phone number must be included in the message text.
Receiving SMS messages: When there are several TC_RECV calls, the first TC_RECV block with a suitable phone number (no number parameterized / fully specified number parameterized) will accept the message. The following ones will not receive messages.	One global receive block with one RC_RECV call is used. All other blocks access its text buffer.
Receiving SMS messages: When a message is received while no TC_RECV is active, it will be lost.	The TC_RECV block in the global receive block is kept permanently active (BUSY).
Sending SMS messages: When sending, the phone number can only be changed on TC_CON.	SMS messages are always sent in three steps: CON → SEND → DISCON

Basic application blocks

In the application, two basic blocks are used that call the above-listed TC blocks:

1. **SMS_Send**
Sends a message text to the parameterized phone number.
2. **SMS_Recv**
Receives messages and stores them in a data area.

Related blocks

Internally, all blocks that are more complex call their own instances of **SMS_Send** and/or access the buffer of **SMS_Recv**.

1. **SMS_Broadcast**
Sends a message text to any number of phone numbers.
2. **Escalation**
 - Sends a message to a phone number.
 - Waits for a parameterized time for the partner's acknowledgement.
 - After the time has elapsed, the next number is contacted.

3. Set/Get

Values can be requested from the process – it is also possible to specify new setpoints.

Functions

In FBD, processing strings is often difficult. Therefore, these program parts are relocated to SCL functions.

1. CheckString

Checks a phone number for the valid characters "0123456789+".

2. CheckMessage

In this application example, all messages to the station must comply with a specified format. The function checks the short text message for this format.

3. Set/Get Commands

A part of the message text contains the commands for setting/getting variables. The command names and the following program execution can be freely selected by the user. This function:

- Contains all commands configured by the user.
- Searches the message part for known commands.
- Executes the actual command.
- Returns the response text depending on the result.

4. Date_to_String

This function reads out the current system time and converts it to a string. This string can be inserted into any message text.

Advantages of this solution

The solution presented here offers you the following advantages:

- Existing S7 1200 stations can be retrofitted.
- Modular design – only blocks that are actually required have to be downloaded to the user program. (When commissioning this application for the first time, however, the full functional scope of the sample project should remain unchanged in order to better understand the blocks and exclude malfunctions.)
- Other user blocks can easily use the defined message format of the related blocks (Broadcast, Escalation, Set/Get) as a basis and use it.

2.3 Hardware and software components used

The application was created and tested with the following components.

Hardware components

Table 2-2

Component	Qty.	Order no.	Note
SIMATIC S7-1200, PM 1207	1	6EP1332-1SH71	2.5A
SIMATIC S7-1200, CPU 1211C	1	6ES7211-1AD30-0XB0	DC/DC/DC Firmware 2.2
S7-1200 SIMULATOR MODULE, 8 INP	1	6ES7274-1XF30-0XA0	For controlling the digital inputs.
SIMATIC CP 1242-7 GPRS,	1	6GK7242-7KX30-0XE0	Firmware 1.2.1
SINAUT ANT 794-4MR, rod aerial	1	6NH9860-1AA00	Alternative: ANT794-3M flat antenna (6NH9870-1AA00)
Ethernet cable TP CORD RJ45/RJ45 2M	1	6XV1870-3QH20	For configuring (2 m of this cable or a comparable cable)
Circuit breaker	1	5SX2116-6	1-pole, B, 16A
Standard mounting rail	1	6ES5 710-8MA11	35 mm
SIM card	1	Available from your wireless service provider	Enabled SMS function GSM sufficient (no GPRS necessary)

Standard software components

Table 2-3

Component	Qty.	Order no.	Note
STEP 7 Basic V11 Update 2 – SP2	1	6ES7822-0AA01-0YA0	Or higher
HSP CP 1242-7 V1.0		-	Necessary only for STEP 7 V11, See \4\

Sample files and projects

The following list contains all files and projects that are used in this example.

Table 2-4

Component	Note
58638283_Set32_SMS_CODE_V10.zip	<This zip file contains the STEP 7 project.>

2.4 Performance data

Memory requirement

The figure below shows the resource consumption using the example of a CPU 1211C.

Figure 2-3

58638283_Set32_SMS_CODE_V10 > SMS_Station [CPU 1211C DC/DC/DC] > Program block				
Resources of SMS_Station				
	Objects	Load memory	Work memory	Retentive memory
1		28.55%	87.75%	0%
2				
3	Total:	1 MB	25600 bytes	2048 bytes
4	Used:	299374 bytes	22465 bytes	0 bytes
5	Details			
6	▶ OB	8779 bytes	948 bytes	
7	▶ FC	40461 bytes	5762 bytes	
8	▶ FB	163861 bytes	10865 bytes	
9	▶ DB	78981 bytes	5090 bytes	0 bytes
10	▶ Data types	7292 bytes	-	
11	PLC tags			0 bytes

For detailed information on the user memory size for all 1200 CPU models, please refer to manual S7-1200 Programmable Controller System Manual → *Table 1-1 Comparing the CPU models*

3 Basics

3.1 GMS network

This section explains the basic principle of operation of sending SMS messages within the GSM network. The description of the processes is greatly simplified.

The most important parties

Mobile Switching Center – MSC

Each MSC manages a large number of cells. They are the central nodes of the GSM network. The most important tasks are to

- manage the dial-up to the network.
- establish the connection to the selected phone number.
- forward SMS messages.

Each connection runs through at least one MSC – even if both subscribers are within one cell.

Home Location Register – HLR

The central database of the system – contains information on all registered subscribers. This information includes:

- Authentication of the terminal units during dial-up.
- All services that are enabled/locked on a SIM card.
- The current location information of the subscribers – thus via which MSC they can currently be reached.

No payload (e.g., language) is transmitted via the connections to the HLR.

Short Message Service Center – SMSC

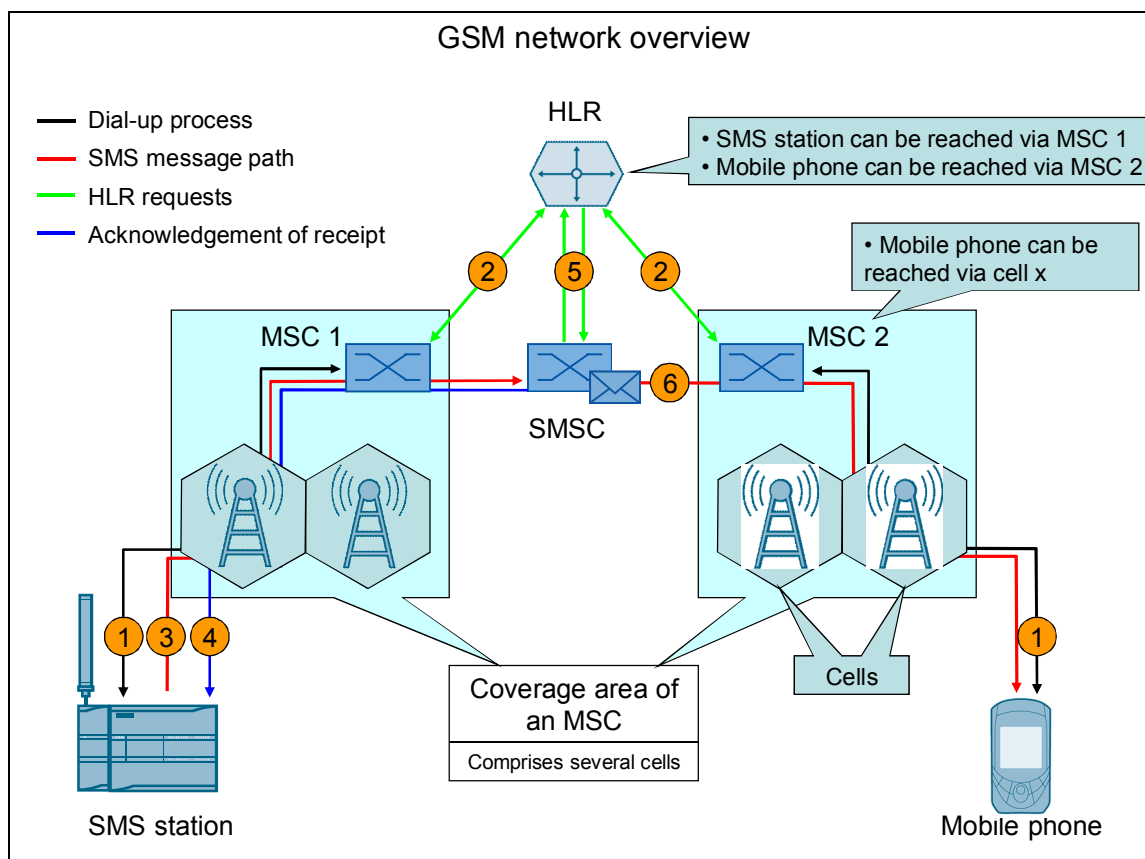
Classified hierarchically in the network at the MSC level. The SMSC does not manage cells itself, it is only used to forward SMS messages.

The most important tasks are to

- confirm the arrival of a message to the requesting MSC.
- determine the current MSC of the desired phone number at the HLR.
- deliver SMS messages.

If the recipient's device is not logged in to the network, the SMSC will save the message. The next time the subscriber dials up, the MSC sends a signal to the SMSC, which is followed by the delivery.

Figure 3-1



SMS message: Send operation in steps

Table 3-1 Explanation of connection establishment

No.	Steps in the functional sequence
1.	The terminal units signal to the responsible MSC that they want to dial up. The units transfer this information to their SIM cards.
2.	With the information of the SIM card, the MSC sends a request to the HLR. If the authentication was successful, the MSC establishes a connection to the terminal unit. Via which MSC the terminal unit can be accessed is stored in the HLR.
3.	The <i>SMS station</i> transmits the message to the current MSC (1). The MSC forwards the contents exactly to the SMSC.
4.	The SMSC confirms the arrival of the message; from this information, it is not yet possible to see if delivery was successful.
5.	The SMSC asks the HLR to which MSC the SIM card of the <i>mobile phone</i> is logged on. If an MSC could not be determined – for example, because the mobile phone is turned off – the message will be stored.
6.	The SMSC transmits the message to the previously determined MSC 2. From there, it is forwarded to the <i>mobile phone</i> . Once the message has been successfully delivered, it is deleted from the SMSC.
	Optionally, the sender receives feedback about successful delivery. However, this function is not a GSM standard function and thus depends on the SMSC provider.

