Aculab SS7

SCCP API Guide

Revision 6.15.1

aculab

PROPRIETARY INFORMATION

Document Revision

Rev	Date	By	Detail
6.10.1	15.09.08	DSL	Draft issue
6.10.3	30.10.08	DSL	First full release
6.11.0	14.09.10	DSL	Additional configuration parameters and support functions
6.12.2	28.06.13	DSL	Remove references to Solaris. Support IPv6 connections to the ss7 driver.
6.13.0	27.10.14	DSL	Minor corrections.
6.14.9	15.09.16	DSL	Minor corrections.
6.15.1	31.08.18	DSL	Minor corrections

CONTENTS

1 Introduction

This document describes the SCCP API.

1.1 Structure of SCCP

The Aculab SCCP is split between the interface library and the ss7 driver. The library is responsible for most of the connection-oriented procedures, with the driver controlling message routing and the actual format of the SCCP messages. The SCCP library communicates with the driver using a proprietary protocol over TCP/IP connecting to the same driver interface code as the TCAP library.

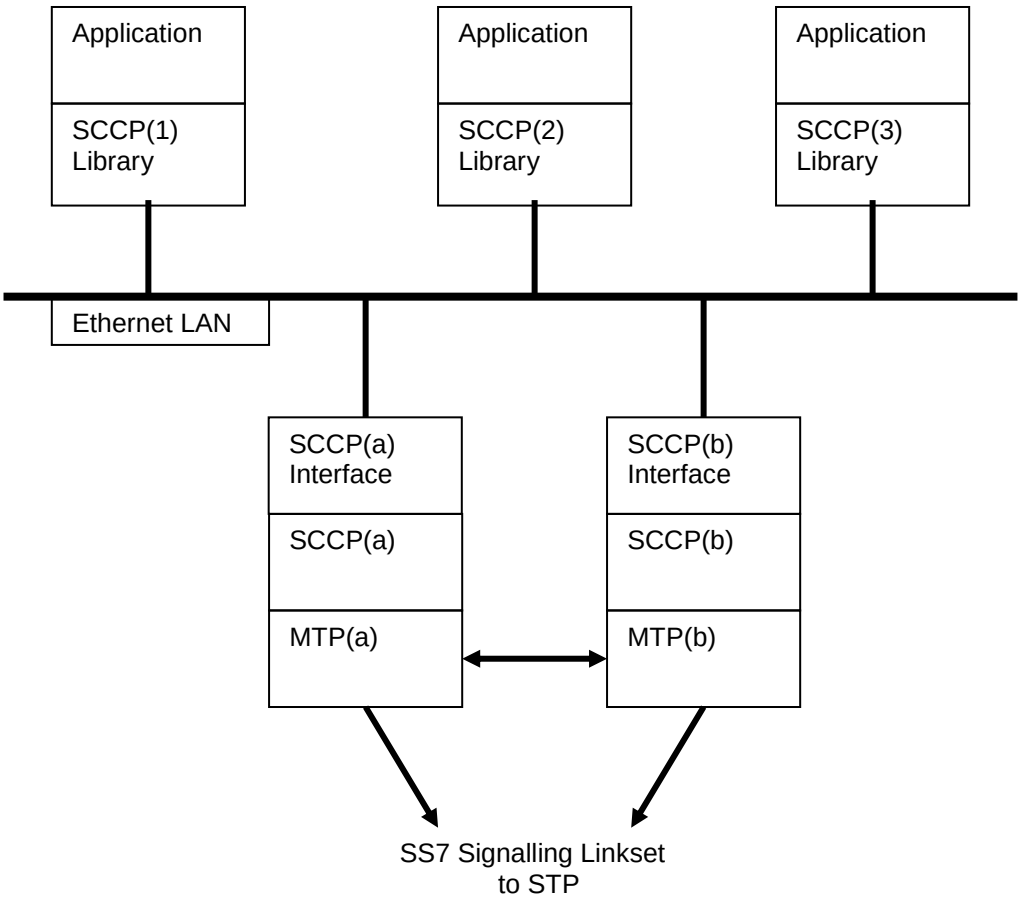
The product supports:

- SCCP classes 0, 1 and 2
- Multiple SCCP and TCAP applications using different SSNs.
- An SCCP application connecting to multiple SCCP endpoints (e.g.: several SSNs or multiple MTP3 local pointcodes).
- SCCP applications in a dual-MTP3 environment
- Multiple copies of the same SCCP application running on multiple chassis.
- Multiple SCCP applications running in a single chassis.
- ITU, ANSI and CHINA SCCP.
- SCCP over MTP2 and MTP3 or M3UA.

The SCCP library is based on the distributed TCAP library and shares common source files. However all symbols in the two libraries differ so an application can use both libraries without any conflicts.

1.2 SCCP with dual resilient MTP3

The diagram below shows the components of a dual resilient MTP3 system.



The SCCP interface, SCCP and MTP3 code all reside in the kernel. The application is shown running in a different system, but can run in the same system as SCCP and MTP3 by

connecting to ‘localhost’.

SCCP must pass inward messages to the correct application. This is done by allocating different ranges of SCCP connection identifiers to each application. Inward connections and Unitdata messages are given to each application in a round-robin fashion.

1.3 SCCP library data structures

The SCCP library defines the following major data structures:

- The ssap structure (SCCP service access point). One of these must be created for each SSN. The TCP/IP connection to SCCP is controlled from this structure. Everything refers (directly or indirectly) to an ssap structure. An application would normally only create a single ssap structure.
- The connection structure. This holds all the information for an SCCP message exchange with the SCCP peer. Multiple connection structures can be created on each ssap.
- The msg structure. This is used for passing SCCP messages between the application and the library.
- The sccp_addr structure. This is used to hold address information.

All the fields of the ssap and connection structures are private to the library.

1.4 Functional Overview

AN SCCP application should perform the following steps:

- Create a ssap structure and initialise the configurable fields (from a configuration file or using the API calls).
- Connect to the SCCP systems.

If the application is going to initiate an SCCP connection:

- Create a connection structure.
- Set the required destination address.

If the application is a server, wait for the first message; the library will allocate a connection area.

- Call the appropriate function to send a connect request/confirm/refused, data or disconnect message.
- Wait for a response on either the ssap queue, or connection queue (the response is added to both).
- When all messages have been sent/received, delete the connection.

Connectionless (class 0 and 1) messages can be sent from any idle connection data area, inbound messages are queued on a single connection which will be created if needed
An application can create any (reasonable) number of connections.

1.5 Relationship between SCCP and TCAP

The Aculab SCCP and TCAP APIs use the same method to communicate with the SCCP driver code, and the two libraries share many source files. There are, however, no run-time dependencies between the libraries and applications can safely link to both libraries.

The driver tracing for SCCP API requests may imply that the requests are being processed as TCAP. This is just the way the driver components are named.

SCCP connection local references are 24bit values. The Aculab SCCP API splits this using the upper 12 bits to select the application and the lower 12 to select the connection within the application. Since TCAP transaction identifiers are 32 bits (and split 12/20) some trace entries from common code will show an SCCP local reference as `xxx00yyy` the inserted zeros are deleted before the data is placed in any SCCP message.

2 API Functions

2.1 SCCP API functions

2.1.1 Abbreviations and nomenclature

The following are used:

condvar	condition variable
pdu	protocol data unit
ssap	SCCP service access point
connection	a set of messages using the same connection-id
con	connection, usually the library data structure

The word 'connection' is overused. In most places it refers to the library data structure that contains the information about an SCCP signalling connection. In some places it will refer to the TCP/IP connection from the application to the SCCP driver code, and in others to the SCCP signalling relation itself.

2.1.2 SCCP Header files

All the definitions start `acu_sccp_` or `ACU_SCCP_` (or similar) in order to avoid polluting other namespaces.

The definitions are all in C, but can be used from C++ applications.

Note **A significant amount of pre-processor 'magic' is used to avoid replicating information.**

2.1.2.1 sccp_api.h

This header file contains all the definitions for the SCCP API.

The majority of the structures are described with the function that uses them. Additional information is in Appendix B:

2.1.2.2 sccp_synch.h

This header file contains the definitions for the synchronisation functions.

2.1.3 Configurable parameters

SCCP's configurable values can either be read from a configuration file when an ssap is created, or set directly on the ssap or connection by function call.

Whenever a connection is created, it gets a copy of its configuration information from its ssap.

Once the SCCP application has connected to the SS7 driver, parameters can also be changed using `ss7maint`. This is particularly useful for changing the trace parameters.

