SCSCP C Library

Reference manual
version 0.3.2
16 January 2009

M. Gastineau
gastineau@imcce.fr

This manual documents how to install and use the SCSCP C Library, version 0.3.2.

M. Gastineau, Astronomie et Systèmes Dynamiques, IMCCE, CNRS, Observatoire de Paris
gastineau@imcce.fr

Table of Contents

SCSCP C Library

1 SCSCP C Library Copying conditions

gastineau@imcce.fr

2 Introduction to SCSCP C Library

This library is an implementation of the Symbolic Computation Software Composibility Protocol (SCSCP). The current implementation is based on the specification version 1.2 (see see Chapter 9 [References], page 33).

This library provides API to develop client applications to access Computer Algebra Systems which support that protocol. So these client applications will be refered as '`SCSCP client`' in this documentation.

Computer Algebra Systems could use the API to provide services to other applications using this protocol. So these Computer Algebra Systems will be refered as '`SCSCP server`' in this documentation.

3 Installing SCSCP C Library

3.1 Installation on a Unix-like system (Linux, Mac OS X, BSD, cygwin, ...)

To build SCSCP C Library, you first have to install Libxml2 version 2.6 or later (see http://xmlsoft.org/) on your computer. You need a C and C++ compiler, such as gcc and g++. And you need a standard Unix '`make`' program, plus some other standard Unix utility programs.

Here are the steps needed to install the library on Unix systems:

1. '`tar xzf scscp-0.3.2.tar.gz`'
2. '`cd scscp-0.3.2`'
3. '`./configure`'

   Running `configure` might take a while. While running, it prints some messages telling which features it is checking for.

   `configure` recognizes the following options to control how it operates.

   '`--help`'
   '`-h`' Print a summary of all of the options to `configure`, and exit.

   '`--prefix=`*dir*'
   Use *dir* as the installation prefix. See the command `make install` for the installation names.
4. '`make`'

   This compiles SCSCP C Library in the working directory.
5. '`make check`'

   This will make sure SCSCP C Library was built correctly.

   If you get error messages, please report them to gastineau@imcce.fr (See Chapter 4 [Reporting bugs], page 6, for information on what to include in useful bug reports).
6. '`make install`'

   This will copy the file '`scscp.h`' and '`scscptypes.h`' to the directory '`/usr/local/include`', the file '`libscscp.a`' to the directory '`/usr/local/lib`', and the file '`scscp.info`' to the directory '`/usr/local/share/info`' (or if you passed the '`--prefix`' option to '`configure`', using the prefix directory given as argument to '`--prefix`' instead of '`/usr/local`'). Note: you need write permissions on these directories.

3.1.1 Other 'make' Targets

There are some other useful make targets:

- '`scscp.info`' or '`info`'

  Create an info version of the manual, in '`scscp.info`'.
- '`scscp.pdf`' or '`pdf`'

  Create a PDF version of the manual, in '`scscp.pdf`'.

- '`scscp.dvi`' or '`dvi`'

  Create a DVI version of the manual, in '`scscp.dvi`'.
- '`scscp.ps`' or '`ps`'

  Create a Postscript version of the manual, in '`scscp.ps`'.
- '`scscp.html`' or '`html`'

  Create an HTML version of the manual, in several pages in the directory '`scscp.html`'; if you want only one output HTML file, then type '`makeinfo --html --no-split scscp.texi`' instead.
- '`clean`'

  Delete all object files and archive files, but not the configuration files.
- '`distclean`'

  Delete all files not included in the distribution.
- '`uninstall`'

  Delete all files copied by '`make install`'.

3.2 Installation on Windows system

To build SCSCP C Library, you first have to install Libxml2 version 2.6 or later (see http://xmlsoft.org/) on your computer. You need a C++ compiler and a Windows SDK. It has been successfully compiled with the Windows Server 2003 R2 Platform SDK, the Windows SDK of Vista, and the Windows Server 2008 Platform SDK.

Here are the steps needed to install the library on Windows systems:

1. Expand the file '`scscp-0.3.2.tar.gz`'
2. Execute the command '`cmd.exe`' from the menu '`Start`' / '`Execute...`'

   This will open a console window
3. '`cd` '*dir*'`\scscp-0.3.2`'

   Go to the directory *dir* where SCSCP C Library has been expanded.
4. '`nmake /f makefile.vc XMLDIR=`*dir*'

   This compiles SCSCP C Library in the working directory. Use *dir* as the installation directory of the libxml2 library.
5. '`nmake /f makefile.vc XMLDIR=`*dir* `check`'

   This will make sure SCSCP C Library was built correctly.

   If you get error messages, please report them to gastineau@imcce.fr (See Chapter 4 [Reporting bugs], page 6, for information on what to include in useful bug reports).
6. '`nmake /f makefile.vc install DESTDIR=`*dir*'

   This will copy the file '`scscp.h`' and '`scscptypes.h`' to the directory *dir*'`\include`', the file '`libscscp.a`' to the directory *dir*'`\lib`', the file '`scscp.info`' and '`scscp.pdf`' to the directory *dir*'`\doc`'. Note: you need write permissions on these directories.

4 Reporting bugs

If you think you have found a bug in the SCSCP C Library, first have a look on the SCSCP C Library web page http://www.imcce.fr/Equipes/ASD/trip/scscp/ , in which case you may find there a workaround for it. Otherwise, please investigate and report it. We have made this library available to you, and it is not to ask too much from you, to ask you to report the bugs that you find.

There are a few things you should think about when you put your bug report together. You have to send us a test case that makes it possible for us to reproduce the bug. Include instructions on how to run the test case.

You also have to explain what is wrong; if you get a crash, or if the results printed are incorrect and in that case, in what way.

Please include compiler version information in your bug report. This can be extracted using '`cc -V`' on some machines, or, if you're using gcc, '`gcc -v`'. Also, include the output from '`uname -a`' and the SCSCP version.

Send your bug report to: gastineau@imcce.fr. If you think something in this manual is unclear, or downright incorrect, or if the language needs to be improved, please send a note to the same address.

5 SCSCP C Library Basics

5.1 Headers and Libraries

All declarations needed to use SCSCP C Library are collected in the include file '`scscp.h`'. It is designed to work with both C and C++ compilers.

