Swisslog Interface Standard

Base Communication Protocol

Maintained by:	Christoph Brändle
Document no.:	SLSW-B31-01
Status:	released
Version:	3.01 of 21.02.2000
Address:	Swisslog Software AG, Bahnhofstrasse 96, CH-5001 Aarau

List of Changes

Version	Date	Author	Remarks
3.01-X1	11.02.2000	Christoph Brändle	• Extended frames with conversion of < and > • Fixed length packets of data on the line
3.01	21.02.2000	Christoph Brändle	Released version V3.01

Table of Contents

1. Introduction

1.1. Purpose of this Document

This document describes the base protocol of the Swisslog Interface Standard between MOVE/SPOC and the subsystems. This covers the basic data exchange protocol, but does not contain any specific application issues.

1.2. Target Readership

This document is designated to be used by the following persons or functions:

- Systems engineers of Swisslog Digitron-OWL and related companies
- Project leaders and software engineers of Swisslog Software AG
- Systems engineers, project leaders and software engineers of other companies that are implementing a crane subsystem interfacing with a Swisslog MOVE/SPOC

1.3. Reference Documents

1.3.1. General Documents

[1] Swisslog Automation Concept, Overview

1.3.2. Swisslog Interface Standard

[2] Swisslog Interface Standard, Base Communication Protocol (this document)

[3] Swisslog Interface Standard, Base Communication with SISCOMLIB

[4] Swisslog Interface Standard, Crane Interface

[5] Swisslog Interface Standard, Conveyor Interface

[6] Swisslog Interface Standard, AGVS Interface

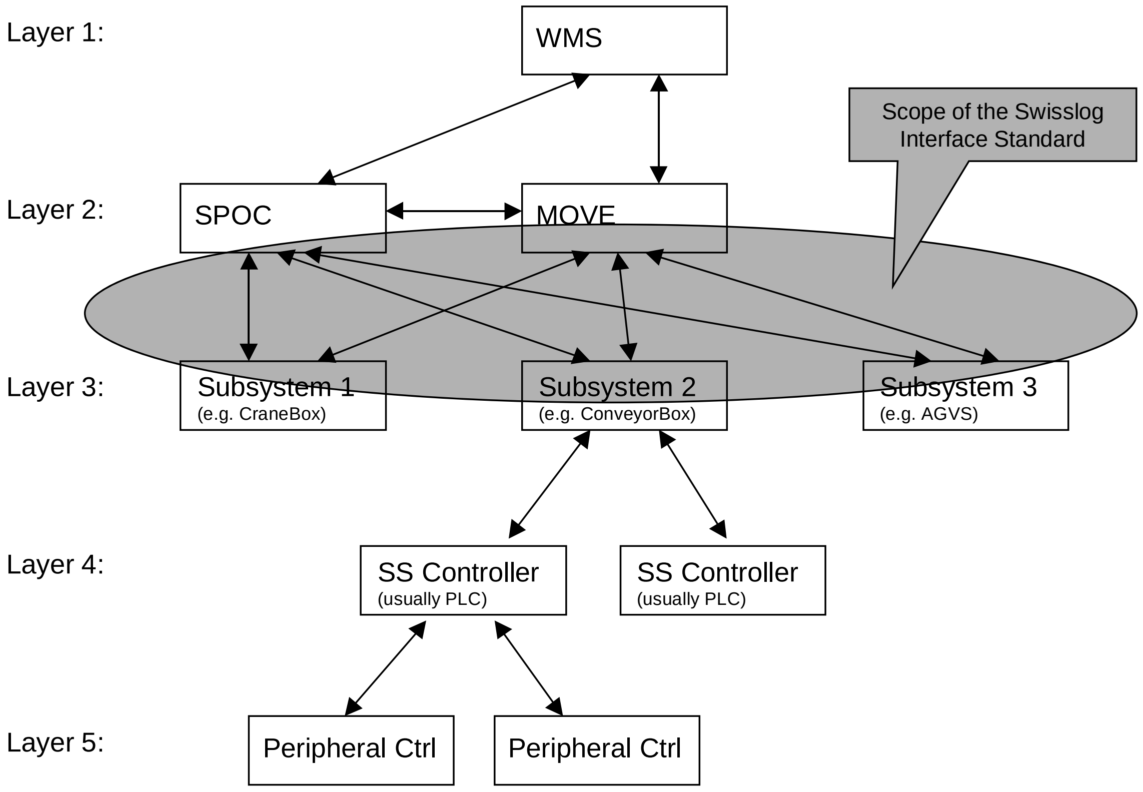
[7] Swisslog Interface Standard, Monorail Interface

[8] Swisslog Interface Standard, Robox Interface

2. System Overview

2.1. Topology of the involved systems

The Swisslog Automation Concept defines a system hierarchy as follows:


Layer 1:
WMS
Scope of the Swisslog Interface Standard
Layer 2:
SPOC
MOVE
Layer 3:
Subsystem 1 (e.g. CraneBox)
Subsystem 2 (e.g. ConveyorBox)
Subsystem 3 (e.g. AGVS)
Layer 4:
SS Controller (usually PLC)
SS Controller (usually PLC)
Layer 5:
Peripheral Ctrl
Peripheral Ctrl

Legend:

WMS	Warehouse Management System
MOVE	Material Flow Management System - The automatic system which controls the basic material flow in a plant
SPOC	Single Point Of Control - The work place of people, which is used to visualize and manually influence the material flow.

The Swisslog Interface Standard is used for communication between systems of layer 2 (MOVE / SPOC) and systems of layer 3 (Subsystem Boxes). It does not cover the communication between MOVE and SPOC, or between MOVE / SPOC and WMS or between a subsystem box and the attached controllers.