Configurable parameters can be placed into three groups

ssap parameters: these control tracing and the connection to the SS7 driver:

Name	Type	Default	Description
`LOGFILE`	string		Name of logfile to open.
`LOGFILE_MAX_SIZE`	integer	`1000000`	Size (bytes) before logfile rotated.
`LOGFILE_APPEND`	boolean	`no`	Append to existing logfile.
`LOGFILE_OLD_KEPT`	integer	`5`	Number of old logfiles kept.
`LOGFILE_FLOCK_INDEX`	boolean	`yes`	flock() logfile.index during log rotation.
`TRACE_TAG`	string		Name for trace entries.
`TRACE_BUFFER_SIZE`	integer	`32768`	Size of cyclic trace buffer.
`TRACE_MODE`	integer	`0`	Determines when trace buffer is written to file, see section 2.1.4
`CO_SERVER`	boolean	`no`	Process inward connections.
`CL_SERVER`	boolean	`no`	Process inward connectionless.
`SINGLE_THREADED`	boolean	`no`	Block on removing messages from ssap queue is not applied.
`NI`	integer	from mtp3	Network Indicator.
`TRACE_LEVEL_ALL`	integer	`5`	Set all trace levels.
`TRACE_LEVEL_xxx`	integer	`5`	Set trace level for source '`xxx`', see section 2.1.4.
`TRACE_LEVEL(n)`	integer	`5`	Set trace level for source '`n`'.
`HOST_A_NAME`	string	`127.0.0.1`	Name and IP addresses of host A (see below).
`HOST_A_PORT`	integer	`8256`	TCP/IP port number.
`HOST_A_PASSWORD`	string		Password for host A.
`HOST_B_NAME`	string		Name and IP addresses of host B (see below).
`HOST_B_PORT`	integer	`8256`	TCP/IP port number.
`HOST_B_PASSWORD`	string		Password for host B.
`RX_BUFLEN`	integer	`130172`	Size (bytes) of TCP/IP receive buffer.
`TX_QUEUE_LEN`	integer	`16`	Number of SCCP messages queued before transmit flow control reported.
`KEEPALIVE_TIMEOUT`	integer	`10`	Seconds between keepalives, set to zero to disable keepalives.
`CONNECT_TIMEOUT`	integer	`10`	Timeout for TCP/IP connection establishment.
`TX_BYTE_WINDOW`	integer	`2920`	Number of data bytes sent to driver before an ack is requested.

Enclose string parameter values that contain spaces (or other special characters) in double quotes.

If the `CO_SERVER` and `CL_SERVER` options are changed after the connection to the driver is made, then the driver is informed of the new value. This allows one node of a distributed application to gracefully shutdown.
General connection parameters are settable on both ssap and connections; connections inherit the values from the ssap:

The `HOST_A_NAME` and `HOST_B_NAME` fields consist of a hostname optionally followed a comma separated list of numeric IP addresses (IPv4 or IPv6). If there are no numeric addresses `getaddrinfo()` is called to resolve the hostname to a list of addresses, otherwise the hostname is ignored unless it is a valid numeric IP address. The returned addresses are tried in turn when connecting to the server.

Name	Type	Default	Description
QOS_RET_OPT	boolean	no	SCCP 'return on error'.
QOS_SEQ_CTRL	boolean	no	SCCP 'sequential delivery', if enabled the 'sls' value is taken from the low bits of the local connection id.
QOS_PRIORITY	integer	~0u	SCCP 'message priority'. ~0u requests the default of 0 for ANSI and absent for ITU.
QOS_RESPONSE_PRI	integer	~0u	SCCP 'message priority' for response messages, default (~0u) is 1 for ANSI and absent for ITU.

SCCP connection-oriented protocol timers. The ssap values will be overwritten with values obtained from the SCCP driver (taken from the driver configuration file). Connections inherit the values from the ssap, all timer values are in seconds:

Name	Type	Range	Description
T_CONN_EST	integer	60-120	Waiting for connect confirm.
T_IAS	integer	300-600	Idle time before IT message send.
T_IAR	integer	660-1320	Waiting to receive any message.
T_REL	integer	10-20	Waiting for release complete.
T_REPEAT_REL	integer	10-20	Retransmit RLSD after initial T_REL.
T_INT	integer	60-120	Waiting for RLC after initial T_REL.

Note **RLSD messages are retransmitted by the driver, so T_REPEAT_REL is unused**

Address parameters, local and remote (replace LOCAL with REMOTE) are settable on ssaps and connections; connections inherit the values from the ssap. See section B.2 for further details:

Name	Type	Description
LOCAL_FLAGS	integer	Address flags.
LOCAL_GTI	integer	Global Title Indicator.
LOCAL_SSN	integer	SSN.
LOCAL_PC	integer	SCCP address pointcode.
LOCAL_RL_PC	integer	MTP routing label pointcode.
LOCAL_TT	integer	Translation Type.
LOCAL_NP	integer	Numbering Plan.
LOCAL_ES	integer	Encoding Scheme.
LOCAL_NAI	integer	Nature of Address Indicator.
LOCAL_GT_DIGITS	BCD	Global Title digits.

The eight address fields (GTI, SSN, PC, RL_PC, TT, NP, ES and NAI) have a 'data valid' bit set whenever they are set via the configuration interface. This bit can be cleared by setting the parameter CLEAR_LOCAL_SSN (etc) to an empty string. This might be needed to stop SCCP including the parameter (e.g. the local ssn) in a message.

When calling the functions to set configuration item, the names above must be preceded by ACU_SCCP_CFG_ (e.g. ACU_SCCP_CFG_REMOTE_PC).

For ANSI/China networks the pointcodes can be specified in 8-8-8 format, although they are currently always traced in decimal.

Note **The configured values for the remote address are overwritten with the actual remote address when the first backwards message arrives.**

2.1.3.1 Configuration file format

The SCCP configuration file has a similar format to that of the ss7 protocol stack. It should contain a single block of configuration data bracketed between `[SCCP]` and `[endSCCP]`. Each line inside the configuration block has the format '*`parameter = value`*', where *`parameter`* is one of the configurable parameter names, and *`value`* is the required value.

Comments can be added to any line by preceding the comment with a '#' character. Blank lines are ignored. The lines before `[SCCP]` and after `[endSCCP]` are currently ignored, but this isn't guaranteed as additional sections may be added at some later release.

The parameter names can be specified in upper or lower case. For compatibility with other parts of the Aculab SS7 protocol stack, the configuration file can contain `localxxx` and `remotexxx` instead of `local_xxx` and `remote_xxx`.

For example:

```
[SCCP]
    trace_tag = program_name
    logfile_append = y
    logfile = sccp2020.log
    localpc = 2020
    localssn = 27
    remote_pc = 7070
    remote_ssn = 143
    co_server = y
    host_a_name = sccp_host_a
    host_a_password = sccp_password
    host_b_name = sccp_host_b
    host_b_password = sccp_password
[EndSCCP]
```

The SS7 stack configuration file on `sccp_host_a` (that for `sccp_host_b` is similar) needs to contain the following:

```
[SP]
    LocalPC = 2020
    [SCCP]
        sccp_listen = y
        password = sccp_password
        master = y
    [EndSCCP]
    [MTP3]
        [DUAL]
            host = sccp_host_b
            ipaddresses = 192.168.1.2
            master = y
            listen = 0
            connect = 1
            password = dual_password
        [EndDUAL]
        [DESTINATION]
             RemotePC = 7070
        [EndDESTINATION]
    [EndMTP3]
[EndSP]
```

2.1.4 Tracing

The SCCP library contains extensive tracing of the API calls and the interface to SCCP. Each trace call specifies a trace source (0 to 63) and trace level (0 to 15). The level of trace output can be set separately for each trace source from the application configuration file, from the program by calling `acu_sccp_ssap_set_cfg_int()`, or from the command line by running `ss7maint sccpconfig`.

Tracing starts when the `LOGFILE` parameter is set for the ssap.

By default the trace buffer is written to the logfile after each trace entry is complete. This can be modified by setting `TRACE_MODE`. to `ACU_SCCP_TRACE_MODE_BLOCK` (`1`) or `ACU_SCCP_TRACE_MODE_CYCLIC` (`2`). In block mode the buffer is written when full, in cyclic mode the buffer just wraps (discarding trace entries). The buffer is always written when a message with trace level 0 or 1 is written, or when the trace mode is set (even if the value doesn't change).

The logfile is always opened in 'append' mode (although it may be truncated). On Linux systems this allows multiple programs and ssaps to log to a common file.

Note **On Windows systems, using a common log file can lead to corrupted log entries.**

If the size of the logfile exceeds the `LOGFILE_MAX_SIZE` bytes parameter, then a new logfile *`logfile`*`.1` (et seq) is opened. The number of old logfiles is restricted to `LOGFILE_OLD_KEPT` (default is 5). The sequence number of the current logfile is kept in *`logfile`*`.index`

On Linix system the logfile rotation uses `flock()` (on the index file) to maintain consistency between multiple applications. Some NFS file systems block the `flock()` call indefinitely, it can be disabled by setting `LOGFILE_FLOCK_INDEX` to `0`.

The logfile is formatted so that '`ss7maint decode`' can be used to pretty-print the sccp messages.

Functions are supplied so that the application can add items to the library log file.