You should include that file in any program using the SCSCP C Library :

```
#include <scscp.h>
```

Note however that the SCSCP constants use `NULL`, the header file '`stdio.h`' must be included before.

```
#include <stdio.h>
#include <scscp.h>
```

5.1.1 Compilation on a Unix-like system

All programs using SCSCP must link against the '`libscscp`' library and the Libxml2 library. On Unix-like system this can be done with '`-lscscp ‘xml2-config --libs‘`', for example

```
gcc myprogram.c -o myprogram -lscscp ‘xml2-config --libs‘
```

If SCSCP C Library has been installed to a non-standard location then it may be necessary to use '`-I`' and '`-L`' compiler options to point to the right directories, and some sort of run-time path for a shared library.

5.1.2 Compilation on a Windows system

All programs using SCSCP must link against the '`libscscp`' library and the Libxml2 library. On Windows system this can be done with '`scscp.lib libxml2_a.lib iconv.lib wsock32.lib ws2_32.lib`', for example

```
cl.exe /out:myprogram myprogram.c scscp.lib libxml2_a.lib iconv.lib \
                                  wsock32.lib ws2_32.lib
```

If SCSCP C Library has been installed to a non-standard location then it may be necessary to use '`/I`' and '`/LIBPATH:`' compiler options to point to the right directories.

5.2 Thread safe

SCSCP C Library is reentrant and thread-safe wit some exceptions:

1. It's safe for two threads to read from the same SCSCP variable simultaneously, but it's not safe for one to read while the another might be writing, nor for two threads to write simultaneously.
2. If the standard I/O functions such as `send` are not reentrant then the SCSCP I/O functions using them will not be reentrant either.

6 SCSCP C Library Interface

6.1 Constants

SCSCP_VERSION_MAJOR

This integer constant defines the major revision of this library. It can be used to distinguish different releases of this library.

SCSCP_VERSION_MINOR

This integer constant defines the minor revision of this library. It can be used to distinguish different releases of this library.

SCSCP_VERSION_PATCH

This integer constant defines the patch level revision of this library. It can be used to distinguish different releases of this library.

```
#if   (SCSCP_VERSION_MAJOR>=2)
  ||  (SCSCP_VERSION_MAJOR>=1 && SCSCP_VERSION_MINOR>=1)
...
#endif
```

SCSCP_PROTOCOL_VERSION_1_1

This string defines the version string for the SCSCP specification version 1.1 .

SCSCP_PROTOCOL_VERSION_1_2

This string defines the version string for the SCSCP specification version 1.2 .

SCSCP_PROTOCOL_DEFAULTPORT

This integer is the default value on which port should listen the SCSCP server. The value 26133 for this port has been assigned to SCSCP by the Internet Assigned Numbers Authority (IANA) in November 2007, see http://www.iana.org/assignments/port-numbers.

6.2 Types

6.2.1 SCSCP_socketserver

`SCSCP_socketserver` [Data type]

This type contains all information of the SCSCP server.

Before using any object of this type, the function `SCSCP_ss_init` must be called.

6.2.2 SCSCP_socketclient

`SCSCP_socketclient` [Data type]

This type contains all information of the SCSCP client about the connection through a socket to a SCSCP server.

Before using any object of this type, the function `SCSCP_sc_init` must be called.

6.2.3 SCSCP_incomingclient

`SCSCP_incomingclient` [Data type]

This type contains all information of an incoming connection accepted by a server.

6.2.4 SCSCP_status

`SCSCP_status` [Data type]

This type contains all information about errors. Each function of the library updates an object of this type if an error occurs during the processing.

The value `SCSCP_STATUS_IGNORE` could be used in order to ignore the returned value by these functions.

The possible values are

'`SCSCP_STATUS_OK`'
: No error occurs.

'`SCSCP_STATUS_CALLOPTIONSOBJECTNULL`'
: The object call options passed to the function is NULL.

'`SCSCP_STATUS_CLIENTOBJECTNULL`'
: The object client passed to the function is NULL.

'`SCSCP_STATUS_ERRNO`'
: The variable errno is set to a system error. The value of errno specifies the error.

'`SCSCP_STATUS_NOMEM`'
: Not enough memory

'`SCSCP_STATUS_RECVCANCEL`'
: The interrupt message "<? scscp cancel ?>" was received.

'`SCSCP_STATUS_RECVQUIT`'
: The interrupt message "<? scscp quit ?>" was received.

'`SCSCP_STATUS_RETURNOPTIONSOBJECTNULL`'
: The object return options passed to the function is NULL.

'`SCSCP_STATUS_SERVEROBJECTNULL`'
: The object server passed to the function is NULL.

'`SCSCP_STATUS_STREAMOBJECTNULL`'
: The object stream passed to the function is NULL.

'`SCSCP_STATUS_VERSIONNEGOTIATIONFAILED`'
: The version negotiation fails.

'`SCSCP_STATUS_USAGEUNKNOWNDEBUGLEVEL`'
: The debug level isn't available in the procedure call message.

'`SCSCP_STATUS_USAGEUNKNOWNMEM`'
: The memory usage isn't available in the procedure return message.

'SCSCP_STATUS_USAGEUNKNOWNMINMEMORY'
: The minimal memory isn't available in the procedure call message.

'SCSCP_STATUS_USAGEUNKNOWNMAXMEMORY'
: The maximal memory isn't available in the procedure call message.

'SCSCP_STATUS_USAGEUNKNOWNRETURNTYPE'
: The return type isn't available in the procedure call message

'SCSCP_STATUS_USAGEUNKNOWNRUNTIME'
: The runtime usage isn't available in the procedure return message.

'SCSCP_STATUS_USAGEUNKNOWNRUNTIMELIMIT'
: The runtime limit usage isn't available in the procedure call message.

6.2.5 SCSCP_calloptions

SCSCP_calloptions [Data type]

This type contains all information about the options for a procedure call. The attribute of the procedure call could be set using the functions SCSCP_co_set_xxx. The attribute of the procedure call could be get using the functions SCSCP_co_get_xxx.

The value SCSCP_CALLOPTIONS_DEFAULT could be used in order to use the default procedure call options.

The value SCSCP_RETURNOPTIONS_IGNORE could be used in order to ignore the returned value by the function SCSCP_ss_callrecvheader, SCSCP_ss_callrecvstr.

Before using any object of this type, the function SCSCP_co_init must be called.

6.2.6 SCSCP_returnoptions

SCSCP_returnoptions [Data type]

This type contains all information about the options for a procedure return. The attribute of the procedure return could be set using the functions SCSCP_ro_set_xxx. The attribute of the procedure return could be get using the functions SCSCP_ro_get_xxx.

The value SCSCP_RETURNOPTIONS_IGNORE could be used in order to ignore the returned value by the function SCSCP_sc_callrecvheader, SCSCP_sc_callrecvstr.