3.2 Characteristic features of the CP device configuration

Authorized phone numbers

The CP accepts only messages from subscribers that are made known to it via the device configuration. This filter is effective **before** the receive block has the opportunity to access the messages. Therefore, messages whose sender number is unknown never become visible in the CPU program. A wildcard * can be used to allow all phone numbers.

SMSC

If messages are to be sent, the provider's SMS center must be entered in the device configuration of the CP. This is **not necessary** in order to **receive** messages.

CP phone number

Does **not have to be configured** for sending/receiving messages.

Background: Remote access to the station via teleservice

When using this function, TIA Portal must know to which phone number the wake-up SMS message is to be sent. Therefore, the phone number must be stored in the project. The field has **no effect whatsoever** on the **SMS functions**.

3.3 Definition of the connection-specific characteristics

Introduction

This chapter explains how the different connection types are defined and where the connection type for sending messages is classified.

Overview of the characteristics of a connection

The following characteristics define the function of the telecontrol system.

Table 3-2

Parameter	Possible values for the parameter	Remark
Operating mode	<ul style="list-style-type: none"> • Telecontrol • GPRS direct 	Set directly in the device configuration and Telecontrol Server Basic. In the following referred to as main connection .
Connection mode	<ul style="list-style-type: none"> • Permanent • Temporary 	
Connection type	<ul style="list-style-type: none"> • Telecontrol connection • UDP • ISO on TCP • SMS • Teleservice 	Programmed in the user program with the aid of the library blocks. In the following referred to as sub-connection . A connection is always reserved for the Teleservice connection type. It does not have to be programmed separately.
Connection parameters	Active/passive connection establishment, connection ID, information on the connection partner	

Definition of main connection

The main connection is defined by the selection of relevant parameters in the device configuration for the CP. This application example does not go into the details of GRPS communication. Therefore, the connection type used is not relevant. Communication via SMS can equally be used in “Telecontrol” and “GPRS direct” mode.

Definition of sub-connection

For the sub-connection, several **connection types** are available for selection that have already been defined by the selection of the main connection.

The desired connection type is programmed directly in the user program with the aid of the library blocks.

A sub-connection with the “SMS” connection type is selected in this application example.

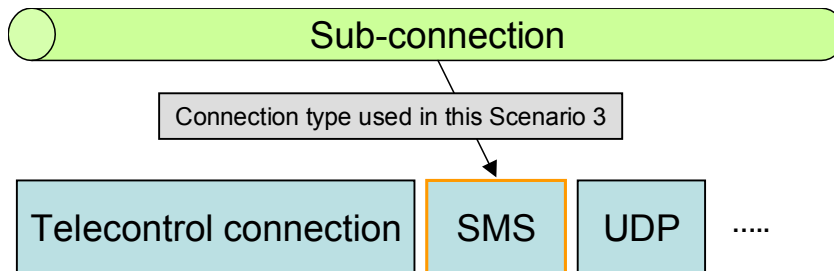
For a more detailed description of the different connection types (SDTs), please refer to document \1\, Chapter 5.4.7.

3.4 Establishing the connection

Connection type in this example

In this application example, the SMS connection type is selected for the sub-connections to exchange messages with any mobile phones.

Figure 3-2



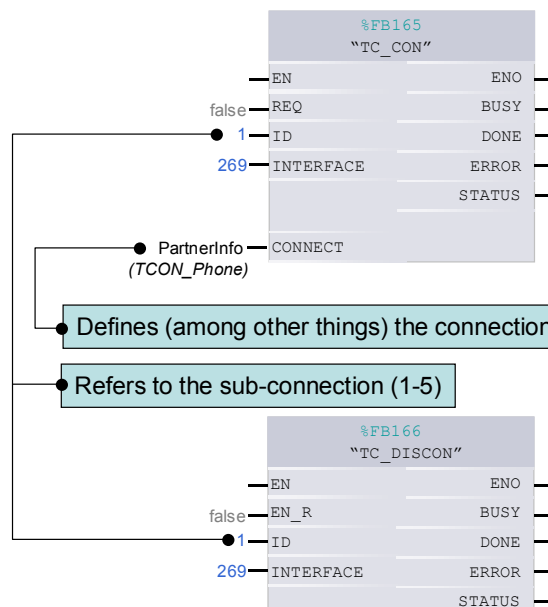
Establishing/terminating the connection

In STEP 7 V11, library blocks are available to control the sub-connection.

From this library, the "TC_CON" instruction is used to establish a sub-connection and the "TC_DISCON" instruction is used to terminate a sub-connection. The information which **connection type** is selected is specified as the "CONNECT" parameter (SDT) on "TC_CON".

With the aid of the ID, all other instructions of the "TC_DISCON", "TC_SEND" and "TC_RECV" library blocks refer to this connection type or this sub-connection.

Figure 3-3 Call of "TC_CON" and "TC_DISCON" to control the sub-connections



The “TCON_phone” SDT

To connect to the GSM network, the TC_CON block must be called with the “TCON_Phone” SDT at the CONNECT input. The SDT contains parameters for the connection and its composition is as follows:

Table 3-3

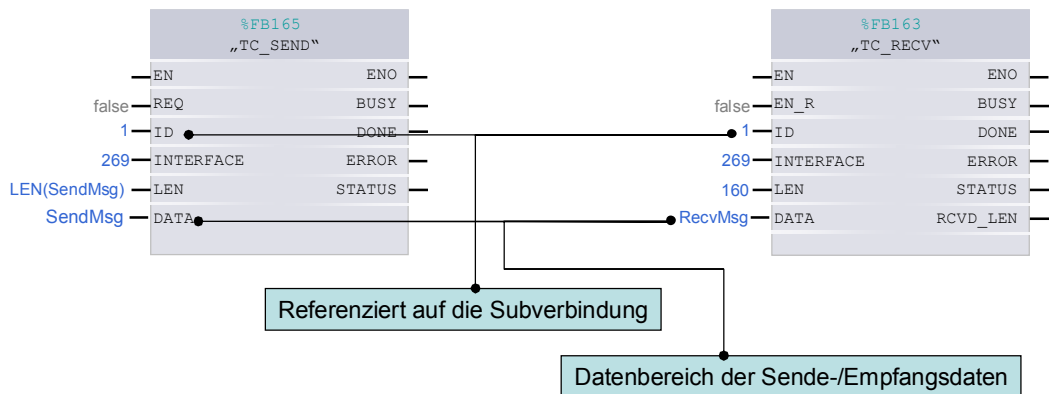
Name	Data type	Comment	Example
Interfaceld	HW_ANY	Hardware ID of the addressed CP.	269/270 Depending on CPU and slot of the CP
ID	CONN_OUC	Connection ID with which the remaining TC blocks address the connection.	1 Must be unique within the project. (1 not on two TCON blocks)
ConnectionType	Byte	Default: <i>W#16#0E</i> – designates the SMS sub-connection type	W#16#0E No changes necessary.
ActiveEstablished	Bool	Default: <i>False</i> – not relevant for SMS connections	False No changes necessary.
Phonenumber	String[22]	Phone number of the communication partner. When receiving, an empty string stands in place of all numbers.	'+49123456' For sending ” For receiving all phone numbers

3.5 Transmitting process data via a sub-connection

Send/receive blocks

To control process data traffic, blocks are available for STEP 7 V11 that are downloaded for the CP with the "Hardware Support Package". From these blocks, the "TC_SEND" instruction for sending and the "TC_RECV" instruction for receiving process data are used via the respective sub-connection.

Figure 3-4 Call of "TC_SEND" and "TC_RECV" to control process data traffic



Note

The blocks are handled in the same way as in Open User Communication in the S7-1200 (TSEND, TRECVC).

Characteristic features of the SMS connection

- The length (LEN) of the data area (DATA) is limited to max. 160 characters (7-bit coded). For other encodings, **8-bit / 16-bit**, it is reduced to **140 / 70** characters.
- Separate connections for TC_SEND/TC_RECV: As a rule, it is to be possible to receive messages from any phone number. For this purpose, TC_CON must connect to a blank phone number, which makes TC_SEND calls to this connection useless. In this case, it is essential that a connection be set up only for TC_RECV.

The String data type

It is useful to store the received characters in the String data type. This data type stores the maximum and the current length of the string in the first two bytes. Therefore, the actual size in the memory is:

Number of characters + 2 bytes of length information

Table 3-4

Text String[10] = 'Hello'										
Byte following start address	1	2	3	4	5	6	7	8	9	10
Contents	8	5	H	e	l	l	o	-	-	-
Corresponds to	Maximum	Current	Actual text					Not relevant		

This structure is important for the following reason:

If the string variable was applied directly to the DATA input of the blocks, the length information would be unintentionally included as characters of the string.

Examples:

1. TC_RECV receives the "Hello" text. The block will enter the value of the letters "H" and "e" in *Maximum* and *Current*. "llo" will be in the actual text. Incorrect lengths can cause software errors.
2. The "Hello" string is applied to TC_SEND. The block will convert the length information 8 and 5 to characters and include them in the sent message text. In most cases, the result is an unwanted special character at the start of the message. For example:  Hello

To prevent this behavior, the DATA parameter must start two bytes to the right of the string's start address:

Text string [0..159] starts at DB1.DBX 0.0

DB1.DBX 2.0 Byte 160 is applied to DATA.

Alternatively, the overlay of the string with **AT** can be used. This creates additional symbolic names via which the same memory area can be accessed. Therefore, the actual **AT** construct does not require memory, it only provides a different view of the data.

This variant is preferable as addressing with names is still possible. There is no need to manually adjust the pointers when there are changes of the start address.

