SIEMENS

SINEC

S7 Programming Interface

Volume 1 of 1

1 The SAPI-S7 Interface

2 Principles of the Programming Interface

3 The Programming Interface

4 Trace and Mini-DB

5 Configuration

6 SAPI-S7 Under MS-DOS/Windows

7 Appendix

Glossary

Index

C79000-G8976-C077 Release 02

SIEMENS

Wir haben den Inhalt der Druckschrift auf Übereinstimmung mit der beschriebenen Hard- und Software geprüft. Dennoch können Abweichungen nicht ausgeschlossen werden, so daß wir für die vollständige Übereinstimmung keine Gewähr übernehmen. Die Angaben in der Druckschrift werden jedoch regelmäßig überprüft. Notwendige Korrekturen sind in den nachfolgenden Auflagen enthalten. Für Verbesserungsvorschläge sind wir dankbar.

Technische Änderungen vorbehalten.

We have checked the contents of this manual for agreement with the hardware described. Since deviations cannot be precluded entirely, we cannot guarantee full agreement. However, the data in this manual are reviewed regularly and any necessary corrections included in subsequent editions. Suggestions for improvement are welcome.

Technical data subject to change.

Nous avons vérifié la conformité du contenu du présent manuel avec le matériel et le logiciel qui y sont décrits. Or, des divergences n'étant pas exclues, nous ne pouvons pas nous porter garants pour la conformité intégrale. Si l'usage du manuel devait révéler des erreurs, nous en tiendrons compte et apporterons les corrections nécessaires dès la prochaine édition. Veuillez nous faire part de vos suggestions.
Nous nous réservons le droit de modifier les caractéristiques techniques.

Siemens Aktiengesellschaft

Elektronikwerk Karlsruhe
Printed in the Federal Republic of Germany

SIEMENS

SINEC
S7 Programming Interface

Description C79000-B8976-C077/02

Note

We would point out that the contents of this product documentation shall not become a part of or modify any prior or existing agreement, commitment or legal relationship. The Purchase Agreement contains the complete and exclusive obligations of Siemens. Any statements contained in this documentation do not create new warranties or restrict the existing warranty.

We would further point out that, for reasons of clarity, these operating instructions cannot deal with every possible problem arising from the use of this device. Should you require further information or if any special problems arise which are not sufficiently dealt with in the operating instructions, please contact your local Siemens representative.

General

This device is electrically operated. In operation, certain parts of this device carry a dangerously high voltage.

WARNING !

!

Failure to heed warnings may result in serious physical injury and/or material damage.

Only appropriately qualified personnel may operate this equipment or work in its vicinity. Personnel must be thoroughly familiar with all warnings and maintenance measures in accordance with these operating instructions.

Correct and safe operation of this equipment requires proper transport, storage and assembly as well as careful operator control and maintenance.

Personnel qualification requirements

Qualified personnel as referred to in the operating instructions or in the warning notes are defined as persons who are familiar with the installation, assembly, startup and operation of this product and who posses the relevant qualifications for their work, e.g.:

- Training in or authorization for connecting up, grounding or labelling circuits and devices or systems in accordance with current standards in safety technology;
- Training in or authorization for the maintenance and use of suitable safety equipment in accordance with current standards in safety technology;
- First Aid qualification.

S7 Programming Interface

SIMATIC S7 system components communicate with each other using the S7 communications protocol. The SAPI-S7 programming interface was developed to allow programming device/personal computer user programs access to SIMATIC S7 system components. With its C programming interface, you have flexible and simple access to the data of SIMATIC S7 system components.

This manual describes the S7 protocol and how to program it.

Notes

1	The SAPI-S7 Interface	5
1.1	Advantages of S7 Compared with Other Protocols	6
1.2	Advantages of the SAPI-S7 Programming Interface	7
2	Principles of the Programming Interface	9
2.1	Synchronous and Asynchronous Job Handling	10
2.2	Advantages of Asynchronous Operation	11
2.3	Receive Call and Processing Functions	12
2.4	Handling S7 Connection Management Services	14
2.5	Error Message Concept	16
2.6	The Trace	17
2.7	The Mini-DB	18
2.8	Multi-CP and Multi-User Operation	19
2.8.1	Assigning VFDs and the S7 Connection List	20
2.9	Installation and Requirements for Operation	21
3	The Programming Interface	23
3.1	Overview of the Programming Interface	24
3.2	Administrative Services	29
3.2.1	s7_get_device	33
3.2.2	s7_get_vfd	35
3.2.3	s7_init	37
3.2.4	s7_get_cref	39
3.2.5	s7_get_conn	40
3.2.6	s7_shut	42
3.3	Receive Call	43
3.3.1	s7_receive	45
3.4	S7 Connection Management Services	47
3.4.1	s7_initiate_req	53
3.4.2	s7_get_initiate_cnf	54
3.4.3	s7_await_initiate_req	55
3.4.4	s7_get_await_initiate_cnf	56
3.4.5	s7_get_initiate_ind	57
3.4.6	s7_initiate_rsp	58
3.4.7	s7_abort	59
3.4.8	s7_get_abort_ind	60
3.5	Variable Services	61
3.5.1	s7_read_req	68
3.5.2	s7_get_read_cnf	71
3.5.3	s7_write_req	73
3.5.4	s7_get_write_cnf	76
3.5.5	s7_multiple_read_req	77
3.5.6	s7_get_multiple_read_cnf	80
3.5.7	s7_multiple_write_req	83
3.5.8	s7_get_multiple_write_cnf	86
3.5.9	s7_cycl_read_init_req	88
3.5.10	s7_get_cycl_read_init_cnf	91
3.5.11	s7_cycl_read_start_req	92
3.5.12	s7_get_cycl_read_start_cnf	93
3.5.13	s7_get_cycl_read_ind	94
3.5.14	s7_get_cycl_read_abort_ind	97
3.5.15	s7_cycl_read_stop_req	98
3.5.16	s7_get_cycl_read_stop_cnf	99
3.5.17	s7_cycl_read_delete_req	100

3.5.18	s7_get_cycl_read_delete_cnf	101
3.5.19	s7_cycl_read	102
3.6	VFD Services	105
3.6.1	s7_vfd_state_req	109
3.6.2	s7_get_vfd_state_cnf	110
3.6.3	s7_get_vfd_ustate_ind	112
4	Trace and Mini-DB	115
4.1	s7_trace	116
4.2	s7_write_trace_buffer	117
4.3	s7_mini_db_set	118
4.4	s7_mini_db_get	125
4.5	s7_last_iec_err_no	127
4.6	s7_last_iec_err_msg	129
4.7	s7_last_detailed_err_no	130
4.8	s7_last_detailed_err_msg	134
4.9	s7_discard_msg	135
5	Configuration	137
5.1	Significance of Configuration	138
5.2	Services With Configuration Data	139
6	SAPI-S7 Under MS-DOS/Windows	141
6.1	General Information	142
6.2	Translating and Linking for MS-DOS	144
6.3	Translating and Linking for Windows 3.x	146
6.4	Translating and Linking for Windows 95 and Windows NT	150
6.5	Environment Variables	151
6.6	The Trace for MS-DOS or Windows	153
6.7	Special Features for Windows	154
6.7.1	s7_set_window_handle_msg	155
7	Appendix	157
7.1	Range of Functions of SAPI-S7	158
7.2	Representation of S7 Variables	160
7.3	Representation of the Standard Data Types	161
7.3.1	Representation of the 'Boolean' Data Type	161
7.3.2	Representation of the Data Type 'Integer'	162
7.3.3	Representation of the 'Unsigned' Data Type	165
7.3.4	Representation of the 'Floating Point' Data Type	167
7.3.5	Representation of the 'Visible String' Data Type	169
7.3.6	Representation of the 'Octet String' Data Type	170
7.3.7	Representation of the 'Bit String' Data Type	171
Glossary		173
Index		175

1 The SAPI-S7 Interface

This chapter explains the advantages of the S7 communications protocol compared with other protocols and the S7 programming interface (SAPI-S7).

1.1 Advantages of S7 Compared With Other Protocols

Advantages of S7 The advantages of S7 compared with other protocols are as follows:

- ➢ Low load on the processor and bus.
  The S7 protocol is optimized for SIMATIC communication.
- ➢ Simplicity
  The S7 protocol elements are adapted to the requirements of SIMATIC and are therefore simple to use.
- ➢ Speed
  Compared with other layer 7 automation protocols, such as MMS, S7 provides a high performance.

1.2 Advantages of the SAPI-S7 Programming Interface

What Does SAPI-S7 Mean?

The acronym SAPI-S7 stands for the following:

- SAPI - **S**imple **A**pplication **P**rogrammers **I**nterface,
- S7 - the layer 7 communications protocol of SIMATIC S7 systems.

What is SAPI-S7?

SAPI-S7

- is a simple C programming interface,
- provides access to the S7 services on PCs and PGs and
- is available as a C library and is operated with Siemens drivers and communications processors.

Advantages of the SAPI-S7 Programming Interface

The SAPI-S7 programming interface has the following advantages:

- SAPI-S7 is a simple but nevertheless flexible and high-performance interface that requires only one transferred parameter field per job and allows simple buffer management. SAPI-S7 is easy to learn.
- The SAPI-S7 programming interface is designed for asynchronous operation since asynchronous calls are more efficient (more jobs can be processed at the same time). It allows status messages to be received at any time and is ideally suited for creating event-driven programs, for example under Windows.
- SAPI-S7 handles sequential services automatically, such as active or passive connection establishment that consists of several individual steps. The user does not need to worry about sequences and error handling in the individual steps. Writing a program is therefore made much simpler.
- SAPI-S7 supports troubleshooting with an integrated trace function that writes the most important transfer parameters and return values to a file. The trace can be activated and deactivated for each individual service class.
- The SAPI-S7 programming interface is compatible with other SAPI programming interfaces, in other words porting SAPI-S7 applications to applications of other SAPI programming interfaces (for example SAPI-FMS) requires little effort.
- The structures of the SAPI-S7 programming interface are designed so that user programs translated with byte or word alignment can access the individual components without problems. This means that the conditions for use, for example, by BASIC or Pascal programs are met.

2 Principles of the Programming Interface

This chapter introduces you to the principles of the SAPI-S7 programming interface, as follows:

- ➢ Implementation of asynchronous calls for improved performance and the creation of event-driven applications.
- ➢ Separation of receive calls and event-specific processing functions to simplify the SAPI-S7 programming interface.
- ➢ Library-internal processing of sequential services to simplify the user programs.
- ➢ Introduction of a logon and logoff function allowing multi-CP and multi-user operation.
- ➢ Implementation of a trace for debugging the program and to allow the functions of the library and user programs to be checked.

When you have worked through this chapter, you will be familiar with the principles of the S7 programming interface and will be in a position to understand the interface for creating programs.

2.1 Synchronous and Asynchronous Job Handling

Synchronous Job Handling

When jobs are handled synchronously (Figure 2.1), the user program is blocked from the time of the request to the reception of the confirmation. Events occurring in the meantime can only be processed if the program is interrupt-driven. This mode is not implemented in SAPI-S7.

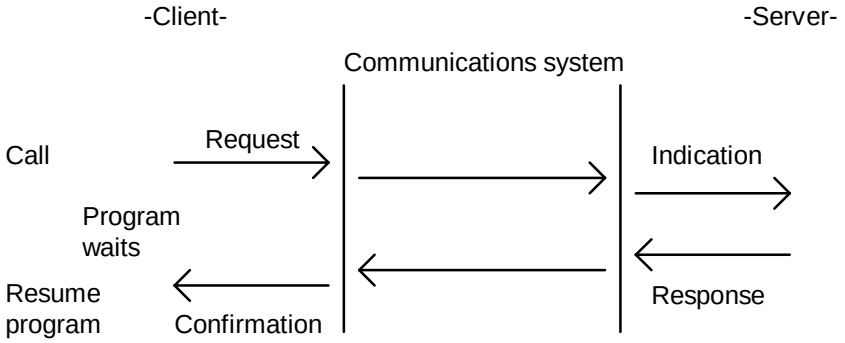

Figure 2.1: Synchronous Operation

Asynchronous Job Handling

In asynchronous job handling (Figure 2.2) a user program is not blocked following a request and can receive any event and react to it. The confirmation is fetched explicitly with a receive call.

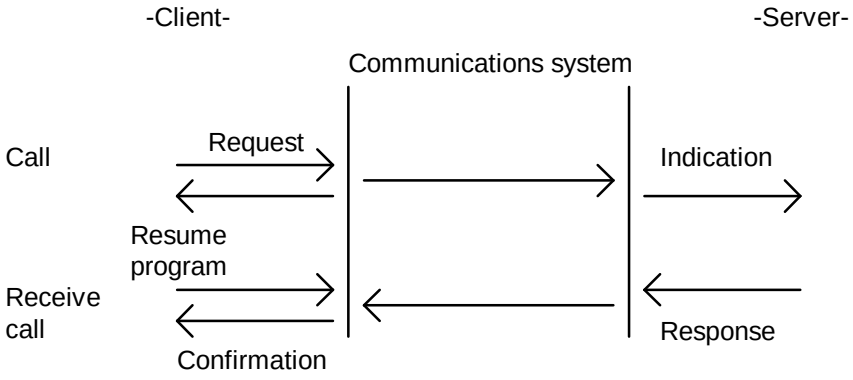

Figure 2.2: Asynchronous Operation

2.2 Advantages of Asynchronous Operation

Asynchronous Jobs as a Basic Principle of the SAPI-S7 Programming Interface

The SAPI-S7 programming interface is designed for asynchronous operation.

- ➢ Asynchronous calls allow a higher data throughput (several jobs can be processed at the same time),
- ➢ An application is not blocked and is free for other tasks such as receiving status messages.
- ➢ Asynchronous mechanisms are ideal for creating event-driven programs, for example under Windows.

From the point of view of the application, you must decide whether or not the services used are dependent on each other. If they are interdependent, the execution of a job must be confirmed before the next job can be sent. For example, a connection must be established before data can be transferred on the connection.

2.3 Receive Call and Processing Functions

How is a Received Message Processed?

A message received from the lower protocol layers is processed by the user program in the two following steps (Figure 2.3):

- ➢ The 's7_receive()' function first fetches an existing message. The return value provides information about the received event. Per event, a constant is defined in the header file 'sapi_s7.h', such as 'S7_INITIATE_IND' that shows that the partner station wants to establish an S7 connection. If 'S7_NO_MSG' is entered, no message was received.
- ➢ In the next step, the user must call the appropriate processing function for the received message: an event identified by 'S7_INITIATE_IND' requires the processing function 's7_get_initiate_ind()' to be called.

Using the receive function, the application program can poll an event. However, between the reception of the event and calling the processing function, no further event can be fetched from the communications system.

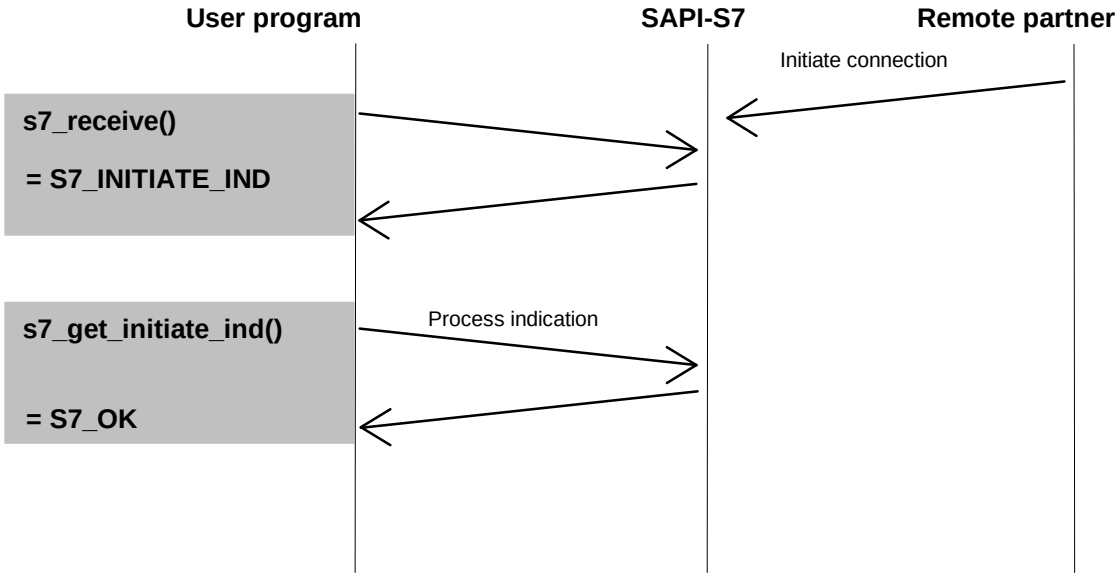

Figure 2.3: Receiving and Processing Messages

Advantages of Separating the Receive Call and Processing Function

Separating the event processing into a receive function and an event-specific processing function achieves the following:

- ➢ Allows a simpler and clearer programming interface. Only the data and parameters relevant to the received event are transferred to the processing functions.
- ➢ Allows the SAPI-S7 programming interface to be extended by additional events by making appropriate processing functions available.
- ➢ Separates the indication (receive call) and response (processing function). This allows user data to be prepared for the response.

2.4 Handling S7 Connection Management Services

Support by the S7 Library

The SAPI-S7 library supports you during active and passive connection establishment that consists of several individual steps. Each step is checked for success or failure. If an error occurs, the library makes sure that steps that have already been executed are corrected. After sending a request, you only need to call the receive function repeatedly until the confirmation of completion is received (see Figure 2.4).

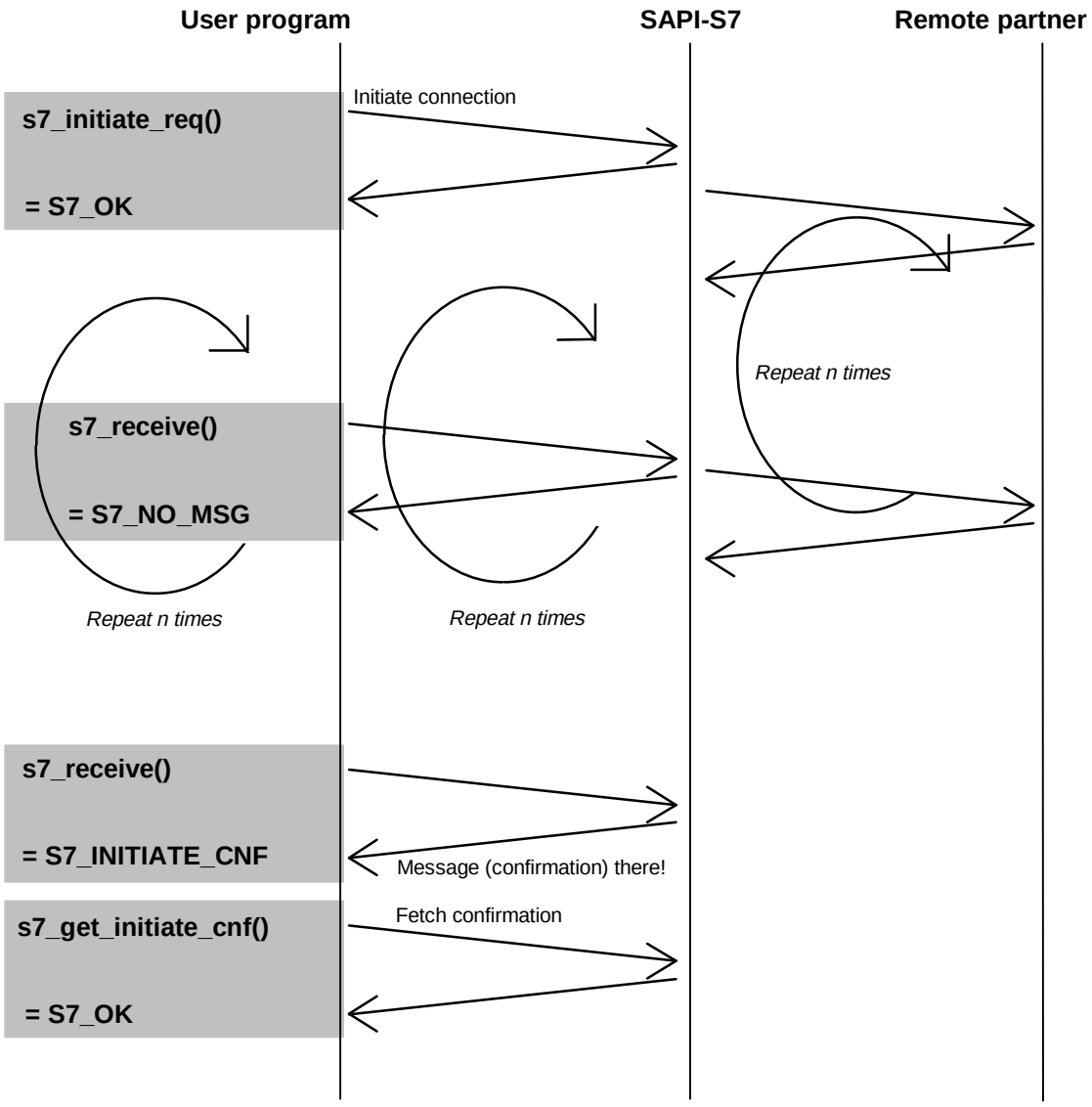

Figure 2.4: Active Connection Establishment

2.5 Error Message Concept

The Three-Stage Error Message Concept of the SAPI-S7 Programming Interface

The SAPI-S7 programming interface supports a three-stage error message concept:

- ➢ Success or failure is only indicated by the return value of the call to simplify error handling.
- ➢ The causes of errors are standardized in compliance with IEC 1131 (**I**nternational **E**lectrotechnical **C**ommission) and the number of error identifiers has been reduced to a more practical size.
- ➢ For more precise error analyses, more detailed error codes and messages are available.