An Automation System in the context of this document consists of one MOVE, one SPOC, and a number of subsystems. For each type (crane, conveyor, AGVS, etc) there is a different subsystem. It may also be that there is more than one subsystem of a particular type.

SPOC has an independent connection to each of the subsystems, the same applies to MOVE. Each of the subsystems has two independent connections, one to MOVE, one to SPOC.

Note: It may very well be that in a control system for a particular plant some of the boxes drawn in the diagram above are not really distinct computers, but some of the functionalities are located in the same computer system. This however does not make any difference to the logical structure.

It may be for example that MOVE and WMS are tightly coupled together, but from the point of view of the SIS the subsystem communicates with the MOVE. It may also be that e.g. MOVE and ConveyorBox functionalities are located on the same computer; in this case SIS is still used, but is just an internal communication within the same computer system.

For more information on the whole SAC structure, and references to other systems and how they communicate with each other, see [1].

2.2. Base communication

2.2.1. SIS Communication Library

Swisslog Software AG provides the "Swisslog Interface Standard Communication Library" (SISCOMLIB).

This is a library that may be linked to an application program on a system. It provides an application programming interface containing functions that are used to transmit a telegram between two systems. In detail, this library performs the following tasks:

- Setup, control and shutdown of the link (TCP/IP stream sockets) between two partner systems.
- Basic data transport on TCP/IP
- Data framing
- Flow control and data buffering
- Sequence numbering
- retransmission in case of temporarily unavailable connections

Using the SISCOMLIB, prevents the application on a system from having to care about low-level communication.

SISCOMLIB is currently available on Windows NT and on various Unix systems.

For more information, see [3].

2.2.2. Base communication protocol

SISCOMLIB implements the "SIS Base Protocol", as described in this document. If a system works on a platform that is not supported by the SISCOMLIB, the functionality has to be implemented on this platform.

A system using the SISCOMLIB is able to work together with another system that implements this protocol.

3. General Information, Overview

3.1. Communication style

SIS is an event oriented, full duplex, bilateral, secure, telegram communication protocol. In detail:

- Data between the involved systems is exchanged in "telegrams", i.e. data packets of a defined rather small length (20 to 120 bytes) with a defined fixed interpretation.
- There is one point-to-point connection between each of the involved systems that have to communicate with each other.
- A connection between two systems is used for data transfer in both directions.
- Each system is allowed to send data at any time, if an event has happened which has to be reported to the partner system. (Here, some restrictions apply to support data security, but from the application point of view, data may be sent at any time).
- There are flow control and handshaking mechanisms provided in SIS which allow a safe data transfer between the systems.

3.2. Data types

All data transfer using SIS is done as a stream of ASCII characters. No binary data is ever used.

If numbers are to be transmitted, they are converted into ASCII character fields with a given length.

There are two basic data types used:

ALPHA *n*	A sequence of exactly *n* ASCII characters. Allowed characters are all in the range from `'!'` to `'~'`, except `'<'` and `'>'`. Space (`' '`) is also not an allowed character.
NUM *n*	A sequence of exactly *n* ASCII characters in the range `'0'` to `'9'`, describing a number. This number is filled up with leading zeroes, if necessary. No sign (`'+'` or `'-'`) is allowed.

If there are other restrictions about the contents of a transmitted data item, these are described in the general description of the particular item.

Notes:

1. Implementations of the base protocol should allow the usage of the space (`' '`) character, and also of other characters (like umlauts) in the data. However, before using such characters, all connected parties must confirm that they are able to handle this properly.
2. Usage of `'>'` and `'<'` in the data: see section "4.4. Extended data frames, < and > in application data (optional)" for more information about this topic

4. Basic Transport - TCP/IP

4.1. Overview

The communication in SIS V3.0 is based on TCP/IP Stream Socket connections.

There is one (and only one) direct connection between each pair of systems that want to communicate with each other. This connection is used both for data from system A to system B, and for data from system B to system A.

If one particular system has a communication to several other systems, a separate connection is used for each link to a partner system.

If two functionnally distinct systems (e.g. SPOC and MOVE) are located on the same computer for any reason, it is still so that each of these systems maintains its connections separately. If these distinct systems on the same computer have to communicate with each other (e.g. a MOVE and a Crane Subsystem might be deployed on the same computer), it is still so that a connection is built between the two systems; here this will be a loopback connection.

If two systems need to communicate with each other, one is defined to be server, the other one client. In the Swisslog Automation Concept Standard Configuration, the "higher level systems" (MOVE and SPOC) are servers, the "subsystems" (Cranes, Conveyors, AGVSs, Monorails) are clients. If SIS communication is used for other purpose, an agreement must be made before who is client and who is server.

The SIS protocol elements like link setup, data framing, etc, are directly built on top of TCP. No other component (e.g. RFC1006) is used in-between.

When a system is started, it immediately tries to (actively or passively) establish a connection to all other systems to which a link is defined. These connections are kept active all the time, even if e.g. there is currently no activity at all. Usually, the link is only disconnected when one of the systems is shut down.

4.2. Link Setup / Disconnect

One system (usually MOVE and SPOC) is defined to be server. The server creates a listening socket with a defined port number, and waits until being called by the client. When an incoming call is recognized, it is accepted by the server, the link is considered ok and data exchange may take place.

The client attempts to establish a connection by calling the server on the defined host and the defined port number. If the connection attempt fails, the client waits a short time, then retries. If the connection is accepted by the server, the link is also considered ok by the client, and data exchange may take place.

The server does not assume any particular client IP address or client port number, i.e. a call from any source is accepted.

