OpenOSP

Interface Specification

10 April 2001

Document Version 1.3

Data Connection Manual MOM-101-0103

Notice

Data Connection Limited
100 Church Street
Enfield
Middlesex
EN2 6BQ
England

+44 20 8366 1177
http://www.dataconnection.com

Contents

1 Introduction

This document describes the external interfaces of the OpenOSP open source Open Settlement Protocol (OSP) server protocol stack jointly developed by Cisco Systems and Data Connection Limited (DCL).

Before starting to use this document, you should read the *OpenOSP Product Overview* for general information about OpenOSP.

The remainder of this document is structured as follows.

- Section 2 gives an overview of the OpenOSP interfaces and how they are used.
- Section 3 provides information that is common to the OSP APIs (the subset of interfaces that your settlement application uses to interface directly to the stack: Control API, UM API, AR API, and SA API).
- Section 4 describes the Control API.
- Section 5 describes the UM API.
- Section 6 describes the AR API.
- Section 7 describes the SA API.
- Section 8 describes the Security API.
- Section 9 describes the Resource API.
- Section 10 describes the SSL API.

1.1 Typographical conventions

In this document, the following typographical conventions are used.

- Bold text indicates an OSP primitive, for example: **AuthorizationRequest**.
- Square brackets indicate a reference, for example: [OSP].
- A line printer typeface indicates a C function or type definition, for example: `osp_init`.
- All C types that are defined by OpenOSP (including function pointer types) are shown in upper case; other C identifiers, such as function names and variables, are shown in lower case.

2 OpenOSP Interfaces Overview

The following diagram illustrates the core OpenOSP server stack together with the APIs and the sample implementations of the other components.

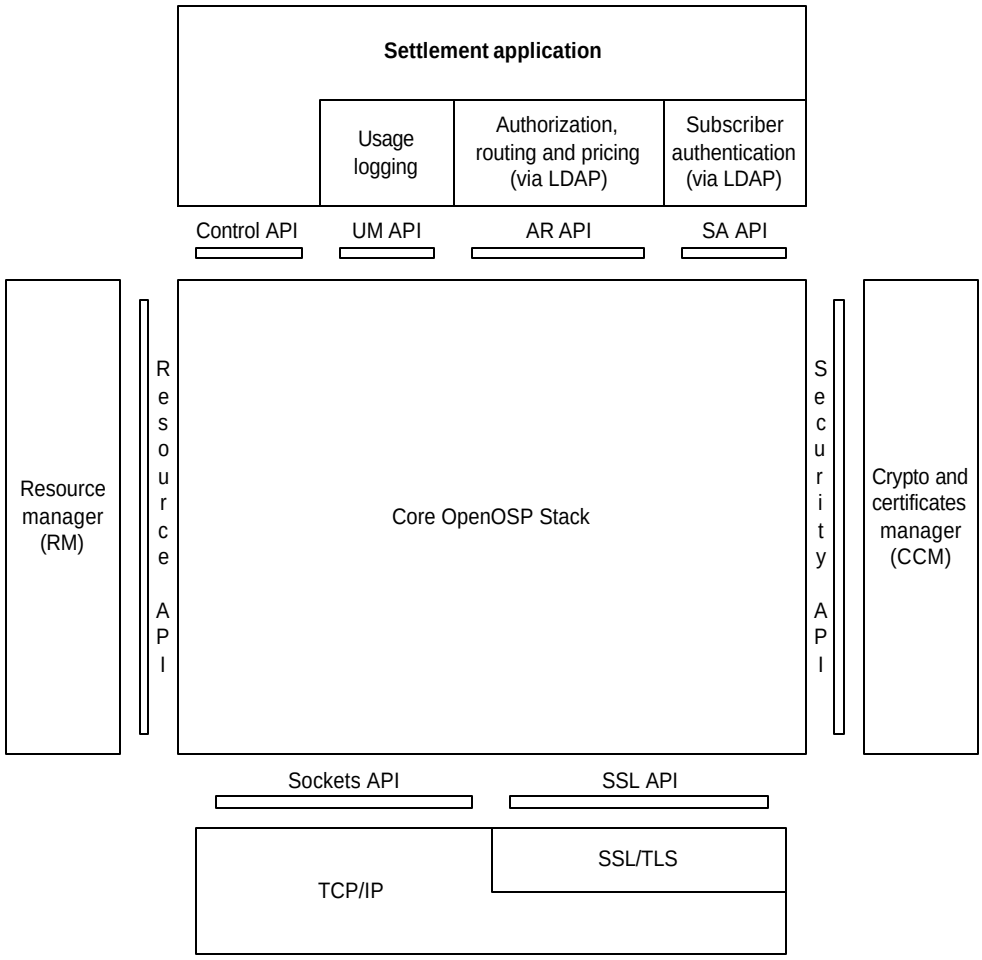

Figure 1 : OpenOSP Server APIs

2.1 OSP APIs

At the top of the diagram are the four **OSP APIs**, which provide access to the core OpenOSP server function from your settlement application.

- The **Control API** allows your application to start and stop OpenOSP, to verify incoming client connections, and to gather statistical information about its operation.

- The **Usage Metering (UM) API** allows your application to collect and store usage information about each call that has been made using the server (for example the call duration and calling / called numbers), for use in billing.

- The **Authorization and Routing (AR) API** allows OpenOSP to request authorization and routing information from your application for a call. This API is also used to pass pricing and capabilities information (gathered from OSP clients) to your application, for use in routing and authorization decisions.

- The **Subscriber Authentication (SA) API** allows OpenOSP to request your application to authenticate subscribers and check whether they are entitled to use the facilities that OSP provides (for example to authenticate users with mobile telephones who are calling from outside their usual network, and to verify whether they are permitted to do so).

Depending on your requirements, your settlement application may not need to use all of these APIs. For example, you may have multiple OSP servers each providing a subset of these functions, or you may have no requirement to support subscriber authentication.

2.1.1 OSP API Parameters And Sequences

The information passed on these APIs is based on the contents of the messages between OSP servers and clients, but OpenOSP breaks it out into individual fields and data structures for ease of use by the application.

The sequence in which different messages will be received by the server is not specified in [OSP]; it will depend upon the configuration of the settlement servers and upon the behaviour of the OSP clients in use.

2.2 Utility APIs

The remaining APIs are used by OpenOSP to access utility functions.

- The **Security API** allows OpenOSP to manage signatures and certificates.

- The **Resource API** allows OpenOSP to make requests for operating system resources (memory and threads), and to free these resources when no longer required. The supplied Resource Manager code maps these requests to basic operating system functions; the API allows you to change this mapping to manage your own resource allocation if required (for example to pre-allocate resources, or to allocate and free them in larger 'chunks' rather than in response to individual requests).

- The **Secure Sockets Layer (SSL) API** provides access to OSP clients through SSL / TLS. It is a subset of the API defined by the OpenSSL open source SSL / TLS implementation, allowing you to use this implementation without change. If you intend to use a different SSL / TLS implementation with OpenOSP, the implementation must provide this interface.