The Return Values of the SAPI-S7 Programming Interface

Defines are located in the header file 'sapi_s7.h' for the return values of each SAPI-S7 call, as follows:

- ➢ The value 'S7_OK' indicates error-free execution of a job.
- ➢ The entry 'S7_ERR_RETRY' indicates that an error occurred when executing a required service. This is a temporary problem as, for example, caused by a shortage of memory. The call can be repeated without modifying the transfer parameters.
- ➢ The entry 'S7_ERR' also indicates an error in the execution of the required service. This is, however, an error that does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as supplying the call with different parameters.

The IEC Error Identifiers

The S7 library provides the functions 's7_last_iec_err_no()' and 's7_last_iec_err_msg()' for reading out an error number and an error message. The causes of the errors are standardized complying with IEC 1131 (**I**nternational **E**lectrotechnical **C**ommission) and reduce the number of error identifiers necessary.

The Detailed Error Codes

For more detailed error analyses, the functions 's7_last_detailed_err_no()' and 's7_last_detailed_err_msg()' functions are implemented on the SAPI-S7 programming interface. These describe the error sources in greater detail and also provide information about eliminating the errors.

2.6 The Trace

What Does Trace Mean?

The trace is a simple and yet effective aid to debugging for the S7 library. This function enters data and events in a ring buffer and a trace file.

The trace can be adapted to a wide range of applications, as follows:

- ➢ The trace file can be given any name to suit your application.
- ➢ The service classes for which the trace will be activated can be selected.
- ➢ The trace depth (trace off, trace only for negative events etc.) can be selected.
- ➢ The target of the trace indicates whether a trace file should be created, an old trace file used or whether the information should be written to the internal ring buffer.

Uses of the Trace

During operation, the S7 library writes important data to the trace file. The content of the file

- ➢ is used as a log of all actions performed by the S7 library,
- ➢ permits the sequence of events to be checked both in the application and the library itself,
- ➢ allows simple debugging and checking of the data and parameters on the SAPI-S7 programming interface.

You can also write to the trace file yourself. With such entries, it is possible to save important data for test purposes, to check the sequence of the program or to synchronize it with the entries by the library.

Time Required

The trace and saving the data in the trace file slows down the user program. For this reason, it is also possible to write the ring buffer to a file as a “snapshot”.

2.7 The Mini-DB

Purpose of the Mini-DB

Using the mini-DB (mini-**d**ata**b**ase), settings different from the standard operating situation can be made and detailed information read out. The mini-DB is a data area within the S7 library

- ➢ in which repeatedly required data (normally unchanged data) can be stored,
- ➢ in which protocol-specific data can be stored or saved during operation,
- ➢ in which identifiers of the last error to occur are saved,
- ➢ that can be read out and modified using functions such as 's7_mini_db_get()', 's7_mini_db_set()' etc.

Advantages of the Mini-DB

The S7 library allows operation with default setting for the trace, the establishment of S7 connections (active and passive), for uploading an OD from the partner station etc. These default settings are adequate for the majority of applications but can be modified and adapted to special requirements at any time by calling the mini-DB.

The use of a mini-DB in the S7 library has the following advantages for the user:

- ➢ The programming interface can be simplified to a few transfer parameters. Values required (possibly always exactly the same values) can be read from the mini-DB. This increases the performance and simplifies the handling of the SAPI-S7 programming interface.
- ➢ The high degree of flexibility provided by the S7 protocol is retained. Changes in the settings and adapting them to the user's requirements are possible at any time by calling a mini-DB.

2.8 Multi-CP and Multi-User Operation

What is Multi-CP Operation?

Multi-CP operation means the capability of addressing

- ➢ several CPs and
- ➢ the partner stations connected via the various CPs on networks

by one application on one computer.

What Does Multi-User Operation Mean?

In multi-user operation, the services and resources of a computer can be used by several applications without interfering with each other.

Requirements

The main condition for using multi-CP and multi-user operation is that the requests and confirmations as well as indications and responses must be unambiguously assigned to each other and to the corresponding application.

This condition is achieved by a logon function implemented on the SAPI-S7 programming interface. There is a logon for

- ➢ a CP and
- ➢ for a VFD of the CP.

The VFD provides VFD-specific S7 connections for the application (Figure 2.5). A further logon for the VFD cannot be made either by the same application or by a different application although multiple logons for a CP are possible.

2.8.1 Assigning VFDs and the S7 Connection List

Relationship between VFDs and the S7 Connection List

An application can log on at several VFDs on one or more CPs. multi-CP and multi-user operation is, however, only possible when a VFD can be assigned unequivocally to an application after the logon (1:n assignment). When the application logs on at the CP and the selected VFD, the connections assigned to the VFD during configuration are available from the S7 connection list of the CP. For example, in the following diagram, communication is possible on connections “C1”, “C2” and “C3” after a logon at “CP 1” and “VFD 1”.

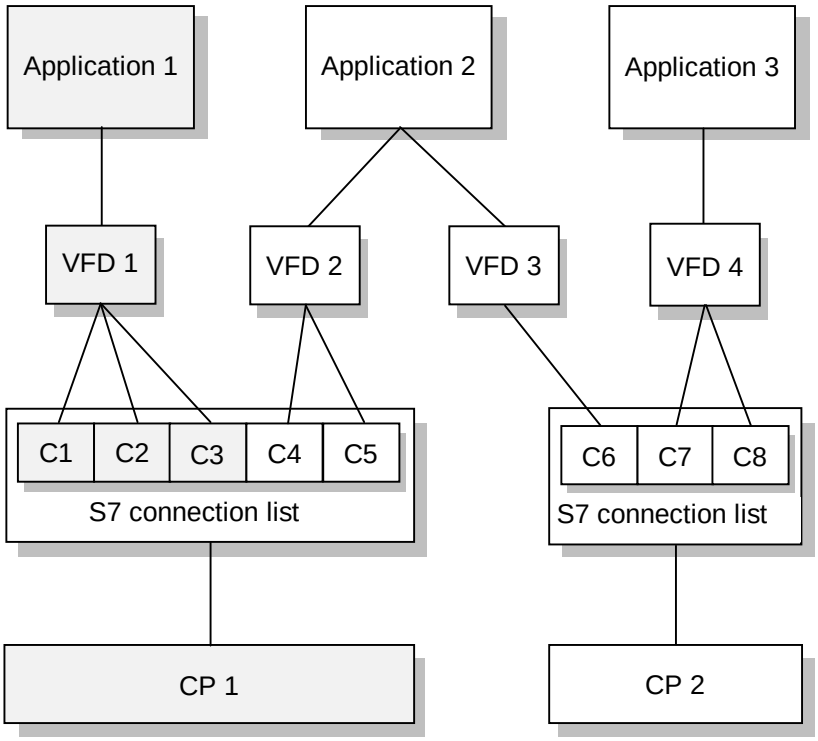

Figure 2.5: Assignment of the S7 Connection List with a Logon at VFD 1 on CP 1

2.9 Installation and Requirements for Operation

Installation

The procedure for installation is described in the documentation for the particular products and is not part of this manual.

Requirements for Operation

To operate the S7 library, the communication system must be ready for operation, for example, VFDs and S7 connections must be configured These tasks are not supported by the SAPI-S7 programming interface.

Notes

3 The Programming Interface

This section introduces you to the practical use of the S7 programming interface for the 'C' programming language.

How to use the calls on the programming interface that provide you with access to S7 services and objects is described in the subsections based on example programs. The examples

- ➢ clearly describe the order that must be maintained for successful communication,
- ➢ introduce the programming interface step by step,
- ➢ supplement each other whereby new functions or modified program segments of the previous example are printed in **bold face**,
- ➢ implement an error handling routine that you can adapt to your own requirements.

The S7 functions are not called directly in the example programs but are separated into their own functions (for example 'my_init()' for 's7_init()'). This is not mandatory, but makes the program more suitable for debugging, error output and subsequent extensions.

At the end of this chapter you will be familiar with the following:

- ➢ Which services are available on a host system,
- ➢ Which services are required for which tasks,
- ➢ Which communications sequences occur as a result of a service request and service confirmation,
- ➢ Which call and sequence structure exists generally on the S7 programming interface.

You will then be familiar with the basics of the S7 programming interface and be in a position to start programming a SAPI-S7 application.

3.1 Overview of the Programming Interface

Administrative Services

The administrative services provide functions for reading out configuration information and for logging on and logging off at the communications system. There are also functions that provide the references for symbolic S7 connection names:

s7_get_device()	With this function, the user program can query the names of all the installed CPs.
s7_get_vfd()	With this function, the user program can query all the configured VFDs of a CP.
s7_init()	With this function, the user program logs on at the communications system.
s7_get_cref()	This function provides a reference to a symbolic S7 connection name. This value selects the real connection in the network and is easier to use than the symbolic name.
s7_get_conn()	This function returns all symbolic S7 connection names of the logged on VFD and their references.
s7_shut()	With this function, the user program logs off at the communications system.

Receive Call Service

s7_receive()	With this function, the user program fetches received messages from the communications system.

S7 Connection Management Services

The S7 connection management services are required to establish and close S7 connections.

Active connection establishment:

s7_initiate_req()	This function passes a request for the active establishment of an S7 connection to the communications system.
s7_get_initiate_cnf()	This function receives the result of the above call.

Passive connection establishment:

s7_await_initiate_req()	This function prepares the communications system to receive a passive S7 connection establishment request.
s7_get_await_initiate_cnf()	This function receives the result of the above call.
s7_get_initiate_ind()	This function completes the reception of an initiate indication.
s7_initiate_rsp()	The connection request can be accepted or rejected with this function.

Closing a connection:

s7_abort()	This function aborts an established S7 connection.
s7_get_abort_ind()	This function completes the reception of an abort indication.

Variable Services

This service class contains functions for reading and writing one or more variables.

Reading and writing a variable:

s7_read_req()	This function initiates a read variable job.
s7_get_read_cnf()	This function receives the value of the variable that was read.
s7_write_req()	This function initiates a write variable job
s7_get_write_cnf()	This function receives the result of the above call.

Reading and writing more than one variable:

s7_multiple_read_req()	This function initiates a job to read multiple variables.
s7_get_multiple_read_cnf()	This function receives the value of the variables that were read.
s7_multiple_write_req()	This function initiates a job to write multiple variables.
s7_get_multiple_write_cnf()	This function receives the result of the above call.

Initiating cyclic reading with multiple calls:

s7_cycl_read_init_req()	This function instructs the server to prepare for the cyclic reading of variables.
s7_get_cycl_read_init_cnf()	This function receives the result of the above call.
s7_cycl_read_start_req()	This function instructs the server to start the cyclic reading of variables.
s7_get_cycl_read_start_cnf()	This function receives the result of the above call.

Receiving data cyclically:

s7_get_cycl_read_ind()	This function receives the data sent by the server.

Stopping and aborting cyclic reading:

s7_get_cycl_read_abort_ind()	This function completes the reception of a cyclic read abort indication.
s7_cycl_read_stop_req()	This function instructs the server to stop cyclic reading of variables.
s7_get_cycl_read_stop_cnf()	This function receives the result of the above call.
s7_cycl_read_delete_req()	This function stops cyclic reading and logs off at the server.
s7_get_cycl_read_delete_cnf()	This function receives the result of the above call.

Initiating cyclic reading with one call:

s7_cycl_read()	This function instructs the server to prepare for cyclic reading of variables and to start reading immediately.

VFD Services

The following VFD services are defined:

s7_vfd_state_req()	This function queries the device/user status.
s7_get_vfd_state_cnf()	This function receives the device/user status.
s7_get_vfd_ustate_ind()	This function receives the device/user status sent unsolicited.

3.2 Administrative Services

Description of the Example of Administrative Services

The following example describes the administrative services for logging on ('**s7_init()**') and logging off ('**s7_shut()**') an S7 application at the communications system. Communication with a local CP and therefore also with a remote CP is only possible after a successful logon. When an application logs on at the local VFD, the VFD-specific resources are reserved for the application so that before the program is terminated, every logon must also be terminated by a logoff.

The transfer parameters required to log on such as the CP name and the VFD name are queried using the functions **'s7_get_device()'** or **'s7_get_vfd()'** and supplied by the communications system.

Example

```
#include "sapi_s7.h"

/* prototypings */
static void my_exit(ord32 cp_descr,char *msg,int32 ret);
static void my_get_cref(ord32 cp_descr,ord16 *cref_ptr);
static void my_init(ord32 *cp_descr_ptr);
static void my_shut(ord32 cp_descr);

/* exit application */
static void my_exit(ord32 cp_descr,char *msg,int32 ret)
{
       printf("\n%s = %lx",msg,ret);
       my_shut(cp_descr);
}

/* get reference for connection 'TEST' */
static void my_get_cref(ord32 cp_descr,ord16 *cref_ptr)
{      int32 ret;

       ret=s7_get_cref(cp_descr,"TEST",cref_ptr);
       if(ret!=S7_OK)
       {
              my_exit(cp_descr,"Error s7_get_cref",ret);
       }
}

/* initialize s7 */
static void my_init(ord32 *cp_descr_ptr)
{      int32 ret;
       char vfd_name[S7_MAX_NAMLEN+1];
       char dev_name[S7_MAX_DEVICELEN+1];
       ord16 number;

       /* only use first device */
       ret=s7_get_device(0,&number,dev_name);
       if((ret!=S7_OK)||(number==0))
       {      /* something has gone wrong */
              printf( "\ns7_get_device = %lx, number = %d",
                     ret,number);
              exit(-1);
       }

       /* only use first vfd of first device */
       ret=s7_get_vfd(dev_name,0,&number,vfd_name);
       if((ret!=S7_OK)||(number==0))
       {      /* something has gone wrong */
              printf( "\ns7_get_vfd = %lx, number = %d",
                     ret,number);
              exit(-2);
       }

       /* initialize s7 */
       ret=s7_init(dev_name,vfd_name,cp_descr_ptr);
       if(ret!=S7_OK)
       {      /* something has gone wrong */
              printf( "\ns7_init = %lx",ret);
              exit(-3);
       }
}
```

```
/* end communication */
static void my_shut(ord32 cp_descr)
{       int32 ret;

        ret=s7_shut(cp_descr);
        if(ret!=S7_OK)
        {       /* error has occurred -> exit */
                printf( "\ns7_shut = %lx",ret);
        }

        /* no error has occurred         */
}

/* main */
void main(void)
{       ord32   cp_descr;
        ord16   cref;

        /* initialize s7 */
        my_init(&cp_descr);

        /* get reference for connection 'TEST' */
        my_get_cref(cp_descr,&cref);

        /* end communication */
        my_shut(cp_descr);
}
```

Sequence Chart

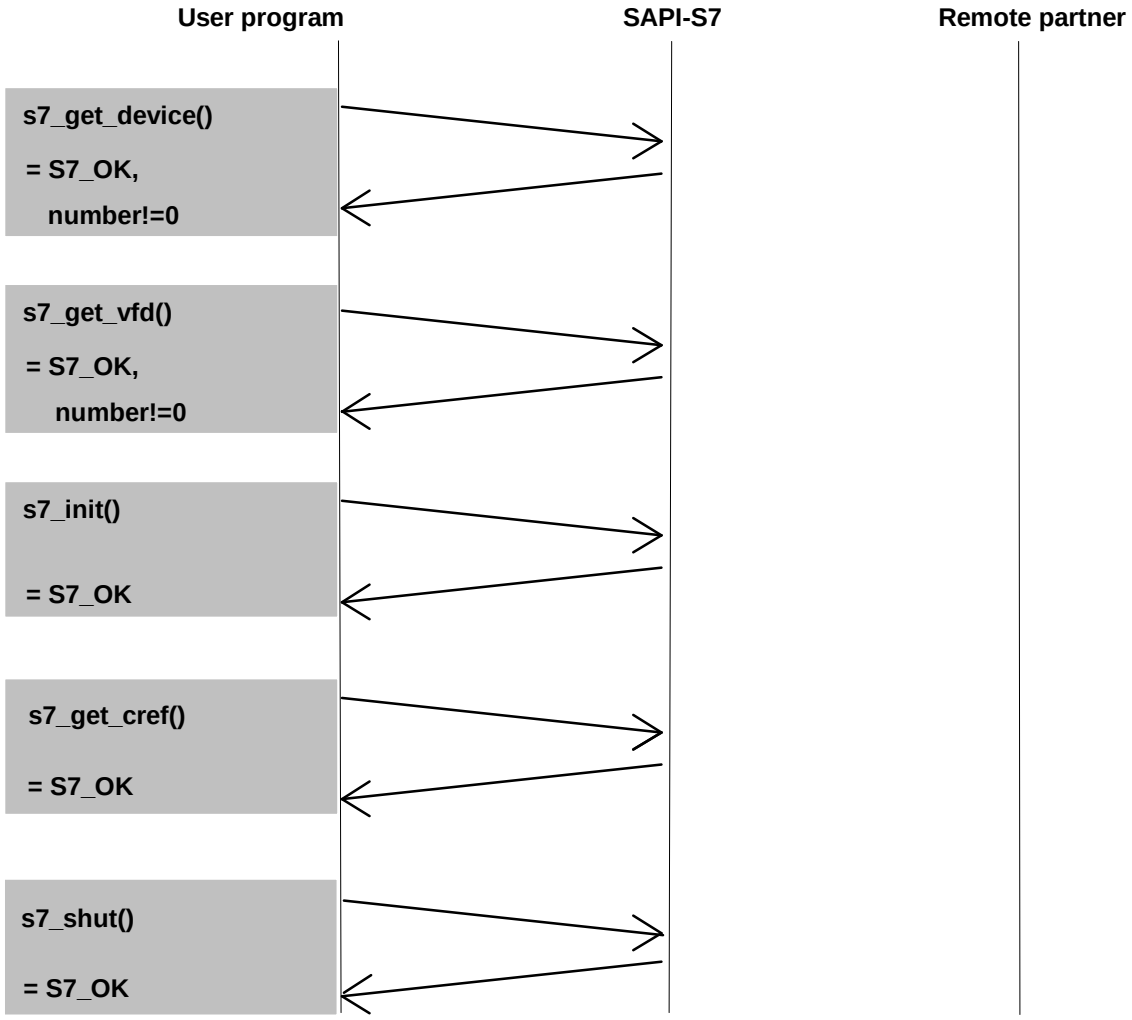

Figure 3.1: Sequence Chart of the Example

3.2.1 s7_get_device

Description

With this call, the user program can query all the configured names of the communications processors installed in the communications system. The names are relevant for logging on with ‘s7_init()’.