Each of the systems may break down the link without further notice. There is no special message exchange or other procedure during link disconnection. Usually, a link is only disconnected in the following cases:

- if a system detects a communication problem on a link (failure when reading from, or sending to the link)
- if a system encounters a timeout or other missing reaction from the partner system, as described below in section 5.1. Link Watching / Keepalive

- if a system, or part of a system (e.g. the communication component) is being shut down

It might also be that a link is disconnected upon request of an operator, in particular due to maintenance or problem solving work.

If a system breaks down the link itself, or detects that the link has been broken down by the partner or some external component, the system goes into the communication setup state as described above. The server just waits until being called again, and the client repeatedly tries to re-establish the connection.

4.3. Data Framing

Since TCP is a stream protocol, there is no direct way to detect the boundaries of a telegram automatically.

There is a data frame layout defined as follows:

- Each telegram starts with a start-of-message character "<".
- Each telegram ends with an end-of-message character ">".

The sender must pack every telegram into the correct frame, in particular start it with a "<" and end it with a ">".

There may be no "<" and ">" inside the application telegram.

The receiver must scan for a valid telegram frame, i.e. take out everything from a "<" to the next ">", and process this as a valid telegram.

Everything after the ">" and before the next "<" is considered noise on the line, and is ignored by the receiver.

If another "<" appears within a telegram frame, this is considered the start of a new frame, and everything from the previous "<" to this "<" is considered a framing error, and is ignored. Actually, the sender should refrain from sending such noise.

The receiver does not generate any action upon detection of noise or other framing errors. A local warning message should be reported.

For details of the frame layout see section 6.1. Telegram framing.

4.4. Extended data frames, < and > in application data (optional)

There are applications that require the possibility to send the "<" and ">" characters in the application data. To achieve this, the SIS Base Protocol V3.1 offers the following option:

A. Use extended data frames

When sending data out to the TCP/IP socket, some special characters are converted as follows:

`">"`	is sent as:	`"%)"`
`"<"`	is sent as:	`"%("`
`"%"`	is sent as:	`"%%"`

All other characters are sent as they are.

Upon reception, character combinations are converted as follows:

`"%)"`	is converted to	`">"`

`"%("` is converted to	`"<"`	
`"%c"` is converted to	`"c"`	where *c* is any character except `")"` or `"("`

all other character combinations are kept as they are.

B. Do not use extended data frames

If extended data frames are not to be used, every character in the application data is sent to the socket as it is. Every character in the data from the socket is kept as it is. The application may not use "<" and ">".

Notes:

1. To be able to be backwards compatible, an implementation that supports extended data frames must be configurable whether such extended frames are used or not.
2. If an application requires that < and > are possible, extended data frames should be used.
3. If an application requires that < and > are possible, but extended data frames are not possible, since the used base protocol implementation does not support this, the application itself must do this conversion.

4.5. Fixed length data packets on the line (optional)

There might be systems that are not able to handle unspecified bytestreams reasonably. This in particular applies to PLCs.

Therefore, the SIS Base Protocol V3.1 offers the option that the sender should be able to send data in fixed packets of a defined size.

If a telegram frame to be sent is shorter than this defined size, the data is padded with a filler character to the defined size. This padding is done outside of the telegram frame.

If a telegram frame to be sent is exactly this defined size, nothing special is done.

If a telegram frame to be sent is longer than this defined size, the data is padded with a filler character to an integral number of the defined size. This padding is done outside of the telegram frame.

Notes:

1. Even if the configured size of the packet is long enough, the sender must not put more than one telegram into one packet.
2. A packet size of 1 means that each byte is handled individually; this means that the byte stream is really treated as a byte stream.
3. A receiver that is configured to handle a plain byte stream (i.e. a packet size of 1) is always able to handle sent packets with a configured bigger size. Since the padding up to the configured length is done outside the data frame, this padding data is simple treated as noise, and is ignored.

4.6. Summary of parameters

For each link, the following data must be known in advance (during configuration time). This is partially dependent on the existing setup of the TCP/IP network.

Usual values:

Parameter	Default value	Usage

Host name / IP address of server	N/A	The name or address of the host where the server component is, and to which the client has to connect.
Port number of server	N/A	The port number of used by the server for accepting connections from the client.
Connection setup retry interval	10 sec	Waiting time after an unsuccessful attempt to establish a connection to the server.
Use extended frames	FALSE	FALSE: No < or > in application data; % in byte stream handled as plain % TRUE: < and > allowed in application data. % in byte stream handled as mapping character.
Fixed packet size	1	Number of bytes sent in one frame packet.

Host name / IP address and port number of the client do not have to be configured.

5. Functionality

5.1. Link Watching / Keepalive

Under certain circumstances it might happen that one partner believes that the connection is established, but the other one believes that it is not. This may lead to a situation that cannot be recovered automatically.

To circumvent such problems, receive timeouts and keepalive telegrams are used.

If a station does not receive any telegram from the partner within a given time ("idle timeout"), a DUM telegram is sent to the partner.

The receiver of a DUM reacts by sending a DUA.

Steps and Telegrams:

No	Telegram	Communication	Description
0			The station did not receive any telegram from the partner within a given time ("idle timeout")
1	DUM	A to B	A DUM telegram is sent to the partner
2	DUA	B to A	The partner responds with a DUA telegram.
3			On system A, the idle timer is reset.

Notes:

1. This behaviour is symmetric, i.e. either partner of a communication link may send a DUM, if its idle timer has expired, regardless of who is server and who is client.
2. Incoming unexpected DUAs (apparently no DUM sent as a request) are ignored (but still reset the idle timer)
3. Any incoming telegrams reset the idle timer, regardless of whether they are correct and useful or not. Therefore, if a station keeps on sending application telegrams where the time between two application telegrams is always shorter than the idle timeout, the station receiving these telegrams will never send a DUM.
4. After sending a DUM, the idle timer is also reset, i.e. after this time has elapsed once again without any incoming telegram, another DUM is sent.
5. If, after sending a DUM, another telegram is received instead of a DUA, this is also considered an useful reaction. The idle timer is reset.
   It is not absolutely required for the receiver of a DUM to send a DUA, the receiver might just send any telegram. However, the DUA is especially designed for this purpose, since it contains no other information.