Table 3-5

Text_Struct <AT Struct>	
Dummy <WORD>	Text <Array of Char [0..159]>
Wildcard	Text field

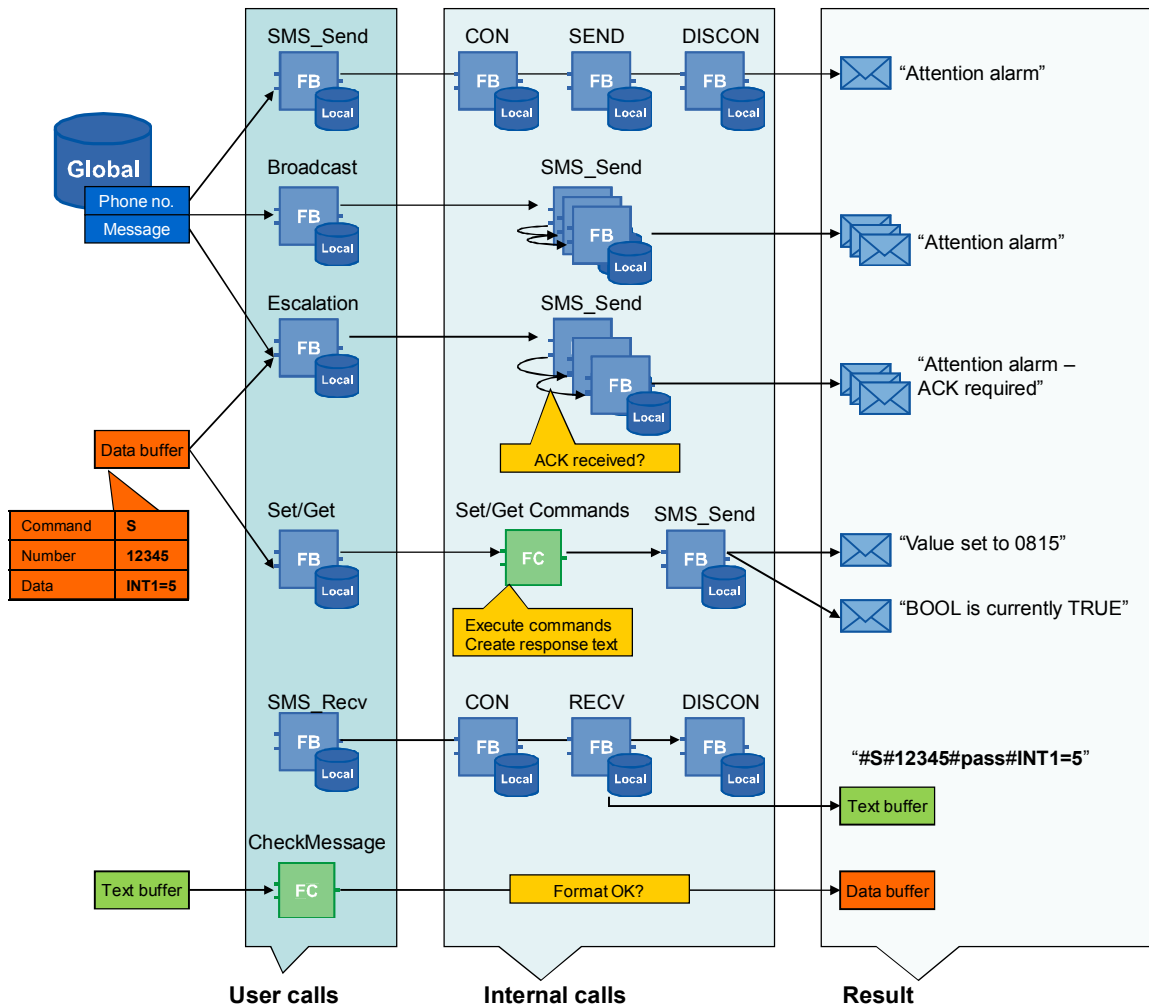
The text can be accessed separately via the array within the structure. Here, **Dummy** is only used to skip the first bytes.

Text_Struct.Text is applied to the DATA parameter.

4 Functional Mechanisms of this Application

Program overview

Figure 4-1



All blocks can be used individually.

The *"Escalation"* and *"Set/Get"* routines necessarily require an *"SMS_Recv"* block in order to detect incoming acknowledgements/commands.

For the *SMS_Recv* block, there is an additional restriction – see 4.1.2.

4.1 Functionality

4.1.1 Basic blocks

SMS_Send

As parameters, the SMS_Send block receives a phone number (*Phonenumber*) and a message text (*MessageText*). A send job is started with each positive edge.

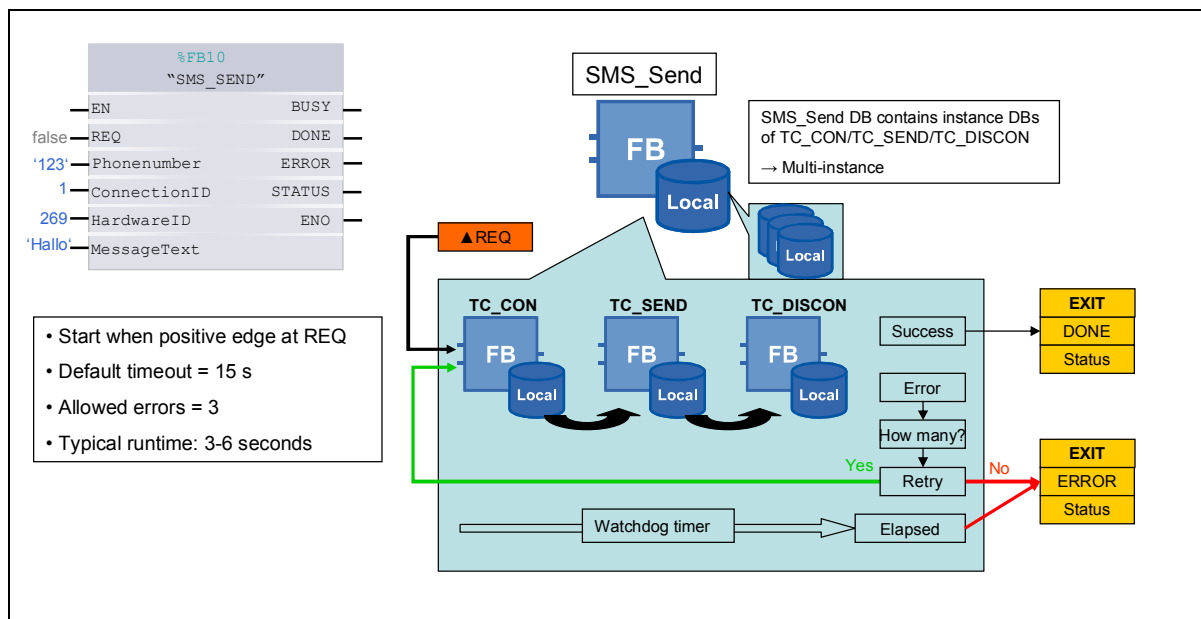
Internally, the required TC_blocks are initialized with the created parameters and called.

SMS_Send terminates when:

- The TC_blocks abort with an error more than three times (Error bit of CON, SEND or DISCON).
- The timeout time of 15 seconds has been reached.

The average runtime is between 3 and 6 seconds.

Figure 4-2 SMS_Send program flow



4 Functional Mechanisms of this Application

4.1 Functionality

SMS_Recv

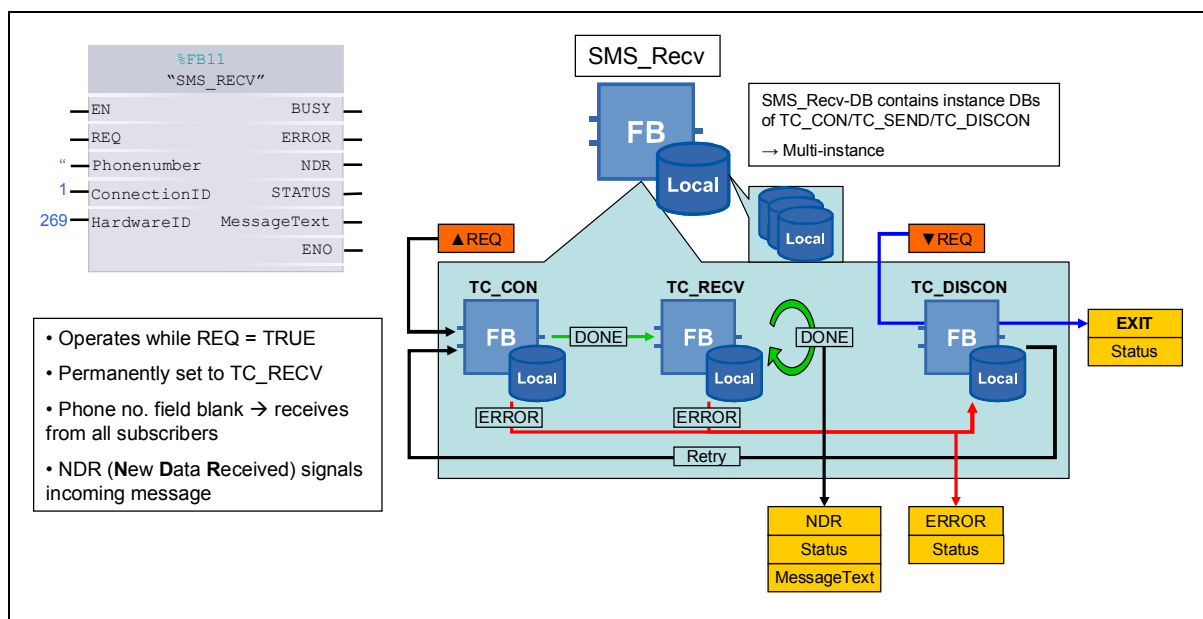
The SMS_Recv block too receives a phone number as a parameter. In addition, a data area is declared as an output in which it stores received messages (*text buffer*). Each new message is signaled at the **NDR (New Data Received)** output.

Internally, the required TC_blocks are initialized with the created parameters and called. If no errors occur, **TC_Recv** is **permanently** active and SMS_RECV is thus ready to receive messages.

SMS_Recv terminates only if a negative edge arrives at the REQ input. If an error occurs, it will be permanently attempted to reestablish the connection.

With an **empty string** as the phone number, **all messages** are received on this block.

Figure 4-3 SMS_Recv program flow



Note

Messages will only be received if the SMS_Recv / TC_RECV block is active when they arrive. Otherwise, the message will be lost – subsequent searching of the memory is not possible.