Declaration

```
int32 s7_get_device(
                ord16  index,          /* In call  */
                ord16  *number_ptr,    /* Returned */
                char   *dev_name       /* Returned */
                      )
```

Parameters

index	The ‘index’ parameter selects one of the existing CPs that are numbered continuously starting at 0.
number_ptr	Address of a variable of the type ‘ord16’ provided by the user program. The number of installed CPs is returned here.
dev_name	Start address of a memory area provided by the user program in which the configured CP name is entered. The memory area should be at least (S7_MAX_DEVICELEN + 1) bytes long for the maximum S7_MAX_DEVICELEN bytes long CP name including the string end identifier.

Querying All CPs

To query all CPs, it is best to use a program loop. The ‘index’ parameter is used as the loop variable and runs from 0 to ‘*number_ptr-1’. The ‘*number_ptr’ has the default 1.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.2.2 s7_get_vfd

Description

With this call, the user program can query all the configured VFDs of a communications processor. The VFD names are relevant for the logon with 's7_init()'.

Declaration

```
int32 s7_get_vfd(
                char   *dev_name,      /* In call  */
                ord16  index,          /* In call  */
                ord16  *number_ptr,    /* Returned */
                char   *vfd_name       /* Returned */
                  )
```

Parameters

dev_name	Configured name of the communications processor via which you want to communicate. For this parameter a CP name read out with 's7_get_device()' is usually used. It must match a configured CP name (for example 'CP_L2_1:').
index	The ‘index’ parameter selects one of the existing CPs that are numbered continuously starting at 0.
number_ptr	Address of a variable of the type 'ord16' provided by the user program. The number of configured VFDs is returned here.
vfd_name	Start address of a memory area provided by the user program in which the configured VFD name is entered. The memory area should be at least (S7_MAX_NAMLEN + 1) bytes long for the maximum S7_MAX_NAMLEN byte long VFD name including the string end identifier.

Querying All VFDs

To query all VFDs, it is best to use a program loop. The ‘index’ parameter is used as the loop variable and runs from 0 to ‘*number_ptr-1’. The ‘*number_ptr’ has the default 1.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.2.3 s7_init

Description

With this call, the user program logs on at a VFD of an underlying communications system.

As input parameters, the user program specifies the configured name of the communications processor via which communication will take place and the configured name of the VFD whose resources and services will be used.

The user program receives a descriptor 'cp_descr' that must be specified for all subsequent calls as an addressing parameter for the selected CP and VFD until the application logs off.

When the program logs on, the configuration information (for example, the list of all S7 connections) is read from a file with a name derived from the name of the CP as follows: the colon completing the CP name is removed and the name extension ‘.LDB’ is added. This assumes that the file exists in the current working directory. As an alternative, the name of the configuration file can also be assigned (using any name) in the environment variable.

If several VFDs are being used at the same time in one or more communications processors by one application, a corresponding number of 's7_init()' calls are necessary.

Declaration

```
int32 s7_init(
            char   *cp_name,       /* In call  */
            char   *vfd_name,      /* In call  */
            ord32  *cp_descr_ptr   /* Returned */
            )
```

Parameters	cp_name	Configured name of the communications processor with which communication will take place. To address a specific module, use a name configured with the SINEC installation tool (for example 'CP_L2_1:'). Normally a CP name read out with 's7_get_device()' is used.
	vfd_name	Configured name of the local VFD at which the application logs on. To address a particular VFD, use a VFD name specified with the SINEC configuration tool. By selecting the VFD, the configured S7 connections are also selected. Here, a VFD name read out with 's7_get_vfd()' is normally used.
	cp_descr_ptr	Address of a variable of the type 'ord32' provided by the user program. Here, a descriptor for addressing the selected communications processor and VFD is entered. This parameter must be used for further communication via the selected communications processor and VFD.
Return Values	S7_OK	The function was processed without errors.
	S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
	S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.2.4 s7_get_cref

Description

From the communication system, the S7 application only sees the symbolic name of the S7 connection that is cumbersome during operation. For this reason, the application uses the 's7_get_cref()' call to obtain a reference to a symbolic connection name.

Declaration

```
int32 s7_get_cref(
                ord32  cp_descr,    /* In call  */
                char   *conn_name,  /* In call  */
                ord16  *cref_ptr    /* Returned */
                  )
```

Parameters

cp_descr	Handle as return value of the 's7_init()' call.
conn_name	Symbolic name of the S7 connection on which communication will take place.
cref_ptr	Address of a variable of the type 'ord16' provided by the user program. The connection reference is returned here.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.2.5 s7_get_conn

Description

With this call, the user program can query all the configured S7 connections of a VFD and their references on a communications processor. This identifier selects the real connection on the network and is easier to use than the symbolic name.

Declaration

```
int32 s7_get_conn(
                ord32  cp_descr,      /* In call   */
                ord16  index,         /* In call   */
                ord16  *number_ptr,   /* Returned  */
                ord16  *cref_ptr,     /* Returned  */
                char   *conn_name     /* Returned  */
                    )
```

Parameters

cp_descr	Handle as return value of the 's7_init()' call.
index	The ‘index’ parameter selects one of the existing S7 connections that are numbered continuously starting at 0.
number_ptr	Address of a variable of the type ‘ord16’ provided by the user program. The number of configured S7 connections is returned here.
cref_ptr	Address of a variable of the type 'ord16' provided by the user program. The connection reference is returned here.
conn_name	Start address of a memory area provided by the user program in which the configured S7 connection name is entered. The memory area should be at least (S7_MAX_NAMLEN + 1) bytes long for the maximum S7_MAX_NAMLEN byte long connection name including string end identifier.

Querying all S7 Connections

To query all S7 connections, it is best to use a program loop. The ‘index’ parameter is used as the loop variable and runs from 0 to ‘*number_ptr-1’. The ‘*number_ptr’ has the default 1.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.2.6 s7_shut

Description

With the 's7_shut()' call, the application cancels the logon (‘s7_init()’) at the communications processor identified by the CP descriptor. All the resources of the communications system are returned and established connections are terminated. This call must therefore be made once for every logon before the program is terminated.

Declaration

```
int32 s7_shut(
                ord32   cp_descr    /* In call  */
              )
```

Parameters

cp_descr	Handle as return value of the 's7_init()' call. This identifies the logon to be canceled.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.3 Receive Call

Description of the Example

To extend the example in Section 3.2 the receive call '**s7_receive()**' is used to check whether a message exists. In this simple case, a message is not expected ('S7_NO_MSG' as return value); the example is simply in preparation for programs coming later.

Example

```
:
:
/* additional prototypings */
static void my_receive(ord32 cp_descr,int32 last_event_expected);

/* receive any message from communication system */
static void my_receive(ord32 cp_descr,int32 last_event_expected)
{       ord16 cref,orderid;
        int32 ret;

        do
        {       ret=s7_receive(cp_descr,&cref,&orderid);
                switch(ret)
                {       case S7_NO_MSG:
                                break;
                        default:
                                printf("\nEvent unexpected: %lx",
                                        ret);
                                break;
                }
        }       while(ret!=last_event_expected);
}

/* main */
void main(void)
{       ord32  cp_descr;
        ord16  cref;

        /* initialize s7 */
        my_init(&cp_descr);

        /* get reference for connection 'TEST' */
        my_get_cref(cp_descr,&cref);

        /* receive message */
        my_receive(cp_descr,S7_NO_MSG);

        /* end communication */
        my_shut(cp_descr);
}
```

Sequence Chart

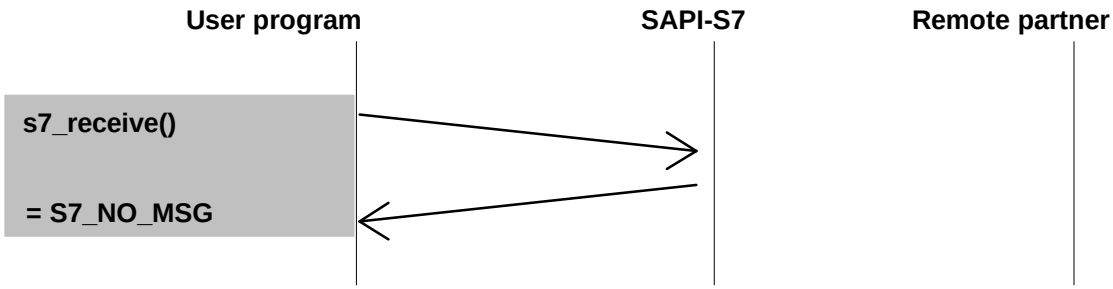

Figure 3.2: Sequence Chart of the Example

3.3.1 s7_receive

Description

The receive function of the S7 library has the central task of analyzing events received from the underlying communications system and to report these directly to the application without any further processing.

The 's7_receive()' call is absolutely necessary when the client

- has sent acknowledged calls and is waiting for the acknowledgment from the server,
- wishes to receive unacknowledged messages from the network,
- has sent sequential jobs (for example establishing an S7 connection) to ensure processing of the service sequence until it is completed.

The return value of the 's7_receive()' function provides a service identifier when a message is received. This identifies the type of service of the received response (for example 'S7_READ_CNF' when a variable was read). After receiving a message with 's7_receive()', the call for the corresponding processing function is mandatory (for example 's7_get_read_cnf()'). Further receive calls are otherwise rejected with an error message.

Declaration

```
int32 s7_receive(
             ord32  cp_descr,       /* In call  */
             ord16  *cref_ptr,      /* Returned */
             ord16  *orderid_ptr    /* Returned */
                )
```

Parameters

cp_descr	Handle as return value of the 's7_init()' call. This identifies the communications processor and the VFD via which an event will be fetched.
cref_ptr	Address of a variable of the type 'ord16' provided by the user program. The reference of the S7 connection on which an indication or a confirmation was received is entered here. This corresponds to the S7 connection reference with which the job was sent.
orderid_ptr	Address of a variable of the type 'ord16' provided by the user program. The job identifier of the received acknowledgment is entered here. This identifies the previously sent job for the user program. The order ID is irrelevant for unacknowledged services or services to establish an S7 connection and is assigned a default value.

Return Values

S7_NO_MSG	No message was received.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

Apart from these values, further service identifiers such as 'S7_READ_CNF' are also returned depending on the message received. You will find the definitions in the ‘sapi_s7.h’ file.

Type of Call

In single-tasking operating systems such as MS-DOS, the receive function is polled. In multitasking operating systems such as Windows, it is possible to include the receive function at a central waiting point allowing all the events to be received.

3.4 S7 Connection Management Services

Description of the Example

To extend the example from Section 3.3, an S7 connection with the symbolic name 'TEST' is established (‘**s7_initiate_req()**’) and then aborted (‘**s7_abort()**’) after the confirmation (‘**s7_get_initiate_cnf()**’) is received.

Example

```
:
:
/* additional prototypings */
static void my_abort(ord32 cp_descr,ord16 cref);
static void my_get_abort_ind(ord32 cp_descr);
static void my_get_initiate_cnf(ord32 cp_descr);
static void my_initiate_req(ord32 cp_descr,ord16 cref);

/* abort connection */
static void my_abort(ord32 cp_descr,ord16 cref)
{       int32 ret;

        if((ret=s7_abort(cp_descr,cref))!=S7_OK)
        {
                my_exit(cp_descr,"Error s7_abort",ret);
        }
}

/* get abort indication */
static void my_get_abort_ind(ord32 cp_descr)
{       int32 ret;

        if((ret=s7_get_abort_ind())!=S7_OK)
        {
                my_exit(       cp_descr,
                               "Error s7_get_abort_ind",ret);
        }
}

/* get initiate confirmation */
static void my_get_initiate_cnf(ord32 cp_descr)
{       int32 ret;

        if((ret=s7_get_initiate_cnf())!=S7_OK)
        {
                my_exit(       cp_descr,
                               "Error s7_get_initiate_cnf",ret);
        }
}

/* initiate connection "TEST" */
static void my_initiate_req(ord32 cp_descr,ord16 cref)
{       int32 ret;

        if((ret=s7_initiate_req(     cp_descr,cref))!=S7_OK)
        {
                my_exit(cp_descr,"Error s7_initiate_req",ret);
        }
}
```

```
/* receive any message from communication system */
static void my_receive(ord32 cp_descr,int32 last_event_expected)
{       ord16 cref,orderid;
        int32 ret;

        do
        {       ret=s7_receive(cp_descr,&cref,&orderid);
                switch(ret)
                {
                        :
                        :
                        case S7_INITIATE_CNF:
                                my_get_initiate_cnf(cp_descr);
                                my_abort(cp_descr,cref);
                                break;
                        case S7_ABORT_IND:
                                my_get_abort_ind(cp_descr);
                                break;
                        default:
                                printf( "Event unexpected",
                                        ret);
                                break;
                }
        }       while( (ret!=last_event_expected)&&
                        (ret!=S7_ABORT_IND));
}

/* main */
void main(void)
{       ord32   cp_descr;
        ord16   cref;

        /* initialize s7 */
        my_init(&cp_descr);

        /* get reference for connection 'TEST' */
        my_get_cref(cp_descr,&cref);

        /* initiate connection */
        my_initiate_req(cp_descr,cref);

        /* receive initiate confirmation */
        my_receive(cp_descr,S7_INITIATE_CNF);

        /* end communication */
        my_shut(cp_descr);
}
```

Sequence Chart for Active Connection Establishment

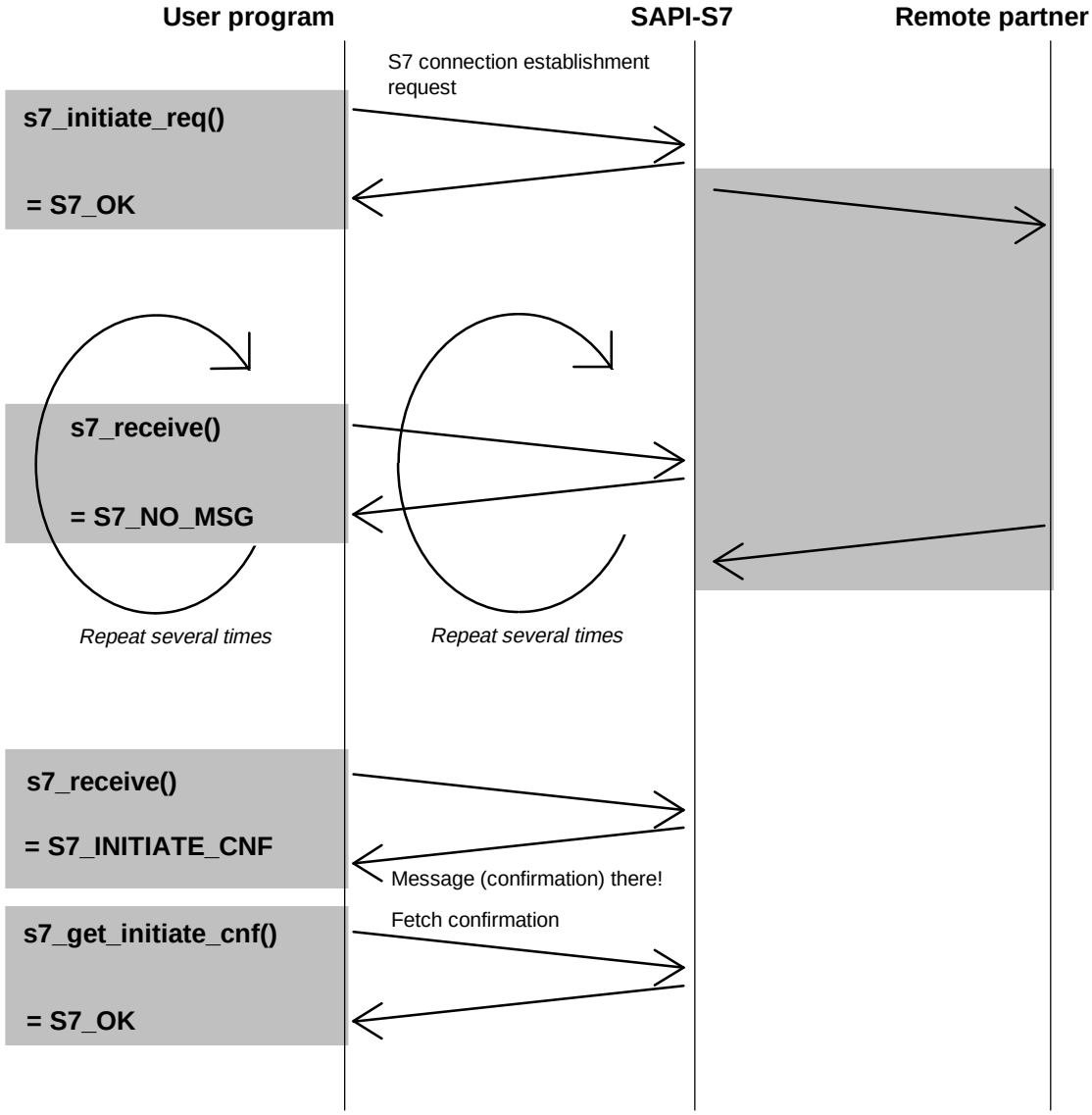

Figure 3.3: Sequence Chart for Active S7 Connection Establishment

Sequence Chart for Preparing for Passive Connection Establishment (not part of example)

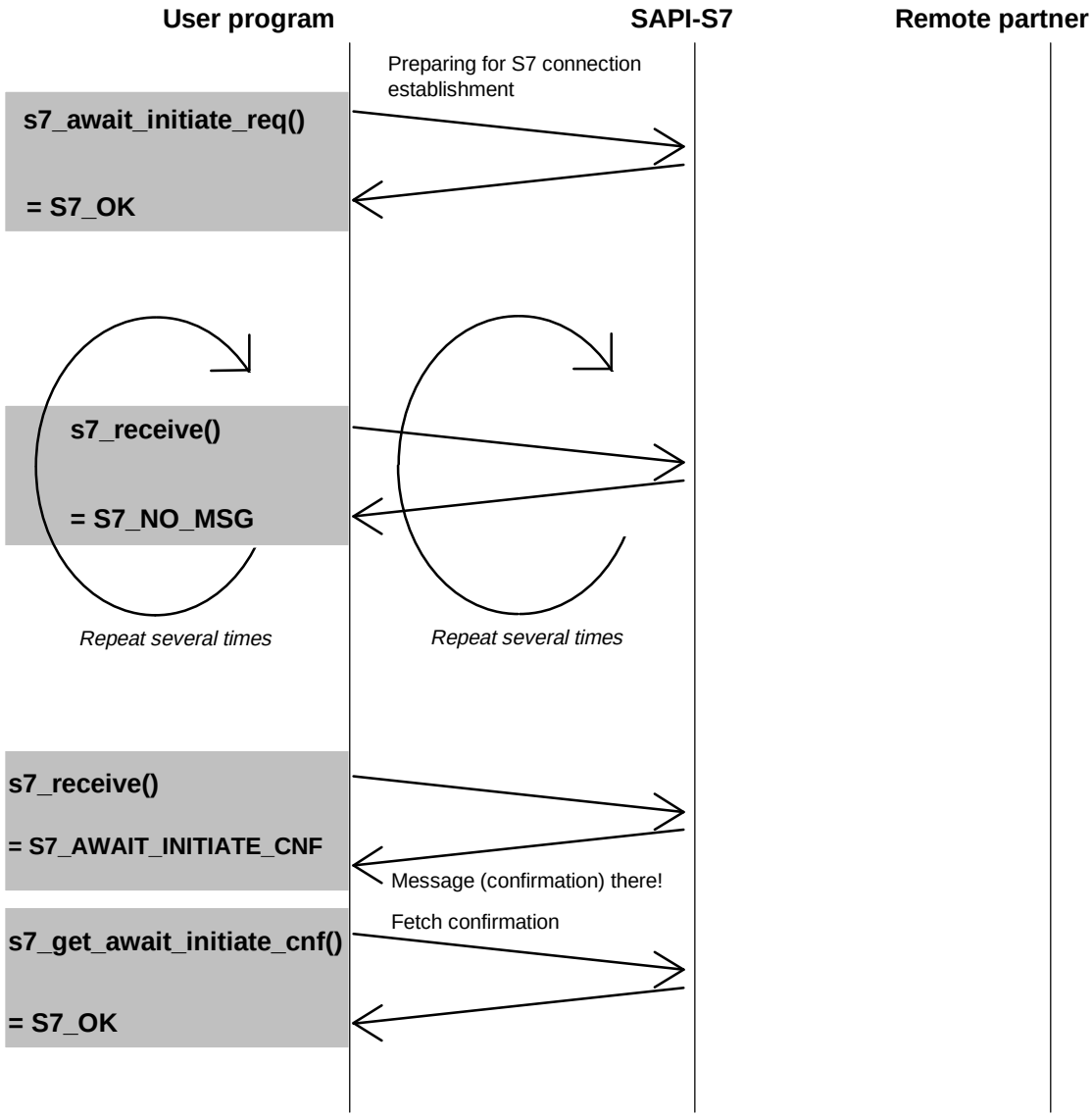

Figure 3.4: Sequence Chart for Preparing for Passive S7 Connection Establishment

Sequence Chart for Passive Connection Establishment (not part of example)