- The **Sockets API** is the standard BSD sockets interface as implemented on Sparc Solaris, and provides access to OSP clients through TCP. This interface is not described in detail in this document; refer to the Solaris documentation for details.

Note that OpenOSP also implements the Simple Certificate Enrollment Protocol (SCEP), but no API is required for its operation. Please refer to [SCEP] for details of the protocol.

3 Common Information for OSP APIs

This chapter provides API usage information that is common to some or all of the OSP APIs (Control API, UM API, AR API, and SA API).

3.1 Callback mechanism

The UM, AR, and SA APIs all use a callback mechanism to allow OpenOSP to make calls into your application. This mechanism operates as follows. (See Chapters 5–7 for details of the specific calls used at each API.)

1. At initialization, the application registers with OpenOSP for a specific type of information (such as usage metering information or authorization requests). This registration includes the address of a callback function provided by the application to handle this information type. Only one callback function can be registered for each type of information.

2. Each time OpenOSP needs to pass information to the application, it calls the callback function that the application has registered for the appropriate type of information.

   The callback function can use the supplied information as required, with the following restrictions:

   - It must not block while processing the information. Blocking on a callback will affect the internal operation of OpenOSP and will reduce overall throughput.

   - It must not modify any of the supplied parameters.

   - If it needs to use the information outside the callback function itself, it must make a copy of the data. The supplied parameters, including pointers to data, are valid only for the lifetime of the callback function, and the application will not be able to access them after the function has completed.

   - It must not call osp_term().

3. The application must then call a 'response' or 'confirm' function to inform OpenOSP of the results of its processing. It can do this synchronously from within the callback function itself, or it can make the call asynchronously from some other thread after the callback function has completed.

At termination, the application calls the registration function with a null address; this indicates that it will no longer handle the appropriate information type, and OpenOSP will no longer make calls to it to do so.

The Control API uses a similar callback mechanism to allow your application to verify the credentials of each client that connects to the server and to receive a copy of each signed OSP request that is received. However, the application must process these callbacks synchronously and there is therefore no 'response' or 'confirm' function – step 3 above does not apply.

3.2 Initialization sequence

To start using OpenOSP, the settlement application must do the following initialization:

1. Call osp_init() to initialize the OpenOSP stack and its associated libraries.
2. Use the register functions at each API to register for each callback that it will handle.
3. Call osp_listen() to instruct OpenOSP to start listening for OSP client connections.

3.3 Termination sequence

To stop using OpenOSP, the settlement application must do the following termination:

1. Use the register functions at each API to deregister for each callback that it is currently handling (by supplying a null pointer for each callback function).
2. Call osp_term() to stop the OpenOSP stack and its associated libraries.

3.4 Common data structures and types

This section describes data structures and types that are used in more than one function call on the OSP APIs. Data structures that are used in only one function call are included in the individual function call descriptions.

3.4.1 OSP_ADDRESS

The OSP_ADDRESS data structure holds address information for the source or destination of a call. This structure corresponds to the **SourceInfo** and **DestinationInfo** elements described in [OSP].

```
typedef struct osp_address
{
  struct osp_address *next;
  char               *type;
  char               *address;
} OSP_ADDRESS;
```

Members:

next — A pointer to the next item if this structure is in a list, or a null pointer if there are no further items.

type — A null-terminated string that specifies the type of addressing information provided. This may take one of the following values:

- e164 — An E.164 telephone number (digits only).
- h323 — An H.323 identifier.

- url — A uniform resource locator.
- email — An electronic mail address.
- transport — A transport address of the form *name*:*nn*, where *name* is the domain name, or the IP address enclosed in square brackets, and *nn* is an optional port number.
- international — An international party number.
- national — A national party number.
- network — A network-specific party number.
- subscriber — A subscriber party number.
- abbreviated — An abbreviated party number.
- e164prefix — The initial (most significant) digits of an E.164 number (digits only).
- iso7818 — *An ISO7818-1 ID card number (digits only).
- pin — *A personal identification number (digits only).
- epin — *A base64-encoded encrypted PIN.

**These values are valid only for source addresses.*

See the descriptions of **SourceInfo** and **DestinationInfo** in [OSP] for further descriptions of the above values.

address — A null-terminated string containing the addressing information for the source or destination.

3.4.2 OSP_CALL_ID

The OSP_CALL_ID data structure holds an H.323 call identifier, which uniquely identifies an individual call. This structure corresponds to the **CallId** element described in [OSP].

```
typedef struct osp_call_id
{
  struct osp_call_id *next;
  char               *encoding;
  char               *data;
} OSP_CALL_ID;
```

Members:

next	A pointer to the next item if this structure is in a list, or a null pointer if there are no further items.
encoding	A null-terminated string that identifies the encoding in which the call identifier is stored. One of the following: • cdata (XML CDATA format) • base64
data	A null-terminated string containing the call identifier.

3.4.3 OSP_CLIENT_ID

The OSP_CLIENT_ID data structure holds details of a client that is connected to the OSP server. It is specific to OpenOSP and does not correspond to any element defined in [OSP].

```
typedef struct osp_client_id
{
  unsigned long  ip_address;
  void          *ver_cookie;
} OSP_CLIENT_ID;
```

Members:

ip_address	The IP address of the client.
ver_cookie	A 'cookie' that the application supplies in the client verification callback routine. This parameter may take any value appropriate for your application. OpenOSP does not make use of the cookie, but simply stores it and supplies it on subsequent callbacks so that the application can identify the client. See section 4.5, POSP_CLIENT_VERIFY_CALLBACK, for details of the client verification callback.

3.4.4 OSP_CORRELATOR

The OSP_CORRELATOR data type is used for a correlator that OpenOSP uses to match requests and responses. It is specific to OpenOSP and does not correspond to any element defined in [OSP].

When it uses a callback function that requires a response from the application, OpenOSP supplies a unique correlator, which the application must return on the response. This ensures that the server can match the response to the original request even if there are multiple requests outstanding.

```
typedef void *OSP_CORRELATOR;
```

3.4.5 OSP_DESTINATION

The OSP_DESTINATION data structure holds information about the destination for a call that is being routed using the OSP server. This structure corresponds to the **Destination** element described in [OSP].

```
typedef struct osp_destination
{
  struct osp_destination *next;
  OSP_ADDRESS            *info;
  OSP_ADDRESS            *alt_list;
  char                   *signal_address;
  OSP_TOKEN              *token_list;
  char                   *valid_after;
  char                   *valid_until;
  OSP_USAGE_DETAIL       *usage_list;
  OSP_SERVICE_URL        *auth_url_list;
  OSP_CALL_ID             call_id;
} OSP_DESTINATION;
```

Members:

next	A pointer to the next item if this structure is in a list, or a null pointer if there are no further items.
info	The address of the call destination, or a null pointer if no destination address has been supplied.
alt_list	A pointer to the first item in a list of OSP_ADDRESS structures, each of which contains an alternative address that could be used to reach the required destination. The addresses are listed in order of preference, highest first.
signal_address	A null-terminated string containing the call signalling address for the destination. It is represented as *name:nn*, where *name* is a domain name or an IP address enclosed in square brackets. The *:nn* is optional and indicates a TCP port number.
token_list	A pointer to the first item in a list of OSP_TOKEN structures, each of which contains an H.235 security tokens.
valid_after	A null-terminated string containing the time at which authorization begins for this destination. This is in the format *YYYY-MM-DD*T*hh:mm:ss*Z (the T and Z are literal characters used as delimiters). A null pointer indicates that authorization is effective immediately.
valid_until	A null-terminated string containing the time at which authorization ends for this destination. This is in the format *YYYY-MM-DD*T*hh:mm:ss*Z (the T and Z are literal characters used as delimiters). A null pointer indicates that authorization continues indefinitely.

usage_list	A pointer to the first item in a list of OSP_USAGE_DETAIL structures, each of which contains usage information.
auth_url_list	A pointer to the first item in a list of OSP_SERVICE_URL structures, each of which contains a uniform resource locator (URL) of an OSP server by which authorization may be verified or refreshed.
call_id	H.323 call identifier that uniquely identifies the call.