The sender's phone number of incoming messages cannot be retrieved on the TC_blocks. If there is to be a response, the phone number must be included in the message text.

4.1.2 Related blocks

Global use of SMS_Recv

It is assumed that messages are to be received from all phone numbers. For this purpose, the connection on which TC_RECV operates must be established to a blank phone number. Such a connection can only be used once per CP. Otherwise, the message will always arrive on the TC_RECV that is called first in the program structure. All other TC_RECV will never receive messages.

Therefore, the application uses a global SMS_RECV block. This block writes the text to a text buffer and signals new messages by means of the *NDR* output bit.

The following applies to all blocks that want to react to an incoming message:

- They consider only the text buffer of the global *SMS_Recv* and do not call other *SMS_Recv* instances.
- They leave this text buffer unchanged.
- They react to the *NDR* bit of *SMS_Recv*.

The text buffer includes the message text. The check for the format – required by the “Escalation” and “Set/Get” blocks – is performed in another step in the *CheckMessage* function.

In the application example, *CheckMessage* is the only function that accesses the text buffer.

Note

Alternatively, different TC_TCON blocks could be used for a single phone number. This severely limits the number of subscribers as the connection resources are limited to max. 5 and all of these connections must always be active.

CheckMessage

For the “Escalation” and “Set/Get” blocks, a fixed format is specified for the incoming message text to facilitate processing. This function checks the message text for compliance with the format and password validity. In addition, the *Phonenumber* substring is limited to the characters “+1234567890”.

4.1 Functionality

Table 4-1

Message format for Escalation and Set/Get							
#	Command	#	Phone number	#	Password	#	Data
	Char		String[22]		String[16]		String

- The system uses “#” as separators between the substrings. These separators must be included.
- The block reacts only to *message texts with leading “#”*. If there is no leading “#”, there will be **no** processing.
- The **commands** are always converted to **capital letters**. Therefore, **S** and **s** have the same effect.
Default selections: **S**: Set value **G**: Get value **Q**: Acknowledge alarm
- A blank password field is permitted. This ensures that all messages are accepted – irrespective of what was sent as the password.
- Messages with an invalid password are not transferred to the UDT at the output.

When successful, the individual substrings are output in an “**SMS_Data**” UDT (User-defined Type). The following blocks such as Set/Get continue to operate with this UDT.

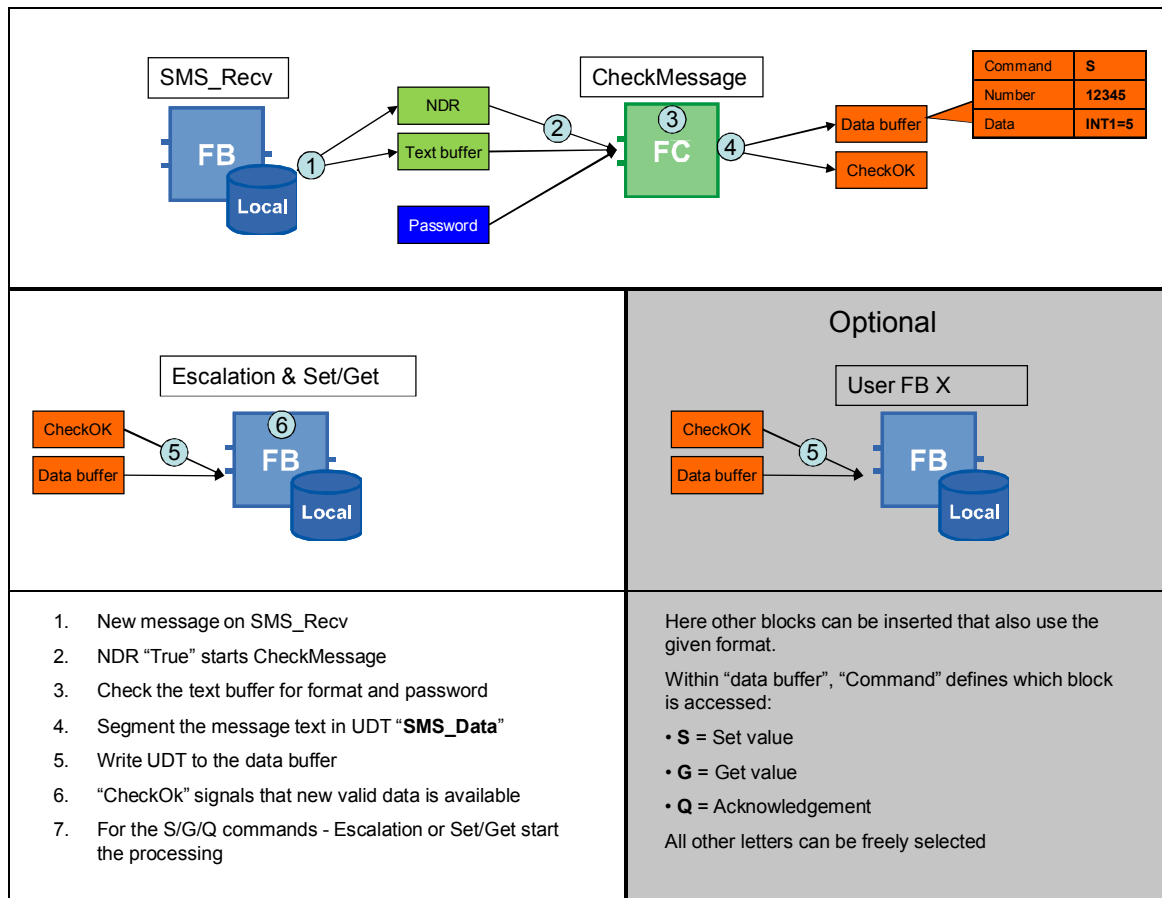
The password is prompted before further processing. Therefore, the password is the same for “*Set/Get*”, “*Escalation*” and self-created user blocks – that want to operate with the “SMS_Data” UDT.

Examples:

#S#12345#pass#INT1=5	→	OK
#S#12345##INT1=5	→	OK if password = ”
S#12345#pass#INT1=5	→	Will not be processed
#S#12345pass#INT1=5	→	Error, only 3 # included
##12345pass#INT1=5	→	Error, no command
#S#12abc345##INT1=5	→	Error, letter in phone number

Overview

Figure 4-4 Global SMS_Recv and CheckMessage program flow



SMS_Broadcast

As parameters, the SMS_Broadcast block receives several phone numbers (*Phonenumber*s) and one message text. A send job to all phone numbers is started with each positive edge. Internally, SMS_Send calls are successively performed with each phone number from the array.

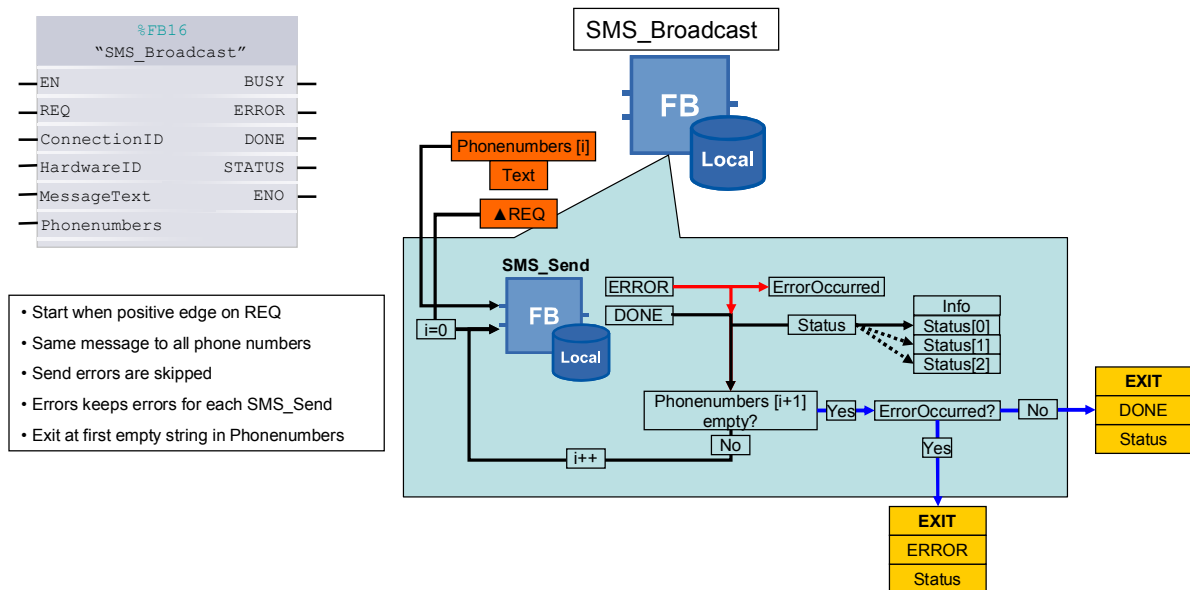
If an error occurs in the SMS_Send call, the block continues with the next phone number. The block stores the result (DWORD status) of each SMS_Send call in the *Errors* array. This enables the user to finally determine which messages were sent and which errors have occurred.

The *Errors* parameter keeps the values until the next job is started.

4 Functional Mechanisms of this Application

4.1 Functionality

Figure 4-5 SMS_Broadcast program flow



Escalation

As parameters, the Escalation block receives an array of phone numbers, a *MessageText* and the data buffer of the CheckMessage block. The sequence is started with a positive edge. Once a message has been successfully sent, the block goes to a wait state. If an acknowledgement with the correct content arrives during this period, the block terminates. Otherwise, the next phone number is contacted after the period has elapsed. If an error occurs on SMS_Send, the phone number will be skipped.

By default, the block also expects the sent *MessageText* to return as an acknowledgement in the data field.

When sending, the block automatically converts the SMS message to the defined format. For security reasons, the password is not included. For acknowledgement, it is only necessary to insert the password (if set) and send the message back to the phone number of the CP.

If the password is disabled, the message can be returned to the CP with the exact same contents.

If none of the phone numbers sends an appropriate acknowledgement, the block will return an error code.