The value SCSCP_RETURNOPTIONS_DEFAULT could be used in order to send default return value to the client. This value could be used for the functions SCSCP_ss_sendcompletedstr, SCSCP_ss_sendterminatedstr, ...

Before using any object of this type, the function SCSCP_ro_init must be called.

6.2.7 SCSCP_msgtype

SCSCP_msgtype [Data type]

This type defines the type of sent messages between the client and the server.

The available values are

'SCSCP_msgtype_ProcedureTerminated'
: The message is a "Procedure terminated". It is defined by the symbol procedure_terminated of the OpenMath Content Dictionary scscp1.

'SCSCP_msgtype_ProcedureCompleted'
: The message is a "Procedure completed". It is defined by the symbol `procedure_completed` of the OpenMath Content Dictionary `scscp1`.

'SCSCP_msgtype_ProcedureCall'
: The message is a "Procedure call". It is defined by the symbol `procedure_call` of the OpenMath Content Dictionary `scscp1`.

6.2.8 SCSCP_error

`SCSCP_error` [Data type]

This type defines the error type of a "procedure terminated". The available values are

'SCSCP_error_memory'
: The message is a memory error of a "procedure terminated" message. It is defined by the symbol `error_memory` of the OpenMath Content Dictionary `scscp1`.

'SCSCP_error_runtime'
: The message is a runtime error of a "procedure terminated" message. It is defined by the symbol `error_runtime` of the OpenMath Content Dictionary `scscp1`.

'SCSCP_error_system_specific'
: The message is a runtime error of a "procedure terminated" message. It is defined by the symbol `error_system_specific` of the OpenMath Content Dictionary `scscp1`.

6.2.9 SCSCP_xmlnodeptr

`SCSCP_xmlnodeptr` [Data type]

This type defines a pointer to a node of a XML tree.

6.2.10 SCSCP_xmlattrptr

`SCSCP_xmlattrptr` [Data type]

This type defines a pointer to an attribute of a node of type SCSCP_xmlnodeptr (noode of a XML tree).

6.2.11 SCSCP_io

`SCSCP_io` [Data type]

This type defines a pointer to a low-level Input/output stream.

6.3 SCSCP server functions

The following functions manage all operations on the `SCSCP_socketserver` and `SCSCP_incomingclient` objects.

6.3.1 SCSCP_ss_init

`int SCSCP_ss_init` [Library Function]
(SCSCP_socketserver* *server*, SCSCP_status* *status* , const char* *servicename*, const char* *serviceversion*, const char* *serviceid*, ...)

It initializes the internal structure of the object *server*. The variadic arguments should be of the type `const char*` and the last argument must be `NULL`. The variadic parameters define the allowed version of SCSCP protocol that could be negotiated with the SCSCP server.

The arguments *servicename*, *serviceversion* and *serviceid* are used as the value of the attribute `service_name`, `service_version` and `service_id` of the *Connection Initiation Message*.

The constants `SCSCP_PROTOCOL_VERSION_x_x` could be used for the variadic parameters.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_ss_init` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

The following example shows how to initialize the server supporting the scscp versions "1.2", "1.1" and "1.001" .

```
SCSCP_status status;
SCSCP_server server;
int res;

res = SCSCP_ss_init(&server, &status, "MYCAS","1","myid",
                        SCSCP_PROTOCOL_VERSION_1_2,
                        SCSCP_PROTOCOL_VERSION_1_1, "1.001", NULL);
```

6.3.2 SCSCP_ss_clear

`int SCSCP_ss_clear` (*SCSCP_socketserver** **`server`**, [Library Function]
*SCSCP_status** **`status`**)

It clears the internal structure of the object *server* and frees allocated memory for this object by the function `SCSCP_ss_init`.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_ss_clear` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.3.3 SCSCP_ss_listen

`int SCSCP_ss_listen` (*SCSCP_socketserver** **`server`**, *int* [Library Function]
`port`, *int* **`firstavailable`**, *SCSCP_status** **`status`**)

server creates an internal queue for the incoming connections and starts to listen on the *port* of "localhost". If the *port* isn't available and *firstavailable* = 0, it fails. If the *port* isn't available and *firstavailable* = 1, it retries with the next port until it finds an available port.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_ss_listen` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.3.4 SCSCP_ss_close

`int SCSCP_ss_close` (*SCSCP_socketserver** **`server`**, *SCSCP_status** **`status`**) [Library Function]

server terminates to listen for the incoming connections.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_ss_close` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.3.5 SCSCP_ss_acceptclient

`int SCSCP_ss_acceptclient` (*SCSCP_socketserver** **`server`**, *SCSCP_incomingclient** **`incomingclient`**, *SCSCP_status** **`status`**) [Library Function]

server extracts the first connection request on the queue of pending connections. If no pending connections are present on the queue, it blocks the caller until a connection is present.

After the *Connection Initiation*, the server returns, in the argument *incomingclient*, an object to manage future exchanged messages.

After the transactions, *incomingclient* must be closed and cleared with the function `SCSCP_ss_closeincoming`.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_ss_acceptclient` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

The following example shows how to implement the main loop of the SCSCP server.

```
SCSCP_status status;
SCSCP_incomingclient incomingclient;
SCSCP_server server;

/*initialization of the server */
SCSCP_ss_init(&server, &status, "MYCAS","1","myid",
                    SCSCP_PROTOCOL_VERSION_1_2,
                    SCSCP_PROTOCOL_VERSION_1_1, "1.001", NULL);

SCSCP_ss_listen(&server, SCSCP_PROTOCOL_DEFAULTPORT, &status);

while (SCSCP_ss_acceptclient(&server, &incomingclient, &status))
{
   ... process incoming message ...

   SCSCP_ss_closeincoming(&incomingclient, &status);
}


/* destroy the server */
SCSCP_ss_close(& server, &status);
SCSCP_ss_clear(&server, &status);
```

6.3.6 SCSCP_ss_closeincoming

`int SCSCP_ss_closeincoming` (*SCSCP_incomingclient** **`incomingclient`**, *SCSCP_status** **`status`**) [Library Function]

It closes the connection with the client and clears the object *incomingclient*.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_ss_closeincoming` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.3.7 SCSCP_ss_callrecvstr

`int SCSCP_ss_callrecvstr` (*SCSCP_incomingclient** **`incomingclient`**, *SCSCP_calloptions** **`options`**, *SCSCP_msgtype** **`msgtype`**, *char*** **`openmathbuffer`**, *SCSCP_status** **`status`**) [Library Function]

It reads the attribute, the type and the content of the message sent by the client *incomingclient*.