3.4.6 OSP_TOKEN

The OSP_TOKEN data structure contains an H.235 security token. This structure corresponds to the **Token** element described in [OSP]. OpenOSP does not process the token data itself, but simply passes it to or from a client.

```
typedef struct osp_token
{
  struct osp_token *next;
  char             *encoding;
  int               token_len
  char             *token_data;
} OSP_TOKEN;
```

Members:

next	A pointer to the next item if this structure is in a list, or a null pointer if there are no further items.
encoding	A null-terminated string that identifies the encoding in which the token is stored. One of the following: • cdata (XML CDATA format) • base64
token_len	The size in bytes of the token data. This field is only valid for tokens that are generated by an application and passed as a parameter to the OpenOSP stack; for such tokens, token_len must be set to the size of the token data. Tokens that are generated by the OpenOSP stack are always null-terminated and so do not require the use of this field.
token_data	A string containing the token data. For tokens generated by the OpenOSP stack, the string is null-terminated.

3.4.7 OSP_USAGE_DETAIL

The OSP_USAGE_DETAIL data structure contains information about resource usage. It corresponds to the **UsageDetail** element described in [OSP].

It is used at the UM API to record details of a call that has been completed, and at the AR API to describe the services that have been authorized for a call.

```
typedef struct osp_usage_detail
{
  struct osp_usage_detail *next;
  OSP_SERVICE_INFO        *service;
  char                    *amount;
  char                    *increment;
  char                    *unit;
  char                    *start_time;
  char                    *end_time;
  OSP_TERM_CAUSE          *term_cause;
  OSP_USAGE_STATISTICS    *statistics;
} OSP_USAGE_DETAIL;
```

Members:

next	A pointer to the next item if this structure is in a list, or a null pointer if there are no further items.
service	A pointer to an OSP_SERVICE_INFO structure describing the type of service. A null pointer indicates basic internet telephony service, which is the only type of service currently defined by [OSP].
amount	A null-terminated string containing a number which, when multiplied by 'increment,' gives the number of units of service used or authorized.
increment	A null-terminated string containing a number which, when multiplied by 'amount,' gives the number of units of service used or authorized.
unit	A null-terminated string containing the units of use by which the usage or authorization period is indicated, for example, seconds, packets, bytes.
start_time	A null-terminated string containing the time at which the service begins. This is in the format *YYYY-MM-DD*T*hh:mm:ss*Z (the T and Z are literal characters used as delimiters). In a call authorization, this parameter can be set to a null pointer, indicating that authorization is effective immediately.
end_time	A null-terminated string containing the time at which the service ends. This is in the format *YYYY-MM-DD*T*hh:mm:ss*Z (the T and Z are literal characters used as delimiters). In a call authorization, this parameter can be set to a null pointer, indicating that authorization continues indefinitely.
term_cause	A pointer to an OSP_TERM_CAUSE structure indicating why the call ended.

statistics	A pointer to an OSP_USAGE_STATISTICS structure containing enhanced usage statistics for the call. This parameter may be null if no enhanced usage statistics were provided by the OSP client.

Example: If 'amount' is '10,' 'increment' is '60,' and 'unit' is 's,' this indicates a usage duration of 10 minutes.

3.4.8 OSP_TERM_CAUSE

The OSP_TERM_CAUSE structure contains information about how and why a call ended. It corresponds to the **TCCode** and **TerminationCause** elements described in [OSP].

```
typedef struct osp_term_cause
{
  char  *term_code;
  char  *term_desc;
} OSP_TERM_CAUSE;
```

Members:

term_code	A null-terminated string containing a numerical code indicating why a call ended. See the description of **TCCode** in [OSP] for the values this can take.
term_desc	A null-terminated string containing a summary description of this termination code. See the description of **TerminationCause** in [OSP] for typical summary descriptions.

3.4.9 OSP_USAGE_STATISTICS

The OSP_USAGE_STATISTICS contains enhanced statistical information about a call. It corresponds to the **Statistics** element described in [OSP].

```
typedef struct osp_usage_statistics
{
  struct osp_usage_statistics *next;
  OSP_USAGE_STATISTICS_PF     *loss_sent;
  OSP_USAGE_STATISTICS_PF     *loss_received;
  OSP_USAGE_STATISTICS_MMVS   *one_way_delay;
  OSP_USAGE_STATISTICS_MMVS   *round_trip_delay;
} OSP_USAGE_STATISTICS;
```

Members:

next	A pointer to the next item if this structure is in a list, or a null pointer if there are no further items.
loss_sent	A pointer to an OSP_USAGE_STATISTICS_PF structure containing statistical information about lost packets that were sent from the originating gateway to the terminating gateway.

loss_received	A pointer to an OSP_USAGE_STATISTICS_PF structure containing statistical information about lost packets that were sent from the terminating gateway to the originating gateway.
one_way_delay	A pointer to an OSP_USAGE_STATISTICS_MMVS structure containing statistical information about the one way delay between the originating gateway and the terminating gateway.
round_trip_delay	A pointer to an OSP_USAGE_STATISTICS_MMVS structure containing statistical information about the round trip delay between the originating gateway and the terminating gateway.

3.4.10 OSP_USAGE_STATISTICS_PF

The OSP_USAGE_STATISTICS_PF data structure is used as a substructure of OSP_USAGE_STATISTICS.

```
typedef struct osp_usage_statistics_pf
{
  struct osp_usage_statistics_pf *next;
  char                           *packets;
  char                           *fraction;
} OSP_USAGE_STATISTICS_PF;
```

Members:

next	A pointer to the next item if this structure is in a list, or a null pointer if there are no further items.
packets	A null-terminated string containing the total number of packets that was lost during the call.
fraction	A null-terminated string containing the fraction of packets that were lost, expressed as a number between 0 (no packets lost) and 255 (all packets lost).