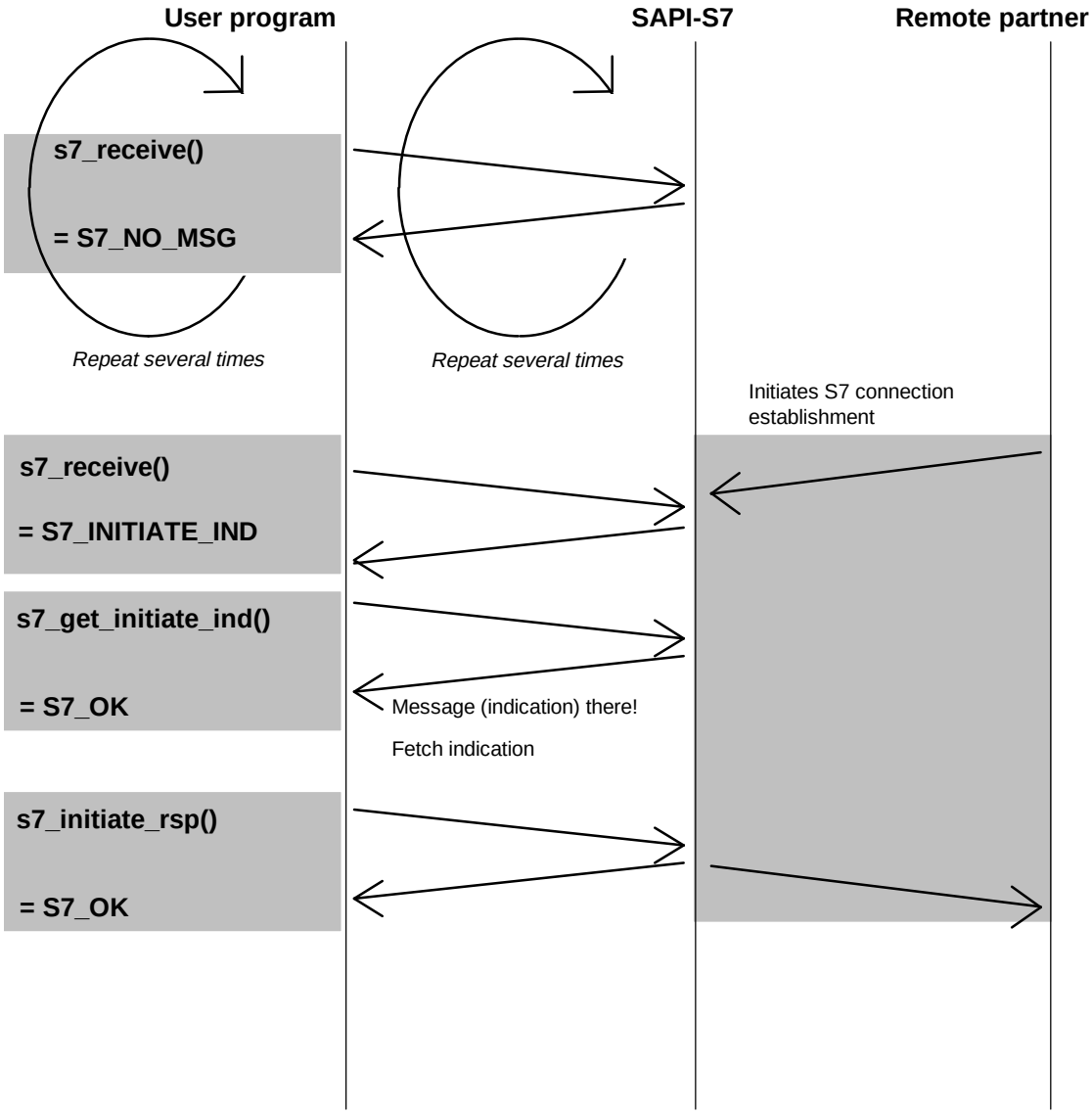

Figure 3.5: Sequence Chart for Passive S7 Connection Establishment

3.4.1 s7_initiate_req

Description

The 's7_initiate_req()' call initiates the establishment of an S7 connection. The initiative for establishing a connection comes from the user program, the required parameters are read from the mini-DB. Here, they can be read out, modified and adapted to the operating requirements by the user program.

On the partner (passive side), the configured capability parameters are compared with the received values of the job. The capability of the S7 connection (send credit, receive credit PDU size..) that can be implemented is determined by the lowest values of these parameters on either station.

Declaration

```
int32 s7_initiate_req(
                ord32  cp_descr,    /* In call  */
                ord16  cref         /* In call  */
                      )
```

Parameters

cp_descr	Handle as return value of the 's7_init()' call.
cref	Reference of the S7 connection to be established.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.4.2 s7_get_initiate_cnf

Description

The 's7_get_initiate_cnf()' call, receives the result of the S7 connection establishment.

With the 's7_receive()' call, the user program receives the 'S7_INITIATE_CNF' confirmation if the establishment request was processed. After this, the corresponding processing function 's7_get_initiate_cnf()' must be called for internal processing in the library.

The negotiated capability parameters (send credit, receive credit, PDU length) are entered in the mini-DB and can then be read out with mini-DB calls.

Declaration

```
int32 s7_get_initiate_cnf(void)
```

Parameters

None

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.4.3 s7_await_initiate_req

Description

The 's7_await_initiate_req()' call prepares the communications system for connection requests from the remote partner. The initiative for establishing the connection comes from the partner station and the required parameters are read from the mini-DB. They can be read, modified and adapted to the current requirements by the user program.

Declaration

```
int32 s7_await_initiate_req(
                ord32  cp_descr,    /* In call  */
                ord16  cref         /* In call  */
                          )
```

Parameters

cp_descr	Handle as return value of the 's7_init()' call.
cref	Reference to the S7 connection that will be established.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.4.4 s7_get_await_initiate_cnf

Description

The 's7_get_await_initiate_cnf()' call receives the result of the 's7_await_initiate_req()' job.

With the 's7_receive()' call, the user program receives the confirmation 'S7_AWAIT_INITIATE_CNF' if the communications system was prepared for a connection establishment. Following this, the corresponding processing function 's7_get_initiate_cnf()' must be called for internal processing in the library.

Declaration

```
int32 s7_get_await_initiate_cnf(void)
```

Parameters

Not required

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.4.5 s7_get_initiate_ind

Description

The 's7_get_initiate_ind()' call receives a request to establish an S7 connection from the remote side.

With the 's7_receive()' call, the user program receives the 'S7_INITIATE_IND' indication if the remote partner wants to establish an S7 connection. Following this, the corresponding processing function 's7_get_initiate_ind()' must be called for internal processing in the library.

The 's7_get_initiate_ind()' call enters the capability parameters of the S7 connection in the mini-DB. Following this, further events can be received from the communication system (for example via other S7 connections). An acceptance or rejection of the establishment request can be delayed with the 's7_initiate_rsp()' call.

By dividing the passive establishment into two parts the following is possible:

- With a simple call structure, the capability parameters can be checked before the connection is established and
- The establishment request can be passed on to a different application that decides whether or not the connection is established (the application has gateway functions).

Declaration

```
int32 s7_get_initiate_ind(void)
```

Parameters

None

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.4.6 s7_initiate_rsp

Description

Using the 's7_initiate_rsp()' call, the user program decides about a passive establishment request.

The passive establishment of an S7 connection is in two parts as follows:

- ➢ With the 's7_get_initiate_ind()' call, the capability parameters are entered in the mini-DB and can be read out. It is then possible to receive further events from the communications system (for example via other S7 connections).
- ➢ The establishment request is granted or rejected with the 's7_initiate_rsp()' call.

Declaration

```
int32 s7_initiate_rsp(
                ord32  cp_descr,    /* In call  */
                ord16  cref,        /* In call  */
                ord16  accept       /* In call  */
                      )
```

Parameters

cp_descr	Descriptor of the CP with which the request to establish an S7 connection was indicated.
cref	Reference of the S7 connection to be established.
accept	The establishment request can be accepted with this parameter ('accept=S7_ACCEPT') or rejected ('accept= S7_NON_ACCEPT').

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.4.7 s7_abort

Description

With the 's7_abort()' call, an existing S7 connection is aborted without confirmation and without any negotiation.

Fetching a response with 's7_receive()' is unnecessary since this is an unacknowledged service. You should also bear in mind that acknowledgments may still arrive later or may not be received at all.

Declaration

```
int32 s7_abort(
                ord32  cp_descr,    /* In call  */
                ord16  cref         /* In call  */
              )
```

Parameters

cp_descr	Handle as return value of the 's7_init()' call.
cref	Reference of the S7 connection to be aborted.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.4.8 s7_get_abort_ind

Description

The 's7_get_abort_ind()' call receives the abort of an S7 connection by a remote station or the CP.

With the 's7_receive()' call, the user program receives the 'S7_ABORT_IND' indication from the remote partner station or from the subordinate communications processor that the S7 connection was aborted. Following this, the corresponding processing function 's7_get_abort_ind()' must be called for internal processing in the library.

In S7, aborting an S7 connection is an unacknowledged service. Jobs that have not yet been acknowledged on this S7 connection will no longer be confirmed.

Declaration

```
int32 s7_get_abort_ind(void)
```

Parameters

None.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.5 Variable Services

Description of the Example

To extend the example from Section 3.4, a job to read a variable cyclically ('**s7_cycl_read()**') is sent on the established S7 connection. The received data are copied to the user memory with the function '**s7_get_cycl_read_ind()**'. The cycle is completed with '**s7_cycl_read_delete_req()**'.

Symbolic Variable Addressing

S7 variables are addressed symbolically. This type of access is oriented on the notation of S7 tools. You do not need to learn different notations for variable addresses.

Examples:

DB5,X12.1	data block 5, data byte 12, data bit 1
DB5,B12	data block 5, data byte 12
DB5,W10	data block 5, data word 10
E1.1,5	5 inputs starting at input bit 1.1 up to and including 1.5
A2.3,7	7 outputs starting at output bit 2.3 up to and including output bit 3.1
MB9,3	3 memory bytes starting at memory byte 9
DB5,W10,9	9 words in data block 5 starting at data word 10

Syntax:

The syntax is defined as follows (upper and lower case irrelevant):

DB<no>, DI<no>, <area>	<type>	<index> <index>.<bitno>	,<number>
mandatory	optional	mandatory	optional

Parameter Description:

DB or DI	data block or instance block	
<no>	number of the data block or instance block	
<area>	**A**	output
	C	counter
	E	input
	M	bit memory
	PA	peripheral output
	PE	peripheral input
	T	timer
	Z	counter

<type>	**B**	byte (unsigned)
	BYTE	byte (unsigned)
	CHAR	byte (signed)
	COUNTER	counter
	D	double word (unsigned)
	DINT	double word (signed)
	DWORD	double word (unsigned)
	INT	word (signed)
	REAL	floating point
	STRING	string
	TIMER	timer
	W	word (unsigned)
	WORD	word (unsigned)
	X	byte for DB and DI area
<index>	element number relative to start of block	
<bitno>	bit within the element number	
<number>	number of variables of one type, whose values are addressed starting at <address> in the <area>	

Example

```
:
:
/* additional prototypings */
static void my_cycl_read(ord32 cp_descr,ord16 cref,ord16 orderid);
static void my_cycl_read_delete_req( ord32 cp_descr,
                                     ord16 cref,
                                     ord16 orderid);
static void my_get_cycl_read_delete_cnf(ord32 cp_descr);
static void my_get_cycl_read_ind(ord32 cp_descr);

/* start cyclic read */
static void my_cycl_read(ord32 cp_descr,ord16 cref,ord16 orderid)
{       struct S7_READ_PARA read_para;
        int32 ret;

        read_para.access=S7_ACCESS_SYMB_ADDRESS;
        strcpy(read_para.var_name,“e0,10“);

        ret=s7_cycl_read(cp_descr,cref,orderid,200,1,&read_para);
        if(ret!=S7_OK)
        {
                my_exit(        cp_descr,
                                "Error s7_cycl_read",
                                ret);
        }
}

/* delete cyclic read */
static void my_cycl_read_delete_req( ord32 cp_descr,ord16 cref,
                                     ord16 orderid)
{       int32 ret;

        ret=s7_cycl_read_delete_req(cp_descr,cref,orderid);
        if(ret!=S7_OK)
        {
                my_exit(        cp_descr,
                                "Error s7_cycl_read",
                                ret);
        }
}

/* get cyclic read delete confirmation */
static void my_get_cycl_read_delete_cnf(ord32 cp_descr)
{       int32 ret;

        if((ret=s7_get_cycl_read_delete_cnf())!=S7_OK)
        {
                my_exit(        cp_descr,
                                "Error s7_get_cycl_read_delete_cnf",
                                ret);
        }
}
```

```
/* get cyclic read indication */
static void my_get_cycl_read_ind(ord32 cp_descr)
{       int32   ret;
        ord16   var_length=10,result;
        char    data_buffer[10];
        char    *value_array[1];

        value_array[0]=data_buffer;

        if((ret=s7_get_cycl_read_ind((void *)0,
                                     &result,
                                     &var_length,
                                     (void *)value_array))!=S7_OK)
        {
                my_exit(        cp_descr,
                                "Error s7_get_cycl_read_ind",
                                ret);
        }
}


/* receive any message from communication system */
static void my_receive(ord32 cp_descr,int32 last_event_expected)
{       ord16 cref,orderid;
        int32 ret;

        do
        {       ret=s7_receive(cp_descr,&cref,&orderid);
                switch(ret)
                {
                        :
                        :
                        case S7_INITIATE_CNF:
                                my_get_initiate_cnf(cp_descr);
                                my_cycl_read(cp_descr,cref,0);
                                break;
                        case S7_CYCL_READ_IND:
                                my_get_cycl_read_ind(cp_descr);
                                my_cycl_read_delete_req(
                                                cp_descr,
                                                cref,
                                                orderid);
                                break;
                        case S7_CYCL_READ_DELETE_CNF:
                                my_get_cycl_read_delete_cnf(
                                                        cp_descr);
                                my_abort(cp_descr,cref);
                                break;
                        default:
                                printf( "Event unexpected",
                                        ret);
                                break;
                }
        }       while(  (ret!=last_event_expected)&&
                        (ret!=S7_ABORT_IND));
}
```

```
/* main */
void main(void)
{       ord32  cp_descr;
        ord16  cref;

        /* initialize s7 */
        my_init(&cp_descr);

        /* get reference for connection 'TEST' */
        my_get_cref(cp_descr,&cref);

        /* initiate connection */
        my_initiate_req(cp_descr,cref);

        /* receive cyclic read delete confirmation */
        my_receive(cp_descr,S7_CYCL_READ_DELETE_CNF);

        /* end communication */
        my_shut(cp_descr);
}
```

Sequence Chart

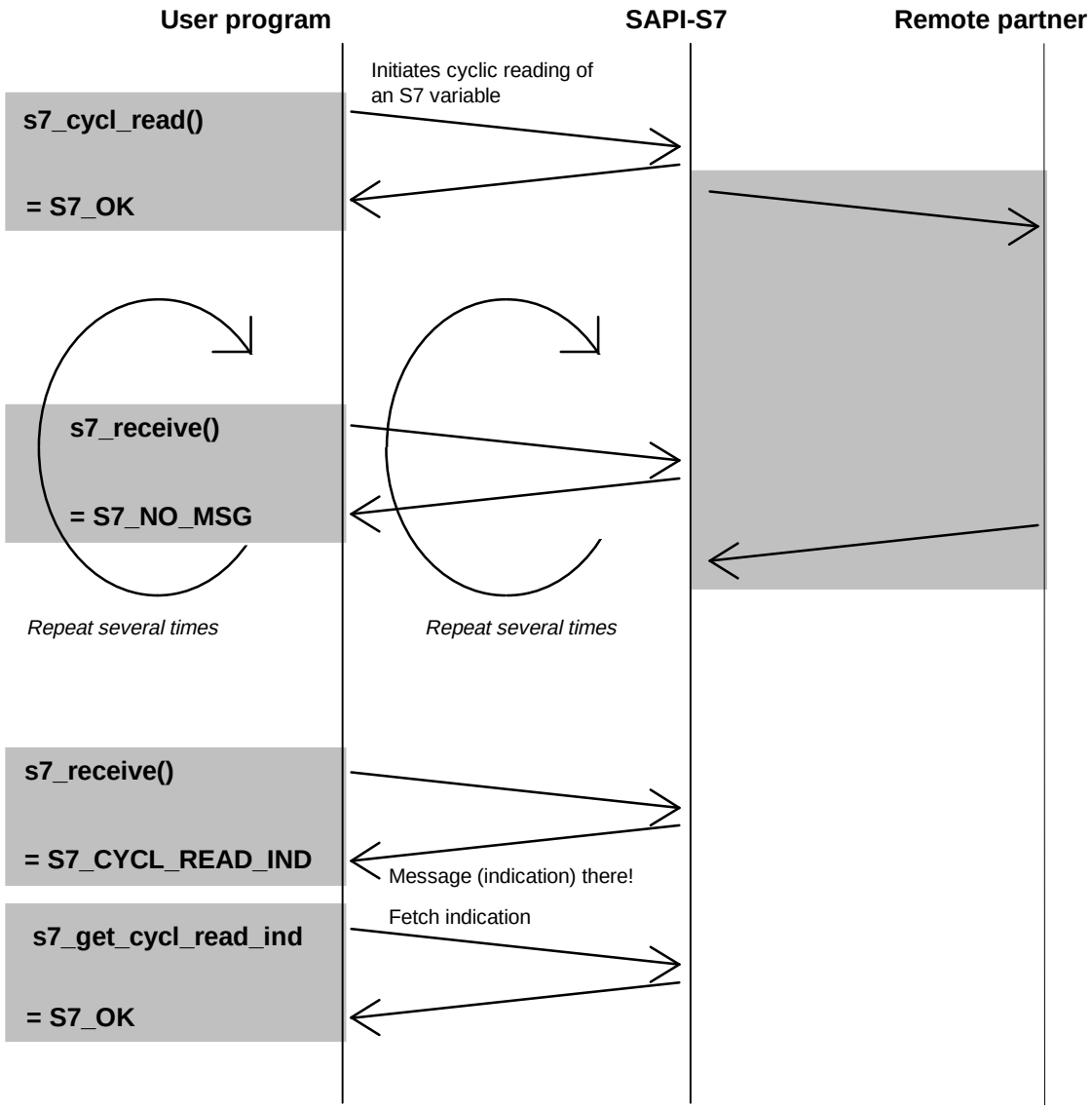

Figure 3.6: Sequence Chart of the Example

3.5.1 s7_read_req

Description

With the 's7_read_req()' call, a client application can read a variable on the server. Access to the variables is only possible using symbolic addresses. The data are transferred to the client in the confirmation from the server and entered in the user buffer by the corresponding processing function ('s7_get_read_cnf()').