The value `SCSCP_CALLOPTIONS_IGNORE` could be used for the parameter *options* in order to ignore the returned value. The call options could be get using the functions `SCSCP_co_get_xxx`. If *options* isn't `SCSCP_CALLOPTIONS_IGNORE`, it must be initialized before with the function `SCSCP_co_init`.

On exit, the argument *msgtype* must be `SCSCP_msgtype_ProcedureCall`. The client sends only "Procedure Call" message.

On exit, the argument *openmathbuffer* contains the content of the message sent by the client. This string must be freed by the system call `free`.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_ss_callrecvstr` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.3.8 SCSCP_ss_callrecvheader

`int SCSCP_ss_callrecvheader` (*SCSCP_incomingclient** ***incomingclient***, *SCSCP_calloptions** ***options***, *SCSCP_msgtype** ***msgtype***, *SCSCP_status** ***status***) [Library Function]

It reads the attribute and type of the message sent by the client *incomingclient*.

The value `SCSCP_CALLOPTIONS_IGNORE` could be used for the parameter *options* in order to ignore the returned value. The return options could be get using the functions `SCSCP_co_get_xxx`. If *options* isn't `SCSCP_CALLOPTIONS_IGNORE`, it must be initialized before with the function `SCSCP_co_init`.

On exit, the argument *msgtype* must be `SCSCP_msgtype_ProcedureCall`. The client sends only "Procedure Call" message.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_ss_callrecvheader` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.3.9 SCSCP_ss_sendterminatedstr

`int SCSCP_ss_sendterminatedstr` (*SCSCP_incomingclient** ***incomingclient***, *SCSCP_returnoptions** ***options***, *SCSCP_error* ***errortype***, *const char ** ***message***, *SCSCP_status** ***status***) [Library Function]

It sends a "procedure terminated" message to the SCSCP client with the *options*. *errortype* is the type error and *message* is the test messge that will be inserted in a OMSTR OpenMath object.

The string *openmathbuffer* must be a valid OpenMath command.

The value `SCSCP_RETURNOPTIONS_DEFAULT` could be used for the parameter *options* in order to use the default procedure return options.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_ss_sendterminatedstr` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.3.10 SCSCP_ss_sendcompletedstr

`int SCSCP_ss_sendcompletedstr` (*SCSCP_incomingclient** ***incomingclient***, *SCSCP_returnoptions** ***options***, *const char ** ***openmathbuffer***, *SCSCP_status** ***status***) [Library Function]

It sends a "procedure completed" message to the SCSCP client with the *options*. The string *openmathbuffer* is the argument of the "procedure completed".

The string *openmathbuffer* must be a valid OpenMath command.

The value `SCSCP_RETURNOPTIONS_DEFAULT` could be used for the parameter *options* in order to use the default procedure return options.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_ss_sendcompletedstr` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.3.11 SCSCP_ss_sendcompletedcallback

`int SCSCP_ss_sendcompletedcallback (` [Library Function]
*SCSCP_incomingclient** **`incomingclient`**, *SCSCP_returnoptions** **`options`**, *int* (**`callbackwriteargs`**)(*SCSCP_io* stream, void *param, SCSCP_status* status*), *void** **`param`**, *SCSCP_status** **`status`**)

It sends a "procedure terminated" message to the client with the *options*. The arguments of the "procedure terminated" message must be written by the callback function *callbackwriteargs*. The function *callbackwriteargs* must use the function `SCSCP_io_write` to write data which are sent to the SCSCP client.

The argument *param* is a pointer which is provided to the *callbackwriteargs* to exchange information. This pointer and its content isn't modified by `SCSCP_sc_callsendcallback`.

The value `SCSCP_RETURNOPTIONS_DEFAULT` could be used for the parameter *options* in order to use the default procedure return options. The procedure return options could be set using the functions `SCSCP_ro_set_xxx`.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_ss_sendcompletedcallback` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.3.12 SCSCP_ss_getxmlnode

`SCSCP_xmlnodeptr SCSCP_ss_getxmlnode (` [Library Function]
*SCSCP_incomingclient** **`incomingclient`**, *SCSCP_status** **`status`**)

The *incomingclient* returns a pointer to the current XML tree. This function could be used to start parsing the message sent by the client.

On exit, it returns `NULL` if an error occurs, otherwise the return value is a valid address. If `SCSCP_ss_getxmlnode` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.4 SCSCP client functions

The following functions manage all operations on the `SCSCP_socketclient` object.

6.4.1 SCSCP_sc_init

`int SCSCP_sc_init` [Library Function]
(SCSCP_socketclient* *client*, SCSCP_status* *status*, ...)

It initializes the internal structure of the object *client*. The variadic arguments should be of the type `const char*` and the last argument must be `NULL`. The variadic parameters define the allowed version of SCSCP protocol that could be negotiated with the SCSCP server.

During the negotiation with the server, the client will choose the first version in the variadic parameters that the server supports too. So the variadic parameters should start by from the highest level of the SCSCP version to the lowest version.

The constants `SCSCP_PROTOCOL_VERSION_x_x` could be used for the variadic parameters.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_sc_init` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

```
res = SCSCP_sc_init(&client, &status, SCSCP_PROTOCOL_VERSION_1_2,
              SCSCP_PROTOCOL_VERSION_1_1, "1.0beta", NULL);
```

6.4.2 SCSCP_sc_clear

`int SCSCP_sc_clear` (*SCSCP_socketclient** **`client`**, *SCSCP_status** **`status`**) [Library Function]

It clears the internal structure of the object *client* and frees allocated memory for this object by the function `SCSCP_sc_init`. If a connection was already opened, the function `SCSCP_sc_close` is called before clearing the object.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_sc_clear` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.4.3 SCSCP_sc_connect

`int SCSCP_sc_connect` [Library Function]
(SCSCP_socketclient* *client*, const char **machine*, int *port*, SCSCP_status* *status*)

It tries to connect to the SCSCP server which is running on the computer *machine* and is listening on the port *port*.

client must be initialized with the function `SCSCP_sc_init` before calling this function. If the connection achieves, *client* is updated on exit.

machine could be any string but it must resolved as an IP address. Its value could be `"localhost"` if the SCSCP server runs on the same computer.