3.4.11 OSP_USAGE_STATISTICS_MMVS

The OSP_USAGE_STATISTICS_MMVS data structure is used as a substructure of OSP_USAGE_STATISTICS.

```
typedef struct osp_usage_statistics_mmvs
{
  struct osp_usage_statistics_mmvs *next;
  char                             *minimum;
  char                             *mean;
  char                             *variance;
  char                             *samples;
} OSP_USAGE_STATISTICS_MMVS;
```

Members:

next	A pointer to the next item if this structure is in a list, or a null pointer if there are no further items.
minimum	A null-terminated string containing the minimum measured delay time in milliseconds.
mean	A null-terminated string containing the mean measured delay time in milliseconds.
variance	A null-terminated string containing the statistical variance of the measured delay time in squared milliseconds.
samples	A null-terminated string containing the number of delay time samples measured by the reporting system.

3.4.12 OSP_STATUS

The OSP_STATUS data structure contains information about the status of an OSP operation. It corresponds to the **Status** element described in [OSP].

It is used by the UM, AR and SA APIs to indicate the status code and description that should be used to form the XML response for an OSP operation.

```
typedef struct osp_status
{
  int           code;
  const char   *desc;
} OSP_STATUS;
```

Members:

code	The OSP status code to be returned to the client. See the description of **Code** in [OSP] for the values this can take.
desc	A null-terminated string containing a summary description of this status code.

3.4.13 OSP_SERVICE_INFO

The OSP_SERVICE_INFO data structure contains information about the type of service that is either requested for a particular operation or supported by a particular gateway. It corresponds to the **Service** element described in [OSP].

```
typedef struct osp_service_info
{
  char                  *bandwidth;
  char                  *service_type;
  OSP_PROTOCOL_TYPE      *prot_type
} OSP_SERVICE_INFO;
```

Members:

bandwidth	A null-terminated string containing the requested or supported bandwidth.
service_type	A null-terminated string containing the requested or supported type of service. For example, this field might contain one of the values “voice” or “fax”.
prot_type	A pointer to the first item in a list of OSP_PROTOCOL_TYPE structures, each of which specifies a requested or supported set of protocol details.

3.4.14 OSP_PROTOCOL_TYPE

The OSP_PROTOCOL_TYPE data structure contains information about the protocol details that are either requested for a particular operation or supported by a particular gateway.

It is used as a part of the OSP_SERVICE_INFO and OSP_SUPP_PROTOCOL structures.

```
typedef struct osp_protocol_type
{
  struct osp_protocol_type *next;
  char                     *prot_name;
} OSP_PROTOCOL_TYPE;
```

Members:

next	A pointer to the next item if this structure is in a list, or a null pointer if there are no further items.
prot_name	A null-terminated string containing the requested or supported protocol name. For example, this field might contain one of the values “sip” or “h323”.

3.4.15 OSP_AUTH_INFO

The OSP_AUTH_INFO data structure contains a subscriber authentication token.

Subscriber authentication tokens are used by the AR and SA APIs to describe the fact that a particular subscriber has been authenticated by the OpenOSP server.

```
typedef struct osp_auth_info
{
  struct osp_auth_info     *next;
  char                     *encoding;
  char                     *auth_data;
} OSP_AUTH_INFO;
```

Members:

next	A pointer to the next item if this structure is in a list, or a null pointer if there are no further items.

encoding	A null-terminated string that identifies the encoding in which the token is stored. One of the following: • cdata (XML CDATA format) • base64
auth_data	A null-termination string containing the token data.

3.5 Return codes

All function calls at the OpenOSP interfaces use a standard set of return codes, to indicate that the function completed successfully or to indicate the type of error that occurred.

- When your application makes a call to OpenOSP, it must check the return value from the function to determine whether it was successful.
- Callback routines supplied by your application must use these return codes to indicate success or error conditions to OpenOSP.

The following return code indicates that the function was successful. The caller can use the values of any parameters that have been modified by the function.

OSP_SUCCESS	The function completed successfully.

The following return codes indicate that the function was unsuccessful. The values of any parameters modified by the function may not be valid, and the caller should not use them.

OSP_ERROR	The function failed for an unspecified reason.
OSP_ERR_ALREADY_INIT	OpenOSP was already initialized.
OSP_ERR_NOT_INIT	OpenOSP was not initialized.
OSP_ERR_BAD_HANDLE	The supplied component handle parameter was not valid.
OSP_ERR_BAD_PARAM	One or more of the parameters was not valid.
OSP_ERR_BAD_REQUEST	The supplied request data was not valid.
OSP_ERR_BAD_CORR	The supplied correlator was not valid.
OSP_ERR_MEMORY	The function failed for lack of system memory.
OSP_ERR_NO_RESOURCES	The function failed for lack of a system resource.
OSP_ERR_FILE	The function failed because of a file access problem.
OSP_ERR_CONFIG	OpenOSP was not configured correctly.
OSP_ERR_BIND_FAILED	OpenOSP could not bind to the directory server.

OSP_ERR_DENIED	The supplied credentials were not acceptable.
OSP_ERR_MISSING_ATTR	The supplied address did not carry a type attribute.
OSP_ERR_FAILED_VERIFY	Verification of the supplied data failed.
OSP_ERR_BUFSIZE	The supplied buffer was not large enough.

3.6 Signal Handling

Each of the OpenOSP API functions sets the thread signal mask on entry to mask all maskable signals, and restores the old mask on exit. This is done to ensure predictable operation of the OpenOSP stack.

3.7 C implementation

All of the OSP APIs, together with their associated types and return codes, are defined in the header file openosp.h. The CCM and RM APIs are defined in the header files ccmextrn.h and rmextrn.h.

The OSPAPI modifier on the OSP APIs is included to allow explicit specification of the calling conventions for these functions. It is defined in openosp.h.

4 Control API

The Control API allows the settlement application to

- initialize the OSP stack, start it listening for client connections, and terminate the stack
- obtain statistical data about the stack.

The settlement application may also register callbacks with the Control API. These callbacks are used to

- verify a client when it connects
- store received messages for non-repudiation purposes.

If the application registers a client verification callback, OpenOSP passes out details of each connecting client, including any certificates received during SSL / TLS negotiation. The application may then return a 'cookie' (that is, some application-specific correlator), which the server will pass back to the application on any subsequent callbacks. When the client connection is closed, the server calls the application's disconnection callback (if one is registered) to allow the application to free any resources associated with the connection.

If no client verification callback is registered, the server will accept all non-secure connections, and it will accept all secure connections subject to a consistent and within-date certificate chain.

OpenOSP calls the non-repudiation callback with all S/MIME-signed OSP requests that it receives. This allows the application to store a complete copy of the request.

4.1 osp_init

The settlement application calls this function to initialize the OpenOSP stack and its associated libraries.

See section 3.2, Initialization sequence, for details of how this function must be used within OpenOSP's initialization sequence.

```
OSP_RC OSPAPI osp_init(void);
```

Parameters:

None.

4.2 osp_listen

The settlement application calls this function to instruct OpenOSP to begin listening for connections from clients.

See section 3.2, Initialization sequence, for details of how this function must be used within OpenOSP's initialization sequence.

```
OSP_RC OSPAPI osp_listen(void);
```

Parameters:

None.

4.3 osp_term

The settlement application calls this function to terminate the OpenOSP stack and its associated libraries. This function blocks until the stack has completely terminated. Note that this means that osp_term() may not be called from within a callback.