5.1.1. Link breakdown if no reaction

If a station does not receive any telegram from the partner within a given time ("link loss timeout"), the partner station is considered dead, and the link is broken down.

Depending on whether this station is configured as client or server on the TCP/IP level (see above), the station either tries to re-connect the link, or waits until the partner re-connects.

Steps and Telegrams:

No	Telegram	Communication	Description
0			The station did not receive any telegram from the partner within a given time ("idle timeout")
1	DUM	A to B	A DUM telegram is sent to the partner
2			The partner does not respond within a given time ("link loss timeout").
3	X X X X X		System A breaks the link.
4			New link setup procedures are started.

Notes:

1. Any incoming telegram resets the link loss timer in the same way as the the idle timer.
2. The link loss timeout must be configured longer than the idle timeout, otherwise the link would be broken down before checking it with a DUM telegarm.
3. The link loss timer measures the time starting from each received telegram, not from the sending of a DUM. For example:

idle timeout	link loss timeout	effect
60 sec	70 sec	After 60 seconds without telegram, a DUM is sent. If no telegram arrives within the next 10 seconds, the link is broken.
60 sec	140 sec	After 60 seconds without telegram, a DUM is sent. If no telegram arrives within the next 60 seconds, another DUM is sent. If no telegram arrives within another 20 seconds, the link is broken.
60 sec	50 sec	After 50 seconds without telegram, the link is broken. No DUM will ever be sent. -> This setting is not very useful

5.2. Flow Control

5.2.1. Generals

The SIS V3.0 protocol supports "confirmed" telegrams and "unconfirmed" telegrams.

Unconfirmed telegrams are just sent by the sender; if the receiver cannot process an incoming unconfirmed telegram due to internal congestion, this telegram is dropped without notice to the sender. This means that unconfirmed telegrams may be lost. After sending an unconfirmed telegram, the sender may just send the next one, no reply has to be awaited.

Confirmed telegrams are sent by the sender; if the telegram is correctly received and can be processed by the receiver, the receiver acknowledges the telegram. If the receiver cannot process an incoming confirmed telegram due to internal congestion, a negative acknowledge is sent back to the sender. Confirmed telegrams cannot just be lost without notice to the sender.

If the sender of a confirmed telegram receives a positive acknowledge, he may send the next telegram.

If the sender receives a negative acknowledge with "buffer full" indication, he waits a short time ("buffer full timeout"), then re-sends the same telegram.

If the sender does not receive any (positive or negative) acknowledge within a given time ("confirmation timeout"), then he re-sends the same telegram.

However, the sender retries sending a telegram only a given number of times ("maximum attempts") without receiving any acknowledge; if after the last attempt still no acknowledge arrives, the sender breaks the link.

Each confirmed telegram contains a sequence number, which may be used by the receiver to find out whether this is now a repetition of an already received telegram, or a really new telegram. For more information about sequence numbering see the next section.

Flow control is kept separate for both directions, i.e. even if for example a telegram has been sent from A to B, but not yet confirmed, B may still send a telegram to station A.

Unconfirmed telegrams may always be sent, even if a confirmation for a confirmed telegram is currently outstanding. However, it does not make much sense in general to send unconfirmed telegrams in this situation, since they will most probably just be lost.

However, keepalive telegrams must always be processed correctly.

5.2.2. Normal data transfer with unconfirmed telegrams

Steps and Telegrams:

No	Telegram	Communication	Description
0			Link is ok
1	*XXX*(1)	A to B	An unconfirmed telegram is sent to the partner
2	*XXX*(2)	A to B	The next unconfirmed telegram is sent to the partner
			...etc...

5.2.3. Normal data transfer with confirmed telegrams

Steps and Telegrams:

No	Telegram	Communication	Description
0			Link is ok
1	*XXX*(1)	A to B	A confirmed telegram is sent to the partner
2	ACK(1)	B to A	The partner confirms reception
3	*XXX*(2)	A to B	The next confirmed telegram is sent to the partner
4	ACK(2)	B to A	The partner confirms reception
			...etc...

5.2.4. Buffer full on the receiver's side

Steps and Telegrams:

No	Telegram	Communication	Description
0			Link is ok
1	*XXX*(1)	A to B	A confirmed telegram is sent to the partner
2	NCK(full)	B to A	The receiver is currently not able to process the telegram, therefore sends a negative acknowledge
3			The sender station waits a certain time (buffer full timeout)
4	*XXX*(1)	A to B	The rejected telegram is sent again to the partner
5	ACK(1)	B to A	The partner confirms reception
6	*XXX*(2)	A to B	The next confirmed telegram is sent to the partner
7	ACK(2)	B to A	The partner confirms reception
			...etc...

Notes:

1. If there is really a problem, the receiver may reject the telegram more than once. The sender will then just indefinitely keep on trying.