In most of the case, the default value *SCSCP_PROTOCOL_DEFAULTPORT* should be used for the port number.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_sc_connect` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.4.4 SCSCP_sc_close

`int SCSCP_sc_close` (*SCSCP_socketclient** **`client`**, *SCSCP_status** **`status`**) [Library Function]

It closes a connection previously opened by a SCSCP client with the function `SCSCP_sc_connect`.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_sc_close` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.4.5 SCSCP_sc_callsendstr

`int SCSCP_sc_callsendstr` (*SCSCP_socketclient** **`client`**, *SCSCP_calloptions** **`options`**, *const char ** **`openmathbuffer`**, *SCSCP_status** **`status`**) [Library Function]

The *client* sends a "procedure call" message to the SCSCP server with the *options*. The string *openmathbuffer* is the argument of the procedure call. A connection must be previously opened with `SCSCP_sc_connect` before preforming this procedure call.

The string *openmathbuffer* must be a valid OpenMath command.

The value `SCSCP_CALLOPTIONS_DEFAULT` could be used for the parameter *options* in order to use the default procedure call options.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_sc_callsendstr` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.4.6 SCSCP_sc_callsendcallback

`int SCSCP_sc_callsendcallback` (*SCSCP_socketclient** **`client`**, *SCSCP_calloptions** **`options`**, *int* (**`callbackwriteargs`**)(*SCSCP_io* stream, void *param, SCSCP_status* status), *void** **`param`**, *SCSCP_status** **`status`**) [Library Function]

The *client* sends a "procedure call" message to the server with the *options*. The arguments of the procedure call must be written by the callback function *callbackwriteargs*. The function *callbackwriteargs* must use the function `SCSCP_io_write` to write data which are sent to the SCSCP server. A connection must be previously opened with `SCSCP_sc_connect` before preforming this procedure call.

The argument *param* is a pointer which is provided to the *callbackwriteargs* to exchange information. This pointer and its content isn't modified by `SCSCP_sc_callsendcallback`.

The value `SCSCP_CALLOPTIONS_DEFAULT` could be used for the parameter *options* in order to use the default procedure call options. The procedure call options could be set using the functions `SCSCP_co_set_xxx`.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_sc_callsendcallback` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.4.7 SCSCP_sc_callrecvheader

`int SCSCP_sc_callrecvheader` (*SCSCP_socketclient** **`client`**, *SCSCP_returnoptions** **`options`**, *SCSCP_msgtype** **`msgtype`**, *SCSCP_status** **`status`**) [Library Function]

The *client* reads the attribute and type of the message returned by the server in response of a procedure call.

The value `SCSCP_RETURNOPTIONS_IGNORE` could be used for the parameter *options* in order to ignore the returned value. The return options could be get using the functions `SCSCP_ro_get_xxx`. If *options* isn't `SCSCP_RETURNOPTIONS_IGNORE`, it must be initialized before with the function `SCSCP_ro_init`.

On exit, the argument *msgtype* contains the message type returned by the server.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_sc_callrecvheader` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.4.8 SCSCP_sc_callrecvstr

`int SCSCP_sc_callrecvstr` (*SCSCP_socketclient** **`client`**, *SCSCP_returnoptions** **`options`**, *SCSCP_msgtype** **`msgtype`**, *char*** **`openmathbuffer`**, *SCSCP_status** **`status`**) [Library Function]

The *client* reads the attribute, the type and the content of the message returned by the server in response of a procedure call.

On exit, the argument *msgtype* contains the message type returned by the server. On exit, the argument *openmathbuffer* contains the content of the message returned by the server. This string must be freed by the system call `free`.

The value `SCSCP_RETURNOPTIONS_IGNORE` could be used for the parameter *options* in order to ignore the returned value. The return options could be get using the functions `SCSCP_ro_get_xxx`. If *options* isn't `SCSCP_RETURNOPTIONS_IGNORE`, it must be initialized before with the function `SCSCP_ro_init`.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_sc_callrecvstr` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.4.9 SCSCP_sc_callrecvterminated

`int SCSCP_sc_callrecvterminated` (*SCSCP_socketclient** **`client`**, *SCSCP_error** **`errortype`**, *char*** **`messagebuffer`**, *SCSCP_status** **`status`**) [Library Function]

The *client* reads the type of the error and the content of the error message if the server replies with a "procedure terminated" message. This function must be called only if `SCSCP_sc_callrecvheader` returns *msgtype*=`SCSCP_msgtype_ProcedureTerminated`.

On exit, the argument *errortype* contains the error type returned by the server. On exit, the argument *messagebuffer* contains the content of the error returned by the server. This string must be freed by the system call `free`.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_sc_callrecvterminated` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.4.10 SCSCP_sc_callrecvcompleted

`int SCSCP_sc_callrecvcompleted ( SCSCP_socketclient* client, char** openmathbuffer, SCSCP_status* status )` [Library Function]

The *client* reads the content of the messages if the server replies with a "procedure completed" message. This function must be called only if `SCSCP_sc_callrecvheader` returns *msgtype*=`SCSCP_msgtype_ProcedureCompleted`.

On exit, the argument *openmathbuffer* contains the content of the message returned by the server. This string must be freed by the system call `free`.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_sc_callrecvcompleted` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.4.11 SCSCP_sc_getxmlnode

`SCSCP_xmlnodeptr SCSCP_sc_getxmlnode ( SCSCP_socketclient* client, SCSCP_status* status )` [Library Function]

The *client* returns a pointer to the current XML tree. This function could be used to start parsing the message sent by the server.

On exit, it returns `NULL` if an error occurs, otherwise the return value is a valid address. If `SCSCP_sc_getxmlnode` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.5 SCSCP I/O functions

6.5.1 SCSCP_io_write

`int SCSCP_io_write ( SCSCP_io* stream, const char *buffer, SCSCP_status* status )` [Library Function]

This function must only be called by the callback function *callbackwritearg* provided to the function `SCSCP_ss_sendcompletedcallback` or `SCSCP_sc_callsendcallback`. The *stream* writes the data *buffer* directly to the socket connected to the SCSCP server. The argument *buffer* can't be NULL.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_io_write` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.6 SCSCP status functions

`int SCSCP_status_is ( SCSCP_status* status )` [Library Function]

This function returns the integer value of the status object. The returned value are defined in See Section 6.2.4 [SCSCP_status], page 9. *status* mustn't be `SCSCP_STATUS_IGNORE`. This function could be defined as a macro in the header file '`scscp.h`'.

```
int res = SCSCP_co_get_maxmemory(&options, &memsize, status);
if (res)
{
  printf("maximum memory = %lld\n", (long long)memsize);
}
else if (SCSCP_status_is(status)==SCSCP_STATUS_USAGEUNKNOWNMAXMEMORY)
{
  printf("maximum memory not available\n");
}
```