See section 3.3, Termination sequence, for details of how this function must be used within OpenOSP's termination sequence.

```
OSP_RC OSPAPI osp_term(void);
```

Parameters:

None.

4.4 osp_client_verify_register

The settlement application calls this function to register two callback routines. One callback handles verification of each connecting client, while the other receives a notification for each disconnecting client. The application does not need to register both callback routines; a null pointer may be supplied in place of either function pointer if appropriate.

If the application issues more than one call to this function, the parameters provided in the most recent call override those provided in previous calls.

The application may deregister either or both callbacks at any time by calling this function with a null pointer in place of the appropriate function pointer (and repeating the existing callback function pointer for a callback that it still wishes to handle).

```
OSP_RC OSPAPI osp_client_verify_register
                                    (POSP_CLIENT_VERIFY_CALLBACK      cv_cb,
                                     POSP_CLIENT_DISCONNECT_CALLBACK  cd_cb);
```

Parameters:

cv_cb	A pointer to the settlement application's client verification callback routine. See section 4.5, POSP_CLIENT_VERIFY_CALLBACK, for details of this routine. To deregister the existing callback routine, the application supplies a null pointer.

cd_cb	A pointer to the settlement application's client disconnection callback routine. See section 4.6, POSP_CLIENT_DISCONNECT_CALLBACK, for details of this routine. To deregister the existing callback routine, the application supplies a null pointer.

4.5 POSP_CLIENT_VERIFY_CALLBACK

The settlement application provides this function to receive details of each connecting OSP client. It may accept or deny the connection attempt based on the information supplied. The application registers this function during initialization by supplying the address of the function on the osp_client_verify_register() call.

OpenOSP calls this callback function only once for each new non-secure connection from an OSP client. Only the parameters `ip_address` and `ver_cookie_ptr` are valid in this case; the other parameters are set to null or zero. The value of the cookie pointed to by `ver_cookie_ptr` is initially null, and the application is free to set it to whatever value is desired. The application may then use the cookie as a correlator to identify this connection in subsequent callbacks across the application API.

OpenOSP calls this callback function one or more times for each new secure connection from an OSP client. In this case, each call supplies information about one of the certificates presented by the client during the SSL / TLS negotiation. The certificates are supplied in order, with the root certificate in the first call and the client's certificate in the last call.

The first time this function is called for a particular secure connection, the cookie is set in the same way as described above for non-secure connections. On subsequent calls to this function for the same connection, the cookie value pointed to by `ver_cookie_ptr` is the same as that returned on the previous call. The application may change the value of the cookie on each call if desired. The last returned cookie value is saved for later use on the application API callbacks and on the disconnection callback.

The application must return OSP_SUCCESS if the client's credentials are acceptable, and OSP_ERR_DENIED otherwise. If the application returns OSP_ERR_DENIED, the client's connection attempt is denied and OpenOSP calls the disconnection callback, if one is registered. OpenOSP will not make further calls to the client verification callback for a connection that has been denied.

```
OSP_RC (OSPAPI *POSP_CLIENT_VERIFY_CALLBACK)
                                        (unsigned long   ip_address,
                                         void          **ver_cookie_ptr,
                                         char           *cert,
                                         unsigned int    cert_len,
                                         unsigned int    chain_len,
                                         unsigned int    chain_pos);
```

Parameters:

ip_address	The IP address of the client.

ver_cookie_ptr	A pointer to a verification 'cookie' that the application may fill in with a private value. OpenOSP will pass this value on all subsequent callbacks across the application API.
cert	A pointer to the certificate to be checked. The certificate is in ASN.1 DER-encoded X.509 format.
cert_len	The number of bytes in the certificate.
chain_len	The total number of certificates in the chain, including the clients certificate and the root certificate.
chain_pos	The position of the supplied certificate in the client's chain of certificates; 0 indicates the client's certificate.

4.6 POSP_CLIENT_DISCONNECT_CALLBACK

The settlement application provides this function so that OpenOSP can inform it when an OSP client has disconnected. The application registers this function during initialization by supplying the address of the function on the osp_client_verify_register() call.

OpenOSP then calls this callback function each time an OSP client disconnects from the server. This allows the application to free any resources associated with the client connection.

This function is also called if the application denies a connection attempt by returning OSP_ERR_DENIED from the client verification callback.

```
void (OSPAPI *POSP_CLIENT_DISCONNECT_CALLBACK)
                                          (OSP_CLIENT_ID  *client_id);
```

Parameters:

client_id	A structure containing the client IP address and the 'cookie' from any earlier client verification.

4.7 osp_non_repudiation_register

The settlement application calls this function to register a callback routine that will handle data for non-repudiation. The application does not need to support non-repudiation; if it does not call this function, the server will not provide this information.

OpenOSP then calls the callback routine each time it processes an S/MIME-signed OSP request. This allows the application to store a copy of the request data.

If the application issues more than one call to this function, the parameter provided in the most recent call overrides those provided in previous calls. The application may deregister the callback at any time by calling this function with a null pointer in place of the function pointer.

```
OSP_RC OSPAPI osp_non_repudiation_register
                                      (POSP_NON_REPUDIATION_CALLBACK nr_cb);
```

Parameters:

nr_cb	A pointer to the settlement application's non-repudiation callback routine. See section 4.8, POSP_NON_REPUDIATION_CALLBACK, for details of this routine. To deregister the existing callback routine, the application supplies a null pointer.

4.8 POSP_NON_REPUDIATION_CALLBACK

The settlement application provides this function to handle non-repudiation data, and registers it during initialization by supplying the address of the function on the osp_non_repudiation_register() call.

OpenOSP then calls this callback function each time it processes an S/MIME-signed OSP request. The parameters to the function include a pointer to the request data buffer, which contains all of the HTTP headers, the request itself, and the S/MIME signature. This allows the settlement application to store a complete copy of the request.

```
void (OSPAPI *POSP_NON_REPUDIATION_CALLBACK)
                                          (char           *buffer,
                                           unsigned int    buf_len,
                                           OSP_CLIENT_ID  *client_id);
```

Parameters:

buffer	A pointer to the complete HTTP request as it was received from the client.
buf_len	The length of the supplied buffer.
client_id	A structure containing the client IP address and the 'cookie' from any earlier client verification.

4.9 osp_get_stack_statistics

The settlement application calls this function to gather statistics about the number of connections (both secure and non-secure) and requests received by OpenOSP since it was initialized.

```
OSP_RC OSPAPI osp_get_stack_statistics(OSP_STATS *stats);

typedef struct osp_stats
{
  unsigned int     ssl_conns;      // secure connections received
  unsigned int     nonssl_conns;   // non-secure connections received
  unsigned int     osp_requests;   // total OSP requests received
  unsigned int     scep_requests;  // total SCEP requests received
} OSP_STATS;
```

Parameters:

stats	A pointer to a statistics structure. OpenOSP fills in this structure with statistics information, as follows:
stats.ssl_conns	The total number of secure connections established with OSP clients since OpenOSP was initialized.
stats.nonssl_conns	The total number of non-secure connections established with OSP clients since OpenOSP was initialized.
stats.osp_requests	The total number of OSP requests received from OSP clients since OpenOSP was initialized.
stats.scep_requests	The total number of SCEP requests received from SCEP clients since OpenOSP was initialized. See [SCEP] for full details of this protocol.