5.2.5. No answer at all from the receiver

Steps and Telegrams:

No	Telegram	Communication	Description
0			Link is ok
1	*XXX*(1)	A to B	A confirmed telegram is sent to the partner
2			No answer from the receiver station.
3			The sender station waits a certain time (confirmation timeout)
4	*XXX*(1)	A to B	The previous telegram is sent again to the partner

No	Telegram	Communication	Description
5	ACK(1)	B to A	The partner confirms reception
6	*XXX*(2)	A to B	The next confirmed telegram is sent to the partner
7			No answer from the receiver station.
8			The sender station waits a certain time (confirmation timeout)
9	*XXX*(2)	A to B	The previous telegram is sent again to the partner
10			No answer from the receiver station.
11			The sender station waits a certain time (confirmation timeout)
12	*XXX*(2)	A to B	The previous telegram is sent again to the partner
13			No answer from the receiver station.
14			The sender station waits a certain time (confirmation timeout)
15	- - - - -		Station A breaks the link
16			New link setup procedures are started.

5.3. Sequence numbering

5.3.1. Generals

As already mentionned, each "confirmed" telegram contains a sequence number, which is used by the receiver to identify whether this incoming telegram is really the one that is expected.

Unconfirmed telegrams do not have a sequence number, i.e. the "sequence number" item in their frame is always set to 0000. For this reason, unconfirmed telegrams are not considered in the following description.

The sequence number ranges from 0001 to 9999. Each new telegram is sent with a sequence number one higher than the one before. If the previous telegram had a sequence number of 9999, a wraparound takes place, and the next one is sent with sequence number 0001. Sequence number 0000 is never used for confirmed telegrams.

The sequence number is kept separate in the directions A -> B and B -> A; each direction is managed on its own.

The sequence number is not telegram type specific, i.e. it is counted up through all types that are sent over a particular link.

If an application on one system is built up to consist of different components for processing different telegrams, the sequence number must be managed by the communication component that handles the link to the partner system.

The sequence numbering must survive all kinds of error situations; in particular program or system crashes may not disturb the sequence number. It must therefore be kept safely on disk or any other permanent storage.

The sequence number is counted up and wrapped around indefinitely; the only situation when the sequence number is re-set to a defined starting value is during a re-synchronisation using a SYN telegram.

Handling of the sequence number by the sender:

- When a new confirmed telegram should be sent, and this can be done, the "next sequence number to be sent" is incremented by one, and assigned to the telegram. This telegram is then sent, and the sender then waits for an acknowledge from the receiver.
  Before receiving the acknowledge, no other confirmed telegram may be sent.
- If the sender receives a positive acknowledge, it may send the next telegram.
- If the sender receives a negative acknowledge with "sequence fault" indication, it goes into "out-of-sync" mode, and does not send any further telegrams.

For a description of negative acknowledge with "buffer full" indication, or "no acknowledge within reasonable time", see above.

Handling of the sequence number by the receiver:

- If the incoming telegram is correct, i.e. has the expected sequence number, this is acknowledged to the sender. The incoming telegram is processed by the receiver.
- If the incoming telegram is a repetition, i.e. the sequence number is the same as the previous telegram had, this is also acknowledged to the sender like a correct telegram (most probably the previous ack telegram has been lost). The incoming telegram is however *not* processed.
- If the incoming telegram has a wrong sequence number (neither the expected one nor the same as before), this is answered with a negative acknowledge with a "sequence error" indication. The receiver then goes to "out-of-sync" mode. The incoming telegram is not processed.

5.3.2. Normal data transfer with confirmed telegrams

See above in section "Flow Control" (5.2.3. Normal data transfer with confirmed telegrams)

5.3.3. Repetition of a telegram

Steps and Telegrams:

No	Telegram	Communication	Description
0			Link is ok
1	*XXX*(1)	A to B	A confirmed telegram is sent to the partner
2	ACK(1)	B to A	The partner confirms reception, and processes the telegram.
3			For any reason, the sender does not receive or not properly recognize the ACK.
4			The sender station waits a certain time (confirmation timeout)
5	*XXX*(1)	A to B	The previous telegram is sent again to the partner
6	ACK(1)	B to A	The partner recognizes that this is the same sequence number as before, confirms reception again, but does not process the telegram this time.
			...etc...

5.3.4. Sequence error

Steps and Telegrams:

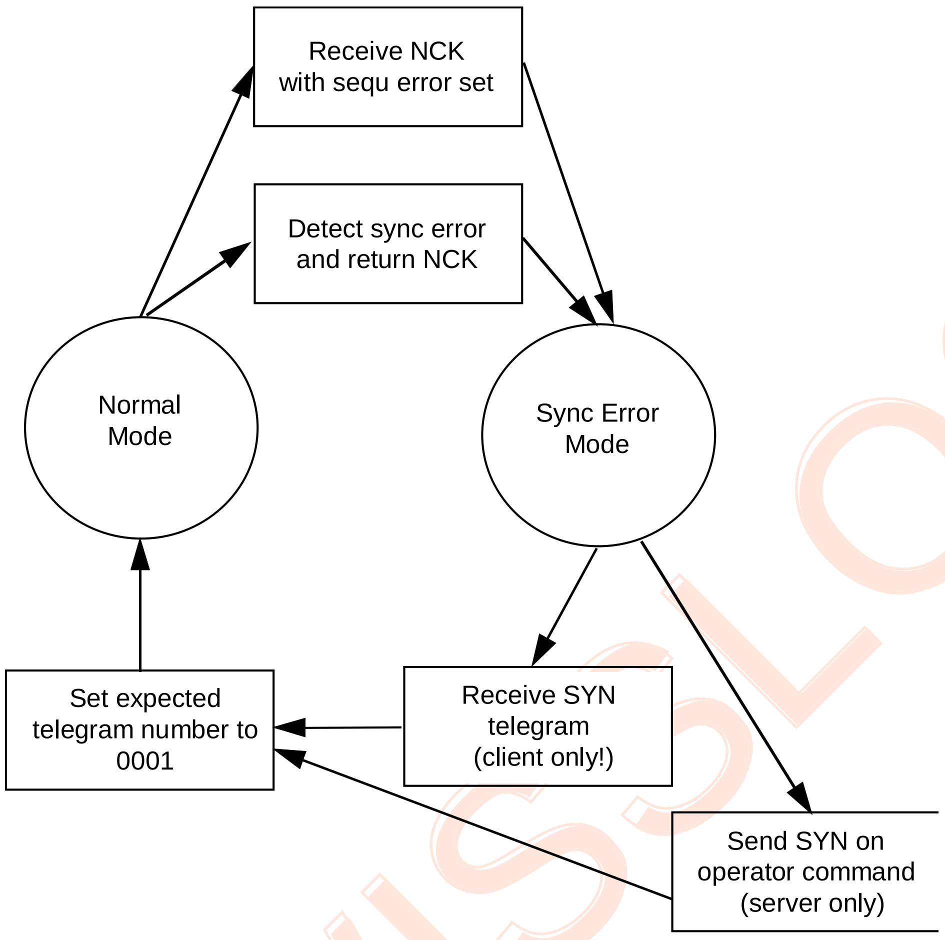
No	Telegram	Communication	Description
0			Link is ok
1	*XXX*(1)	A to B	A confirmed telegram is sent to the partner
2	ACK(1)	B to A	The partner confirms reception
3	*XXX*(7)	A to B	Another telegram is sent to the partner, with a wrong sequence number.
4	NCK(seq)	B to A	The receiver does not expect this sequence number, sends a negative acknowledge, and goes into out-of-sync mode.
5			Upon reception of the NCK, the sender station also goes into out-of-sync mode, and does not send any further telegrams.