Declaration

```
int32 s7_read_req(
                ord32  cp_descr,       /* In call  */
                ord16  cref,           /* In call  */
                ord16  orderid,        /* In call  */
                struct S7_READ_PARA *read_para_ptr
                                       /* In call  */

                    )
```

Parameters

cp_descr	Handle as return value of the 's7_init()' call.
cref	Reference of the S7 connection via which the job will be sent.
orderid	Job identifier to identify the job to be sent and the corresponding confirmation.

read_para_ptr Pointer to a buffer provided by the user program for the following structure:

```
struct S7_READ_PARA
{       ord16   access;
        char    var_name[S7_MAX_NAMLEN+2];
        ord16   index;
        ord16   subindex;
        ord16   address_len;
        ord8    address[S7_MAX_ADDRESSLEN];
}
```

The '**access**' parameter indicates the type of access. With 'S7_ACCESS_SYMB_ADDRESS', the symbolic address in the ‘var_name’ is expected.

The '**var_name**' parameter specifies the symbolic address of the variable to be read and is evaluated if the variable is to be accessed by its symbolic address ('access=S7_ACCESS_SYMB_ADDRESS'). (Please refer to the general information about variable addressing at the start of this section.)

The **'index'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

The **'subindex'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

The **'address_len'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

The **'address'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.5.2 s7_get_read_cnf

Description

The 's7_get_read_cnf()' call receives a variable value that has be read.

With the 's7_receive()' call, the user program receives the 'S7_READ_CNF' confirmation if the read job could be executed. Following this, the corresponding processing function 's7_get_read_cnf()' must be called for internal processing in the library so that the values can be copied to the user buffer.

Declaration

```
int32 s7_get_read_cnf(
                void *od_ptr,          /* In call  */
                ord16 *var_length_ptr,/* Call and */
                                       /* Returned */
                void   *value_ptr      /* Returned */
                       )
```

Parameters

od_ptr — The 'od_ptr' parameter is implemented to ensure compatibility with other SAPI interfaces and must be assigned the NULL pointer, in other words, the variable values are transferred on the SAPI-S7 programming interface in the network representation.

var_length_ptr — Address of a variable of the type 'ord16' provided by the user program. Here, the length of the data buffer is specified. After the call, the parameter contains the length of the variable that was read.

value_ptr — Pointer to a buffer provided by the user program. Here, the variable content that was read is saved in the network representation. When evaluating the content of the buffer, the data type of the variable must be taken into account.

It is also important to take into account that the variable values are saved byte aligned, in other words without padding bytes between two components.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.5.3 s7_write_req

Description

With the 's7_write_req()' call, a client application can write to a variable of a server. The variable can only be accessed using its symbolic address. The data are transferred in the request from the client to the server.

Declaration

```
int32 s7_write_req(
                ord32  cp_descr,       /* In call */
                ord16  cref,           /* In call  */
                ord16  orderid,        /* In call  */
                struct S7_WRITE_PARA *write_para_ptr
                                       /* In call  */
                void   *od_ptr         /* In call  */
                      )
```

Parameters

cp_descr — Handle as return value of the 's7_init()' call.

cref — Reference of the S7 connection via which the job will be sent.

orderid — Job identifier to identify the job to be sent and the corresponding confirmation.

write_para_ptr — Pointer to a buffer provided by the user program for the following structure:

```
struct S7_WRITE_PARA
{       ord16   access;
        char    var_name[S7_MAX_NAMLEN+2];
        ord16   index;
        ord16   subindex;
        ord16   address_len;
        ord8    address[S7_MAX_ADDRESSLEN];
        ord16   var_length;
        ord8    value[S7_MAX_BUFLEN];
}
```

The '**access**' parameter indicates the type of access. With 'S7_ACCESS_SYMB_ADDRESS', the symbolic address in the ‘var_name’ field is expected.

The '**var_name**' parameter specifies the symbolic address of the variable to be written and is evaluated if the variable is to be accessed by its symbolic address ('access=S7_ACCESS_SYMB_ADDRESS'). (Please refer to the general information about variable addressing at the start of this section.)

The **'index'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

The **'subindex'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

The **'address_len'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

The **'address'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

The '**var_length**' parameter specifies the number of relevant and valid bytes in the data buffer 'value'.

The '**value**' buffer contains the value of the variable to be written in the network representation. When evaluating the content of the buffer, the data type of the variable must be taken into account.

It is also important to take into account that the variable values are saved byte aligned, in other words without padding bytes between two components.

od_ptr

The 'od_ptr' parameter is implemented to ensure compatibility with other SAPI interfaces and must be assigned the NULL pointer, in other words, the variable values are transferred on the SAPI-S7 programming interface in the network representation.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.5.4 s7_get_write_cnf

Description

The 's7_get_write_cnf()' call receives the result of the write variable job.

With the 's7_receive()' call, the user program receives the 'S7_WRITE_CNF' confirmation if the write job was executed. Following this, the corresponding processing function 's7_get_write_cnf()' must be called for internal processing in the library.

Declaration

```
int32 s7_get_write_cnf(void)
```

Parameters

None

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.5.5 s7_multiple_read_req

Description

With the 's7_multiple_read_req()' call, a client application can read one or more variables on the server at the same time. Variables can only be accessed using symbolic addresses. The data are transferred in the confirmation from the server to the client and transferred to the user buffer by the corresponding processing function ('s7_get_multiple_read_cnf()').

Declaration

```
int32 s7_multiple_read_req(
                ord32  cp_descr,       /* In call  */
                ord16  cref,           /* In call  */
                ord16  orderid,        /* In call  */
                ord16  number,         /* In call  */
                struct S7_READ_PARA *read_para_array
                                       /* In call  */

                        )
```

Parameters

cp_descr	Handle as return value of the 's7_init()' call.
cref	Reference of the connection via which the job will be sent.
orderid	Job identifier to identify the job to be sent and the corresponding confirmation.
number	Number of variables to be read.
read_para_array	Pointer to an array provided by the user program with a total of 'number' elements of the following structure, where the nth element describes the nth variable:

```
struct S7_READ_PARA
{       ord16   access;
        char    var_name[S7_MAX_NAMLEN+2];
        ord16   index;
        ord16   subindex;
        ord16   address_len;
        ord8    address[S7_MAX_ADDRESSLEN];
}
```

The '**access**' parameter indicates the type of access. With 'S7_ACCESS_SYMB_ADDRESS', the symbolic address in the ‘var_name’ field is expected.

The '**var_name**' parameter specifies the symbolic address of the variable to be read and is evaluated if the variable is to be accessed by its symbolic address ('access=S7_ACCESS_SYMB_ADDRESS'). (Please refer to the general information about variable addressing at the start of this section.)

The **'index'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

The **'subindex'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

The **'address_len'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

The **'address'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

Return Values	S7_OK	The function was processed without errors.
	S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
	S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.5.6 s7_get_multiple_read_cnf

Description

The 's7_get_multiple_read_cnf()' call receives the variable values that have been read.

With the 's7_receive()' call, the user program receives the 'S7_MULTIPLE_READ_CNF' confirmation if the read job was executed. Following this, the corresponding processing function 's7_get_multiple_read_cnf()' must be called for internal processing in the library. The call copies the values that were read into the user buffer.

Declaration

```
int32 s7_get_multiple_read_cnf(
                void   *od_ptr,        /* In call  */
                ord16  *result_array,  /* Returned */
                ord16  *var_length_array,
                                       /* Call and */
                                       /* Returned */
                void   *value_array    /* Returned */
                                    )
```

Parameters

od_ptr

The 'od_ptr' parameter is implemented to ensure compatibility with other SAPI interfaces and must be assigned the NULL pointer, in other words, the variable values are transferred on the SAPI-S7 programming interface in the network representation.

result_array

Address of an array of the type 'ord16' provided by the user program. The array must contain at least as many elements as variables that were read. The array elements contain the results of the read job in the order in which the variables were specified in the request. The following results are possible:

S7_RESULT_OK

This value indicates that the access was possible and no error occurred.

S7_RESULT_HW_ERROR

A hardware problem occurred.

S7_RESULT_OBJ_ACCESS_DENIED

Access to a variable was denied.

S7_RESULT_OBJ_ADDRESS_INVALID

The specified address is invalid.

S7_RESULT_OBJ_TYPE_NOT_SUPPORTED

The server does not support the data type.

S7_RESULT_OBJ_TYPE_INCONSISTENT

The data type of the variable is not consistent.

S7_RESULT_OBJ_NOT_EXISTS

The variable does not exist.

var_length_array Address of an array of the type 'ord16' provided by the user program. The array must contain at least as many elements as variables that were read. The individual array elements contain the length of the data buffer. After the call, the elements of this array contain the lengths of the variables that were read. The value '0' means that the corresponding variable could not be read.

value_array Pointer to buffers provided by the user program. The variable values that were read are entered in the buffers. Once again the order is the same as specified in the request. When evaluating the buffer contents, the data type must be taken into account.

It is also important to take into account that the variable values are saved byte aligned, in other words without padding bytes between two components.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.5.7 s7_multiple_write_req

Description

With the 's7_multiple_write_req()' call, a client application can write to one or more variables of a server at the same time. The variables can only be accessed using their symbolic addresses. The data are transferred in the request from the client to the server.

Declaration

```
int32 s7_multiple_write_req(
                ord32  cp_descr,       /* In call  */
                ord16  cref,           /* In call  */
                ord16  orderid,        /* In call  */
                ord16  number,         /* In call  */
                struct S7_WRITE_PARA *write_para_array,
                                       /* In call  */
                void   *od_ptr         /* In call  */
                                )
```

Parameters

cp_descr	Handle as return value of the 's7_init()' call.
cref	Reference of the S7 connection via which the job will be sent.
orderid	Job identifier to identify the job to be sent and the corresponding confirmation.
number	Number of variables to be written.
write_para_array	Pointer to an array provided by the user program with a total of 'number' elements of the following structure, where the nth element describes the nth variable:

```
struct S7_WRITE_PARA
{       ord16   access;
        char    var_name[S7_MAX_NAMLEN+2];
        ord16   index;
        ord16   subindex;
        ord16   address_len;
        ord8    address[S7_MAX_ADDRESSLEN];
        ord16   var_length;
        ord8    value[S7_MAX_BUFLEN];
}
```

The '**access**' parameter indicates the type of access. With 'S7_ACCESS_SYMB_ADDRESS', the symbolic address in the ‘var_name’ field is expected.

The '**var_name**' parameter specifies the symbolic address of the variable to be read and is evaluated if the variable is to be accessed by its symbolic address ('access=S7_ACCESS_SYMB_ADDRESS'). (Please refer to the general information about variable addressing at the start of this section.)

The **'index'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

The **'subindex'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

The **'address_len'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

The **'address'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

The '**var_length**' parameter specifies the number of relevant and valid bytes in the data buffer 'value'.

The '**value**' buffer contains the value of the variable to be written in the network representation. When evaluating the content of the buffer, the data type of the variable must be taken into account.

It is also important to take into account that the variable values are saved byte aligned, in other words without padding bytes between two components.

od_ptr

The 'od_ptr' parameter is implemented to ensure compatibility with other SAPI interfaces and must be assigned the NULL pointer, in other words, the variable values are transferred on the SAPI-S7 programming interface in the network representation.

Return Values	S7_OK	The function was processed without errors.
	S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
	S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.5.8 s7_get_multiple_write_cnf

Description

The 's7_get_multiple_write_cnf()' call receives the result of the write variable job.

With the 's7_receive()' call, the user program receives the 'S7_MULTIPLE_WRITE_CNF' confirmation if the write job was executed. Following this, the corresponding processing function 's7_get_multiple_write_cnf()' must be called for internal processing in the library.

Declaration

```
int32 s7_get_multiple_write_cnf(
                ord16 *result_array    /* Returned */
                                    )
```

Parameters

result_array

Address of an array of the type 'ord16' provided by the user program. The array must contain at least as many elements as variables that were read. The array elements contain the results of the read job in the order in which the variables were specified in the request. The following results are possible:

S7_RESULT_OK

This value indicates that the access was possible and no error occurred.

S7_RESULT_HW_ERROR

A hardware problem occurred.

S7_RESULT_OBJ_ACCESS_DENIED

Access to a variable was denied.

S7_RESULT_OBJ_ADDRESS_INVALID

The specified address is invalid.

S7_RESULT_OBJ_TYPE_NOT_SUPPORTED

The server does not support the data type.

S7_RESULT_OBJ_TYPE_INCONSISTENT

The data type of the variable is not consistent.

S7_RESULT_OBJ_NOT_EXISTS

The variable does not exist.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.5.9 s7_cycl_read_init_req

Description

With this request, an S7 application instructs the server to prepare for cyclic reading of variables. The request contains the cycle time, the number of variables and the variables to be read.

Declaration

```
int32 s7_cycl_read_init_req(
                ord32  cp_descr,       /* In call  */
                ord16  cref,           /* In call  */
                ord16  orderid,        /* In call  */
                ord16  cycl_time,      /* In call  */
                ord16  number,         /* In call  */
                struct S7_READ_PARA *read_para_array
                                       /* In call  */
                              )
```

Parameters

cp_descr	Handle as return value of the 's7_init()' call
cref	Reference of the S7 connection via which the job will be sent. The variable values are transferred on this connection.
orderid	Job identifier to identify the job to be sent and the corresponding confirmation. This job identifier must be used for further jobs such as start (‘s7_cycl_read_start_req()’), stop (‘s7_cycl_read_stop_req()’) and delete (‘s7_cycl_read_delete_req()’) the cyclic reading.
cycl_time	Cycle time as a multiple of 10ths of seconds. SAPI-S7 rounds down to a permitted value (1, 2 to 9 and 10, 20 to 90 and 100, 200 to 900.
number	Number of variables whose values will be read cyclically.

read_para_array Pointer to an array provided by the user program with a total of 'number' elements of the following structure, where the nth element describes the nth variable:

```
struct S7_READ_PARA
{       ord16   access;
        char    var_name[S7_MAX_NAMLEN+2];
        ord16   index;
        ord16   subindex;
        ord16   address_len;
        ord8    address[S7_MAX_ADDRESSLEN];
}
```

The '**access**' parameter indicates the type of access. With 'S7_ACCESS_SYMB_ADDRESS', the symbolic address in the ‘var_name’ field is expected.

The '**var_name**' parameter specifies the symbolic address of the variable to be read and is evaluated if the variable is to be accessed by its symbolic address ('access=S7_ACCESS_SYMB_ADDRESS'). (Please refer to the general information about variable addressing at the start of this section.)

The **'index'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

The **'subindex'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

The **'address_len'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

The **'address'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.5.10 s7_get_cycl_read_init_cnf

Description

The 's7_get_cycl_read_init_cnf()' call receives the result of a 's7_cycl_read_init_req()' job.

With the 's7_receive()' call, the user program receives the 'S7_CYCL_READ_INIT_CNF' confirmation if the remote partner executed the job to prepare for cyclic reading of a variable. Following this, the corresponding processing function 's7_get_cycl_read_init_cnf()' must be called for internal processing in the library.

Declaration

```
int32 s7_get_cycl_read_init_cnf(void)
```

Parameters

Not required

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.5.11 s7_cycl_read_start_req

Description

With this request, an S7 application instructs the server to start cyclic reading of variables. The server must already have been prepared with the 's7_cycl_read_init_req()' call.

Declaration

```
int32 s7_cycl_read_start_req(
                ord32   cp_descr,     /* In call  */
                ord16   cref,         /* In call  */
                ord16   orderid       /* In call  */
                                )
```

Parameters

cp_descr	Handle as return value of the 's7_init()' call. This parameter must match the corresponding parameter of the 's7_cycl_read_init_req()' call.
cref	Reference of the connection via which the job will be sent. This parameter must match the corresponding parameter of the 's7_cycl_read_init_req()' call.
orderid	Job identifier to identify the job to be sent and the corresponding confirmation. This parameter must match the corresponding parameter of the 's7_cycl_read_init_req()' call.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.5.12 s7_get_cycl_read_start_cnf

Description

The 's7_get_cycl_read_start_cnf()' call receives the result of a 's7_cycl_read_start_req()' job.

With the 's7_receive()' call, the user program receives the 'S7_CYCL_READ_START_CNF' confirmation if the remote partner executed the job to start cyclic reading of the variable. Following this, the corresponding processing function 's7_get_cycl_read_start_cnf()' must be called for internal processing in the library.

Declaration

```
int32 s7_get_cycl_read_start_cnf(void)
```

Parameters

Not required

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.5.13 s7_get_cycl_read_ind

Description

The 's7_get_cycl_read_ind()' call receives the data sent by the server.

With the 's7_receive()' call, the user program receives the 'S7_CYCL_READ_IND' indication if a remote partner read a variable cyclically. Following this, the corresponding processing function 's7_get_cycl_read_ind()' is called to copy the values that were read into the user buffer.

Declaration

```
int32 s7_get_cycl_read_ind(
                void   *od_ptr,        /* In call  */
                ord16  *result_array,  /* Returned */
                ord16  *var_length_array,
                                       /* Call and */
                                       /* Returned */
                void   *value_array    /* Returned */
                      )
```

Parameters

od_ptr

The 'od_ptr' parameter is implemented to ensure compatibility with other SAPI interfaces and must be assigned the NULL pointer, in other words, the variable values are transferred on the SAPI-S7 programming interface in the network representation.

result_array

Address of an array of the type 'ord16' provided by the user program. The array must contain at least as many elements as variables that were read. The array elements contain the results of the read job in the order in which the variables were specified in the request. The following results are possible:

S7_RESULT_OK

This value indicates that the access was possible and no error occurred.

S7_RESULT_HW_ERROR

A hardware problem occurred.

S7_RESULT_OBJ_ACCESS_DENIED

Access to a variable was denied.

S7_RESULT_OBJ_ADDRESS_INVALID

The specified address is invalid.

S7_RESULT_OBJ_TYPE_NOT_SUPPORTED

The server does not support the data type.

S7_RESULT_OBJ_TYPE_INCONSISTENT

The data type of the variable is not consistent.

S7_RESULT_OBJ_NOT_EXISTS

The variable does not exist.

var_length_array Address of an array of the type 'ord16' provided by the user program. The array must contain at least as many elements as variables that were read. The individual array elements contain the length of the data buffer. After the call, the elements of this array contain the lengths of the variables that were read. The value '0' means that the corresponding variable could not be read.

value_array Pointer to buffers provided by the user program. The variable values that were read are entered in the buffers. Once again the order is the same as specified in the request. When evaluating the buffer contents, the data type must be taken into account.

It is also important to take into account that the variable values are saved byte aligned, in other words without padding bytes between two components.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.5.14 s7_get_cycl_read_abort_ind

Description

The 's7_get_cycl_read_abort_ind()' receives a cyclic read abort indication.

With the 's7_receive()' call, the user program receives the 'S7_CYCL_READ_ABORT_IND' indication if the cyclic reading of the variable was aborted. Following this, the corresponding processing function 's7_get_cycl_read_delete_cnf()' must be called for internal processing in the library.

Using the functions describes in the previous sections, cyclic reading of variables can be initiated again.

Declaration

```
int32 s7_get_cycl_read_abort_ind(void)
```

Parameters

Not required

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.5.15 s7_cycl_read_stop_req

Description

With this request, an S7 application instructs the server to stop cyclic reading of variables. The server must already have been requested to read variables cyclically.

Declaration

```
int32 s7_cycl_read_stop_req(
                ord32  cp_descr,    /* In call  */
                ord16  cref,        /* In call  */
                ord16  orderid      /* In call  */
                          )
```

Parameters

cp_descr	Handle as return value of the 's7_init()' call. This parameter must match the corresponding parameter of the 's7_cycl_read_init_req()'- or 's7_cycl_read()' call.
cref	Reference of the S7 connection via which the job will be sent. This parameter must match the corresponding parameter of the 's7_cycl_read_init_req()'- or 's7_cycl_read()' call.
orderid	Job identifier to identify the job to be sent and the corresponding confirmation. This parameter must match the corresponding parameter of the 's7_cycl_read_init_req()'- or 's7_cycl_read()' call.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.5.16 s7_get_cycl_read_stop_cnf

Description

The 's7_get_cycl_read_stop_cnf()' call receives the result of a 's7_cycl_read_stop_req()' call.