5 Usage Metering API

The Usage Metering (UM) API allows OpenOSP to pass usage information to your settlement application. The information may be passed into a permanent storage facility (for later retrieval by an offline settlement application), or directly to an external usage metering application.

5.1 osp_um_register

The application calls this function to register a callback routine that will handle information from OSP **UsageIndication** messages. OpenOSP then calls this UM callback routine each time it receives a **UsageIndication** from an OSP client.

If the application issues more than one call to this function, the parameter provided in the most recent call overrides those provided in previous calls.

The application may deregister the callback at any time by calling this function with a null pointer in place of the function pointer. After deregistering the callback, the application will not receive any further usage information unless it registers a new UM callback routine.

```
OSP_RC OSPAPI osp_um_register(POSP_UM_CALLBACK um_cb);
```

Parameters:

um_cb	A pointer to the settlement application's usage metering callback routine. See section 5.2, POSP_UM_CALLBACK, for details of this routine. To deregister the existing callback routine, the application supplies a null pointer.

5.2 POSP_UM_CALLBACK

The application provides this function to handle information from OSP **UsageIndication** messages, and registers it during initialization by supplying the address of the function on the osp_um_register() call.

OpenOSP then calls this function each time it receives a **UsageIndication** from an OSP client. The parameters passed to the routine provide usage information for a call that has been made using the OSP server.

The settlement application must respond to the callback by calling osp_um_response().

```
OSP_RC (OSPAPI *POSP_UM_CALLBACK)(OSP_CORRELATOR     corr,
                                  char              *timestamp,
                                  char              *role,
                                  char              *transaction_id,
                                  OSP_CALL_ID       *call_id,
                                  OSP_ADDRESS       *source_info,
                                  OSP_ADDRESS       *source_alt_list,
                                  OSP_ADDRESS       *dest_info,
                                  OSP_ADDRESS       *dest_alt_list,
                                  OSP_USAGE_DETAIL  *use_detail_list,
                                  OSP_CLIENT_ID     *client_id);
```

For details of the data structures OSP_CALL_ID, OSP_ADDRESS, OSP_USAGE_DETAIL, and OSP_CLIENT_ID, see section 3.4, Common data structures and types.

Parameters:

corr — A correlator to be passed back to OpenOSP on the subsequent call to osp_um_response().

timestamp — A null-terminated string containing the message's timestamp. This is in the format *YYYY-MM-DD*T*hh:mm:ss*Z (the T and Z are literal characters used as delimiters).

role — A null-terminated string describing the client's role in the call. This may take one of the following values:

- source — The client was the source.
- destination — The client was the destination.
- other — The client was a system other than the source or the destination.

transaction_id — A null-terminated string containing the transaction identifier provided by the client from its previous authorization by the server. This was provided by the application on a previous osp_ar_auth_response() call.

call_id — A structure containing the H.323 call ID provided by the client.

source_info — A structure containing details about the source of the call, e.g. the originating phone number.

source_alt_list — A pointer to the first item in a list of OSP_ADDRESS structures, each of which contains alternate information about the source of the call, e.g. the address of the source gateway.

dest_info — A structure containing details about the destination of the call, e.g. the destination phone number.

dest_alt_list	A pointer to the first item in a list of OSP_ADDRESS structures, each of which contains alternate information about the destination, e.g. the address of the termination gateway.
use_detail_list	A pointer to the first item in a list of OSP_USAGE_DETAIL structures, each of which contains the usage details of the call, e.g. the duration of the call and the reason it ended.
client_id	A structure containing the client IP address and the 'cookie' from any earlier client verification.

5.3 osp_um_response

When OpenOSP calls the settlement application's implementation of POSP_UM_CALLBACK(), the application must respond by calling osp_um_response(). This function uses the information provided to construct an OSP **UsageConfirmation** message, to be sent to the OSP client from which the indication originated. This indicates acceptance or rejection of the information in the indication.

```
OSP_RC OSPAPI osp_um_response(OSP_CORRELATOR  corr,
                              OSP_STATUS     *status,
                              unsigned int    audit_signal,
                              unsigned int    sig_required);
```

Parameters:

corr	The correlator supplied by OpenOSP on the corresponding usage metering callback.
status	A pointer to an OSP_STATUS structure that specifies the status code and description that should be used to form the **UsageConfirmation** response.
audit_signal	This parameter controls the use of Cisco's **AuditSignal** extension to OSP. Set it to 0 if OpenOSP should not include an **AuditSignal** element in the response. Otherwise, the following values are valid: • OSP_UM_AUDIT_START: include an **AuditSignal** element in the response that tells the client to sign future **UsageIndication** messages using S/MIME. • OSP_UM_AUDIT_STOP: include an **AuditSignal** element in the response that tells the client not to sign future **UsageIndication** messages using S/MIME.
sig_required	Set this parameter to 1 if the server should S/MIME sign the response, 0 if it should not.

For details of the OSP_STATUS structure, see section 3.4, Common data structures and types.

6 Authorization and Routing API

The Authorization and Routing (AR) API allows OpenOSP to request authorization and routing information from your application for a call. This API is also used to pass pricing and capabilities information (gathered from OSP clients) to your application, for use in routing and authorization decisions.

Authorization and pricing are grouped into the same API because both sets of functions need to access the routing and pricing database.

Information from five different OSP requests and indications is handled at this API, and each request or indication is associated with a different callback routine provided by the application. The application can choose to support only a subset of these if appropriate.

- **AuthorizationRequest**: OpenOSP can request the application to provide authorization and routing information for a call.

- **AuthorizationIndication**: OpenOSP can request the application to verify authorization information received by a client that is accepting an incoming call.

- **ReauthorizationRequest**: OpenOSP can request the application to extend the authorization period for an existing call (for example if additional payments have been made to extend a prepaid call).

- **PricingIndication**: OpenOSP can provide information to the application about the pricing for calls routed through different gateways (for use by the application in determining the best route for a call).

- **CapabilitiesIndication**: OpenOSP can provide information to the application about the capabilities of different gateways (for use by the application in determining the best route for a call).

6.1 osp_ar_register

The application calls this function to register callback routines that will handle each of the five OSP requests described above. The application does not need to support all five; it can provide callback routines for only the requests it needs to handle.

OpenOSP then calls the appropriate callback routine each time it receives a request of a type for which the application has registered.

If the application issues more than one call to this function, the parameters provided in the most recent call override those provided in previous calls.