Note **The default level of tracing has a significant performance penalty.**

Trace sources:

`APPLICATION(0)` to `APPLICATION(15)`	`0x00` `0x0f`	16 trace sources available for application use
`API_ENTRY`	`0x10`	Entry to API routine (not all functions make trace calls)
`API_EXIT`	`0x11`	Normal exit from API function
`API_ERROR`	`0x12`	Error exit from API function (might be an internal function)
`API_EVENT`	`0x13`	Significant event
`API_INFO`	`0x14`	Additional information
`API_CONFIG`	`0x15`	Configuration changes
`API_OP_TIMER`	`0x16`	Connection timers
`TCP`	`0x30`	TCP/IP connection establishment and control
`TCP_SEND`	`0x31`	TCP/IP messages being sent
`TCP_RECV`	`0x32`	TCP/IP messages being received

Other values are reserved for future use.

2.1.4.1 acu_sccp_trace/trace_v/trace_buf

```
void acu_sccp_trace_v(acu_sccp_ssap_t *ssap, unsigned int flags,
     const void *buf, int buf_len, const char *fmt, va_list ap);
void acu_sccp_trace(acu_sccp_ssap_t *ssap, unsigned int flags,
     const char *fmt, ...);
void acu_sccp_trace_buf(acu_sccp_ssap_t *ssap, unsigned int flags,
     const void *buf, int buf_len, const char *fmt, ...);
```

Purpose

These functions output text to the trace buffer, `acu_sccp_trace_buf()` adds a hexdump of `buf` following the text output.

Parameters

`ssap` the ssap structure the trace is for

`flags` usually `ACU_SCCP_TRF(part, source, level)`

- `part` One of `FIRST`, `MIDDLE`, `LAST` or `ONLY` indicating which part of the trace entry is being generated.
- `source` `APPLICATION(n)` for n between 0 and 9, identifying the source of the trace.
- `level` 0 to 15 indicating the level (high number for more verbose trace) of this call, the default is usually 5

`buf` address of buffer area to hexdump following the format output

`buf_len` number of bytes to hexdump

`fmt` printf format for trace arguments

`ap` variable argument list for underlying printf call

The `flags` parameter specifies the trace source and level and also indicates which part of a trace entry is being generated (allowing a single trace entry to be generated by multiple calls to the trace functions). A short header including the system time is output at the start of each trace entry. The trace is locked while a trace entry is generated (i.e. from the call specifying `FIRST` to that specifying `LAST`) to avoid trace output from different threads being intermixed – even when multiple threads try to write concurrently to the same log file.

The trace is output if the `level` in the call is less than that set using `acu_sccp_ssap_set_cfg_int()` for the same source.

Note **The trace is formatted by a local version of `snprintf()` which does not support floating point format specifiers.**

2.1.4.2 acu_sccp_trace_error

```
int acu_sccp_trace_error(acu_sccp_ssap_t *ssap, const char *fname, int rval,
     const char *fmt, ...);
```

Purpose

This function is used to write a trace entry when one of the SCCP error codes is generated. It is loosely equivalent to calling `acu_sccp_trace()` with flags of `ACU_SCCP_TRF(ONLY, API_ERROR, 5)`.

Parameters

`ssap` the ssap structure the trace is for

`fname` name of function that is returing the error

`rval` SCCP error number (one of `ACU_SCCP_ERROR_xxx`)

`fmt` printf style format string, followed by the arguments

Return value

Always `rval`.

2.1.4.3 acu_sccp_strerror

```
const char *acu_sccp_strerror(int rval, unsigned int flags);
```

Purpose

This function returns a text string that describes an SCCP library error code.

Parameters

`rval` SCCP error number (one of `ACU_SCCP_ERROR_xxx`)

`flags` 0 => return descriptive text, see B.1
1 => return the C name `"ACU_SCCP_ERROR_xxx"`

Return value

A pointer to a static const string describing the error, unless the error number is unknown in which case the address of a static array filled with the text "`error %d unknown`" is returned.

The error text strings are defined by the `ACU_SCCP_ERRORS` define in `sccp_api.h`.

2.1.5 SCCP access functions

2.1.5.1 acu_sccp_ssap_create

```
acu_sccp_ssap_t *acu_sccp_ssap_create(const char *cfg_file,
     acu_sccp_ssap_flags_t flags);
```

Purpose

This function creates a new SCCP access point without establishing the connection to the driver. The application may set parameters from its own configuration information before the connection to the driver is established.

Parameters

`cfg_file` Name of the configuration file to use, may be `NULL`
If the file cannot be opened, and the name doesn't contain a '/' (or '\') then the library will look for the file in the directories `${HOME}` and `${ACULAB_ROOT}/ss7`

`flags` Bitwise OR of:

`ACU_SCCP_CO_SERVER`	Application is a server process and will be given new connections
`ACU_SCCP_CL_SERVER`	Application will be given inward connectionless messages.
`ACU_SCCP_LOG_APPEND`	Append to the log file.
`ACU_SCCP_STATUS_IND`	The application will be given all the status indications from SCCP.
`ACU_SCCP_LOG_STDERR`	Write initialisation errors to `stderr`

The `CO_SERVER`, `CL_SERVER` and `LOG_APPEND` flags can also be set from the configuration.

Return value

The address of an initialised `acu_sccp_ssap_t` structure, or `NULL` if `malloc()` fails or the configuration file cannot be accessed.

2.1.5.2 acu_sccp_ssap_delete

```
void acu_sccp_ssap_delete(acu_sccp_ssap_t *ssap);
```

Purpose

This function deletes an SCCP access point, and any SCCP connections created on it.

Parameters

`ssap` The address of the `acu_sccp_ssap_t` structure to delete

Return value

None.

2.1.5.3 acu_sccp_ssap_connect_driver

```
int acu_sccp_ssap_connect_driver(acu_sccp_ssap_t *ssap);
```

Purpose

This function causes the SCCP library to try to establish a TCP/IP connection between the ssap library and the SCCP driver code.

The local SSN and POINTCODE must be set before this is called.

After this function completes the TCP connection attempt continues asynchronously, and it may subsequently succeed or fail. When the connection attempt completes, a message of type `ACU_SCCP_MSG_CON_STATE` will be sent to the ssap, indicating a state transition. When that message is seen, the application should check the ssap connection state, using `acu_sccp_get_con_state()`, to see whether the connection was successfully established.

Note **SCCP connections cannot be created until the connection to the driver has been established.**

Parameters

`ssap` The address of the `acu_sccp_ssap_t` structure to connect to the driver.

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.5.4 acu_sccp_ssap_set_cfg_int/str

```
int acu_sccp_ssap_set_cfg_int(acu_sccp_ssap_t *ssap,
     acu_sccp_cfg_param_t param, unsigned int i_val);
int acu_sccp_ssap_set_cfg_str(acu_sccp_ssap_t *ssap,
     acu_sccp_cfg_param_t param, const char *s_val);
```

Purpose

These functions set a configurable value of the ssap.

Integer parameters can be set using either function.

Refer to section 2.1.3 for a list of configurable parameters.
Connections inherit their configuration from the ssap.

Parameters

`ssap` The address of the `acu_sccp_ssap_t` structure to modify
`param` Configuration parameter to modify
`i_val` Integer value for parameter
`s_val` String value for parameter

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.5.5 acu_sccp_ssap_get_locaddr/remaddr

```
acu_sccp_addr_t *acu_sccp_ssap_get_locaddr(acu_sccp_ssap_t *ssap);
acu_sccp_addr_t *acu_sccp_ssap_get_remaddr(acu_sccp_ssap_t *ssap);
```

Purpose

These functions return a pointer to the local/remote SCCP address information for this ssap. The application can change the structure through the returned pointer. The values can also be set from the configuration file and by the configuration functions.

The local SSN and POINTCODE values are used when connecting to the driver.

Parameters

`ssap` The address of the `acu_sccp_ssap_t` structure.

Return value

The address of the `acu_sccp_addr_t` structure within the ssap data area, or NULL if the ssap pointer is invalid.

See section B.2 for details of the `acu_sccp_addr_t` structure.

2.1.6 Connection structure functions

2.1.6.1 acu_sccp_con_create

```
acu_sccp_con_t *acu_sccp_con_create(acu_sccp_ssap_t ssap);
```

Purpose

This function creates a new SCCP connection data area on the specified ssap.

Parameters

`ssap`	The ssap on which to create a connection

Return value

The address of an initialised `acu_ccp_con_t` structure, or `NULL` if the ssap isn't connected to SCCP or if `malloc()` fails.

2.1.6.2 acu_sccp_con_delete

```
void acu_sccp_con_delete(acu_sccp_con_t *con);
```

Purpose

This function deletes an SCCP connection data area and all memory associated with it.

This has the effect of a 'pre-arranged' end on any active SCCP connection.

Parameters

`con`	The address of the `acu_sccp_con_t` structure to delete

Return value

None.