6.7 SCSCP procedure call options functions

6.7.1 SCSCP_co_init

`int SCSCP_co_init` (*SCSCP_calloptions** **`options`**, [Library Function]
*SCSCP_status** **`status`**)

It initializes the internal structure of the object *options*.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_co_init` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.7.2 SCSCP_co_clear

`int SCSCP_co_clear` (*SCSCP_calloptions** **`options`**, [Library Function]
*SCSCP_status** **`status`**)

It clears the internal structure of the object *options* and frees allocated memory for this object by the function `SCSCP_co_init`.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_co_clear` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.7.3 SCSCP_co_set_callid

`int SCSCP_co_set_callid` (*SCSCP_calloptions** **`options`**, [Library Function]
*const char **`buffer`*, *SCSCP_status** **`status`**)

This function sets the call id of the options of the procedure call. This call id is defined as the symbol `call_ID` of the OpenMath Content Dictionary `scscp1`. The argument *buffer* can't be NULL and won't be duplicated by these function. So *buffer* mustn't be destroyed until the function `SCSCP_co_clear` is called on the object *options*.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_co_set_callid` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.7.4 SCSCP_co_get_callid

`int SCSCP_co_get_callid` (*SCSCP_calloptions** **`options`**, *const char ***`buffer`, *SCSCP_status** **`status`**) [Library Function]

This function returns, in the argument *buffer*, the call id of the options of the procedure call. This call id is defined as the symbol `call_ID` of the OpenMath Content Dictionary `scscp1`. The argument *buffer* can't be NULL. The returned string mustn't be modified.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_co_get_callid` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.7.5 SCSCP_co_set_runtimelimit

`int SCSCP_co_set_runtimelimit` (*SCSCP_calloptions** **`options`**, *size_t* **`time`**, *SCSCP_status** **`status`**) [Library Function]

This function sets the amount of time in milliseconds, with the value *time*, that the SCSCP server should spend on this call. This runtime limit is defined by the symbol `option_runtime` of the OpenMath Content Dictionary `scscp1`.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_co_set_runtimelimit` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.7.6 SCSCP_co_get_runtimelimit

`int SCSCP_co_get_runtimelimit` (*SCSCP_calloptions** **`options`**, *size_t** **`time`**, *SCSCP_status** **`status`**) [Library Function]

This function returns, in the argument *time*, the amount of time in milliseconds that the server should spend on this call. This amount of time is defined as the symbol `option_runtime` of the OpenMath Content Dictionary `scscp1`. If the amount of time isn't available (not supplied by the server), the function fails and *status* is set to `SCSCP_STATUS_USAGEUNKNOWNRUNTIMELIMIT`.

The argument *time* can't be NULL.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_co_get_runtimelimit` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.7.7 SCSCP_co_set_minmemory

`int SCSCP_co_set_minmemory` (*SCSCP_calloptions** **`options`**, *size_t* **`memsize`**, *SCSCP_status** **`status`**) [Library Function]

This function sets the minimum amount of memory in bytes, with the value *memsize*, that the server should use on this call. This memory limit is defined by the symbol `option_min_memory` of the OpenMath Content Dictionary `scscp1`.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_co_set_minmemory` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.7.8 SCSCP_co_get_minmemory

`int SCSCP_co_get_minmemory ( SCSCP_calloptions* options, size_t* memsize, SCSCP_status* status )` [Library Function]

This function returns, in the argument *memsize*, the minimum amount of memory in bytes that the server should use on this call. This amount of memory is defined as the symbol `option_min_memory` of the OpenMath Content Dictionary `scscp1`. If the amount of memory isn't available (not supplied by the server), the function fails and *status* is set to `SCSCP_STATUS_USAGEUNKNOWNMINMEMORY`.

The argument *memsize* can't be NULL.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_co_get_minmemory` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.7.9 SCSCP_co_set_maxmemory

`int SCSCP_co_set_maxmemory ( SCSCP_calloptions* options, size_t memsize, SCSCP_status* status )` [Library Function]

This function sets the maximum amount of memory in bytes, with the value *memsize*, that the SCSCP server should use on this call. This memory limit is defined by the symbol `option_max_memory` of the OpenMath Content Dictionary `scscp1`.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_co_set_maxmemory` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.7.10 SCSCP_co_get_maxmemory

`int SCSCP_co_get_maxmemory ( SCSCP_calloptions* options, size_t* memsize, SCSCP_status* status )` [Library Function]

This function returns, in the argument *memsize*, the maximum amount of memory in bytes that the server should use on this call. This amount of memory is defined as the symbol `option_max_memory` of the OpenMath Content Dictionary `scscp1`. If the amount of memory isn't available (not supplied by the server), the function fails and *status* is set to `SCSCP_STATUS_USAGEUNKNOWNMAXMEMORY`.

The argument *memsize* can't be NULL.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_co_get_maxmemory` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.7.11 SCSCP_co_set_debuglevel

`int SCSCP_co_set_debuglevel ( SCSCP_calloptions* options, int debuglevel, SCSCP_status* status )` [Library Function]

This function sets the debug level, with the value *debuglevel*, that the client is interested. This debug level is defined by the symbol `option_debuglevel` of the OpenMath Content Dictionary `scscp1`.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_co_set_debuglevel` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.7.12 SCSCP_co_get_debuglevel

`int SCSCP_co_get_debuglevel` (*SCSCP_calloptions** *options*, *int** *debuglevel*, *SCSCP_status** *status*) [Library Function]

This function returns, in the argument *debuglevel*, the debug level that the client is interested. This debug level is defined as the symbol `option_max_memory` of the OpenMath Content Dictionary `scscp1`. If the debug level isn't available (not supplied by the server), the function fails and *status* is set to `SCSCP_STATUS_USAGEUNKNOWNDEBUGLEVEL`.