The application may deregister one or more of the callbacks at any time by calling this function with null pointers in place of the appropriate function pointers (and repeating the existing callback function pointers for any callbacks that it still wishes to handle).

```
OSP_RC OSPAPI osp_ar_register( POSP_AR_AUTH_RQ_CALLBACK     ar_cb,
                               POSP_AR_AUTH_IND_CALLBACK    ai_cb,
                               POSP_AR_REAUTH_RQ_CALLBACK   ra_cb,
                               POSP_AR_PRICING_IND_CALLBACK pi_cb,
                               POSP_AR_CAPS_IND_CALLBACK    cp_cb);
```

Parameters:

ar_cb	A pointer to the authorization request callback routine, or a null pointer if the application does not support authorization requests. See section 6.2, POSP_AR_AUTH_RQ_CALLBACK, for details of this routine.
ai_cb	A pointer to the authorization indication callback routine, or a null pointer if the application does not support authorization indications. See section 6.4, POSP_AR_AUTH_IND_CALLBACK, for details of this routine.
ra_cb	A pointer to the reauthorization request callback routine, or a null pointer if the application does not support reauthorization requests. See section, 6.6, POSP_AR_REAUTH_RQ_CALLBACK for details of this routine.
pi_cb	A pointer to the pricing indication callback routine, or a null pointer if the application does not support pricing indications. See section 6.8, POSP_AR_PRICING_IND_CALLBACK, for details of this routine.
cp_cb	A pointer to the capabilities indication callback routine, or a null pointer if the application does not support capabilities indications. See section 6.10, POSP_AR_CAPS_IND_CALLBACK, for details of this routine.

6.2 POSP_AR_AUTH_RQ_CALLBACK

The application provides this function to handle information from OSP **AuthorizationRequest** messages, and registers it during initialization by supplying the address of the function on the osp_ar_register() call.

OpenOSP then calls this callback routine each time it receives an **AuthorizationRequest** from an OSP client. The parameters passed to the routine provide information about the request.

The application must respond to the callback by calling osp_ar_auth_response().

```
OSP_RC (OSPAPI *POSP_AR_AUTH_RQ_CALLBACK)
                                    (OSP_CORRELATOR    corr,
                                     char             *timestamp,
                                     OSP_CALL_ID      *call_id_list,
                                     OSP_ADDRESS      *source_info,
                                     OSP_ADDRESS      *source_alt_list,
                                     OSP_ADDRESS      *dest_info,
                                     OSP_ADDRESS      *dest_alt_list,
                                     OSP_SERVICE_INFO *service,
                                     char             *max_dests,
                                     OSP_TOKEN        *token_list,
                                     OSP_AUTH_INFO    *authentication_list,
                                     OSP_CLIENT_ID    *client_id);
```

For details of the data structures OSP_CALL_ID, OSP_ADDRESS, OSP_SERVICE_INFO, OSP_TOKEN, OSP_AUTH_INFO and OSP_CLIENT_ID, see section 3.4, Common data structures and types.

Parameters:

corr	A correlator to be passed back to OpenOSP on the subsequent call to osp_ar_auth_response().
timestamp	A null-terminated string containing the message's timestamp. This is a string in the format *YYYY-MM-DD*T*hh:mm:ss*Z (the T and Z are literal characters used as delimiters).
call_id_list	A pointer to the first item in a list of OSP_CALL_ID structures, each of which contains an H.323 call ID provided by the client.
source_info	A structure containing details about the source of the call, e.g. the originating phone number.
source_alt_list	A pointer to the first item in a list of OSP_ADDRESS structures, each of which contains alternate information about the source of the call, e.g. the address of the source gateway. This may contain subscriber information, in which case the settlement application should authenticate and authorize the subscriber.
dest_info	A structure containing details about the destination of the call, e.g. the destination phone number.
dest_alt_list	A pointer to the first item in a list of OSP_ADDRESS structures, each of which contains alternate information about the destination, e.g. the address of the termination gateway.
service	A pointer to an OSP_SERVICE_INFO structure describing the type of service requested. A null pointer indicates basic internet telephony service, which is the only type of service currently defined by [OSP].

max_dests	The maximum number of destinations to be returned.
token_list	A pointer to the first item in a list of OSP_TOKEN structures, each of which contains a authorization token.
authentication_list	A pointer to the first item in a list of OSP_AUTH_INFO structures, each of which contains a subscriber authentication token.
client_id	A structure containing the client IP address and the 'cookie' from any earlier client verification.

6.3 osp_ar_auth_response

When OpenOSP calls the settlement application's implementation of POSP_AR_AUTH_RQ_CALLBACK(), the application must respond by calling osp_ar_auth_response(). This function uses the information provided to construct an OSP **AuthorizationResponse** message, to be sent to the OSP client from which the request originated. This provides routing information to allow the OSP client to route the requested call, and authorization information for the call.

```
OSP_RC OSPAPI osp_ar_auth_response(OSP_CORRELATOR    corr,
                                   OSP_STATUS       *status,
                                   char             *transaction_id,
                                   OSP_TOKEN        *token_list,
                                   OSP_DESTINATION  *dest_list,
                                   unsigned int      sig_required);
```

For details of the data structures OSP_STATUS, OSP_DESTINATION and OSP_TOKEN, see section 3.4, Common data structures and types.

Parameters:

corr	The correlator supplied by OpenOSP on the corresponding authorization request callback.
status	A pointer to an OSP_STATUS structure that specifies the status code and description that should be used to form the **AuthorizationResponse** response.
transaction_id	A null-terminated string containing an identifier assigned by the application to this authorization. The format of this parameter is defined by the application; OpenOSP does not make use of it, but passes it back to the application on subsequent calls in order to identify the transaction.
token_list	A pointer to the first item in a list of OSP_TOKEN structures, each of which contains an authorization token that is valid for any of the destinations in the dest_list parameter.

dest_list	A pointer to the first item in a list of OSP_DESTINATION structures, each of which contains details of a destination to which service has been authorized.
sig_required	Set this parameter to 1 if the server should S/MIME sign the response, 0 if it should not.

6.4 POSP_AR_AUTH_IND_CALLBACK

The application provides this function to handle information from OSP **AuthorizationIndication** messages, and registers it during initialization by supplying the address of the function on the osp_ar_register() call.

OpenOSP then calls this function each time it receives an **AuthorizationIndication** from an OSP client (requesting the application to verify authorization information received by a client that is accepting an incoming call). The parameters passed to the routine provide information about the indication.

The application must respond to the callback by calling osp_ar_auth_confirm().

```
OSP_RC (OSPAPI *POSP_AR_AUTH_IND_CALLBACK)
                                        (OSP_CORRELATOR    corr,
                                         char             *timestamp,
                                         char             *role,
                                         OSP_CALL_ID      *call_id,
                                         OSP_ADDRESS      *source_info,
                                         OSP_ADDRESS      *source_alt_list,
                                         OSP_ADDRESS      *dest_info,
                                         OSP_ADDRESS      *dest_alt_list,
                                         OSP_SERVICE_INFO *service,
                                         OSP_TOKEN        *auth_token_list,
                                         OSP_CLIENT_ID    *client_id);
```

For details of the data structures OSP_CALL_ID, OSP_ADDRESS, OSP_SERVICE_INFO, OSP_TOKEN, and OSP_CLIENT_ID, see section 3.4, Common data structures and types.