Note **The connection data isn't actually deleted until the last message that references the connection is freed.**

2.1.6.3 acu_sccp_ssap_get_unitdata_con

```
acu_sccp_con_t *acu_sccp_ssap_get_unitdata_con(acu_sccp_ssap_t ssap);
```

Purpose

This function returns the address of the connection on which received connectionless unitdata and notice messages are queued.

The connection is created either by this call, or when the first connectionless message is received. If the connection is deleted it will be re-created when needed.

Note **An application will only be given connectionless messages if '`cl_server = y`' is set in the ssap's configuration.**

Connectionless messages can be sent from this connection, or from another connection created by `acu_sccp_con_create()`.

Parameters

`ssap`	The ssap whose unidirectional connection is required

Return value

The address of an `acu_sccp_con_t` structure, or `NULL` if `malloc` fails.

2.1.6.4 acu_sccp_con_set_userptr

```
void acu_sccp_con_set_userptr(acu_sccp_con_t *con, void *userptr);
```

Purpose

This function saves the pointer to an application data area for this connection.

Parameters

`con`	The address of the `acu_sccp_con_t` structure to modify
`userptr`	The pointer to save

2.1.6.5 acu_sccp_con_get_userptr

```
void *acu_sccp_con_get_userptr(acu_sccp_con_t *con);
```

Purpose

This function retrieves the pointer saved by `acu_sccp_set_userptr()`.

Parameters

`con`	The address of the `acu_sccp_con_t` structure

Return value

The pointer saved previously.

2.1.6.6 acu_sccp_con_get_ids

```
int acu_sccp_con_get_ids(acu_sccp_con_t *con, unsigned int *loc_ref,
     unsigned int *rem_ref, unsigned int *rem_ref_len);
```

Purpose

This function gets the connection local references assigned to the connection.

Parameters

`con`	The address of the `acu_sccp_con_t` structure to modify
`loc_ref`	Address of location to write the reference assigned by this system
`rem_ref`	Address of location to write the reference assigned by the remote system
`rem_id_len`	Address of location to write the length of the remote reference

Any of `loc_ref`, `rem_ref` and `rem_ref_len` may be `NULL` in which case nothing is returned. `*rem_ref_len` will be set to zero if the remote connection identifier is unknown.

For SCCP the references are always 3 bytes long.

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.6.7 acu_sccp_con_set_cfg_int/str

```
int acu_sccp_con_set_cfg_int(acu_sccp_con_t *con, acu_sccp_cfg_param_t param,
     unsigned int i_val);
int acu_sccp_con_set_cfg_str(acu_sccp_con_t *con, acu_sccp_cfg_param_t param,
     const char *s_val);
```

Purpose

These functions set a configurable value of the connection data area. The default values for these are inherited from the ssap when a connection is created.
`acu_sccp_con_set_cfg_str()` can be used to set an integer parameter from a character string value.

Refer to section 2.1.3 for a list of the configurable parameters.

Parameters

`con`	The address of the `acu_sccp_con_t` structure to modify
`param`	Configuration parameter to modify
`i_val`	Integer value for parameter
`s_val`	String value for parameter

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.6.8 acu_sccp_con_get_locaddr/remaddr

```
acu_sccp_addr_t *acu_sccp_con_get_locaddr(acu_sccp_con_t *con);
acu_sccp_addr_t *acu_sccp_con_get_remaddr(acu_sccp_con_t *con);
```

Purpose

These functions return a pointer to the local/remote SCCP address information for this connection. The application can change the structure through the returned pointer.

The default values for these are inherited from the ssap when a connection is created.

The remote address will be set from information in the first message received for each connection. For connection-oriented SCCP, only the remote point code is significant once the SCCP connection has been established.

To respond from the destination address in a received connectionless message (rather than from the configured address) set the connections local address with:

```
*acu_sccp_con_get_locaddr(connection) = *msg->tm_local_addr;
```

when processing the received message.

Parameters

`con`	Connection

Return value

The address of the structure or NULL if the `con` pointer is invalid.

See section B.2 for details of the `acu_sccp_addr_t` structure.

2.1.7 Message sending functions

2.1.7.1 acu_sccp_connect_request

```
int acu_sccp_connect_request(acu_sccp_con_t *con, const void *user_data,
     unsigned int data_len);
```

Purpose

This function sends an SCCP CR message to the remote system to request that a new connection be established.

The local address will be encoded unless its `sa_valid` field is zero.

Parameters

`con`	Address of connection data area
`user_data`	Address of optional connect user data
`data_len`	Length in bytes of the connect user data

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.7.2 acu_sccp_connect_confirm

```
int acu_sccp_connect_confirm(acu_sccp_con_t *con, const void *user_data,
     unsigned int data_len);
```

Purpose

This function sends an SCCP CC message to the remote system in order to accept an incoming connection.

The local address will be encoded unless its `sa_valid` field is zero.

Parameters

`con`	Address of connection data area
`user_data`	Address of optional connect user data
`data_len`	Length in bytes of the connect user data

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.7.3 acu_sccp_connect_refused

```
int acu_sccp_connect_refused(acu_sccp_con_t *con, int cause,
     const void *user_data, unsigned int data_len);
```

Purpose

This function sends an SCCP CREF message to the remote system in order to reject an incoming connection.

The local address will be encoded unless its `sa_valid` field is zero.

Parameters

`con`	Address of connection data area
`cause`	Refusal cause value from Q.713 section 3.15
`user_data`	Address of optional connect user data
`data_len`	Length in bytes of the connect user data

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.7.4 acu_sccp_disconnect

```
int acu_sccp_disconnect(acu_sccp_con_t *con, int cause,
     const void *user_data, unsigned int data_len);
```

Purpose

This function sends an SCCP RLSD message to the remote system in order to release a connection.

Parameters

`con`	Address of connection data area
`cause`	Release cause value from Q.713 section 3.11
`user_data`	Address of optional disconnect userdata
`data_len`	Length in bytes of the disconnect user data

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.7.5 acu_sccp_data_request

```
int acu_sccp_data_request(acu_sccp_con_t *con, const void *user_data,
     unsigned int data_len);
```

Purpose

This function sends one or more SCCP DT1 messages to the remote system containing the user data.

If more than 255 bytes of user data are supplied, multiple DT1 messages are sent with all but the last having the 'M' bit (of the Segmenting/reassembling parameter) set to 1.

Note **Class 2 SCCP cannot detect missing DT1 messages, if a packet is lost from (or mis-sequenced in) an M-bit sequence then the remote system will receive corrupted data.**

This function may fail reporting `ERROR_CONNECTION_OUTSTATE` if an inwards disconnect occurs.

Parameters

`con`	Address of connection data area
`user_data`	Address of the user data to send
`data_len`	Length in bytes of the user data

Return value

Zero if successful, a positive number indicating the number of bytes sent if only part of an M-bit sequence is sent before an error occurs, `ACU_SCCP_ERROR_xxx` on failure before any bytes are sent.

2.1.7.6 acu_sccp_unitdata_request

```
int acu_sccp_unitdata_request(acu_sccp_con_t *con, const void *user_data,
     unsigned int data_len);
```

Purpose

This function sends an SCCP UDT, XUDT or LUDT message to the remote system containing the user data.

Any 'con' structure (in any state) can be used.

Parameters

`con`	Address of connection data area
`user_data`	Address of the user data
`data_len`	Length in bytes of the user data

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.8 Message receiving functions

SCCP messages received from the driver (via TCP/IP) are queued on, and can be retrieved from queues on both the ssap and connection data areas.

Every message must be freed at some point by calling `acu_sccp_msg_free()`.

The application must normally call `acu_sccp_con_unblock()` after processing messages that refer to a connection in order to make any further messages for that connection available from the ssap queue. The block is applied in order to stop an application having more than one thread processing messages for a single connection.

If the application only ever uses a single thread to access SCCP then `SINGLE_THREADED=y` can be configured and the block will not be applied.

The data bytes of the message itself are within a circular buffer used to receive data from the TCP/IP connection. The application must call `acu_sccp_msg_free()` or `acu_sccp_msg_copy_rx_buffer()` in a timely manner to avoid blocking messages for other SCCP connections.

The initial elements of `acu_sccp_msg_t` are exposed in the header file and can be read by the application.

As well as received SCCP messages, other indications from the library to the application are passed through this interface. These additional messages are only added to the ssap queue.

2.1.8.1 acu_sccp_ssap_msg_get

```
int acu_sccp_ssap_msg_get(acu_sccp_ssap_t *ssap, int tmo_ms,
     acu_sccp_msg_t **msgp);
```

Purpose

This function retrieves the next inbound sccp message from the queue associated with the specified ssap.

A new connection structure is automatically allocated when a CR message is retrieved.