Sample parameters:

MessageText = "Alarm 101"

Phonenumbers[1] = "12345"

Password (An CheckMessage) = "pass"

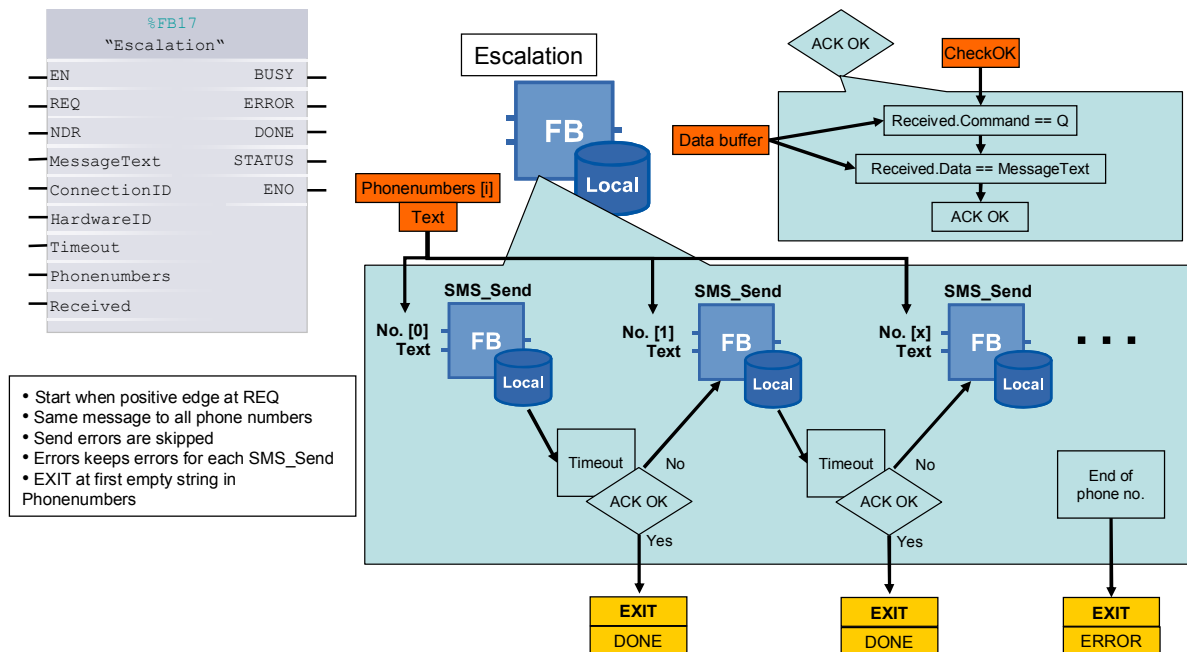
Outgoing alarm message: #Q#12345##Alarm 101

Expected response: #Q#12345#pass#Alarm 101

With *Password* = " (blank)

Expected response: #Q#12345##Alarm 101 (→ corresponds to an outgoing SMS message)

Figure 4-6 Escalation program flow



Set/Get

As a parameter, the Set/Get block receives information on a received message ("SMS_Data" Received UDT). Processing starts with a positive edge at NDR (**New Data Received**). The block searches the data field of the message for known commands – this step is performed in the relocated SCL function Set/Get Commands. The respective response is sent back to the phone number depending on the return value of the function.

In addition, the block has **three** memory locations for incoming "SMS_Data" data records. When a new request arrives while the block is BUSY, it will be put into a memory location and then processed. Without this mechanism, one request would always be lost – if there are simultaneous requests.

Set/Get Commands

This function includes all commands known to the system. This applies to both setting and getting values from the process. The distinction is made based on the transferred Command parameter from the received message.

Then the *FIND* string function is used to search the data field of the message for known commands. If one of the FIND calls was successful, the respective action will be performed and the message text will be returned. Depending on the use case, all numerical values must be converted either to or from a string.

Examples with comments are stored in the Set/Get Commands function and can be customized with little effort.

4 Functional Mechanisms of this Application

4.1 Functionality

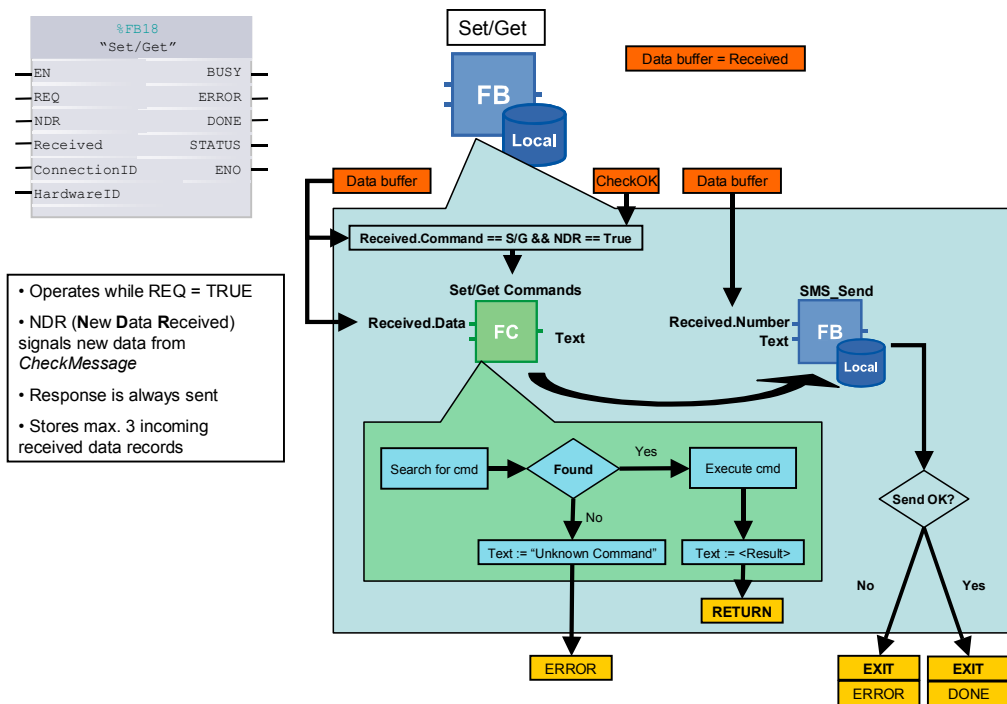
Note

The FIND string function is **case sensitive** – it distinguishes between uppercase and lowercase letters.

→ Int1=5 ≠ INT1=5

In this case, the block does not find the command and returns “*Unknown Command*”.

Figure 4-7 Set/Get & Set/GetCommands program flow



4.1.3 Interfaces

Where possible, the blocks behave similarly to the communication blocks from the library. The interfaces too largely follow this pattern (REQ, BUSY, etc.).

Frequent parameters

Table 4-2 Input

Name	Type	Info
REQ	Bool	Start of processing. Reacts to a positive edge in all send jobs. For SMS_Recv and Set/Get , the block operates only as long as Req is set to TRUE.
ConnectionID	CONN_OUC	The user can freely select the connection resource used. However, it must be ensured that one ID is not simultaneously used by two blocks.
HardwareID	HW_ANY	The hardware ID of the CP used for communication. Automatically set in the device configuration.
Phonenumber	String[22]	The phone number to which a message is sent.

Table 4-3 InOut

Name	Type	Info
MessageText	String[160]	The message contents.
Phonenumbers	Array[0..10] of String[22]	A list of 10 phone numbers that are successively contacted. Used for Broadcast and Escalation.

Note

These parameters are not changed. The declaration as InOut only reduces the memory requirement.

Table 4-4 Output

Name	Type	Info
DONE	Bool	If the block was processed successfully, set to TRUE for one cycle.
ERROR	Bool	If an error occurs, set to TRUE for one cycle. In this case, Status provides information on the error.
BUSY	Bool	Set to TRUE as long as a job is still being processed.
NDR	Bool	Signals the arrival of new data on receive blocks.
Status	DWord	Divided into two bytes and one word. Provides information on the current processing status of the block. If an error occurs, the status of the TC_block or SMS_block causing the error is exactly stored in the word. The two bytes provide additional information. Details can be found in the respective block comment.

4.1.4 Status codes of the blocks

Structure of function blocks

All function blocks return a DWord as status that provides details on the block status.

Table 4-5

Status		
Byte	Byte	Word
Info	Source	Status Word

Table 4-6

Meaning	
Info	Additional information, varies depending on the block.
Source	<p>If an error occurs, the ID of the TC_block causing the error:</p> <ul style="list-style-type: none"> • 01: TC_CON • 02: TC_SEND • 03: TC_RECV • 04: TC_DISCON • 00: Message of the actual FB
StatusWord	<p>If an error occurs, the status of the TC_block causing the error: When Source = 00, message of the actual FB:</p> <ul style="list-style-type: none"> • 7000: Ready – job can be started • 7001: Being processed (BUSY) • 7002: New job request while BUSY • 0000: Job successfully completed (DONE)

Note

In the event of an error, the **first** error that has occurred is always reported.

If – due to an incorrect HW ID – TC_CON and then TC_DISCON fail, the error of the TC_CON block is provided on the StatusWord.

The possible values and their meaning can be found in the comment of the respective block title. The “Info” parameter is partially replaced by the related blocks.

Example: Broadcast

The phone number at position[2] contains an invalid character.

SMS_Send will make three attempts and provide **820280EF** on the status (together with the set ERROR bit)

Table 4-7

Info: 82	Source: 02	Status: 80EF
Number of retries reached	TC_Send failed	Sending was not possible. See TC_Send help

Broadcast, for its part, will enter **820280EF** on Errors[2] and continue processing. At this moment, it does not provide an ERROR bit. It will signal, by **00008000**, that not all subscribers could be reached not before the end of the phone number.

Structure of CheckMessage

The function has five possible return values:

Status	Meaning	Remark
0	Valid format.	
1	Incorrect number of #.	Must include exactly 4.
2	No command letter between the first two #.	Must contain one letter, which one will not be checked. (for example for S/G/Q).
3	The phone number contains invalid characters.	The following characters are permitted: +0123456789
4	The password is incorrect.	Case sensitive.

4.2 Extensions and adaptations

Increasing/reducing the size of the Phonenumbers array

By default, the Phonenumbers array can contain 10 phone numbers. If you need more or less phone numbers:

1. Open the *SMS_Broadcast* or *Escalation* FB.
2. In the interfaces, change to **InOut**.
3. Adjust the data type of **Phonenumbers** to the desired size: *Array[0..x] of String 22*.
4. Update existing FB calls.
5. Change the data area created on the call also to the length [0..x] so that the data types match.