Notes:

1. It does not matter whether the sender just sends a wrong telegram because of incorrect internal data in the sender, or the receiver mis-interprets the telegram sequence number because of incorrect internal data in the receiver; the effect is the same: both stations do not send telegrams any more.
2. If a receiving station is in out-of-sync mode, no application telegrams are accepted any more, not even a telegram with a now correct sequence number. Low-level telegrams (DUM, DUA, SYN) however are still accepted and processed.
3. If a receiving station in out-of-sync mode receives a confirmed application telegram, this is answered by a NCK with "sequence fault" indication without any further checks. The telegram is ignored.
4. If a sending station is in out-of-sync mode, no application telegrams are sent any more. Low-level telegrams (DUM, DUA, NCK, SYN) however are still sent.
5. An out-of-sync situation always blocks the traffic in both directions, i.e. if something went wrong in the direction A->B, and the stations go into out-of-sync mode, this also means that no confirmed telegrams are sent or accepted in direction B->A.

5.3.5. Recovering from out-of-sync situations

The only way to recover from an out-of-sync situation is to do a re-synchronisation.


Receive NCK with sequ error set
Detect sync error and return NCK
Normal Mode
Sync Error Mode
Set expected telegram number to 0001
Receive SYN telegram (client only!)
Send SYN on operator command (server only)

This is manually initiated on the server system (usually MOVE / SPOC). This manual intervention is intended because out-of-sync situations do not happen just so; usually there is something wrong which must be corrected first.

Steps and Telegrams:

No	Telegram	Communication	Description
0			Both stations are in out-of-sync mode
1			The operator on the server station performs a "resynchronize" action
2			The server station returns from out-of-sync mode to normal mode. It sends the "next expected incoming seqno" and the "next seqno to be sent" both to 0001.
3	SYN	A to B	A SYN telegram is sent to the client.
4			The client station also returns from out-of-sync mode to normal mode. The "next expected incoming seqno" and the "next seqno to be sent" are also both set to 0001.
			Normal telegram exchange may continue; the first telegram in each direction will have a sequence number of 0001.

Notes:

1. The operator must make sure that the connection between the two stations is up and running, otherwise it might happen that only one station goes to normal mode, the other one stays in out-of-sync.
2. A (client) station also accepts a SYN telegram if it is not in out-of-sync mode. The processing is the same, i.e. the "next expected incoming seqno" and the "next seqno to be sent" are both set to 0001.
   If there is an outstanding telegram which is not yet acknowledged, this is re-sent with the new sequence number 0001.
3. A resynchronisation always influences both directions, i.e. even if for example only the sequence numbers in the direction A->B have fallen out of sync, the following resynchronisation resets the sequence numbers to 0001 in both directions A->B and B->A.

5.4. Actions after link setup

When a connection to a partner system is established, the following is performed:

1. The timers (receive timer, link loss timer) are reset.
2. If there is a sent confirmed telegram, for which no acknowledgement has been received yet, this telegram is re-sent.
3. If there is no open acknowledgement, the next telegram is sent, if there is any at all.

5.5. Summary of parameters

Usual values:

Parameter	Default value	Usage
idle timeout	60 sec	After this time without any incoming telegram, a station sends a DUM telegram to check the link.
link loss timeout	70 sec	After this time without any incoming telegram, a station breaks the link down.
buffer full timeout	10 sec	After an incoming NCK (buffer full), the sender waits this time, then re-sends the telegram
confirmation timeout	10 sec	After this time without any (positive or negative) confirmation, the sender re-sends the telegram
maximum communication attempts	3	The sender sends a telegram at most this number of times without receiving a confirmation. After this number of times, the sender breaks the link down.