Note **An application will only be given CR messages if '`co_server = y`' is set in the ssap's configuration.**

Note **An application will only be given UDT messages if '`cl_server = y`' is set in the ssap's configuration.**

Parameters

`ssap`	Address of ssap data area
`tmo_ms`	Time in milliseconds to wait for a message, 0 => don't wait, -1 => wait forever
`msgp`	Address of parameter where the message structure address will be written

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.
`*msgp` will be set to `NULL` if the function fails.

The following fields of the message are set:

`tm_msg_type` Type of message/indication, one of:

`ACU_SCCP_MSG_DATA_IND`	Connection data from remote SCCP (DT1 messages)
`ACU_SCCP_MSG_CONNECT_IND`	
`ACU_SCCP_MSG_CONNECT_CONF`	
`ACU_SCCP_MSG_CONREF_CONF`	
`ACU_SCCP_MSG_DISCON_IND`	
`ACU_SCCP_MSG_DISCON_CONF`	
`ACU_SCCP_MSG_ERROR_IND`	
`ACU_SCCP_MSG_UNITDATA`	Unitdata from remote SCCP ({L|X}UDT messages)
`ACU_SCCP_MSG_NOTICE`	Error report from remote SCCP ({L|X}UDTS message)
`ACU_SCCP_MSG_TIMEOUT`	Timer expired

ACU_SCCP_MSG_CON_STATE	Change in state of TCP/IP connections to the driver
ACU_SCCP_MSG_USER_STATUS	Change in status of remote user (from local SCCP)
ACU_SCCP_MSG_SP_STATUS	Change in status of remote signalling point from local SCCP
tm_ssap	Address of associated ssap
tm_con	Address of associated connection (may be NULL)

For indications from the remote SCCP the following are also set:

tm_local_addr	Destination (ie our) address from SCCP message, NULL if not present
tm_remote_addr	Source (ie remote) address from SCCP message, NULL if not present
tm_data	Address of any associated user data, NULL if none present
tm_data_length	Number of bytes of user data
tm_ret_opt	Received SCCP 'return on error' option (UDT messages)
tm_class	Received message class (UDT, CR, CC messages)
tm_cause	Received 'cause' (RLSD, UDTS, ERR, or CREF message)
tm_priority	ANSI message priority, ITU importance (0xff if ITU option not present)

For TIMEOUT message the following is set:

tm_timer_id	Number of timer that expired.

For CON_STATUS call acu_sccp_msg_get_con_state() to find the connection states at the time the message was generated, or acu_sccp_get_con_state() to find the current state.

For USER_STATUS and SP_STATUS call acu_sccp_msg_get_sccp_status() to determine the concerned pointcode and SSN.

Failure to establish an outward connect may be indicated by an ERROR_IND or DISCON_IND as well as the more usual CONREF_IND. Similarly disconnection of an active connection might be signalled by a ERROR_IND. The message generated depends on the type of message received from the network (and the tm_cause value coding depends on the message type).

If the library or driver initiates a disconnect then the message type passed to the application matches that sent to the remote system, but the cause value is or'ed with 0x80.

2.1.8.2 acu_sccp_con_msg_get

```
int acu_sccp_con_msg_get(acu_sccp_con_t *con, int tmo_ms,
     acu_sccp_msg_t **msgp);
```

Purpose

This function retrieves the next inbound sccp message from the queue associated with the specified connection.

Refer to acu_sccp_ssap_msg_get() for information on the possible message types.

Parameters

con	Address of connection data area
tmo_ms	Time in milliseconds to wait for a message, 0 => don't wait, -1 => wait forever
msgp	Address of parameter where the message structure address will be written

Return value

Zero if successful, ACU_SCCP_ERROR_xxx on failure.
msgp will be set to NULL if the function fails.

2.1.8.3 acu_sccp_event_msg_get

```
int acu_sccp_event_msg_get(acu_sccp_event_t *event, acu_sccp_msg_t **msgp);
```

Purpose

This function retrieves the next inbound sccp message from one of the queues associated with `event`. Refer to section 2.1.13 for more information on the event mechanism.

Refer to `acu_sccp_ssap_msg_get()` for information on the possible message types.

Parameters

`event`	Address of an event data area
`msgp`	Address of parameter where the message structure address will be written

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.
`msgp` will be set to `NULL` if the function fails.

2.1.8.4 acu_sccp_msg_free

```
void acu_sccp_msg_free(acu_sccp_msg_t *msg);
```

Purpose

This function releases all resources associated with the specified `msg`.

Parameters

`msg`	Address of message to free

Note **Every message must be explicitly freed using this function.**

2.1.8.5 acu_sccp_msg_copy_rx_buffer

```
int acu_sccp_msg_copy_rx_buffer(acu_sccp_msg_t *msg);
```

Purpose

This function copies any data that `msg` references that is in the TCP/IP receive buffer area to a malloced memory area and updates all of the pointers within the message structure to reference the correct locations in the new buffer.

The memory will be freed when `msg` is freed.

Parameters

`msg`	Address of message to process

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.8.6 acu_sccp_con_unblock

```
void acu_sccp_con_unblock(acu_sccp_con_t *con);
```

Purpose

This function removes the block that stops inbound messages for the given connection from being retrieved from the corresponding ssap queue.

The block exists so that a pool of threads can be used to process messages from the ssap queue without having to worry about multiple threads processing messages from the same connection. It also allows the application to use a separate thread for each connection, although this is discouraged because of the resource issues with large numbers of threads.

Parameters

`con` Address of connection data area

2.1.8.7 acu_sccp_con_block

```
int acu_sccp_con_block(acu_sccp_con_t *con);
```

Purpose

This function sets the block that stops inbound messages for the given connection from being retrieved from the corresponding ssap queue.

The block is automatically set whenever a message is retrieved for a connection unless `SINGLE_THREADED=y` is configured.

It may be necessary to manually set the block on a newly created connection.

Parameters

`con` Address of connection data area

Return value

One if the block was already set, zero otherwise.

2.1.8.8 acu_sccp_ssap_wakeup_msg_get

```
void acu_sccp_ssap_wakeup_msg_get(acu_sccp_ssap_t *ssap);
```

Purpose

This function wakeup up all threads sleeping in acu_sccp_ssap_msg_get() for the specified ssap.

Parameters

`ssap` Address of ssap data area

2.1.8.9 acu_sccp_can_wakeup_msg_get

```
void acu_sccp_con_wakeup_msg_get(acu_sccp_con_t *con);
```

Purpose

This function wakeup up all threads sleeping in acu_sccp_con_msg_get() for the specified connection.

Parameters

`con` Address of the connection data area

2.1.9 Timer functions

There are 256 timers defined for each connection structure. Timers 250 and above are reserved for use by the connection-oriented SCCP protocol code, the other timers can be used for any purpose by the application.

The timer resolution is 1 second and the maximum timeout 9 hours. Timers are guaranteed not to expire in less than the specified period. Even if the system is idle they may not expire until almost 2 seconds after the nominal expiry time.

When a timer expires, a message with `tm_msg_type` set to `ACU_SCCP_MSG_TIMEOUT` will be queued. The timer number is placed in the `tm_timer_id` field.

Note **The timer resolution is 1 second. A 1 second timer is guaranteed to sleep for at least 1 second, but may sleep for almost 3 seconds.**

2.1.9.1 acu_sccp_con_timer_start

```
int acu_sccp_con_timer_start(acu_sccp_con_t *con, int timer_id,
     unsigned int tmo_secs);
```

Purpose

This function starts the requested timer.

This can be used by an application to run a timer for its own purposes.

Parameters

`con`	Connection data area
`timer_id`	Timer identifier, 0 to 249
`tmo_secs`	Required timeout in seconds

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.9.2 acu_sccp_con_timer_restart

```
int acu_sccp_con_timer_restart(acu_sccp_con_t *con, int timer_id,
     unsigned int tmo_secs);
```

Purpose

This function restarts the requested timer.
An error will be returned it the timer isn't running (e.g. if it has just expired).

Parameters

`con`	Connection data area
`timer_id`	Timer identifier
`tmo_secs`	Required timeout in seconds

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.9.3 acu_sccp_con_timer_cancel

```
int acu_sccp_con_timer_cancel(acu_sccp_con_t *con, int timer_id);
```

Purpose

This function cancels the requested timer.
An error will be returned it the timer isn't running (e.g. if it has just expired).

Parameters

`con`	Connection data area
`timer_id`	Timer identifier

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.10 TCP/IP connection status functions

The SCCP library connects to the driver using TCP/IP. It connects asynchronously and will automatically attempt to reconnect if the connection fails for any reason.

Changes in the TCP/IP connection's state are reported by queueing an `ACU_SCCP_MSG_CON_STATE` message onto the ssap message queue. The application must wait until the `IN_SERVICE` state is reported before calling `acu_sccp_con_create()`.

Note **The `IDLE` -> `CONNECTING` and `CONNECTING` -> `CONNECTED` transitions are not reported.**

2.1.10.1 acu_sccp_get_con_state

```
int acu_sccp_get_con_state(acu_sccp_ssap_t *ssap, int con_id,
     const acu_sccp_con_state_t **con_state);
```

Purpose

This function returns information about the current state of one of the TCP/IP connections to the driver.