As the block terminates with the first blank element and does not consider the length, no further changes are necessary.

Filtering blanks in the message text

It may be advisable to remove included blanks from the message text. The basic procedure has already been implemented in FC12 CheckMessage – however, it is disabled by default.

To enable the function, select lines 119-126 and select the “Enable Code” button.

→ With a similar loop, for example, also the outgoing messages in *Set/Get Commands* can be freed from unwanted blanks. These are frequently inserted when executing the conversion functions such as *INT_TO_STRING*.

Note

For the Escalation FB, blanks must not be removed as it expects the outgoing text to be returned with the exact same contents. If the blanks are missing, the Escalation FB will not detect the message as a valid acknowledgement and will continue to operate.

Set/Get commands need more than one cycle

The examples assume that the commands can be processed within one cycle. Otherwise, you have to customize the Set/Get FB.

1. Navigate to network 4 of the Set/Get FB:
→ **MOVE 2 to #Step** is called immediately after Set/Get Commands.
2. Adapt the condition for the MOVE command to your use case.

Workaround for e-mails

Currently, the CP does not natively support sending e-mails. However, many providers offer an **SMS2Mail** function. To use it, send the message to a provider-specific phone number and specify the desired e-mail address in the message text.

Different message texts for Broadcast

By default, the block sends the same message text to all phone numbers. If the message text is to vary for each phone number, proceed as follows:

1. Open the *SMS_Broadcast* FB.
2. In the interfaces, change to **InOut**.
3. Change the data type on **MessageText** from “*String*” to “*Array[0..10] of String*”.
4. In network 2, change the parameter on *SMS_Send* from *#MessageText* to *#MessageText[i]* (i is the index variable of the array).
5. Update existing FB calls.
6. Change the data area created on the call also to the “*Array[0..10] of String*” type so that the data types match.

Setting the password

By default, the password function is disabled (blank password). To enable the function, proceed as follows:

1. Open OB1.
2. Navigate to the *CheckMessage* call in network 2.
3. Create the desired string on the “Password” parameter.

Note

The password prompt is now globally active. All incoming messages – that comply with the format – are checked for it.

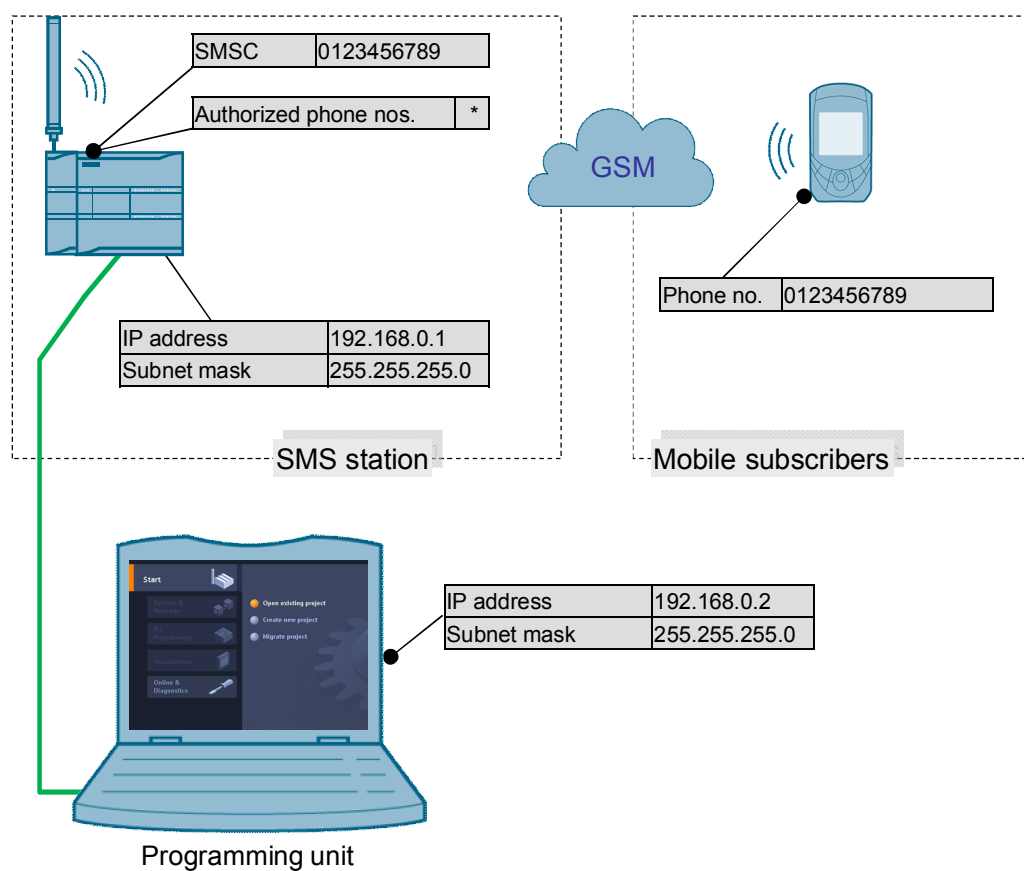
5 Commissioning of the Application

5.1 Installing and wiring the hardware

Network plan

The figure below shows all network-relevant information you need to interconnect all components.

Figure 5-1 Network plan: Addresses and phone numbers



Installing the hardware

Table 5-1

No.	Action	Remark
1	Mount all required components of the SMS station on the top-hat rail.	<ul style="list-style-type: none"> SMS station: List of components 2.3
2	Wire and connect all required components for the SMS station as described. Insert the SIM card into the CP. Make sure that the ground connections of the components are correct and do not activate the power supply for the SIMATIC PM 1207 before the end.	<ul style="list-style-type: none"> SMS: Configuration display, Chapter 2.1 To avoid invalid PIN entries by the CP, you should make sure that no old configuration is loaded to the CPU.

Note CP and CPU only start correctly if the power supply of both devices is switched on simultaneously.

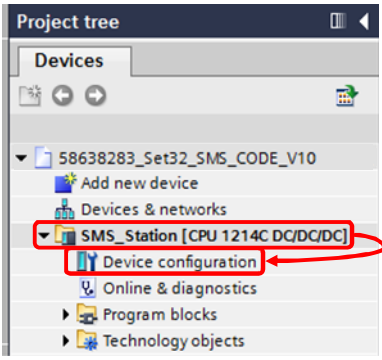
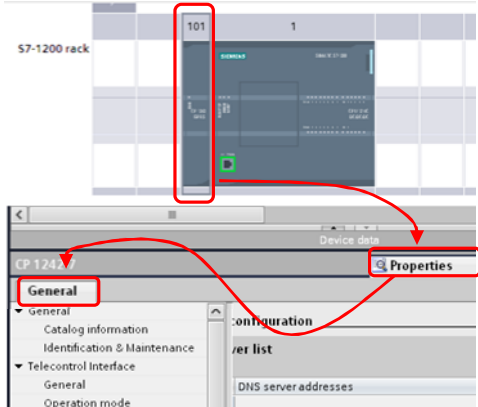
→ It is recommended that the power supply of the CP be connected to the CPU via the 24V terminals with internal bridging.

Note Do not insert or remove the SIM card while the CP is in operation.

5.2 Configuration instructions

Configuring the SMS station

Table 5-2

No.	Action	Remark
1.	Install the Hardware Support Package V1.0 of the CP for STEP7 V11.	Download at: \4\ Installation guide: → Chapter 3.5
2.	Network the S7-1200 controller with your programming device. Assign the Ethernet parameters shown in Figure 5-1.	Assign IP address to S7-1200: S7-1200 Programmable Controller System Manual → Chapter 5.6.4
3.	Open the project file (ap11) with STEP 7 V11.	Table 2-4
4.	Open the device configuration of the “PLC_1” controller.	
5.	Select the CP and open the “Properties” tab to enter the connection parameters. Assign the parameters as explained in the following steps. For a complete description of the parameters, please refer to document \1\, Chapter 5.2.	

5 Commissioning of the Application

5.3 Error handling

No.	Action	Remark
6.	Enter the PIN of the SIM card that is inserted in the modem.	Telecontrol interface >> Modem settings >> <i>PIN</i> , <i>confirm PIN</i> and <i>activate PIN</i>
7.	The short message service center of the SIM card provider. The phone number of the short message service center can be obtained from your wireless service provider.	Telecontrol interface >> Modem settings >> <i>Enable SMS</i> und <i>SMSC</i>
8.	The phone numbers of the mobile subscribers from which messages are accepted. Enter the desired phone numbers individually or use * to receive phone numbers from all mobile subscribers.	Telecontrol interface >> Authorized phone numbers >> <i>Phone number 1-10</i>
9.	Check the hardware ID of the CP. It must be created as the HardwareID input parameter for all blocks to uniquely address the CP.	Telecontrol interface >> Hardware identifier >> Hardware identifier
10.	In <i>Data_Blocks</i> , open Send[DB2] and insert the phone numbers to which messages are to be sent.	Phonenumber for the SMS_Send block. Phonenumbers for the Escalation and Set/Get blocks.
11.	Save the project. Select the program folder of the S7-1200 and transfer the program to the controller via "Online/Download to device". Make sure that the LED of the S7-1200 controller indicates the "RUN" status.	Online >> Download to device

Note

In order to use the project in STEP 7 V12 you need to upgrade it. Go to "STEP 7 V12> Project view> Project> Upgrade".

Note

The CPUs of the S7 1200 series apply changes to **DB start values** only during the next **STOP-RUN** transition.

Please consider this behavior when making changes to the data blocks.

5.3 Error handling

TC_blocks

You can retrieve the status messages of the library blocks directly in the help function of STEP 7 V11. To do so, select the relevant TC_block in the user program and press F1.

SMS_blocks

The general structure of the Status parameter is described in the following chapter: *Status codes of the blocks*. Details on the individual codes can be found in the comment field of the block header.