The argument *debuglevel* can't be NULL.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_co_get_debuglevel` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.7.13 SCSCP_co_set_returntype

`int SCSCP_co_set_returntype` (*SCSCP_calloptions** *options*, *SCSCP_option_return* *returntype*, *SCSCP_status** *status*) [Library Function]

This function sets the return type, with the value *returntype*, of the procedure call that the server should send. The available value for *returntype* are

- `SCSCP_option_return_object`. The return value is an OpenMath object. It's defined by the symbol `option_return_object` of the OpenMath Content Dictionary `scscp1`.
- `SCSCP_option_return_cookie`. The return value is a cookie (a reference to an OpenMath object). It's defined by the symbol `option_return_cookie` of the OpenMath Content Dictionary `scscp1`.
- `SCSCP_option_return_nothing`. The procedure call returns no value. It's defined by the symbol `option_return_nothing` of the OpenMath Content Dictionary `scscp1`.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_co_set_returntype` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.7.14 SCSCP_co_get_returntype

`int SCSCP_co_get_returntype` (*SCSCP_calloptions** *options*, *SCSCP_option_return** *returntype*, *SCSCP_status** *status*) [Library Function]

This function returns, in the argument *returntype*, the return type of the "procedure call" message that the server should send. The possible value of *returntype* are described in the function `SCSCP_co_set_returntype`. If the return type isn't available (not supplied by the server), the function fails and *status* is set to `SCSCP_STATUS_USAGEUNKNOWNRETURNTYPE`.

The argument *returntype* can't be NULL.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_co_get_returntype` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.8 SCSCP procedure return options functions

6.8.1 SCSCP_ro_init

`int SCSCP_ro_init` (*SCSCP_returnoptions** **`options`**, [Library Function]
*SCSCP_status** **`status`**)

It initializes the internal structure of the object *options*.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_ro_init` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.8.2 SCSCP_ro_clear

`int SCSCP_ro_clear` (*SCSCP_returnoptions** **`options`**, [Library Function]
*SCSCP_status** **`status`**)

It clears the internal structure of the object *options* and frees allocated memory for this object by the function `SCSCP_ro_init`.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_ro_clear` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.8.3 SCSCP_ro_set_callid

`int SCSCP_ro_set_callid` (*SCSCP_returnoptions** **`options`**, [Library Function]
*const char **`buffer`*, *SCSCP_status** **`status`**)

This function sets the call id, with the value *buffer*, of the options of the procedure return. This call id is defined as the symbol `call_ID` of the OpenMath Content Dictionary `scscp1`. The argument *buffer* can't be NULL and won't be duplicated by this function. So *buffer* can't be destroyed until the function `SCSCP_ro_clear` is called on the object *options*.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_ro_set_callid` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.8.4 SCSCP_ro_get_callid

`int SCSCP_ro_get_callid` (*SCSCP_returnoptions** **`options`**, [Library Function]
*const char ***`buffer`*, *SCSCP_status** **`status`**)

This function returns, in the argument *buffer*, the call id of the options of the procedure return. This call id is defined as the symbol `call_ID` of the OpenMath Content Dictionary `scscp1`. The argument *buffer* mustn't be NULL. The returned string mustn't be modified.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_ro_get_callid` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.8.5 SCSCP_ro_set_runtime

`int SCSCP_ro_set_runtime` (*SCSCP_returnoptions** **`options`**, *size_t* **`time`**, *SCSCP_status** **`status`**) [Library Function]

This function sets the amount of time in milliseconds, with the value *time*, that the server spent on this call. This amount of time is defined as the symbol `info_runtime` of the OpenMath Content Dictionary `scscp1`.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_ro_set_runtime` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.8.6 SCSCP_ro_get_runtime

`int SCSCP_ro_get_runtime` (*SCSCP_returnoptions** **`options`**, *size_t** **`time`**, *SCSCP_status** **`status`**) [Library Function]

This function returns, in the argument *time*, the amount of time in milliseconds that the server spent on this call. This amount of time is defined as the symbol `info_runtime` of the OpenMath Content Dictionary `scscp1`. If the amount of time isn't available (not supplied by the server), the function fails and *status* is set to `SCSCP_STATUS_USAGEUNKNOWNRUNTIME`.

The argument *time* can't be NULL.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_ro_get_runtime` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.8.7 SCSCP_ro_set_memory

`int SCSCP_ro_set_memory` (*SCSCP_returnoptions** **`options`**, *size_t* **`memsize`**, *SCSCP_status** **`status`**) [Library Function]

This function sets the amount of memory in bytes, with the value *memsize*, that the server used on this call. This amount of memory is defined as the symbol `info_memory` of the OpenMath Content Dictionary `scscp1`.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_ro_set_memory` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.8.8 SCSCP_ro_get_memory

`int SCSCP_ro_get_memory` (*SCSCP_returnoptions** **`options`**, *size_t** **`memsize`**, *SCSCP_status** **`status`**) [Library Function]

This function returns, in the argument *memsize*, the amount of memory in bytes that the server used for this call. This amount of memory is defined as the symbol `info_memory` of the OpenMath Content Dictionary `scscp1`. If the amount of memory isn't available (not supplied by the server), the function fails and *status* is set to `SCSCP_STATUS_USAGEUNKNOWNMEM`.

The argument *memsize* can't be NULL.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value. If `SCSCP_ro_get_runtime` fails, the value of *status* is set to the corresponding error value. `SCSCP_STATUS_IGNORE` could be used for *status* in order to ignore the returned value.

6.9 XML parsing functions

The following functions are useful to parse incoming messages after reading the header of the message with the functions `SCSCP_sc_callrecvheader` or `SCSCP_ss_callrecvheader`. The following example prints each node and attribute of a XML tree.

```
void printelements(SCSCP_xmlnodeptr node, int tab)
{
   SCSCP_xmlattrptr attr;
   const char *name;
   const char *value;
   int j;

   while (node!=NULL)
   {
      for(j=0; j<tab; j++) putchar(' ');
      printf ("node : '%s'\n",SCSCP_xmlnode_getname(node));

      for (attr = SCSCP_xmlnode_getattr(node);
           attr!=NULL;
           attr = SCSCP_xmlattr_getnext(attr))
      {
         SCSCP_xmlattr_getvalue(attr, &name, &value);
         for(j=0; j<tab+1; j++) putchar(' ');
         printf ("attribute : '%s '  = '%s'\n",name, value);
      }
      printelements(SCSCP_xmlnode_getchild(node),tab+4);
      node = SCSCP_xmlnode_getnext(node);
   }
}
```

6.9.1 SCSCP_xmlnode_getnext

`SCSCP_xmlnodeptr SCSCP_xmlnode_getnext (` *SCSCP_xmlnodeptr* `curnode` `)` [Library Function]

This function returns a pointer to the next node. If *curnode* is the last element, this function returns `NULL`.

6.9.2 SCSCP_xmlnode_getname

`const char*` *SCSCP_xmlnode_getname* `(` *SCSCP_xmlnodeptr* `curnode` `)` [Library Function]

This function returns the name of the node *curnode*.