Parameters

`ssap`	ssap data area
`con_id`	0 for the connection to 'host a', 1 for that to 'host b'
`con_state`	Pointer filled with address of connection state structure.

The `acu_sccp_con_state_t` structure contains the following fields:

`cs_ipaddr`	IP address of the connected driver (host order)
`cs_tcpport`	TCP/IP port number of the connected driver
`cs_state`	The current state of the connection, one of:
`ACU_SCCP_CON_STATE_IDLE`	Not configured or connect failed
`ACU_SCCP_CON_STATE_CONNECTING`	TCP/IP connection being made
`ACU_SCCP_CON_STATE_CONNECTED`	Initial message handshake in progress
`ACU_SCCP_CON_STATE_IN_SERVICE`	Available for SCCP traffic
	If `IN_SERVICE` the following bits can also be set:
`ACU_SCCP_CON_STATE_RX_BLOCKED`	No space in receive ring buffer area
`ACU_SCCP_CON_STATE_RX_FLOW`	Receive flow controlled off
`ACU_SCCP_CON_STATE_TX_BLOCKED`	No TCP transmit window
`ACU_SCCP_CON_STATE_TX_FLOW`	Transmit flow controlled off
`cs_failure`	The reason why the last connection (or connect attempt) failed, one of:
`ACU_SCCP_CON_FAIL_SSAP_DELETED`	ssap deleted
`ACU_SCCP_CON_FAIL_CON_TIMEOUT`	TCP/IP connect timed out
`ACU_SCCP_CON_FAIL_CON_REJECTED`	TCP/IP connection rejected
`ACU_SCCP_CON_FAIL_LOGIN_REJECTED`	Login sequence failed
`ACU_SCCP_CON_FAIL_INWARD`	Inward disconnected by the driver
`ACU_SCCP_CON_FAIL_KEEPALIVE`	No response to keepalive
`ACU_SCCP_CON_FAIL_BAD_MESSAGE`	Corrupt message received
`cs_fail_text`	Textual description of `cs_failure`, or one of the following texts when the login fails:
`Bad Request`	Major discrepancy between the versions of the SCCP library and the driver
`Responder has gone`	The driver is no longer waiting for connections on the requested TCP/IP port. Driver is probably shut down.
`Unknown service`	SCCP isn't configured in the ss7 driver configuration.
`Unknown service parameter`	SCCP isn't configured on the requested pointcode.
`Incorrect password`	The passwords in the application and driver configuration files do not match.
`Rejected by server`	Connection rejected by SCCP driver stub.
`Bad hash in response`	Three-way login handshake failed.
`cs_tx_qlen`	The number of outbound sccp messages queued within the library.

Application level acknowledgements are used on the TCP/IP connection in order to avoid blocking the TCP/IP connection itself. Thus the `BLOCKED` states should not happen.

Receive flow control is most likely to occur if the application fails to free receive messages – which have pointers directly into the receive ring buffer area.

If transmit flow control is reported the application should take steps to avoid sending further messages. However all messages sent will be queued by the library.

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.11 acu_sccp_msg_get_con_state

```
int acu_sccp_msg_get_con_state(acu_sccp_msg_t *msg,
     const acu_sccp_con_state_t **cs_a, const acu_sccp_con_state_t **cs_b)
```

Purpose

This function resolves pointers to the connection state field(s) in messages of type `ACU_MSG_SCCP_CON_STATE`.

This information relates to the state of the TCP/IP connections to the driver at the time the indication was generated.

Refer to `acu_sccp_get_con_state()` for details of the `acu_sccp_con_state_t` structure.

Parameters

`msg`	Message structure address (from one of the msg_get() functions)
`cs_a`	Address of parameter where the ‘host a’ connection state structure address will be written.
`cs_b`	Address of parameter where the ‘host b’ connection state structure address will be written (where SCCP is configured in ‘dual’ mode).

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

Note **The addresses written to `cs_a` and `cs_b` point into the message itself.**

2.1.12 Remote SP and SSN status functions

The SCCP library receives status indications from the driver that show the accessibility of remote entities. The information is saved so that the application can synchronously determine the current status.

The application can also ask to be notified when the status of a remote pointcode or ssn changes. Such changes are reported by queueing an `ACU_SCCP_MSG_SCCP_STATUS` message onto the ssap message queue.

Additionally the application can request to be given all of the raw status events from SCCP by setting the `ACU_SCCP_STATUS_IND` flag when the ssap is created.

Note **The SCCP protocol does not distinguish between the accessibility of connection-oriented and connectionless users.**

2.1.12.1 acu_sccp_get_sccp_status

```
int acu_sccp_get_sccp_status(acu_sccp_ssap_t *ssap, unsigned int pointcode,
     unsigned int ssn, const acu_sccp_sccp_status_t **sccp_status);
```

Purpose

This function returns information about the current state of the pointcode and ssn.

Parameters

`ssap`	ssap data area
`pointcode`	SS7 pointcode of the remote system.
`ssn`	ssn of remote application
`sccp_status`	Pointer filled with address of the sccp and user state structure.

The `acu_sccp_sccp_status_t` structure contains the following fields:

`tsp_pc`	Remote pointcode	
`tsp_ssn`	ssn of remote application	
`tsp_host`	Either ‘a’ or ‘b’ depending of which SCCP host the information came from.	
`tsp_user_status`	Status of the ssn, one of:	
	`ACU_SCCP_UIS`	User In Service
	`ACU_SCCP_UOS`	User Out of Service
`tsp_sp_status`	Status of the signalling point (from MTP3), one of:	
	`ACU_SCCP_SP_PROHIBIT`	Prohibited
	`ACU_SCCP_SP_ACCESS`	Accessible
`tsp_sccp_status`	Status of the remote SCCP, one of:	
	`ACU_SCCP_REM_SCCP_PROHIBIT`	Prohibited
	`ACU_SCCP_REM_SCCP_UNAVAIL`	Unavailable, reason unknown
	`ACU_SCCP_REM_SCCP_UNEQUIP`	Unequipped
	`ACU_SCCP_REM_SCCP_INACCESS`	Inaccessible
	`ACU_SCCP_REM_SCCP_CONGEST`	Congested
	`ACU_SCCP_REM_SCCP_AVAIL`	Available
`tsp_tx_cong_cost`	A measure of the level of congestion of the remote node.	

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.12.2 acu_sccp_msg_get_sccp_status

```
int acu_sccp_msg_get_sccp_status(acu_sccp_msg_t *msg,
     const acu_sccp_sccp_status_t **sccp_status);
```

Purpose

This function returns the information about the state of a pointcode and ssn from an `ACU_SCCP_MSG_USER_STATUS` or `ACU_SCCP_MSG_SP_STATUS` message.

Parameters

`msg`	Message data area
`sccp_status`	Pointer filled with address of the sccp and user state structure (embedded in the msg).

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.12.3 acu_sccp_enable_user_status

```
int acu_sccp_enable_user_status(acu_sccp_ssap_t *ssap,
     unsigned int pointcode, unsigned, int ssn);
```

Purpose

This function enables the receipt of `ACU_SCCP_MSG_USER_STATUS` messages for the specified pointcode and ssn.

Parameters

`ssap`	ssap data area
`pointcode`	Remote SS7 pointcode from which user status indications are required.
`ssn`	Associated remote SCCP ssn.

The pointcode and/or ssn may be specified as `~0u` in which case indications will be given for all pointcodes/ssns.

Note **User status is only reported if the SS7 stack configuration file contains an SCCP [CONCERNED] section for the pointcode and ssn.**

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.12.4 acu_sccp_enable_sp_status

```
int acu_sccp_enable_sp_status(acu_sccp_ssap_t *ssap, unsigned int pointcode);
```

Purpose

This function enables the receipt of `ACU_SCCP_MSG_SP_STATUS` messages for the specified pointcode.

Parameters

`ssap`	ssap data area
`pointcode`	Remote SS7 pointcode from which user status indications are required.

The pointcode may be specified as `~0u` in which case indications will be given for all pointcodes.

Note **The 'unavailable', 'unequipped', 'inaccessible' and 'congested' statuses are only reported if the SS7 stack configuration file contains an SCCP [CONCERNED] section for the pointcode.**

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.13 SCCP message events

The message receiving functions allow an application to wait for messages on an ssap or a connection, however there are cases where an application may need to wait for messages on a group of connections, or wait for messages from SCCP and events from some other part of the system. The event mechanism described here solves both these problems.

On Microsoft Windows events are implemented using manual-reset events, on Linux systems pipes are used. This allows the application to use `WaitForMultipleObjects` or `poll`/`select` to wait for SCCP messages. Due to scalability problems with both of these it is inappropriate to allocate an event for each connection. The application can create an event which can be signalled by messages being queued at several SCCP connections, or queued at the ssap itself.