6 Operation of the Application

Inputs

The individual blocks are connected to digital inputs with which you can start the respective function:

Table 6-1

Input	Block	Remark
I 0.0	SMS_Recv	REQ must be permanently set to TRUE
I 0.1	SMS_Send	Start when there is a positive edge at REQ
I 0.2	SMS_Broadcast	Start when there is a positive edge at REQ
I 0.3	Escalation	Start when there is a positive edge at REQ
I 0.4	Set/Get	REQ must be permanently set to TRUE

Note

To avoid unwanted costs, the sample program does not start any automatic sending of messages in the as-supplied state.

Watch table

To monitor the sequence, the “**Overview**” symbol table has already been created in the project. It provides you with an overview of different program parameters:

- Process values simulated by the program
- Contents of the send and receive buffers
- The status of the block (here only **BUSY**)
- For example, the *Status* outputs or the *ERROR/DONE* bits can be added for more details.

6.1 Overview

Figure 6-1

i	Name	Address	Display format	Monitor value
1	"Generate_Simulated_Values_DB".BottlesSlot1	%DB3.DBW10	DEC_signed	98
2	"Generate_Simulated_Values_DB".BottlesSlot2	%DB3.DBW12	DEC_signed	97
3	"Generate_Simulated_Values_DB".BottlesSlot3	%DB3.DBW14	DEC_signed	96
4	"Generate_Simulated_Values_DB".DesiredTemp	%DB3.DB04	Floating-point number	20.0
5	"Generate_Simulated_Values_DB".Temperature	%DB3.DB00	Floating-point number	0.2
6	"Receive_Enabled"	%I0.0	Bool	<input type="checkbox"/> FALSE
7	"SMS_Recv_DB".BUSY	%DB10.DBX30.0	Bool	<input type="checkbox"/> FALSE
8	"Receive".TextBuffer	P#DB1.DBX0.0	String	" "
9	"Receive".DataBufer.Command	%DB1.DB8348	Character	" "
10	"Receive".DataBufer.Data	P#DB1.DBX162.0	String	" "
11	"Receive".DataBufer.Number	P#DB1.DBX324.0	String	" "
12	"Start_Send"	%I0.1	Bool	<input type="checkbox"/> FALSE
13	"SMS_Send_DB".BUSY	%DB11.DBX30.0	Bool	<input type="checkbox"/> FALSE
14	"Send".MessageText_Send	P#DB2.DBX288.0	String	" "
15	"Start_Broadcast"	%I0.2	Bool	<input type="checkbox"/> FALSE
16	"SMS_Broadcast_DB".BUSY	%DB12.DBX6.0	Bool	<input type="checkbox"/> FALSE
17	"Send".MessageText_Broadcast	P#DB2.DBX450.0	String	" "
18	"Start_Escalation"	%I0.3	Bool	<input type="checkbox"/> FALSE
19	"Escalation_DB".BUSY	%DB13.DBX172.0	Bool	<input type="checkbox"/> FALSE
20	"Escalation_DB".AckWait.PT	%DB13.DB0200	Time	T#0MS
21	"Escalation_DB".AckWait.ET	%DB13.DB0204	Time	T#0MS
22	"Send".MessageText_Escalation	P#DB2.DBX612.0	String	" "
23	"Set/Get_Enabled"	%I0.4	Bool	<input type="checkbox"/> FALSE
24	"Set/Get_DB".BUSY	%DB14.DBX194.0	Bool	<input type="checkbox"/> FALSE
25	"Set/Get_DB".SendMsg	P#DB14.DBX206.0	String	" "
26	"SMS_Broadcast_DB".Errors[0]	%DB12.DB030	Hex	16#0000_0000
27	"SMS_Broadcast_DB".Errors[1]	%DB12.DB034	Hex	16#0000_0000
28	"SMS_Broadcast_DB".Errors[2]	%DB12.DB038	Hex	16#0000_0000
29	"Send".Phonenumbers[0]	P#DB2.DBX24.0	String	"0173"
30	"Send".Phonenumbers[1]	P#DB2.DBX48.0	String	'Errorrest'
31	"Send".Phonenumbers[2]	P#DB2.DBX72.0	String	" "


Table 6-2

No.	Description
1.	Three simulated values for the number of bottles per slot. Start at 100 and count to 0 with different clock memory bits. If all three are 0, they will be reset to the initial value 100.
2.	The simulated temperature with actual value and setpoint. If there is a difference, the actual value adapts to the specified setpoint in steps of 0.1.
3.	Overview of the global SMS_Recv block and the associated buffers.
4.	Overview of SMS_Send and SMS_Broadcast.
5.	Overview of Escalation, including timer status for wait time.
6.	Overview of Set/Get.
7.	Overview of the status messages in the SMS_Broadcast block and the Phonenumbers array.

6.1.1 SMS_Send

Figure 6-2

12	1	"Start_Send"	%I0.1	Bool	<input checked="" type="checkbox"/> TRUE
13		"SMS_Send_DB".BUSY	%DB11.DBX30.0	Bool	<input checked="" type="checkbox"/> TRUE
14		"Send".MessageText_Send	P#DB2.DBX288.0	String	'Alarm: Slot nearly out of Bottles'



12	2	"Start_Send"	%I0.1	Bool	<input checked="" type="checkbox"/> TRUE
13		"SMS_Send_DB".BUSY	%DB11.DBX30.0	Bool	<input type="checkbox"/> FALSE
14		"Send".MessageText_Send	P#DB2.DBX288.0	String	'Alarm: Slot nearly out of Bottles'

Table 6-3

No.	Action	Remark
1	Set input 0.1 to TRUE.	In FB1, the configured message text is written to the send buffer. Then the block starts the send operation and changes to the BUSY status.
2	Automatic	After 3-6 seconds, the send operation is completed and the BUSY bit disappears. In addition, the DONE bit is set for one cycle.

6.1.2 SMS_Broadcast

Figure 6-3

15	1	"Start_Broadcast"	%I0.2	Bool	<input checked="" type="checkbox"/> TRUE
16		"SMS_Broadcast_DB".BUSY	%DB12.DBX6.0	Bool	<input checked="" type="checkbox"/> TRUE
17		"Send".MessageText_Broadcast	P#DB2.DBX450.0	String	'Alarm: Break-in Attempt'
■ ■ ■					
26		"SMS_Broadcast_DB".Errors[0]	%DB12.DBX30	Hex	16#FFFF_FFFF
27		"SMS_Broadcast_DB".Errors[1]	%DB12.DBX34	Hex	16#FFFF_FFFF
28		"SMS_Broadcast_DB".Errors[2]	%DB12.DBX38	Hex	16#FFFF_FFFF
29		"Send".Phonenumbers[0]	P#DB2.DBX24.0	String	'0173 [REDACTED]'
30		"Send".Phonenumbers[1]	P#DB2.DBX48.0	String	'Errorrest'
31		"Send".Phonenumbers[2]	P#DB2.DBX72.0	String	''

↓

15	2	"Start_Broadcast"	%I0.2	Bool	<input checked="" type="checkbox"/> TRUE
16		"SMS_Broadcast_DB".BUSY	%DB12.DBX6.0	Bool	<input type="checkbox"/> FALSE
17		"Send".MessageText_Broadcast	P#DB2.DBX450.0	String	'Alarm: Break-in Attempt'
■ ■ ■					
26		"SMS_Broadcast_DB".Errors[0]	%DB12.DBX30	Hex	16#0000_0000
27		"SMS_Broadcast_DB".Errors[1]	%DB12.DBX34	Hex	16#8202_80EF
28		"SMS_Broadcast_DB".Errors[2]	%DB12.DBX38	Hex	16#FFFF_FFFF
29		"Send".Phonenumbers[0]	P#DB2.DBX24.0	String	'0173 [REDACTED]'
30		"Send".Phonenumbers[1]	P#DB2.DBX48.0	String	'Errorrest'
31		"Send".Phonenumbers[2]	P#DB2.DBX72.0	String	''

Table 6-4

No.	Action	Remark
1.	Set input 0.2 to TRUE.	In FB1, the configured message text is written to the send buffer. The block starts the send operation and first writes FFFF_FFFF to all Error fields. Then SMS_Send calls are performed for each phone number in the array. The block terminates at the first blank phone number[2].
2.	Automatic, block has completed processing.	In the example, an invalid phone number, "Errorrest", was deliberately entered. In Errors, the following can be seen: <ul style="list-style-type: none"> • [0]: The message was successfully sent. • [1]: Abort after the maximum number of errors (82) caused by TC_SEND (02) with error (80EF). • [2] Not processed due to blank phone number.

6.1 Overview

6.1.3 SMS_Recv

Figure 6-4

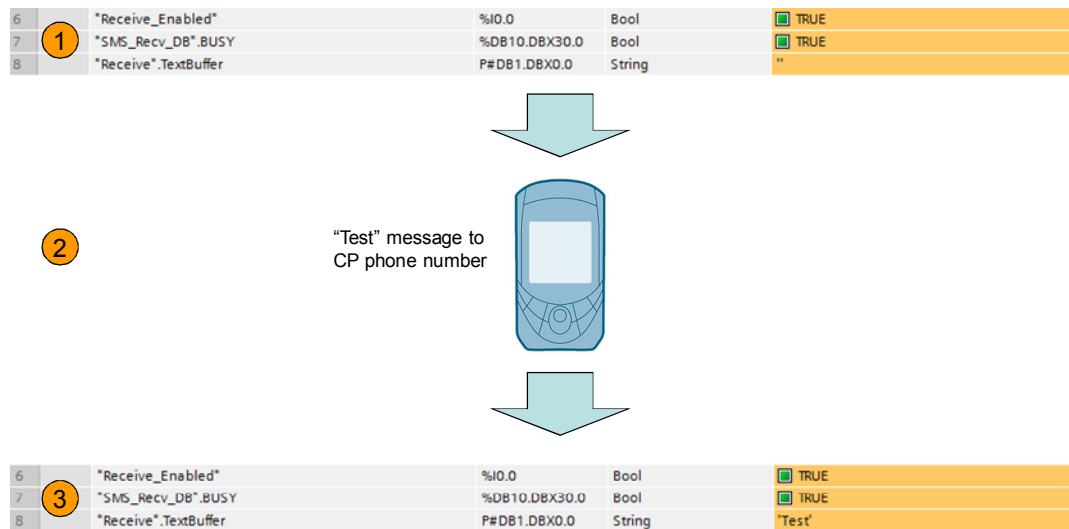


Table 6-5

No.	Action	Remark
1.	Set input 0.0 to TRUE.	The SMS_Recv block establishes the connection and is then permanently connected to TC_RECV. The block is now ready to receive.
2.	Send a message from your mobile phone to the phone number of the CP.	
3.	Automatic	After a few seconds, the message text appears in the text buffer of the CPU.