6. Telegrams

6.1. Telegram framing

Each telegram is built up as follows:

Data description	Field type	Value	Remarks
STX	ALPHA 1	<	
Confirmation request flag	NUM 1	0: unconfirmed 1: confirmation required	
Sequence number	NUM 4	0000, or 0001 to 9999	For unconfirmed telegrams: always 0000. For confirmed telegrams: between 0001 and 9999
Destination (Receiver)	ALPHA 6		as defined plant specific
Source (Sender)	ALPHA 6		as defined plant specific
Application telegram data	ALPHA n		
CRC	NUM 2	00	Reserved for future use
ETX	ALPHA 1	>	

Notes:

1. For unconfirmed telegrams (Confirmation request flag == 0), there is no direct low-level answer. The sender may send the next telegram immediately.
2. For confirmed telegrams (Confirmation request flag == 1), the receiver will answer with a low-level ACK (or NCK). The sender may only send the next (confirmed) telegram after this one has been acknowledged.
3. Framing errors are not reported back to the sender. The receiver only generates a local error, and does not accept this telegram.
4. The SIS V3.0 does not contain any routing and forwarding procedures. The receiver only accepts telegrams where the destination matches his identity. Other telegrams are ignored.
5. If extended data frames are used, the characters <, >, and % must be handled in a special way. For details see section "4.4. Extended data frames, < and > in application data (optional)".
   Example:
   An application telegram containing the following data:
   `XTC0001<XML-Data>99%0102`
   would be transmitted as
   `<00000ddddddssssssXTC0001%(XML-Data%)99%%010200>`

6.2. Link Check / Keepalive Telegrams

6.2.1. Dummy request, DUM

Request from either system to give a dummy answer.

Sent from either system if no telegram has been received within a given time.

Layout of the telegram:

Data description	Field type	Value	Remarks
Telegram type	ALPHA 3	DUM	
Fault Indicator	NUM 1	0	Reserved for future use
Acknowledged sequence no	NUM 4	0000	Reserved for future use

Notes:

1. DUM (and DUA) telegrams are used for detecting link failures in times where there is no other traffic on the line.
2. Each station may send DUM telegrams at any time. The receiver of a DUM telegram must answer this with a DUA telegram.
3. DUM telegrams are always sent as unconfirmed telegrams (i.e. in the framing part, "Confirmation request flag" == 0 and "Sequence number" == 0000).

6.2.2. Dummy answer, DUA

Answer from the receiver of a DUM.

Sent from either system as a reaction to an incoming DUM telegram.

Layout of the telegram:

Data description	Field type	Value	Remarks
Telegram type	ALPHA 3	DUA	
Fault Indicator	NUM 1	0	Reserved for future use
Acknowledged sequence no	NUM 4	0000	Reserved for future use

Notes:

1. (DUM and) DUA telegrams are used for detecting link failures in times where there is no other traffic on the line.
2. If an expected DUA telegram does not arrive, the waiting station must perform appropriate recovery procedures.
3. DUA telegrams are always sent as unconfirmed telegrams (i.e. in the framing part, "Confirmation request flag" == 0 and "Sequence number" == 0000).

6.3. Flow Control / Acknowledge Telegrams

6.3.1. Positive Acknowledge, ACK

Indication of a successful receipt of a telegram.

Sent from either station when a telegram to be confirmed has been received succesfully.

Layout of the telegram:

Data description	Field type	Value	Remarks
Telegram type	ALPHA 3	ACK	
Fault Indicator	NUM 1	0	Reserved for future use
Acknowledged sequence no	NUM 4	0001..9999	Sequence number of the telegram to be confirmed

Notes:

1. When a station receives a correct application telegram with a "Confirmation request flag" == 1 in the framing part, an ACK telegram with the sequence number of the telegram to be confirmed is sent back to the original sender.
2. When a station receives a repeated confirmable application telegram (i.e. with a sequence number one below the expected one), an ACK telegram with the sequence number of this repeated telegram is sent back to the original sender.
3. An incoming (correct) ACK telegram allows the sender of the original telegram to remove it from the send queue, and to send the next confirmed telegram.
4. ACK telegrams are always sent as unconfirmed telegrams (i.e. in the framing part, "Confirmation request flag" == 1 and "Sequence number" == 0000).

6.3.2. Negative Acknowledge, NCK

Indication of an unsuccessful receipt of a telegram.

Sent from either station when a telegram to be confirmed has been received, but rejected.

Layout of the telegram:

Data description	Field type	Value	Remarks
Telegram type	ALPHA 3	NCK	
Fault Indicator	NUM 1	1: sequence fault 2: buffer full	
Next sequence no	NUM 4	0001..9999	Next expected sequence number

Notes:

1. When a station receives a confirmable application telegram (i.e. with a "Confirmation request flag" == 1 in the framing part), but the sequence number is not as expected and also not the sequence number of the last correctly received one, a NCK telegram with a "Fault Indicator" == 1 and the next expected sequence number is sent back to the original sender.

2. The receiver of a NCK - Sequence Fault telegram enters "Sequence Error Mode". For more details about this, an what has to happen then, see above.
3. When a station receives a confirmable application telegram (i.e. with a "Confirmation request flag" == 1 in the framing part), but cannot store this at the moment because of no space in the receive buffer, a NCK telegram with a "Fault Indicator" == 2 and the next expected sequence number is sent back to the original sender.
4. The receiver of a NCK - Buffer Full telegram waits a short time, then re-sends the application telegram that caused this fault. No other confirmable telegram may be sent in-between.
5. An incoming (correct) ACK telegram allows the sender of the original telegram to remove it from the send queue, and to send the next confirmed telegram.
6. NCK telegrams are always sent as unconfirmed telegrams (i.e. in the framing part, "Confirmation request flag" == 1 and "Sequence number" == 0000).

6.3.3. Sequence Number Synchronisation, SYN

Request to re-synchronize the sequnce number.

Sent from MOVE/SPOC to recover from "synchronisation error" mode.