With the 's7_receive()' call, the user program receives the 'S7_CYCL_READ_STOP_CNF' confirmation if the remote partner executed the job to stop cyclic reading of variables. Following this, the corresponding processing function 's7_get_cycl_read_stop_cnf()' must be called for internal processing in the library.

Declaration

```
int32 s7_get_cycl_read_stop_cnf(void)
```

Parameters

Not required

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.5.17 s7_cycl_read_delete_req

Description This function stops cyclic reading and logs off at the server.

Declaration

```
int32 s7_cycl_read_delete_req(
                ord32  cp_descr,    /* In call  */
                ord16  cref,        /* In call  */
                ord16  orderid      /* In call  */
                              )
```

Parameters

cp_descr	Handle as return value of the 's7_init()' call. This parameter must match the corresponding parameter of the 's7_cycl_read_init_req()'- or ‘s7_cycl_read()’ call.
cref	Reference of the S7 connection via which the job will be sent. This parameter must match the corresponding parameter of the 's7_cycl_read_init_req()'- or ‘s7_cycl_read()’ call.
orderid	Job identifier to identify the job to be sent and the corresponding confirmation. This parameter must match the corresponding parameter of the 's7_cycl_read_init_req()'- or ‘s7_cycl_read()’ call.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.5.18 s7_get_cycl_read_delete_cnf

Description

The 's7_get_cycl_read_delete_cnf()' call receives the result of a 's7_cycl_read_delete_req()' job.

With the 's7_receive()' call, the user program receives the 'S7_CYCL_READ_DELETE_CNF' confirmation if the remote partner executed the job to delete cyclic reading of variables. Following this, the corresponding processing function 's7_get_cycl_read_delete_cnf()' must be called for internal processing in the library.

Declaration

```
int32 s7_get_cycl_read_delete_cnf(void)
```

Parameters

Not required

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.5.19 s7_cycl_read

Description

With this job, an S7 application instructs the server to read variables cyclically. This job includes the sequence consisting of the calls 's7_cycl_read_init_req()' (logon at server) and 's7_cycl_read_start_req()' (start reading variables). The cycle time, the number of variables and the variables to be read are specified.

Important: in S7, this is an unacknowledged service (in contrast to the individual jobs that make up this call).

Declaration

```
int32 s7_cycl_read(
                ord32  cp_descr,       /* In call  */
                ord16  cref,           /* In call  */
                ord16  orderid,        /* In call  */
                ord16  cycl_time,      /* In call  */
                ord16  number,         /* In call  */
                struct S7_READ_PARA *read_para_array
                                       /* In call  */

                )
```

Parameters

cp_descr	Handle as return value of the 's7_init()' call
cref	Reference of the S7 connection via which the job will be sent. The variable values are transferred on this connection.
orderid	Job identifier to identify the job to be sent and the corresponding confirmation. This job identifier must be used for further jobs such as stop (‘s7_cycl_read_stop_req()’), start (‘s7_cycl_read_start_req()’) and delete (‘s7_cycl_read_delete_req()’) cyclic reading.
cycl_time	Cycle time as a multiple of 10ths of seconds. SAPI-S7 rounds down to a permitted value (1, 2 to 9 and 10, 20 to 90 and 100, 200 to 900.
number	Number of variables whose values will be read cyclically.

read_para_array Pointer to an array provided by the user program with a total of 'number' elements of the following structure, where the nth element describes the nth variable:

```
struct S7_READ_PARA
{       ord16   access;
        char    var_name[S7_MAX_NAMLEN+2];
        ord16   index;
        ord16   subindex;
        ord16   address_len;
        ord8    address[S7_MAX_ADDRESSLEN];
}
```

The '**access**' parameter indicates the type of access. With 'S7_ACCESS_SYMB_ADDRESS', the symbolic address in the ‘var_name’ field is expected.

The '**var_name**' parameter specifies the symbolic address of the variable to be read and is evaluated if the variable is to be accessed by its symbolic address ('access=S7_ACCESS_SYMB_ADDRESS'). (Please refer to the general information about variable addressing at the start of this section.)

The **'index'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

The **'subindex'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

The **'address_len'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

The **'address'** parameter is irrelevant and only implemented for reasons of compatibility with other SAPI interfaces.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.6 VFD Services

Description of the Example

To extend the example from Section 3.5, after stopping the cyclic reading of variables, the status of the remote VFD is queried. The status provides information about whether or not the remote communication partner is ready for operation.

Example

```
:
:
/* additional prototypings */
static void my_get_vfd_state_cnf(ord32 cp_descr);
static void my_get_vfd_ustate_ind(ord32 cp_descr);
static void my_vfd_state_req(ord32 cp_descr,ord16 cref);

/* get vfd state confirmation */
static void my_get_vfd_state_cnf(ord32 cp_descr)
{       ord16 log_state,phy_state;
        ord8  local_detail[3];
        int32 ret;

        ret=s7_get_vfd_state_cnf(     &log_state,
                                      &phy_state,
                                      local_detail);
        if(ret!=S7_OK)
        {
                my_exit(       cp_descr,
                               "Error s7_get_vfd_state_cnf",
                               ret);
        }
}

/* get unsolicited state indication */
static void my_get_vfd_ustate_ind(ord32 cp_descr)
{       ord16 log_state,phy_state;
        ord8  local_detail[3];
        int32 ret;

        ret=s7_get_vfd_ustate_ind(    &log_state,
                                      &phy_state,
                                      local_detail);
        if(ret!=S7_OK)
        {
                my_exit(       cp_descr,
                               "Error s7_get_vfd_ustate_ind",
                               ret);
        }
}

/* send vfd state request */
static void my_vfd_state_req(ord32 cp_descr,ord16 cref)
{       int32  ret;

        ret=s7_vfd_state_req(
                        cp_descr,             /* cp_descr     */
                        cref,0                /* cref,orderid */);
        if(ret!=S7_OK)
        {
                my_exit(       cp_descr,
                               "Error s7_vfd_state_req",
                               ret);
        }
}
```

```
/* receive any message from communication system */
static void my_receive(ord32 cp_descr,int32 last_event_expected)
{       ord16 cref,orderid;
        int32 ret;

        do
        {       ret=s7_receive(cp_descr,&cref,&orderid);
                switch(ret)
                {
                        :
                        :
                        case S7_CYCL_READ_DELETE_CNF:
                                my_get_cycl_read_delete_cnf(
                                                cp_descr);
                                my_vfd_state_req(cp_descr,cref);
                                break;
                        case S7_VFD_STATE_CNF:
                                my_get_vfd_state_cnf(cp_descr);
                                my_abort(cp_descr,cref);
                                break;
                        case S7_VFD_USTATE_IND:
                                my_get_vfd_ustate_ind(cp_descr);
                                break;
                        default:
                                printf( "Event unexpected",
                                        ret);
                        break;
                }
        }       while(  (ret!=last_event_expected)&&
                        (ret!=S7_ABORT_IND));
}

/* main */
void main(void)
{       ord32   cp_descr;
        ord16   cref;

        /* initialize s7 */
        my_init(&cp_descr);

        /* get reference for connection 'TEST' */
        my_get_cref(cp_descr,&cref);

        /* initiate connection */
        my_initiate_req(cp_descr,cref);

        /* receive vfd state confirmation */
        my_receive(cp_descr,S7_VFD_STATE_CNF);

        /* end communication */
        my_shut(cp_descr);
}
```

Sequence Chart

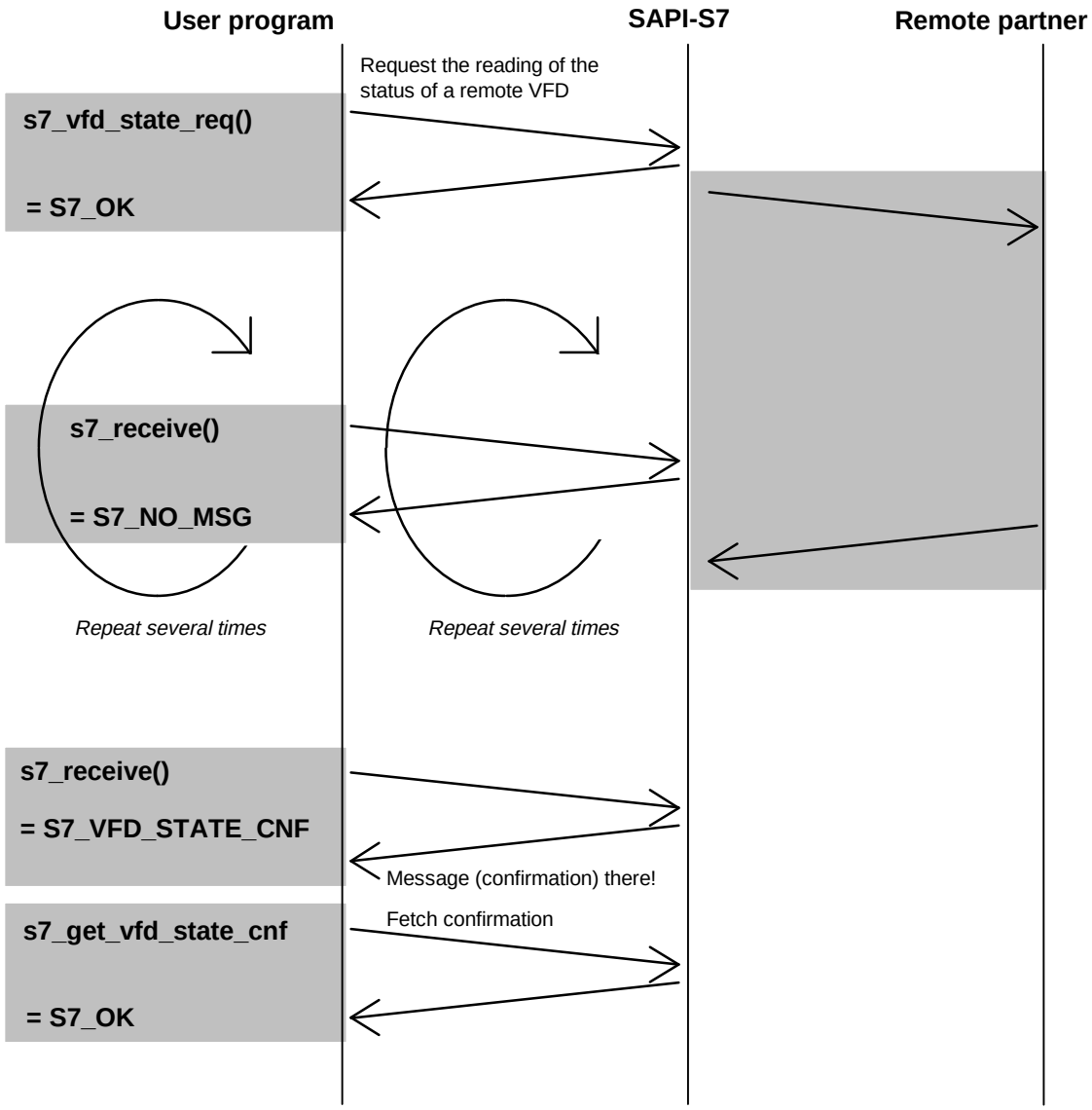

Figure 3.7: Sequence Chart of the Example

3.6.1 s7_vfd_state_req

Description

With the 's7_vfd_state_req()' call, a client application can read the logical and physical status of the local or a different (remote) virtual field device (VFD).

Declaration

```
int32 s7_vfd_state_req(
                ord32  cp_descr,    /* In call  */
                ord16  cref,        /* In call  */
                ord16  orderid      /* In call  */
                       )
```

Parameters

cp_descr	Handle as return value of the 's7_init()' call.
cref	Reference of the S7 connection on which the job will be sent.
orderid	Job identifier to identify the job to be sent and the corresponding confirmation.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.6.2 s7_get_vfd_state_cnf

Description

The 's7_get_vfd_state_cnf()' call receives the result of a VFD status job.

With the 's7_receive()' call, the user program receives the 'S7_VFD_STATE_CNF' confirmation if the status job was executed. Following this, the corresponding processing function 's7_get_vfd_state_cnf()' must be called for internal processing in the library.

With the 's7_get_vfd_state_cnf()' call, the values that were read (the physical and logical status of the VFD and the local status of the application) are copied to the user buffer.

Declaration

```
int32 s7_get_vfd_state_cnf(
                ord16  *log_state_ptr,
                                /* Returned */
                ord16  *phy_state_ptr,
                                /* Returned */
                ord8   *local_detail_ptr
                                /* Returned */

                     )
```

Parameters

log_state_ptr	Address of a variable of the type 'ord16' provided by the user program. The logical status of the VFD is entered here. This parameter specifies which services can currently be used. 'S7_STATE_CHANGES_ALLOWED' is the only possible state meaning that all services are permitted.
phy_state_ptr	Address of a variable of the type 'ord16' provided by the user program. The physical status of the VFD is entered here. The value of this parameter is derived from the states of the resources. If 'S7_OPERATIONAL' is set, the VFD is completely functional. In the status 'S7_NEEDS_COMMISSIONING' local intervention is necessary to change to an operable status.
local_detail_ptr	Pointer to a buffer provided by the user program that must have a minimum size of 3 bytes. The local status of the application and device are entered here. The meaning of the 3 bytes depends on the particular VFD and can be found in the description of the VFD.

Return Values	S7_OK	The function was processed without errors.
	S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
	S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

3.6.3 s7_get_vfd_ustate_ind

Description

The 's7_get_vfd_ustate_ind()' call receives an unsolicited status indication from the server.

With the 's7_receive()' call, the user program receives the 'S7_VFD_USTATE_IND' indication if the remote partner reports its status without being solicited. Following this, the corresponding processing function 's7_get_vfd_ustate_ind()' must be called for internal processing in the library.

With the 's7_get_vfd_ustate_ind()' call, the logical and physical status of the VFD are transferred to the user buffer.

Declaration

```
int32 s7_get_vfd_ustate_ind(
                ord16  *log_state_ptr,
                                /* Returned */
                ord16  *phy_state_ptr,
                                /* Returned */
                ord8   *local_detail_ptr
                                /* Returned */
                      )
```

Parameters

log_state_ptr	Address of a variable of the type ‘ord16’ provided by the user program. The logical status of the VFD is entered here. This parameter specifies which services can currently be used. 'S7_STATE_CHANGES_ALLOWED' is the only possible state meaning that all services are permitted.
phy_state_ptr	Address of a variable of the type 'ord16' provided by the user program. The physical status of the VFD is entered here. The value of this parameter is derived from the states of the resources. If 'S7_OPERATIONAL' is set, the VFD is completely functional. In the status 'S7_NEEDS_COMMISSIONING' local intervention is necessary to change to an operable status.
local_detail_ptr	Pointer to a buffer provided by the user program that must have a minimum size of 3 bytes. The local status of the application and device are entered here. The meaning of the 3 bytes depends on the particular VFD and can be found in the description of the VFD.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

Notes

4 Trace and Mini-DB

In this chapter, you will learn how to use the trace and to call the mini-DB. The chapter explains the following:

- ➢ How to enable entries in the library's own trace.
- ➢ How to check or modify settings in the mini-DB.
- ➢ How to obtain information about the last error that occurred.

When you have worked through this chapter you will be in a position to do the following:

- ➢ Use all the functions provided by S7 for your application while retaining a simple SAPI-S7 programming interface.
- ➢ Recognize and eliminate errors using your application.

4.1 s7_trace

Description

With this call, the user can make entries in the trace of the S7 library. This makes it possible to save important data for analysis, to check the program sequence or to synchronize with the trace entries.

Declaration

```
void s7_trace(
             char       *msg        /* In call */
              )
```

Parameters

msg	String with the user message to be entered in the trace. A trace line can contain up to 78 characters of which, however, 14 characters are reserved for the time since the trace was initialized and the line number. This leaves a net data length of 64 characters per trace call. Longer strings are truncated.

Return Value

None

4.2 s7_write_trace_buffer

Description

With high-performance applications, writing the trace to a file is inefficient. Nevertheless, entries should be available in the trace, for example if errors occur. For this reason, it is possible to configure the trace using the mini-DB so that all the trace entries are made in an internal ring buffer. The total information can then be written to a file with the ‘s7_write_trace_buffer()’ function and is therefore available for error analysis and evaluation.

Declaration

```
void s7_write_trace_buffer(
                char   *filename     /* In call  */
                                )
```

Parameters

filename	Name of the file to which the internal ring buffer will be written.

Return Value

None

4.3 s7_mini_db_set

Description

With this call, settings in the mini-DB are overwritten to be able to adapt the establishment of an S7 connection, the trace and querying errors in a wide range of differing situations. There is a limited number of combinations of data and values as described below.

Declaration

```
int32 s7_mini_db_set(
                ord16   type,       /* In call  */
                char    *value      /* In call  */
                     )
```

Parameters

type	Identifier for the setting to be modified. The possible transfer values are described below.
value	New value for the setting to be modified. The value is always transferred as a string. Defines are available for this in the header file 'sapi_s7.h' that correspond to the numbers as ASCII character strings. If you require combinations of individual values, you must first convert the individual defines into integers or perform logic operations on them and then reconvert the result to a string.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

Combinations of Values for the Trace

The trace is a simple but effective aid to debugging for the S7 library. It can be adapted to an extremely wide range of applications. The permitted combinations of values are described below starting with the ‘type’ parameter.

S7_MINI_DB_TRACE_FILENAME

This parameter value specifies the name of the trace file. The file name is transferred as 'value' (default: 's7trace.txt' in the current working directory).

S7_MINI_DB_TRACE_TARGET

This value specifies the target for the trace.

Parameter 'value'	Description
S7_TRACE_TARGET_BUFFER	The trace entries are written to an internal ring buffer (default setting).
S7_TRACE_TARGET_OLD_FILE	The trace entries are written to a file. Any trace file that already exists remains unmodified. Trace entries that follow are appended to the trace file.
S7_TRACE_TARGET_NEW_FILE	The trace entries are written to a file. A new file is created. Subsequent trace entries are appended. It is possible with this setting to avoid the trace file becoming far too large.
S7_TRACE_TARGET_CONSOLE	The trace entries are passed on to another application. What then takes place depends on the operating system.

S7_MINI_DB_TRACE_DEPTH

This sets the trace depth.

Parameter 'value'	Description
S7_TRACE_DEPTH_OFF	Switches the trace off.
S7_TRACE_DEPTH_USER	With this setting (default) only the strings specified by the user program with the 's7_trace()' function are entered in the trace.
S7_TRACE_DEPTH_EXCEPT	This setting only allows trace entries following error events or incorrect results.
S7_TRACE_DEPTH_INTERFACE	With this setting, parameters transferred to the SAPI-S7 programming interface are entered in the trace. This allows incorrect parameters to be detected quickly and without using a debugger.
S7_TRACE_DEPTH_OTHER	This setting provides further information.

S7_MINI_DB_TRACE_SELECT

To be able to activate the trace for specific service classes, a define is available in the header file 'sapi_s7.h' for each service class. The defines can be combined as explained above.

Parameter 'value'	Description
S7_TRACE_SELECT_ADMIN_SERVICES	Activates the trace for the administrative services.
S7_TRACE_SELECT_CONN_SERVICES	Activates the trace for the S7 connection management services,
S7_TRACE_SELECT_VAR_SERVICES	Activates the trace for the variable services.
S7_TRACE_SELECT_CYCL_VAR_SERVICES	Activates the trace for the cyclic variable services.
S7_TRACE_SELECT_RECEIVE_SERVICES	Activates the trace for the receive call.
S7_TRACE_SELECT_VFD_SERVICES	Activates the trace for the VFD services.
S7_TRACE_SELECT_OTHER_SERVICES	Activates the trace for functions accessing the mini-DB.
S7_TRACE_SELECT_ALL	Activates the trace for all service classes.

S7_MINI_DB_TRACE_NO_LINES

This parameter is used to modify the number of lines in the internal ring buffer. For applications that require a lot of memory, the ring buffer can be reduced in size or increased in size to record the history when errors occur. The function ‘s7_write_trace_buffer()’ allows the ring buffer to be written to a file.