Note

An active **SMS_Recv.** is mandatory for the *Escalation* and *Set/Get* blocks.

Figure 6-5

Figure 6-5

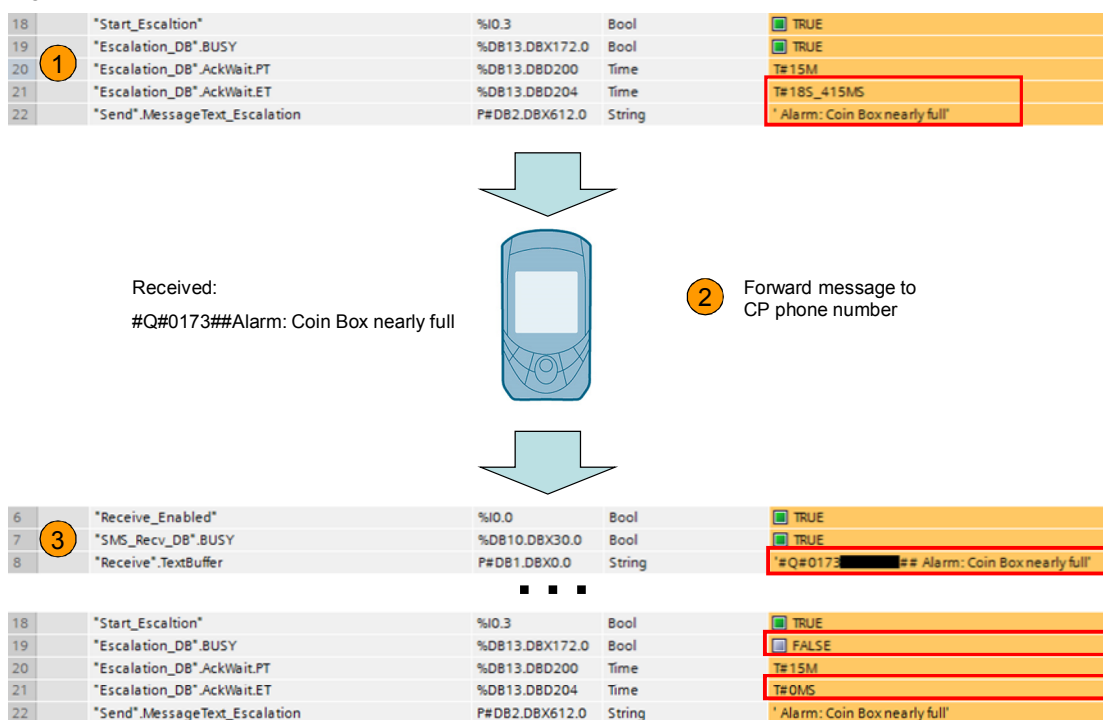


Table 6-6

No.	Action	Remark
1.	Set input 0.3 to TRUE.	The block accepts the message text and starts the send operation with the first phone number. Then the wait time starts.
2.	Forward the alarm message to the CP phone number.	Direct forwarding is only possible with a blank password field. Otherwise, the password must be inserted between ##.
3.	Automatic	The incoming message is written to the text buffer and then further processed. SMS_Escalation checks the data field and detects the appropriate response. The acknowledgement was successful, the timer is stopped and the block terminates with the DONE bit.

6.1.5 Set/Get

Predefined examples

For the predefined sample commands, there is a distinction between assignments and queries. Queries are marked with ? / assignments are marked with =:

6 Operation of the Application

6.1 Overview

Command	Command in the text	
S/G	Help?	Provides a list of known command strings.
G	CurrTemp?	Returns the current temperature.
G	Slots?	Returns the current levels.
G	Output0.0?	Provides the status of output 0.0
S	Output0.0=	Sets output 0.0 to a status. Valid values: TRUE/FALSE (capitalized)
S	DesTemp=	Specifies a new setpoint for the temperature. Valid values: Floating-point numbers. (The String_to_Real function aborts at the first character that is not a digit or period).

Figure 6-6

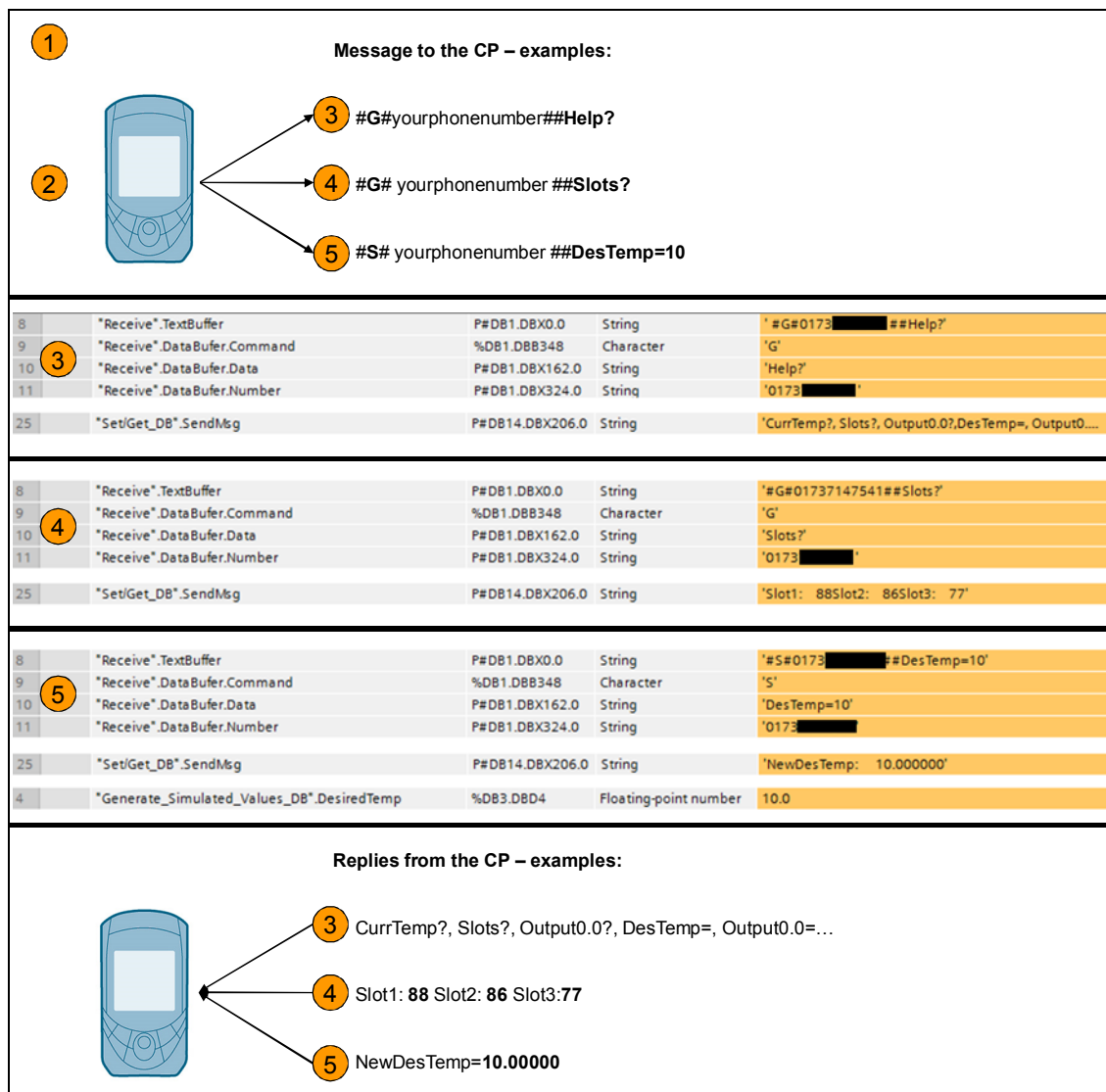


Table 6-7

No.	Action	Remark
1.	Set input 0.4 to TRUE.	The block starts checking the data buffer for the “ G ” and “ S ” commands.
2.	Send a message with a known command to the CP phone number.	The known commands are defined in the Set/Get Commands FC (line 6-9). If necessary, customize the search text and the respective response.
3.	Send: #G#yourphonenumber##Help?	The Set/Get Commands FC also contains a string that includes all known commands. With Help? , it is sent as the response text.
4.	Send: #G#yourphonenumber##Slots?	Set/Get Commands retrieves the current levels of the individual slots (3 x INT) and inserts them in the response message.
5.	Send: #S#yourphonenumber##DesTemp=10	Specifies a new setpoint for the temperature. The value is applied to the CPU program. Then you receive a confirmation of successful execution: NewDesTemp:10.00000

Note Basically, the distinction between assignment and query is already given by the command letter (**S/G**). The additional **=/?** are only used to make this distinction clearer to the user.

Note Set (**S**) and Get (**G**) commands are processed in **separate IF statements**.
 Commands with **?** are only known when command == **G**.
 Commands with **=** are only known when command == **S**.
 A request with **S** as the command and **Slots?** in the data field will return “Unknown Command”.

7 References

This list is by no means complete and only provides a selection of useful information.

Table 7-1

	Topic	Title
\1\	CP Operating Instructions	http://support.automation.siemens.com/WW/view/en/55631071
\2\	S7-1200 Programmable Controller System Manual	http://support.automation.siemens.com/WW/view/en/36932465
\3\	CP Firmware 1.2.1	http://support.automation.siemens.com/WW/view/en/58565570
\4\	CP Hardware Support Package (HSP)	http://support.automation.siemens.com/WW/view/en/52788225
\5\	Step 7 V11 System Manual	http://support.automation.siemens.com/WW/view/en/57185407

8 History

Table 8-1

Version	Date	Modification
V1.0	June 2013	New set number, project tested with STEP7 V12
V1.0	April 2013	New layout
V1.0	06/2012	First edition