Layout of the telegram:

Data description	Field type	Value	Remarks
Telegram type	ALPHA 3	SYN	
Fault Indicator	NUM 1	0	Reserved for future use
Acknowledged sequence no	NUM 4	0000	Reserved for future use

Notes:

1. SYN telegrams are only sent from MOVE or SPOC, not from subsystems.
2. SYN telegrams are only sent upon manual request from an operator.
3. The SYN telegram re-synchronizes both sequence numbers (for telegrams from MOVE/SPOC to subsystem, and vice-versa)
4. The sender of a SYN sets its "next expected sequence number" to 0001, this means that the receiver of a SYN will have to send the next telegram with a sequence number of 0001.
5. The receiver of a SYN also sets its "next expected sequence number" to 0001, this means that the sender of a SYN will also have to send the next telegram with a sequence number of 0001.
6. If the receiver of a SYN is currently not in "synchronisation error mode", it will nevertheless accept and process the SYN telegram.
7. There is no confirmation telegram to the SYN, i.e. the operator performing the re-synchronisa-tion has to make sure that the link to the subsystem is ok, and the subsystem is ready to accept a SYN telegram.
8. SYN telegrams are always sent as unconfirmed telegrams (i.e. in the framing part, "Confirmation request flag" == 1 and "Sequence number" == 0000).

7. Data Security / Data Buffering

7.1. Generals

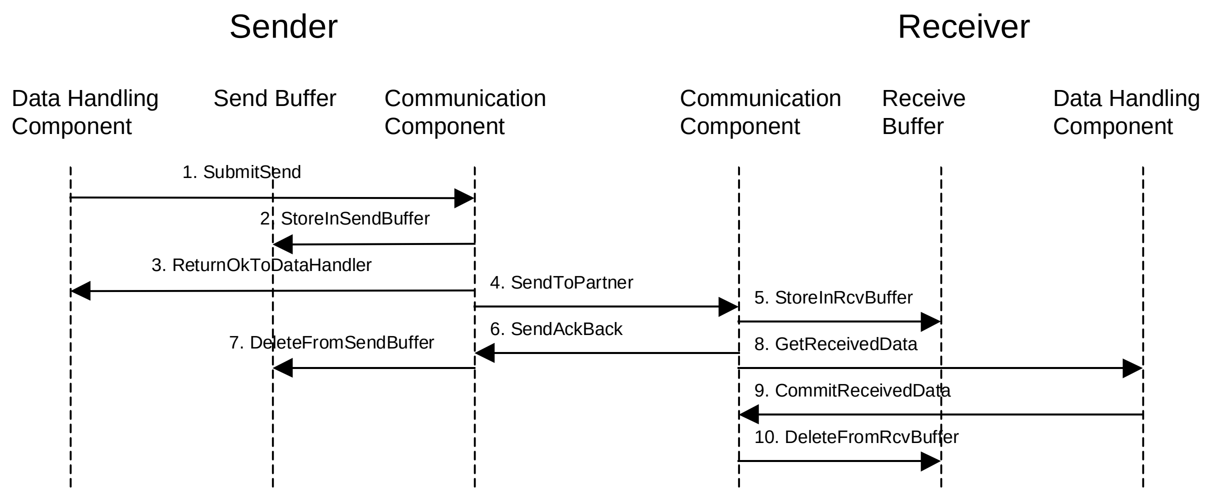
Each system is responsible for telegram data to be sent, until the telegram is acknowledged from the receiver. After the acknowledgement, the receiver is now responsible. This is also true in exceptional cases, e.g. network breakdowns, system crashes etc.

It depends on the general software architecture on a system what the implications onto the communication component are.

If communication component and data handling components are only loosely coupled, data buffers must be implemented in the communication component.

If a data handling component generates a telegram to an other system, and this telegram cannot be sent at the moment for any reason (e.g. link not established, previous telegram not yet acknowledged), this telegram must be kept safely on the sender side. This has to survive system crashes and also software errors (at least to a useful extent). In most cases it should be possible for the sending data handling component to continue working even if the telegram could not be sent at the moment.

If an incoming telegram arrives, and this telegram cannot currently be passed over from the communication component to the appropriate data handling component, the communciation component has to store this telegram in a buffer. This buffer also has to be safe. Only when the telegram has been stored, the acknowledgement may be sent back to the sending station.



Notes:

1. The sender may only (3) return ok to the data handling component, and (4) send the data to the partner, after (2) storing the data in the send buffer has been successful. If the data cannot be stored in the send buffer for any reason (e.g. the buffer is full), "not ok" must be returned to the data handling component, and the data is not sent.
2. Whether (3) return ok to the data handling component, or (4) send the data to the partner, is performed first, does not matter. However, (3) should be performed before (6) send ack back has been recognized, since it may take quite a long time until (6) happens.
3. The receiver may only (6) send an ACK back to the sender after (5) storing the data in the receive buffer has been successful. If the data cannot be stored in the receive buffer for any reason (e.g. the buffer is full), NCK must be returned to the sender.

4. (7) Delete data from the send buffer is performed automatically by the communication component, as a reaction on the incoming ACK (6).
5. (10) Delete data from the receive buffer is performed (9) upon request from the data handling component.

7.2. Recovery in case of errors

Until the sender has received an ACK, it must be possible for the sender under any circumstances (including communication breakdown, program failure, system crash etc) to send the data telegram again.

After the sender has received an ACK, the sender will not be requested to send the telegram again. The data may therefore be deleted.

After the receiver has sent the ACK, the receiver may not lose this telegram under any circumstances (including communication breakdown, program failure, system crash etc), since the sender will probably not be able to re-send the telegram.

8. Miscellaneous topics

8.1. Logging

Each system should provide a way to log all transmitted and received data, as well as all other events, in a logfile. This ensures that in case of unclear cases the situation can be analyzed later, and necessary actions may be taken to prevent problems.