Changing this parameter is only effective if the change precedes the first S7 library call.

Possible Combinations of Values for Establishing S7 Connections

For the S7 connection establishment, it is possible to assign defaults for various connection parameters negotiated by the stations. The permitted combinations of values are described below starting with the 'type' parameter.

S7_MINI_DB_INIT_REQ_AMQ_CALLING

This value specifies how many acknowledged jobs can be received at the same time on the connection by the active partner (default: ‘3’). The value is a proposal that can be accepted or reduced by the partner station. The negotiated value can be read out by the function ‘s7_mini_db_get()’.

S7_MINI_DB_INIT_REQ_AMQ_CALLED

This value specifies how many acknowledged jobs can be sent at the same time on the connection by the active partner (default: ‘3’). The value is a proposal that can be accepted or reduced by the partner station. The negotiated value can be read out by the function ‘s7_mini_db_get()’.

S7_MINI_DB_INIT_REQ_PDU_SIZE

This value specifies the maximum size of a PDU on this connection for the active partner (default: ‘0x100’). The value is a proposal that can be accepted or reduced by the partner station. The negotiated value can be read out by the function ‘s7_mini_db_get()’.

S7_MINI_DB_INIT_RSP_AMQ_CALLING

This value specifies how many acknowledged jobs can be sent at the same time on the connection by the passive partner (default: ‘3’). The lower of the values set for the active or passive side is negotiated.

S7_MINI_DB_INIT_RSP_AMQ_CALLED

This value specifies how many acknowledged jobs can be received at the same time on the connection by the passive partner (default: ‘3’). The lower of the values set for the active or passive side is negotiated.

S7_MINI_DB_INIT_RSP_PDU_SIZE

This value specifies the maximum size of a PDU on this connection for the passive partner (default: ‘0x100’). The lower of the values set for the active or passive side is negotiated.

S7_MINI_DB_PERSISTANCE_COUNT

This value sets the number of attempts at active connection establishment (default: ‘5’). When the partner station has rejected the establishment request this number of times, the establishment is terminated and a negative acknowledgment sent to the user program.

S7_MINI_DB_ABORT_TIMEOUT

This value specifies how long a station can attempt to establish a connection if the remote station does not respond. The value is set in multiples of 51 ms (default: ‘300’). The parameter applies both to the connection establishment and to the data transfer phase.

4.4 s7_mini_db_get

Description

With this call, the settings are read out of the mini-DB. The user specifies an identifier for the setting to be read and receives a string that must be interpreted depending on the identifier.

Declaration

```
const char *s7_mini_db_get(
             ord16      type        /* In call  */
                          )
```

Parameters

type — Identifier for the data to be read. The possible values are described below.

Return Values for Trace Settings

The mini-DB allows all the modifiable settings of the trace to be read at any time. These are as follows:

S7_MINI_DB_TRACE_FILENAME

With this parameter, the name of the trace file is returned.

S7_MINI_DB_TRACE_TARGET

With this parameter, the target of the trace is output.

S7_MINI_DB_TRACE_DEPTH

With this parameter, the trace depth can be queried.

S7_MINI_DB_TRACE_SELECT

With this parameter, the service classes that were activated for the trace are indicated.

The return values correspond to the specified values in the 's7_mini_db_set()' call.

Return Values for Establishing an S7 Connection

After receiving an initiate confirmation, the performance parameters negotiated between client and server are entered in the mini-DB by the corresponding processing function 's7_get_initiate_cnf()'.

S7_MINI_DB_INIT_IND_AMQ_CALLING

After receiving the initiate indication, this value informs the passive partner how many jobs the active partner on the connection can receive at the same time.

S7_MINI_DB_INIT_IND_AMQ_CALLED

After receiving the initiate indication, this value informs the passive partner how many jobs the active partner on the connection can send at the same time.

S7_MINI_DB_INIT_IND_PDU_SIZE

After receiving the initiate indication, this value informs the passive partner how much data the active partner can receive on this connection.

S7_MINI_DB_INIT_CNF_AMQ_CALLING

After active connection establishment, this value indicates the number of acknowledged jobs that can be received at the same time on this connection. The value was negotiated by the partners when the connection was established.

S7_MINI_DB_INIT_CNF_AMQ_CALLED

After active connection establishment, this value indicates the number of acknowledged jobs that can be sent at the same time on this connection. The value was negotiated by the partners when the connection was established.

S7_MINI_DB_INIT_CNF_PDU_SIZE

After active connection establishment, this value indicates the maximum PDU size on this connection. The value was negotiated by the partners when the connection was established.

4.5 s7_last_iec_err_no

Description The error identifiers are reduced to a practical number by the S7 library to allow error handling to be implemented more simply in applications. According to IEC 1131 (**I**nternational **E**lectrotechnical **C**ommission), there are standard error codes that can be read out with this call

Declaration

```
ord16 s7_last_iec_err_no(void)
```

Parameters None

Return Values Possible return values and their significance:

S7_ERR_IEC_NO

No error occurred.

S7_ERR_IEC_DATA_TYPE_MISMATCH

The data types do not match.

S7_ERR_IEC_INVALID_REF

The specified S7 connection reference does not exist.

S7_ERR_IEC_LOWER_LAYER

An error occurred in the lower layers.

S7_ERR_IEC_NEG_RESPONSE

The client has received a negative response from the communications partner.

S7_ERR_IEC_NO_ACCESS_TO_REM_OBJECT

Access to an object was rejected.

S7_ERR_IEC_PARTNER_IN_WRONG_STATE

The partner station is in a status in which the requested job cannot be processed.

S7_ERR_IEC_RECEIVER_DISABLED

The server is not responding.

S7_ERR_IEC_RECEIVER_OVERRUN

The resources in the server are exhausted.

S7_ERR_IEC_RESET_RECEIVED

A reset request has been received.

4.6 s7_last_iec_err_msg

Description This call returns a string that describes the error indicated by the IEC error code. The error string can, for example, be displayed on an operator console or written to a log file.

Declaration

```
const char *s7_last_iec_err_msg(void)
```

Parameters None

4.7 s7_last_detailed_err_no

Description With this call, the caller receives an error number that provides more detailed information about the cause of the error than the standard IEC error codes.

Declaration

```
ord16 s7_last_detailed_err_no(void)
```

Parameters None

Return Values The possible return values and their significance:

S7_ERR_NO_ERROR

No error occurred.

S7_ERR_CONN_ABORTED

The S7 connection was aborted.

S7_ERR_CONN_CNF

The S7 connection could not be established.

S7_ERR_CONN_NAME_NOT_FOUND

The specified S7 connection name was not found.

S7_ERR_FW_ERROR

A firmware error has occurred on the communications processor.

S7_ERR_INSTALL

When installing the SINEC driver or initializing the communications processor an error occurred that makes communication impossible.

S7_INTERNAL_ERROR

During communication, library-internal data were overwritten making it impossible to continue operating the application.

S7_ERR_INVALID_CONN_STATE

The job that has been sent is not permitted in the current status of the S7 connection.

S7_ERR_INVALID_CREF

The specified S7 connection reference is invalid.

S7_ERR_INVALID_CYCL_READ_STATE

The job is not permitted in the current status of the cyclic read job.

S7_ERR_INVALID_DATA_SIZE

The data buffer provided by the user program is too small.

S7_ERR_INVALID_ORDERID

There is no job with the specified job identifier (parameter 'orderid').

S7_ERR_INVALID_PARAMETER

A transferred parameter or a specified value in a transferred structure is invalid.

S7_ERR_MAX_REQ

The maximum number acknowledged jobs negotiated during connection establishment has already been sent.

S7_ERR_MINI_DB_TYPE

The 'type' parameter is not permitted in a mini-DB call.

S7_ERR_MINI_DB_VALUE

The 'value' parameter is not permitted in a mini-DB call.

S7_ERR_NO_LICENCE

The license required for the product could not be found.

S7_ERR_NO_RESOURCE

The resources available are currently exhausted.

S7_ERR_NO_SIN_SERV

The SINEC server required for S7 applications under Windows that sends messages to the relevant application could not be started.

S7_ERR_OBJ_ACCESS_DENIED

Access to the required object was rejected.

S7_ERR_OBJ_ATTR_INCONSISTENT

The OD or the attributes of the addressed object are inconsistent.

S7_ERR_OBJ_UNDEFINED

The object to be accessed does not exist.

S7_ERR_ORDERID_USED

The job identifier transferred with the call (parameter 'orderid') is already being used.

S7_ERR_RECEIVE_BUFFER_FULL

A message was received however the corresponding processing function has not yet been called.

S7_ERR_SERVICE_NOT_SUPPORTED

The requested service is not supported.

S7_ERR_SERVICE_VFD_ALREADY_USED

The application or a different process has already logged on at the VFD.

S7_ERR_SYMB_ADDRESS

The symbolic address transferred in the job is incorrect.

S7_ERR_SYMB_ADDRESS_INCONSISTENT

The size of the user data contained in the symbolic address and the size of the user buffer are contradictory.

S7_ERR_TOO_LONG_DATA

There are more data to be written than permitted by the standard.

S7_ERR_UNKNOWN_ERROR

An unknown error has occurred.

S7_ERR_WRONG_CP_DESCR

The CP descriptor in the call is incorrect.

S7_ERR_WRONG_IND_CNF

The wrong processing function was called for a received message.

4.8 s7_last_detailed_err_msg

Description This call provides an error message about the detailed error number that describes the error that has occurred and provides information about eliminating the error. This is an error string that can, for example, be displayed on an operator console or written to a log file etc.

Declaration

```
const char *s7_last_detailed_err_msg(void)
```

Parameters None

4.9 s7_discard_msg

Description

With this call, the user program can discard a received message without having called the corresponding processing function.

Per S7 connection, there is a maximum number of acknowledged jobs that can be processed simultaneously, this maximum number is specified during configuration and is negotiated when the connection is established. Ignoring (for example unexpected) events and the consequent absence of responses therefore permanently reduces the number of acknowledged jobs that can be used at the same time.

Declaration

```
void s7_discard_msg(void)
```

Parameters

None

Notes

5 Configuration

In this chapter

- ➢ you will learn the meaning of configuration,
- ➢ you will obtain an overview of the configuration parameters necessary for operation,
- ➢ you will find a list of the services that use configuration parameters.

When you have worked through this chapter, you will be in a position to match your SAPI-S7 application to the configuration.

5.1 Significance of Configuration

Configuration Increases the Flexibility of Your Application

To avoid involving applications in adaptations made necessary by changes in the communications system (network), the creation and assignment of S7 connections is configured. Configuration is a standardized method of setting address parameters etc. for all applications. Generally, the installation of software and its integration in the network is not done by the software developer.

When you reconfigure a system, you must make sure that the configuration parameters relevant to the applications are retained. The name of an S7 connection, for example, must continue to exist even if a different partner station is addressed on this connection and under certain circumstances even with a different S7 connection reference. By keeping to these rules, modifications can be made using the appropriate configuration tools at any time without needing to change user programs.

5.2 Services With Configuration Data

Administrative Services

The logon function 's7_init()' requires two items of information from the administrative services:

- ➢ Firstly, the CP name that identifies a CP must be specified and must match the name selected during installation. The installation is performed with SINEC products. The installation procedures are described in the documentation accompanying these products.
- ➢ Secondly, the name of the local VFD at which the user wants to log on is required. The logon makes the VFD-specific S7 connections and the VFD-specific objects accessible to the application. The VFD name is specified during configuration.

The names of the installed CPs or the names of VFDs configured on a CP can be queried with the ‘s7_get_device()’ or ‘s7_get_vfd()’ calls.

Management Services for S7 Connection Lists

Only the 's7_get_cref()' call requires a configuration parameter from the management services for S7 connection lists in the form of the S7 connection name. This function returns the configured reference for an S7 connection. Once this reference has been obtained, the application should continue to use the reference in future communication.

The names of all configured S7 connections can be read out with the ‘s7_get_conn()’ call.

Notes

6 SAPI-S7 Under MS-DOS/Windows

This chapter explains the characteristics of the SAPI-S7 programming interface specific to MS-DOS and Windows. In this context Windows stands for the Microsoft operating systems Windows 3.x in enhanced 386 mode, Windows 95 and Windows NT if not specified otherwise.

You will learn the following:

- ➢ The compilers and memory models for which the S7 library is available under MS-DOS and Windows.
- ➢ Which compiler options are useful when translating your own applications.
- ➢ Which linker options are required when linking your program modules to the S7 library.
- ➢ How to control the trace using environment variables without having to change your application.

When you have worked through this chapter, you will be in a position to

- ➢ translate your own program modules and to link them with the S7 library to form an executable program,
- ➢ control the trace outputs of your application.

6.1 General Information

Memory Models

The SAPI-S7 programming interface is available to the user in the form of libraries. The definitions required to use the programming interface are located in the 'sapi_s7.h' file. Libraries are available for MS-DOS, Windows 3.x, Windows 95 and Windows NT for various compilers.

The names of the libraries for MS-DOS and Windows 3.x are as follows:

<Memorymodel><Operatingsystem>s7<Compiler>.lib

The following table explains the components making up the library names.

Format Name	Format Entry	Meaning
<Memory model>	l	Large model
	h	Huge model
<Operating system>	d	MS DOS
	w	Windows 3.x
<Compiler>	msc	MSC compiler 7.0
	tc	Turbo C compiler 1.0
	bc	Borland C compiler 3.1

For example, the file ‘lds7tc.lib’ is the library translated with the Turbo C compiler in the ‘Large’ memory model for the MS-DOS operating system.

For the Windows 3.x operating system, the ‘s7.dll’ file provides a DLL version (**D**ynamic **L**ink **L**ibrary) with the import libraries ‘s7msc.lib’ and ‘s7bc.lib’ for the Microsoft C or Borland C compilers.

For Windows 95 and Windows NT, a 32-bit DLL ‘s732.dll’ and the corresponding import library ‘s7msc.lib’ are available.

Alignment

Normally, compilers save variables in memory in a form that seems the most suitable to the compiler. During this procedure, gaps can occur between the components of a variable (padding bytes).

The structures available on the SAPI-S7 programming interface are designed so that they can access the individual components of user programs translated with byte or word alignment. Double word alignment is not supported by the S7 library.

Program Abort

To allow communication, user programs must log on at the communications system that occupies resources for management. If an application is aborted with the key combination 'CTRL+C' the resources still remain reserved for the process and the logon is still effective. To avoid this, a 'CTRL+C' handler should be implemented in the user program to handle all the logoffs at the communications system if the program is aborted.

6.2 Translating and Linking for MS-DOS

Requirements

The following sections list compilation examples that illustrate the required compiler and linker options for your applications.

You must adapt the generate instruction to suit the situation on your system selecting the correct drive and path.

Working With the MSC Compiler 7.0

The S7 library for the MSC Compiler 7.0 under MS-DOS has the name 'lds7msc.lib'. The following example shows how a typical program 'exp.c' is translated and linked with the 'Large' memory model for MS-DOS:

```
cl /c /AL /Os /Iinc src\exp.c
link @exp.lnk
```

The instructions for the linker are in the file ‘exp.lnk’:

```
exp.obj,
exp.exe,
exp.map,
lds7msc.lib+
\msc70\lib\oldnames.lib+
\msc70\lib\llibce.lib
;
```

Working With the MS Visual C++ Compiler 1.0

If you want to use the MS Visual C++ Compiler 1.0 for your MS-DOS applications, you can use the same S7 library as for the MSC Compiler 7.0. The compile instructions for the 'Large' memory model then appear as follows under MS-DOS:

```
cl /c /AL /Os /Iinc src\exp.c
link @exp.lnk
```

The instructions for the linker are in the file ‘exp.lnk’:

```
exp.obj,
exp.exe,
exp.map,
lds7msc.lib+
\msvc\lib\oldnames.lib+
\msvc\lib\llibce.lib
;
```

Working With the Turbo C Compiler 1.0

For the Turbo C Compiler 1.0, there are two versions of the S7 library available for MS-DOS: 'lds7tc.lib' for the 'Large' memory model and 'hds7tc.lib' for the 'Huge' memory model. The following example shows how a typical program 'exp.c' is translated and linked with the 'Large' memory model for MS-DOS:

```
tcc -c -ml -Iinc src\exp.c
tlink @exp.lnk.
```

The instructions for the linker are in the file ‘exp.lnk’:

```
\tc10\lib\c0l.obj exp.obj
exp.exe
exp.map
\tc10\lib\emu.lib \tc10\lib\mathl.lib \tc10\lib\cl.lib lds7tc.lib
```

6.3 Translating and Linking for Windows 3.x

Requirements

The following sections list compilation examples that illustrate the required compiler and linker options for your applications.

You must adapt the generate instruction to suit the situation on your system selecting the correct drive and path.

Working with the MSC Compiler 7.0

The S7 library for the MSC Compiler 7.0 under Windows 3.x has the name 'lws7msc.lib'. The following example shows how a typical program 'exp.c' is translated and linked with the 'Large' memory model for Windows 3.x:

```
cl /c /AL /Os /Iinc src\exp.c
link @exp.lnk
```

The instructions for the linker are in the file ‘exp.lnk’:

```
exp.obj,
exp.exe,
exp.map,
lws7msc.lib+
\msc70\lib\oldnames.lib+
\msc70\lib\llibcew.lib+
\msc70\lib\libw.lib
exp.def;
```

The module definition file ‘exp.def’ appears as follows:

```
NAME        EXP
EXETYPE     WINDOWS
CODE        PRELOAD MOVEABLE DISCARDABLE
DATA        PRELOAD MOVEABLE MULTIPLE
HEAPSIZE    1024
STACKSIZE   10240
EXPORTS
```

Working with the MS Visual C++ Compiler 1.0

If you want to use the MS Visual C++ Compiler 1.0 for your Windows 3.x applications, you can use the same S7 library as for the MSC Compiler 7.0. The compile instructions for the 'Large' memory model then appear as follows under MS-DOS:

```
cl /c /AL /Os /Iinc src\exp.c
link @exp.lnk
```

The instructions for the linker are in the file 'exp.lnk' :

```
exp.obj,
exp.exe,
exp.map,
lws7msc.lib+
\msvc\lib\oldnames.lib+
\msvc\lib\llibcew.lib+
\msvc\lib\libw.lib
exp.def;
```

The module definition file ‘exp.def’ appears as follows:

```
NAME        EXP
EXETYPE     WINDOWS
CODE        PRELOAD MOVEABLE DISCARDABLE
DATA        PRELOAD MOVEABLE MULTIPLE
HEAPSIZE    1024
STACKSIZE   10240
EXPORTS
```

Working with the Borland C Compiler 3.1

The S7 library for the Borland C Compiler 3.1 has the name 'lws7bc.lib' under Windows 3.x . The following example shows how a typical program 'exp.c' is translated and linked with the 'Large' memory model for Windows:

```
bcc -c -ml -Os -Iinc src\exp.c
tlink /Twe @exp.lnk
```

The instructions for the linker are in the file ‘exp.lnk’:

```
\bc31\lib\c0wl.obj exp.obj
exp.exe
exp.map
lws7bc.lib \bc31\lib\mathwl.lib \bc31\lib\import.lib
\bc31\lib\cwl.lib
exp.def
```

The module definition file ‘exp.def’ appears as follows:

```
NAME        EXP
EXETYPE     WINDOWS
CODE        PRELOAD MOVEABLE DISCARDABLE
DATA        PRELOAD MOVEABLE MULTIPLE
HEAPSIZE    1024
STACKSIZE   10240
EXPORTS
```

SAPI-S7 as DLL Version

Under the Windows operating system, in addition to the libraries listed above there is a DLL version (**D**ynamic **L**ink **L**ibrary) available (file ‘s7.dll’) and the required import libraries for the MSC Compiler 7.0 and MS Visual C++ Compiler 1.0 (file ‘s7msc.lib’) or the Borland C Compiler 3.1 (file ‘s7bc.lib’).