Note **The connections must all be on the same ssap**

2.1.13.1 acu_sccp_event_create

```
acu_sccp_event_t *acu_sccp_event_create(acu_sccp_ssap_t *ssap);
```

Purpose

This function creates an event structure.

Parameters

`ssap`	ssap data area

Return value

Address of an initialised event structure. `NULL` if one cannot be allocated or the ssap pointer is invalid.

2.1.13.2 acu_sccp_event_delete

```
void acu_sccp_event_delete(acu_sccp_event_t *event);
```

Purpose

This function unlinks the event from any message queues and then deletes the structure itself.

Parameters

`event`	Address of event structure

Return value

None.

2.1.13.3 acu_sccp_event_wait

```
int acu_sccp_event_wait(acu_sccp_event_t *event, int tmo_ms);
```

Purpose

This function waits for the specified event to be signalled.

Parameters

`event`	Address of event data area
`tmo_ms`	Time to wait in milliseconds, 0 => don't wait, -1 => wait for ever

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.13.4 acu_sccp_event_get_os_event

```
acu_sccp_os_event_t acu_sccp_event_get_os_event(acu_sccp_event_t *event);
```

Purpose

This function returns the operating system data item underlying the given event.

The return type is actually `HANDLE` for Windows and `int` for Linux systems.

Parameters

`event`	Address of event data area

Return value

For Windows the `HANDLE` of the windows event.
For Linux the file descriptor number of the read side of a pipe.
If the call is invalid 0 is returned, care is taken to ensure the pipe fd number isn't zero, one or two.

2.1.13.5 acu_sccp_event_clear

```
void acu_sccp_event_clear(acu_sccp_event_t *event);
```

Purpose

This function clears (i.e.: returns to the non-signalled state) the operating system item underlying the given event.

The event is automatically cleared if when `acu_sccp_event_msg_get()` returns the last message or fails because no messages are present.

Parameters

`event`	Address of event data area

Return value

None.

2.1.13.6 acu_sccp_event_ssap_attach

```
int acu_sccp_event_ssap_attach(acu_sccp_event_t *event,
     acu_sccp_ssap_t *ssap);
```

Purpose

This function adds the message queue for `ssap` as a source for the `event`.

Note **The ssap specified must be the same one specified when the event was created.**

Parameters

`event`	Address of event data area
`ssap`	Address of corresponding ssap data area

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.13.7 acu_sccp_event_ssap_detach

```
int acu_sccp_event_ssap_detach(acu_sccp_event_t *event,
     acu_sccp_ssap_t *ssap);
```

Purpose

This function removes the message queue for the `ssap` from the sources for `event`. It reverses the effect of `acu_sccp_event_ssap_attach()`

Parameters

`event`	Address of event structure
`ssap`	Address of ssap data area

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.13.8 acu_sccp_event_ssap_detach_all

```
int acu_sccp_event_ssap_detach_all(acu_sccp_ssap_t *ssap);
```

Purpose

This function detaches the message queue for the ssap from all events. It is implicitly called if the ssap is deleted.

Parameters

`ssap`	ssap data area

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.13.9 acu_sccp_event_con_attach

```
int acu_sccp_event_con_attach(acu_sccp_event_t *event, acu_sccp_con_t *con);
```

Purpose

This function adds the message queue for `con` as a source for the `event`.

Note **The connection and event must have been created on the same ssap.**

Parameters

`event`	Address of event structures
`con`	Connection data area

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.13.10 acu_sccp_event_con_detach

```
int acu_sccp_event_con_detach(acu_sccp_avent_t *event, acu_sccp_con_t *con);
```

Purpose

This function removes the message queue for `con` from the sources for `event`. It reverses the effect of `acu_sccp_event_con_attach()`

Parameters

`event`	Address of event data area
`con`	Connection data area

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.1.13.11 acu_sccp_event_con_detach_all

```
int acu_sccp_event_con_detach_all(acu_sccp_con_t *con);
```

Purpose

This function detaches the message queue for con from all events. It is implicitly called if the connection is deleted.

Parameters

`con`	Connection data area

Return value

Zero if successful, `ACU_SCCP_ERROR_xxx` on failure.

2.2 Thread support functions

Support functions are provided for multi-threaded applications. They provide an operating independent interface to threads and thread synchronization functions.

Some of the functions are actually `#defines` within the header file `sccp_synch.h`. Because of this, the function arguments may be evaluated more than once.

Additional error information may be available in an operating system dependant manner (e.g.: by inspecting `errno`).

These functions are used within the SCCP library itself. They are exposed by its interface, and portable applications may decide to use them internally.

On Linux systems the functions use the pthread library routines.

Note **Do not cancel threads that are using the SCCP API library.**

2.2.1 Mutex functions

Mutexes are used to protect data areas from concurrent access by more than one thread.

The mutex functions are non-recursive under Linux. Under Windows an error message will be output to `stderr` if a mutex is acquired recursively.

On Windows systems mutexes are implemented using the critical-section functions so that acquiring an uncontested mutex does not require a system call.

2.2.1.1 acu_sccp_mutex_create

```
int acu_sccp_mutex_create(acu_sccp_mutex_t *mutex);
```

Purpose

This function initialises the mutex, allocating any operating system resources needed.

Parameters

`mutex` Address of the mutex to initialise.

Return value

Zero on success, `-1` on failure.

2.2.1.2 acu_sccp_mutex_delete

```
void acu_sccp_mutex_delete(acu_sccp_mutex_t *mutex);
```

Purpose

This function frees all the operating system resources associated with the mutex. The mutex must not be locked when it is deleted.

Parameters

`mutex` Address of the mutex delete.

2.2.1.3 acu_sccp_mutex_lock

```
int acu_sccp_mutex_lock(acu_sccp_mutex_t *mutex);
```

Purpose

This function locks the mutex. If the mutex is already locked the thread will block until the mutex is unlocked.

Parameters

`mutex` Address of the mutex to lock

Return value

Zero on success, `-1` on failure.

2.2.1.4 acu_sccp_mutex_trylock

```
int acu_sccp_mutex_trylock(acu_sccp_mutex_t *mutex);
```

Purpose

This function attempts to lock the mutex. If the mutex is already locked then it will return immediately with a non-zero return value.

Parameters

`mutex` Address of the mutex to lock

Return value

Zero on success, `-1` on failure.

2.2.1.5 acu_sccp_mutex_unlock

```
void acu_sccp_mutex_unlock(acu_sccp_mutex_t *mutex);
```

Purpose

This function unlocks the mutex. A mutex can only be unlocked by the thread that locked it

Parameters

`mutex` Address of mutex to unlock.

2.2.2 Condition variable functions

Condition variables allow one thread to wait until signalled by a different thread. To avoid timing windows all accesses to a condition variable must be protected by the same mutex.

Under Windows, a condition variable is implemented using two manual reset events that are used alternately, with the last thread to exit resetting the event. This avoids any problems associated with `PulseEvent()` and kernel mode APC. It also allows the mutex to be implemented using the critical section functions – avoiding a system call when the mutex is available.

2.2.2.1 acu_sccp_condvar_create

```
int acu_sccp_condvar_create(acu_sccp_cond_t *condvar);
```

Purpose

This function initialises the condition variable, allocating any operating system resources needed.

Parameters

`condvar`	Address of the condition variable to initialise

Return value

Zero on success, `-1` on failure.

2.2.2.2 acu_sccp_condvar_delete

```
void acu_sccp_condvar_delete(acu_sccp_cond_t *condvar);
```

Purpose

This function frees the operating system resources allocated to the condition variable. No threads must be waiting for a condition variable when it is deleted.

Parameters

`condvar`	Condition variable to delete

2.2.2.3 acu_sccp_condvar_wait

```
int acu_sccp_condvar_wait(acu_sccp_cond_t *condvar, acu_sccp_mutex_t *mutex);
```

Purpose

This function waits for the condition variable to be signalled. The `mutex` is released atomically with the wait and re-acquired before the function returns.

Parameters

`condvar`	Condition variable to wait for
`mutex`	Mutex associated with this condvar

Return value

Zero on success, `-1` on failure.

2.2.2.4 acu_sccp_condvar_wait_tmo

```
int acu_sccp_condvar_wait_tmo(acu_sccp_cond_t *condvar,
     acu_sccp_mutex_t *mutex, int millisecs);
```

Purpose

This function waits for the condition variable to be signalled. If the condition variable isn't signalled within the specified timeout it will return `-1`.

Parameters

`condvar`	Condition variable to wait for

`millisecs` The maximum time to wait in milliseconds
`mutex` Mutex associated with this condvar

Return value

Zero on success, `-1` on failure or if the wait times out.

2.2.2.5 acu_sccp_condvar_broadcast

```
int acu_sccp_condvar_broadcast(acu_sccp_cond_t *condvar);
```

Purpose

This function signals the condition variable. All threads blocked in `acu_sccp_condvar_wait()` or `acu_sccp_condvar_wait_tmo()` on the specified condition variable are woken up.