6.9.3 SCSCP_xmlnode_getchild

`SCSCP_xmlnodeptr SCSCP_xmlnode_getchild (` [Library Function]
SCSCP_xmlnodeptr **`curnode`**)

This function returns the first child of the node *curnode*. The function returns NULL if it has no child.

6.9.4 SCSCP_xmlnode_getcontent

`const char * SCSCP_xmlnode_getcontent (` [Library Function]
SCSCP_xmlnodeptr **`curnode`**)

This function returns as a string the content of the node *curnode*.

6.9.5 SCSCP_xmlnode_getattr

`SCSCP_xmlattrptr SCSCP_xmlnode_getattr (` [Library Function]
SCSCP_xmlnodeptr **`curnode`**)

This function returns the first attribute of the node *curnode*.

6.9.6 SCSCP_xmlattr_getnext

`SCSCP_xmlattrptr SCSCP_xmlattr_getnext (` [Library Function]
SCSCP_xmlattrptr **`attr`**)

This function returns a pointer to the next attribute. If *attr* is the last attribute, this function returns `NULL`.

6.9.7 SCSCP_xmlattr_getvalue

`int SCSCP_xmlattr_getvalue (` *SCSCP_xmlattrptr* **`attr`**, [Library Function]
*const char ** **`name`**, *const char ** **`value`**)

This function returns, in the argument *name*, a pointer to the name of the attribute *attr* and returns, in the argument *value*, a pointer to the value of the attribute *attr*.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value.

7 Design a SCSCP server

The file '`examples/decodeserver.c`' shows the server which decodes each node of the OpenMath expression received from the client. It sends an answer to the client depending on the call options.

A simple SCSCP server could be done with the following operations. This server supports the scscp versions "1.1", "1.2" and "1.5beta".

- Initialize the server

```
SCSCP_socketserver server;
SCSCP_status status;

SCSCP_ss_init(&server, &status, "MYCAS", "1.0", "myid",
              SCSCP_PROTOCOL_VERSION_1_2,
              SCSCP_PROTOCOL_VERSION_1_1,
              "1.5beta",
              NULL);
```

- Listen for incoming client

```
SCSCP_ss_listen(&client, SCSCP_PROTOCOL_DEFAULTPORT, 0, &status);
```

- Loop over new clients

```
SCSCP_incomingclient incomingclient;

while (SCSCP_ss_acceptclient(&server, &incomingclient, &status))
{
```

  - Receive the "procedure call" message : 2 solutions

```
SCSCP_calloptions calloptions;
SCSCP_returnoptions returnoptions;
SCSCP_msgtype msgtype;

SCSCP_co_init(&options, status);
SCSCP_ro_init(&returnopt, status);
```

    - solution 1 : read the header and decode each node of the openmath stream

```
SCSCP_ss_callrecvheader(&incomingclient, &calloptions,
                        &msgtype, &status);

SCSCP_xmlnodeptr ptrnode;
ptrnode = SCSCP_sc_getxmlnode(&incomingclient, &status);
```

    - solution 2 : read the header and store the content in a string buffer

```
char *openmathbuffer;
SCSCP_ss_callrecvstr(&incomingclient, &calloptions, &msgtype,
                     &openmathbuffer, &status);
```

  - Send the answer : procedure completed or terminated ?
    - Send a "procedure completed" message

```
const char *openmathanswer='<OM...>";
SCSCP_ss_sendcompletedstr(&incomingclient, &returnopt,
                          openmathanswer, &status);
```

- Send a "procedure terminated" message

```
const char *openmathanswer='<OM...>";
SCSCP_ss_sendterminatedstr(&incomingclient, &returnopt,
                           SCSCP_error_system_specific,
                           openmathanswer,
                           &status);
```

- clear the options object

```
SCSCP_co_clear(&options, status);
SCSCP_ro_clear(&returnopt, status);
```

- Close the connection

```
  SCSCP_ss_closeincoming(&incomingclient, &status);
}
```

- Stop to listen for incoming clients

```
SCSCP_ss_close(&server, &status);
```

- Clear the server

```
SCSCP_ss_clear(&server, &status);
```

8 Design a SCSCP client

The file '`examples/simplestclient.c`' shows the simplest client which stores the value 6177887 on the server and prints the answer of the server.

The file '`examples/decodeclient.c`' shows the client which decodes each node of the OpenMath expression received from the server.

A simple SCSCP client could be done with the following operations. This simple client will connect with the SCSCP server located on "localhost" and listening on port 26133.

- Initialize the client

```
SCSCP_socketclient client;
SCSCP_status status;

SCSCP_sc_init(&client, &status, SCSCP_PROTOCOL_VERSION_1_2, NULL);
```

- Open the Connection

```
SCSCP_sc_connect(&client, "localhost",
                 SCSCP_PROTOCOL_DEFAULTPORT, &status);
```

- Send a procedure call

```
SCSCP_sc_callsendstr(&client, &options,
    "<OMA><OMS cd=\"scscp2\" name=\"get_allowed_heads\" /></OMA>",
    &status);
```

- Receive the answer of the procedure call : 2 solutions
  - solution 1 : read the header and decode each node of the openmath stream

```
SCSCP_msgtype msgtype;
SCSCP_returnoptions options;
SCSCP_ro_init(&options, status);
SCSCP_sc_callrecvheader(client, &options, &msgtype,
                        &status);
if (msgtype==SCSCP_msgtype_ProcedureTerminated)
{
    char *messagebuffer;
    SCSCP_error errortype;
    SCSCP_sc_callrecvterminated(client, &errortype,
                                &messagebuffer, status);

}
```

  - solution 2 : read the header and store the content in a string buffer

```
SCSCP_msgtype msgtype;
SCSCP_returnoptions options;
char *buffer;
SCSCP_ro_init(&options, status);
SCSCP_sc_callrecvstr(client, &options, &msgtype,
                     &buffer, &status);
```

- Close the connection

```
SCSCP_sc_close(&client,&status);
```

- Clear the client

```
SCSCP_sc_clear(&client,&status);
```

9 References

- Symbolic Computation Software Composability Protocol (SCSCP) specification
  Version 1.2, 2008
  S.Freundt, P.Horn, A.Konovalov, S.Linton, D.Roozemond.
  http://www.symbolic-computation.org/scscp
- OpenMath content dictionary scscp1
  D. Roozemond
  http://www.win.tue.nl/SCIEnce/cds/scscp1.html
- OpenMath content dictionary scscp2
  D. Roozemond
  http://www.win.tue.nl/SCIEnce/cds/scscp2.html