The compilation rules for SAPI-S7 applications that are based on the DLL version of SAPI-S7 are similar to those for applications that use the SAPI-S7 libraries. The SAPI-S7 libraries above ‘lws7msc.lib’ and ‘lws7b.lib’ must be replaced by the import libraries ‘s7msc.lib’ and ‘s7bc.lib’. In addition to this, the define ‘S7_DLL’ must be set when compiling source files that use the SAPI-S7 functions.

For using this DLL in other programming languages, such as BASIC or Pascal, refer to the corresponding manuals.

6.4 Translating and Linking for Windows 95 and Windows NT

Requirements

The following sections list compilation examples that illustrate the required compiler and linker options for your applications.

You must adapt the compile instruction to suit the situation on your system selecting the correct drive and path.

Working with the MSVC Compiler 2.2

The S7 import library for the Microsoft Visual C++ Compiler 2.2 under Windows 95 and Windows NT has the name 's7msc.lib'. The corresponding DLL has the name 's732.dll'. The following example shows how a typical program 'exp.c' is translated and linked for Windows 95. For Windows NT, simply replace the directory 'sapi_s7.w95' with 'sapi_s7.nt'.

```
cl /c /MT /W3 /GX /Zp1 /Od -DSTRICT -DWIN32 -DWINDOWS
   -I\sinec\sapi_s7.w95\include -I\msvc20\include src\exp.c
link /NODEFAULTLIB /OUT:"exp.exe" @exp.dat
```

The instructions for the linker are in the file ‘exp.dat’:

```
exp.obj,
\sinec\sapi_s7.w95\lib\s7msc.lib
\msvc20\lib\kernel32.lib
\msvc20\lib\user32.lib
\msvc20\lib\gdi32.lib
\msvc20\lib\libc.lib
/SUBSYSTEM:windows /MACHINE:I386
```

6.5 Environment Variables

What Does Environment Mean?

The term environment means a memory area in which the parameters are saved that are set with the MS-DOS commands 'path', 'prompt' and 'set'. The area consists of a series of ASCII strings that are completed by a NULL character. The area is used to hold information about the entire computer system.

Controlling the Trace

The S7 library can be controlled by a total of three environment variables. Using the variables 'S7_TRACE_SELECT', 'S7_TRACE_DEPTH' and 'S7_TRACE_TARGET' the service classes for which entries will be made in the trace, the trace depth and the target of the trace can be set (see file ‘sapi_s7.h’).

Example: set S7_TRACE_DEPTH=104

The existence of the environment variable is checked and evaluated when the trace is initialized. Trace settings made earlier are then overwritten.

It is advisable to set the trace using the environment variables and not with mini-DB calls. This allows the default values to be overwritten for debugging without the user having to modify his application and retranslate it.

Under Windows 95 and Windows NT, you can also specify the value of the variables by making entries with the names of the environment variables in the registration database under the key “HKEY_LOCAL_MACHINE\SOFTWARE\SIEMENS\SINEC\SAPI_S7”.

How to Access the Configuration

Under Windows 95 and Windows NT, an entry is made in the registration database under the key “HKEY_LOCAL_MACHINE\SOFTWARE\-SIEMENS\SINEC\SAPI_S7” for every configured CP. The name of the entry is created by appending the string ‘_S7LDB’ to the CP name. For a CP with the name ‘CP_L2_1:’, this would be ‘CP_L2_1:_S7LDB’. The value of the entry is the drive, path and the name of the configuration file. For example with the value “C:\SINEC\LDB\EXP.LDB”, the configuration data are expected in the file “C:\SINEC\LDB\EXP.LDB” when the ‘s7_init’ function is called.

Under MS-DOS and Windows 3.x, or if there is no entry in the registration database for the CP, the SAPI S7 library attempts to take the path of the configuration file from an environment variable. The name of the environment variable corresponds to the entry in the registration database under Windows 95 and Windows NT. For a CP with the name ‘CP_L2_1:’ this would be ‘CP_L2_1:_S7LDB’.

If no corresponding environment variable is found either when the ‘s7_init()’ function is called, the SAPI S7 library attempts to read out the configuration data from a file in the currently active directory.

The file name is obtained from the transfer parameter ‘cp_name’ by removing the colon completing the CP name and appending the extension ’.LDB’. If a CP with the name CP ‘CP_L2_1:’ logs on, for example, the file ‘CP_L2_1.LDB’ is read out.

6.6 The Trace for MS-DOS or Windows

General

It is possible to make a setting for a specific operating system for the trace of the SAPI-S7 programming interface. Using a mini-DB call, the target of the trace can be modified to the value 'S7_TRACE_TARGET_CONSOLE'. The way in which the trace then reacts depends on the specific operating system and is explained below.

The Trace Setting for Windows

Under Windows, the trace setting described above means that the entire trace is output on the monitor in its own window. You can then adapt the window to suit your needs. How you configure the window and the number of trace lines displayed can be found in the on-line help texts.

The Trace Setting for MS-DOS

With the trace setting above, no trace entries whatsoever are made under the MS-DOS operating system. As a single-user operating system, MS-DOS is not in a position to start a second application parallel to the S7 user program that edits and represents the trace on the monitor.

6.7 Special Features for Windows

Differences Between MS-DOS and Windows Programs

One of the differences between Windows applications and MS-DOS programs is that they receive events at a central point in the main program and pass them on to a suitable Windows procedure ('WndProc') for further processing. This Windows procedure must be made accessible to Windows management when the program is started.

Example of a Typical Windows Application

```
:
:
#define MY_MSG_ID    1500

WndProc(hWnd,msg,...)
{
        :
        :
        switch(msg)
        {
                case ....:     /* init code */
                        s7_init("CP_L2_1:","MY_VFD",&cp_descr);
                        s7_set_window_handle_msg(
                                cp_descr,hWnd,MY_MSG_ID);
                        break;

                case ....:
                        s7_....(cp_descr,...);
                        break;

                case MY_MSG_ID:
                        s7_receive(cp_descr,&cref,&orderid);
                        break;
        }
}
:
:
```

In a Windows application, following 's7_init()', the routine 's7_set_window_handle_msg()' must be called with a Windows handle and a message ID so that the underlying communications system can send a message to the application. When a frame is received, the application is informed by a message. The Windows procedure that is then called in turn calls 's7_receive()' and the appropriate processing function.

6.7.1 s7_set_window_handle_msg

Description

In a Windows program, after a successful ‘s7_init()’, the user must call the routine ‘s7_set_window_handle_msg()’. This informs the underlying communication system to which window and with which ID it should send its messages.

Declaration

```
int32 s7_set_window_handle_msg(
                ord32     cp_descr,   /* In call  */
                ord32     hWnd,       /* In call  */
                ord32     msgID       /* In call  */
                                      )
```

Parameters

cp_descr	Handle as return value of the ‘s7_init()’ call.
hWnd	Handle of the window to which the SINEC message will be sent.
msgID	ID of the SINEC message to be sent to the window specified above.

Return Values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

Notes

7 Appendix

This chapter introduces you to the following:

- ➢ Which S7 subset is covered by the SAPI-S7 library,
- ➢ Which conditions must be adhered to when operating the SAPI-S7 library.
- ➢ How S7 variables and the standard data types are represented by S7 (both on the host and on the network).

At the end of the chapter you will find a list of the most common abbreviations used in this manual and a list of documentation available for further reading.

7.1 Range of Functions of SAPI-S7

SAPI-S7 as a Subset of S7

The SAPI-S7 programming interface provides access to the following services (abbreviation: ‘req’ for Request, ‘ind’ for Indication, ‘con’ for Confirmation and ‘rsp’ for Response):

PICS Serial Number: 1 PICS Part 1 Implementation in the system	
System Parameters	Detail
Implementation’s Vendor Name	- (can be set by COML)
Implementation’s Model Name	VFD-Name (can be set by COML)
Implementation’s Revision Identifier	- (can be set by COML)
Vendor Name of S7	Siemens AG
Controller Type of S7	ASPC2
Hardware Release of S7	A_._ (can be found on type plate)
Software Release of S7	V_._
Profile Number	0
Calling S7 User (enter ‘YES’ or ‘NO’)	YES
Called S7 User (enter ‘YES’ or ‘NO’)	YES

PICS Part 2 Supported Services	
Service	Primitive
Initiate	req, con, ind, rsp
Abort	req, ind
Status	req, con
Unsolicited-Status	ind
Read	req, con
Write	req, con

PICS Part 3	
S7 Parameters and Options	Detail
Addressing by names	YES
Maximum length for names	32
Access-Protection Supported	-
Maximum length for Extension	32
Maximum length for Extension Arguments	0

PICS Part 4	
Local Implementation Values	Detail
Maximum length of S7-PDU	1024
Maximum number of Services Outstanding Calling	-
Maximum number of Services Outstanding Called	-
Syntax and semantics of the Execution Argument	-
Syntax and semantics of Extension	-

Conditions for Operation

When operating the SAPI-S7 programming interface, the following conditions must be kept to:

➢ Up to a maximum of **150** jobs can be processed **simultaneously** per application.

7.2 Representation of S7 Variables

Byte Alignment

The SAPI-S7 programming interface requires that variables are saved byte aligned in main memory. This means that there must be no padding bytes, for example between the individual components of a structure. This is achieved with suitable compiler options or by pragmas.

Network Representation and Range of Values

The SAPI-S7 library supplies data that have been read or variable values to be written in the network representation. A conversion between the host and network representation is intended for future versions of the library. For this reason, all the affected functions have been extended by a transfer parameter ‘od_ptr’ that must be assigned the NULL pointer in the current version of the library.

This parameter (‘od_ptr’) will control the representation of variables in a later version of the library. The representation depends on the following:

- Whether or not data should be converted from the host to the network representation and vice versa (currently not implemented in the SAPI-S7 library), or whether the network representation is required on the host.
- Which host CPU is being used (for example Intel).

7.3 Representation of the Standard Data Types

7.3.1 Representation of the 'Boolean' Data Type

Network Representation and Range of Values

Bit / Octet	MSB 8	7	6	5	4	3	2	LSB 1
1	0	0	0	0	0	0	0	x

For 'FALSE', 'x' has the value '0', for 'TRUE' the value '1'.

Host Representation and Range of Values (Intel CPU)

Bit / Octet	MSB 8	7	6	5	4	3	2	LSB 1
1	x8	x7	x6	x5	x4	x3	x2	x1

If at least 1 bit of the bits ‘x8’ to ‘x1’ are set, the value ‘TRUE’ is assumed. For 'FALSE', all bits must be ‘0’

7.3.2 Representation of the Data Type 'Integer'

Range of Values

With the 'integer' data type, a distinction must be made according to the length of the data type.

Data Type	Range of Values	Length
8-bit integer	-128 ... 127	1 octet
16-bit integer	- 32768 ... 32767	2 octets
32-bit integer	-2^{31} ... $2^{31}-1$	4 octets

Network Representation

Network representation (2's complement representation) for 8-bit integers ('SI' represents the sign bit and has the value '1' for negative numbers, otherwise the value '0'):

Octet \ Bit	MSB							LSB
	8	7	6	5	4	3	2	1
1	SI	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Network representation (2's complement representation) for 16-bit integers ('SI' represents the sign bit and has the value '1' for negative numbers, otherwise the value '0'):

Octet \ Bit	MSB							LSB
	8	7	6	5	4	3	2	1
1	SI	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8
2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Network representation (2’s complement representation) for 32-bit integers ('SI' represents the sign bit and has the value '1' for negative numbers, otherwise the value '0'):

Octet \ Bit	MSB 8	7	6	5	4	3	2	LSB 1
1	SI	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

Host Representation (Intel CPU)

Host representation (2’s complement representation) for 8-bit integers ('SI' represents the sign bit and has the value '1' for negative numbers, otherwise the value '0'):

Octet \ Bit	MSB 8	7	6	5	4	3	2	LSB 1
1	SI	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

Host representation (2’s complement representation) for 16-bit integers ('SI' represents the sign bit and has the value '1' for negative numbers, otherwise the value '0'):

Octet \ Bit	MSB 8	7	6	5	4	3	2	LSB 1
1	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
2	SI	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}

Host representation (2's complement representation) for 32-bit integers ('SI' represents the sign bit and has the value '1' for negative numbers, otherwise the value '0'):

Bit / Octet	MSB 8	7	6	5	4	3	2	LSB 1
1	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
2	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8
3	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
4	SI	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}

7.3.3 Representation of the 'Unsigned' Data Type

Range of Values

With the 'unsigned' data type, a distinction must be made according to the length of the data type.

Data Type	Range of Values	Length
8-bit unsigned	0 ... 255	1 octet
16-bit unsigned	0 ... 65535	2 octets
32-bit unsigned	0 ... $2^{32}-1$	4 octets

Network Representation

Network representation for 8-bit unsigned:

Bit / Octet	MSB 8	7	6	5	4	3	2	LSB 1
1	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Network representation for 16-bit unsigned:

Bit / Octet	MSB 8	7	6	5	4	3	2	LSB 1
1	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8
2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Network representation for 32-bit unsigned:

Bit / Octet	MSB 8	7	6	5	4	3	2	LSB 1
1	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8
4	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Host Representation (Intel CPU)

Host representation for 8-bit unsigned:

Bit / Octet	MSB 8	7	6	5	4	3	2	LSB 1
1	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Host representation for 16-bit unsigned:

Bit / Octet	MSB 8	7	6	5	4	3	2	LSB 1
1	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
2	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8

Host representation for 32-bit unsigned:

Bit / Octet	MSB 8	7	6	5	4	3	2	LSB 1
1	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
2	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8
3	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
4	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}

7.3.4 Representation of the 'Floating Point' Data Type

Range of Values

Range of Values	Length
$-3.37*10^{38}$ ... $-8.43*10^{-37}$	4 octets
0	
$8.43*10^{-37}$ ... $3.37*10^{38}$	

Network Representation

Network representation ('SI' is the sign of the mantissa, the shaded fields belong to the exponent, the remaining fields to the mantissa).

Octet \ Bit	MSB 8	7	6	5	4	3	2	LSB 1
1	SI	2^7	2^6	2^5	2^4	2^3	2^2	2^1
2	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
3	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}
4	2^{-16}	2^{-17}	2^{-18}	2^{-19}	2^{-20}	2^{-21}	2^{-22}	2^{-23}

Host Representation (Intel CPU)

Host representation ('SI' is the sign of the mantissa, the shaded fields belong to the exponent, the remaining fields to the mantissa).

Octet \ Bit	MSB 8	7	6	5	4	3	2	LSB 1
1	2^{-16}	2^{-17}	2^{-18}	2^{-19}	2^{-20}	2^{-21}	2^{-22}	2^{-23}
2	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}
3	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
4	SI	2^7	2^6	2^5	2^4	2^3	2^2	2^1

Value Calculation The variable value is calculated as follows:

Exponent	Variable Value
0	$(-1)^{SI}*((0.mantissa)*2^{-126})$
≠ 0	$(-1)^{SI}*((1.mantissa)*2^{(exponent-127)})$

Example of the representation of the variable value '0.5' in the host as ‘C’ data type ‘float’:

Bit / Octet	MSB 8	7	6	5	4	3	2	LSB 1
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	1	1	1	1	1	1

7.3.5 Representation of the 'Visible String' Data Type

Range of Values

For variables of the type 'visible string', all letters (upper and lower case), numbers, the underscore ('_') and the Dollar sign ('$') are permitted.

Network Representation

Bit / Octet	MSB 8	7	6	5	4	3	2	LSB 1
1	1st character							
2	2nd character							
..								
n	nth character							

Host Representation (Intel CPU)

Bit / Octet	MSB 8	7	6	5	4	3	2	LSB 1
1	1st character							
2	2nd character							
..								
n	nth character							
n+1	NULL terminator							

A variable of the type ‘visible string' corresponds to a string of the programming language 'C in the host, in other words it is terminated with NULL.

7.3.6 Representation of the 'Octet String' Data Type

Network Representation

A variable of the type 'octet string' is represented on the network like a ‘visible string'. In this case, however, all characters are permitted.

Bit / Octet	MSB 8	7	6	5	4	3	2	LSB 1
1	1st character							
2	2nd character							
..								
n	nth character							

Host Representation (Intel CPU)

Bit / Octet	MSB 8	7	6	5	4	3	2	LSB 1
1	Length information							
2	1st character							
..								
n	(n-1)th character							
n+1	nth character							

The host representation differs from the network representation in that the length information is stored in the first byte.

7.3.7 Representation of the 'Bit String' Data Type

Network Representation

A variable of the type ‘bit string' is represented on the network as follows:

Bit / Octet	MSB 8	7	6	5	4	3	2	LSB 1
1	7	6	5	4	3	2	1	0
2	15	14	13	12	11	10	9	8
..								

Host Representation (Intel CPU)

Bit / Octet	MSB 8	7	6	5	4	3	2	LSB 1
1	Length information							
2	7	6	5	4	3	2	1	0
3	15	14	13	12	11	10	9	8
..								

The host representation differs from the network representation in that the length information is stored in the first byte.

Notes

Glossary

ASCII	**A**merican **S**tandard **C**ode of **I**nformation **I**nterchange
CP	**C**ommunications **P**rocessor. communications module for installation in computers or programmable logic controllers.
CRL	**C**ommunication **R**elationship **L**ist.
IEC	**I**nternational **E**lectrotechnical **C**ommission.
FDL	**F**ieldbus **D**ata **L**ink (data link layer of PROFIBUS complying with ISO/OSI).
FMS	**F**ieldbus **M**essage **S**pecification.
LLI	**L**ower **L**ayer **I**nterface (with PROFIBUS, this handles the most important functions of layers 3 to 6).
OD	**O**bject **D**ictionary.
PDU	**P**rotocol **D**ata **U**nit.
PROFIBUS	Network for the cell and field area of the mid-performance range with its main application in an industrial environment complying with DIN 19245 Part 1 and Part 2.
Profile	Parameter record grouped under a profile name (during configuration) describing a subfunction of the standard.
SAPI	**S**imple **A**pplication **P**rogrammers **I**nterface.
SINEC	**Si**emens **Ne**twork and **C**ommunication, product range for industrial communications from Siemens.
SINEC L2	SINEC bus system for industrial applications based on PROFIBUS.
VFD	**V**irtual **F**ield **D**evice.

Notes

Index

s7_abort() 25; 59
s7_await_initiate_req() 25; 55
s7_cycl_read() 27; 102
s7_cycl_read_delete_req() 27; 100
s7_cycl_read_init_req() 26; 88
s7_cycl_read_start_req() 26; 92
s7_cycl_read_stop_req() 27; 98
s7_discard_msg() 135
s7_get_abort_ind() 25; 60
s7_get_await_initiate_cnf() 25; 56
s7_get_conn() 24; 40
s7_get_cref() 24; 39
s7_get_cycl_read_abort_ind() 27; 97
s7_get_cycl_read_delete_cnf() 27; 101
s7_get_cycl_read_ind() 27; 94
s7_get_cycl_read_init_cnf() 26; 91
s7_get_cycl_read_start_cnf() 26; 93
s7_get_cycl_read_stop_cnf() 27; 99
s7_get_device() 24; 33
s7_get_initiate_cnf() 25; 54
s7_get_initiate_ind() 25; 57
s7_get_multiple_read_cnf() 80
s7_get_multiple_write_cnf() 26; 86
s7_get_read_cnf() 26; 71
s7_get_vfd() 24; 35
s7_get_vfd_state_cnf() 28; 110
s7_get_vfd_ustate_ind() 28; 112
s7_get_write_cnf() 26; 76
s7_init() 24; 37
s7_initiate_req() 25; 53
s7_initiate_rsp() 25; 58
s7_last_detailed_err_msg() 134
s7_last_detailed_err_no() 130
s7_last_iec_err_msg() 129
s7_last_iec_err_no() 127
s7_mini_db_get() 125
s7_mini_db_set() 118
s7_multiple_read_req() 26; 77
s7_multiple_write_req() 26; 83
s7_read_req() 26; 68
s7_receive() 24; 45
s7_set_window_handle_msg() 155
s7_shut() 24; 42
s7_trace() 116
s7_vfd_state_req() 28; 109
s7_write_req() 26; 73
s7_write_trace_buffer() 117

Notes

❑