The calling thread must hold the mutex associated with the condvar.

Parameters

`condvar` Address of the condition variable to signal

Return value

Zero on success, `-1` on failure.

2.2.3 Thread functions

2.2.3.1 acu_sccp_thread_create

```
int acu_sccp_thread_create(acu_sccp_thrd_id_t *id,
     ACU_SCCP_THREAD_FN((*fn), arg), void *fn_arg);
```

Purpose

This function creates a new thread to run the caller supplied function. The thread function can be defined portably as:

```
static ACU_SCCP_THREAD_FN(fn, arg)
{
    ...
    acu_sccp_thread_exit(1, 0);
    return 0;
}
```

Parameters

`id`	Data area to hold thread identification
`fn`	Function to call in the new thread
`fn_arg`	Argument to pass `fn`

Return value

Zero on success, `-1` on failure.

2.2.3.2 acu_sccp_thread_exit

```
void acu_sccp_thread_exit(int detach, unsigned int rval);
```

Purpose

This function causes the current thread to terminate itself.

Parameters

`detach`	If non-zero the thread will exit and free all associated system resources. If zero `acu_sccp_thread_join()` must be called to free the resources.
`rval`	Return value to pass to the caller of `acu_sccp_thread_join()`.

If a thread function returns (instead of calling `acu_sccp_thread_exit`) then it is not detached and `acu_sccp_thread_join` must be called to free the operating system resources.

2.2.3.3 acu_sccp_thread_join

```
void acu_sccp_thread_join(acu_sccp_thrd_id_t *id, unsigned int *rval);
```

Purpose

This function waits for the specified thread to terminate, saves the thread return code, and frees all the system resources associated with the thread.

Parameters

`id`	Thread identification data for the thread (from `acu_sccp_thread_create`)
`rval`	Pointer to a where the thread return code will be written

2.2.3.4 acu_sccp_thread_id

```
int acu_sccp_thread_id(void);
```

Purpose

This function returns an operating system supplied identifier for the current thread.

Return value

The operating system identifier for the current thread. This has the same value as the `att_thrd_id` field of the `acu_sccp_thrd_id_t` structure.

Appendix A: Building SCCP applications

A.1 Linux

The SCCP API header file includes all the necessary system headers.
Compile with `-D_REENTRANT`.
Link with `-lpthread -Wl,--enable-new-dtags`.
Link with `-Wl,-rpath,$ACULAB_ROOT/lib` (or `lib64`) to get the location of the libraries embedded in the application image (`$ACULAB_ROOT` here must be expanded at program link time).

A.2 Windows

To obtain the correct definitions the symbol `_WINSOCKAPI_` must be defined before `windows.h` is included. One way to achieve this is to specify `-D_WINSOCKAPI_=` on the compiler command line.

Since the SCCP library itself creates threads, the program must be compiled as a threaded program. ie: build with `-MT` (or `-MTd`) not `-ML`.

The application must also include `windows.h` and `winsock2.h` before the SCCP API header file.

Appendix B: sccp_api.h

B.1 Error Codes

The error codes returned by the SCCP library functions are small negative integers. API functions may return any of the error codes below, not just those identified in the section for the API function itself.

In most cases more detailed information is written to the logfile.

`ACU_SCCP_ERROR_SUCCESS`
: API call succeeded (guaranteed to be zero).

`ACU_SCCP_ERROR_TIMEDOUT`
: Request timed out.

`ACU_SCCP_ERROR_NO_MESSAGE`
: There are no messages on the specified queue.

`ACU_SCCP_ERROR_NO_INFORMATION_AVAILABLE`
: Requested information isn't available.

`ACU_SCCP_ERROR_MALLOC_FAIL`
: The library failed to allocate memory for a data item.
Check the application for memory leaks.

`ACU_SCCP_ERROR_NO_THREADS`
: The SCCP library failed to create a thread.
Check that the application isn't using more threads than the operating system can support.

`ACU_SCCP_ERROR_BAD_CONNECTION`
: The `acu_sccp_con_t` parameter doesn't reference a valid connection data area.

`ACU_SCCP_ERROR_BAD_SSAP`
: The `acu_sccp_ssap_t` parameter doesn't reference a valid ssap data area.

`ACU_SCCP_ERROR_BAD_MESSAGE`
: The `acu_sccp_msg_t` parameter doesn't reference a valid message data area.

`ACU_SCCP_ERROR_BAD_EVENT`
: The `acu_sccp_event_t` parameter doesn't reference a valid event data area.

Note **The above four errors are likely to be caused by the application using a stale pointer.**

`ACU_SCCP_ERROR_BAD_OS_EVENT`
: An internal function tried to use an invalid operating system event or file descriptor.

`ACU_SCCP_ERROR_WRONG_MSG_TYPE`
: The message buffer isn't the correct type for the called function.

`ACU_SCCP_ERROR_ALREADY_CONNECTED`
: The ssap is already connected to (or is trying to connect to) the driver.

`ACU_SCCP_ERROR_NO_LOCAL_SSN`
: No local SCCP sub-system number has been set.

`ACU_SCCP_ERROR_NO_LOCAL_POINTCODE`
: No local MTP3 pointcode has been set.

`ACU_SCCP_ERROR_UNKNOWN_CONFIG_PARAM`
: Configuration parameter not known.

`ACU_SCCP_ERROR_INVALID_CONFIG_VALUE`
: Invalid, or out of range, configuration parameter value.

`ACU_SCCP_ERROR_CANNOT_OPEN_CONFIG_FILE`
: The configuration file cannot be opened.

`ACU_SCCP_ERROR_BAD_CONNECTION_STATE`
: The connection isn't in the correct state for the request.

`ACU_SCCP_ERROR_CONNECTION_OUTSTATE`
: Connection (or timer) is not in the correct state.

`ACU_SCCP_ERROR_CONNECTION_IDLE`
: Connection (or timer) is idle.

Note **Do not confuse the above error codes with call control error codes that have the same numeric values.**

B.2 SCCP addresses

The `acu_sccp_addr_t` structure has the following fields:

`sa_flags`	bitwise 'or' of the following:
`ACU_SCCP_SA_FLAGS_ROUTE_SSN`	route on SSN (even if global title present)
`ACU_SCCP_SA_FLAGS_RAW_GT`	raw global title (unknown `sa_gti`)
`sa_valid`	indicates which address elements contain valid data, bitwise 'or of:
`ACU_SCCP_SA_VALID_GTI`	
`ACU_SCCP_SA_VALID_SSN`	
`ACU_SCCP_SA_VALID_PC`	
`ACU_SCCP_SA_VALID_RL_PC`	
`ACU_SCCP_SA_VALID_TT`	
`ACU_SCCP_SA_VALID_NP`	
`ACU_SCCP_SA_VALID_ES`	
`ACU_SCCP_SA_VALID_NAI`	
`sa_gti`	global title indicator (4 bits)
`sa_ssn`	subsystem number
`sa_pc`	SS7 signalling pointcode for/from SCCP address buffer
`sa_rl_pc`	SS7 signalling pointcode from MTP3 routing label
`sa_tt`	translation type (8 bits)
`sa_np`	numbering plan (4 bits)
`sa_es`	encoding scheme (4 bits)
`sa_nai`	nature of address indicator (7 bits)
`sa_gt.sag_num`	number of digits in `sa_gt.sag_digits`
`sa_gt.sag_digits[]`	global title address information, two digits per byte

The global title indicator placed in outbound messages depends on which of the `ss_tt`, `sa_np`, `sa_es` and `sa_nai` fields are marked as valid, not on the value of `sa_gti`.

The `sa_gt.sag_num` field contains the number of digits (not bytes) in the global title. The application need not care about the odd/even field of the encoded global title.

The `sa_rl_pc` field contains the pointcode from the MTP3 routing label of received messages, it has no effect on outward messages.

When routing using global titles, if the sa_pc field is set then the SCCP driver will not perform global title translation and will send the message to that point code, if the sa_pc is not set then global title translation is performed.

The SCCP protocol constrains the valid combinations of TT, NP, ES and NAI. NP and ES must always be specified together. NAI is not valid for ANSI SCCP, and, for ITU and China SCCP, must be specified on its own or with TT, NP and ES.

The first digit of the global title is encoded in the least significant 4 bits of `sa_gt.sag_digits[0]` and the second digit in the most significant 4 bits. This matches the protocol encoding, but is reversed from a normal hexdump of the address buffer.

Appendix C: System limits

The following limits are inherent in the design of the SCCP product, however other constraints (e.g. lack of memory) may apply first:

Dimension	Limit	Notes
Connections to an SCCP system	4094	Also constrained by available server-side resources.
Connections per library ssap	3840	Cost a few kb per connection.
Timers per connection	250	No additional cost per timer.