Parameters:

corr	A correlator to be passed back to OpenOSP on the subsequent call to osp_ar_auth_confirm().
timestamp	A null-terminated string containing the message's timestamp.
role	A null-terminated string describing the client's role in the call. This may take one of the following values: • source The client is the source. • destination The client is the destination. • other The client is a system other than the source or the destination.

call_id	A structure containing the H.323 call ID provided by the client.
source_info	A pointer to a structure containing details about the source of the call, e.g. the originating phone number.
source_alt_list	A pointer to the first item in a list of OSP_ADDRESS structures, each of which contains alternate information about the source of the call, e.g. the address of the source gateway.
dest_info	Pointer to a structure containing details about the destination of the call, e.g. the destination phone number.
dest_alt_list	A pointer to the first item in a list of OSP_ADDRESS structures, each of which contains alternate information about the destination, e.g. the address of the termination gateway.
service	A pointer to an OSP_SERVICE_INFO structure describing the type of service requested. A null pointer indicates basic internet telephony service, which is the only type of service currently defined by [OSP].
auth_token_list	A pointer to the first item in a list of OSP_TOKEN structures, each of which contains an authorization token.
client_id	A structure containing the client IP address and the 'cookie' from any earlier client verification.

6.5 osp_ar_auth_confirm

When OpenOSP calls the settlement application's implementation of POSP_AR_AUTH_IND_CALLBACK(), requesting the application to verify authorization information received by a client that is accepting an incoming call, the application must respond by calling osp_ar_auth_confirm(). This function uses the information provided to construct an OSP **AuthorizationConfirmation** message, to be sent to the OSP client from which the indication originated. This indicates whether or not the authorization in the indication is valid.

```
OSP_RC OSPAPI osp_ar_auth_confirm(OSP_CORRELATOR  corr,
                                  OSP_STATUS     *status,
                                  char           *valid_after,
                                  char           *valid_until,
                                  unsigned int    sig_required);
```

Parameters:

corr	The correlator supplied by OpenOSP on the corresponding authorization indication callback.
status	A pointer to an OSP_STATUS structure that specifies the status code and description that should be used to form the **AuthorizationConfirmation** response.

valid_after	A null-terminated string containing the time that authorization begins; a null pointer means immediately.
valid_until	A null-terminated string containing the time that authorization ends; a null pointer means that it is indefinite.
sig_required	Set this parameter to 1 if the server should S/MIME sign the response, 0 if it should not.

For details of the data structure OSP_STATUS, see section 3.4, Common data structures and types.

6.6 POSP_AR_REAUTH_RQ_CALLBACK

The application provides this function to handle OSP **ReauthorizationRequest** messages, and registers it during initialization by supplying the address of the function on the osp_ar_register() call.

OpenOSP then calls this callback routine each time it receives a **ReauthorizationRequest** from an OSP client. The parameters passed to the routine provide information about the request.

The application must respond to the callback by calling osp_ar_reauth_response().

```
typedef OSP_RC (OSPAPI *POSP_AR_REAUTH_RQ_CALLBACK)
                                        (OSP_CORRELATOR     corr,
                                         char              *timestamp,
                                         char              *role,
                                         OSP_CALL_ID       *call_id,
                                         OSP_ADDRESS       *source_info,
                                         OSP_ADDRESS       *source_alt_list,
                                         OSP_ADDRESS       *dest_info,
                                         OSP_ADDRESS       *dest_alt_list,
                                         char              *transaction_id,
                                         OSP_USAGE_DETAIL  *use_detail_list,
                                         OSP_TOKEN         *auth_token_list,
                                         OSP_CLIENT_ID     *client_id);
```

For details of the data structures OSP_CALL_ID, OSP_ADDRESS, OSP_USAGE_DETAIL, OSP_TOKEN, and OSP_CLIENT_ID, see section 3.4, Common data structures and types.

Parameters:

corr	A correlator to be passed back to OpenOSP on the subsequent call to osp_ar_reauth_response().
timestamp	A null-terminated string containing the message's timestamp.
role	A null-terminated string describing the client's role in the call. This may take one of the following values: • source The client is the source. • destination The client is the destination.

	• other — The client is a system other than the source or the destination.
call_id	Structure containing the H.323 call ID provided by the client.
source_info	Pointer to a structure containing details about the source of the call, e.g. the originating phone number.
source_alt_list	A pointer to the first item in a list of OSP_ADDRESS structures, each of which contains alternate information about the source of the call, e.g. the address of the source gateway.
dest_info	Pointer to a structure containing details about the destination of the call, e.g. the destination phone number.
dest_alt_list	A pointer to the first item in a list of OSP_ADDRESS structures, each of which contains alternate information about the destination, e.g. the address of the termination gateway.
transaction_id	A null-terminated string containing the transaction identifier provided by the client from its previous authorization by the server.
use_detail_list	A pointer to the first item in a list of OSP_USAGE_DETAIL structures, each of which contains usage details of the call, e.g. the duration of the call and the reason it ended.
auth_token_list	A pointer to the first item in a list of OSP_TOKEN structures, each of which contains an authorization token.
client_id	A structure containing the client IP address and the 'cookie' from any earlier client verification.

6.7 osp_ar_reauth_response

When OpenOSP calls the settlement application's implementation of POSP_AR_REAUTH_RQ_CALLBACK(), the application must respond by calling osp_ar_reauth_response(). This function uses the information provided to construct an OSP **ReauthorizationResponse** message, to be sent to the OSP client from which the request originates. This indicates acceptance or rejection of the request.

```
OSP_RC OSPAPI osp_ar_reauth_response(OSP_CORRELATOR    corr,
                                     OSP_STATUS       *status,
                                     char             *transaction_id,
                                     OSP_DESTINATION  *dest_list,
                                     unsigned int      sig_required);
```

For details of the data structures OSP_STATUS and OSP_DESTINATION, see section 3.4, Common data structures and types.

Parameters:

corr	The correlator supplied by OpenOSP on the corresponding reauthorization request callback.
status	A pointer to an OSP_STATUS structure that specifies the status code and description that should be used to form the **ReauthorizationResponse** response.
transaction_id	A null-terminated string containing an identifier assigned by the application to this reauthorization. The format of this parameter is defined by the application; OpenOSP does not make use of it, but passes it back to the application on subsequent calls in order to identify the transaction.
dest_list	A pointer to the first item in a list of OSP_DESTINATION structures, each of which contains details of a destination to which service has been authorized.
sig_required	Set this parameter to 1 if the server should S/MIME sign the response, 0 if it should not.

6.8 POSP_AR_PRICING_IND_CALLBACK

The application provides this function to handle OSP **PricingIndication** messages, and registers it during initialization by supplying the address of the function on the osp_ar_register() call.

OpenOSP then calls this callback routine each time it receives a **PricingIndication** from an OSP client. The parameters passed to the routine provide information about the indication.

The application must respond to the callback by calling osp_ar_pricing_confirm().

```
OSP_RC (OSPAPI *POSP_AR_PRICING_IND_CALLBACK)
                              (OSP_CORRELATOR    corr,
                               char             *timestamp,
                               OSP_ADDRESS      *source_info,
                               OSP_ADDRESS      *dest_info,
                               char             *currency,
                               char             *amount,
                               char             *increment,
                               char             *unit,
                               OSP_SERVICE_INFO *service,
                               char             *valid_after,
                               char             *valid_until,
                               OSP_CLIENT_ID    *client_id);
```

For details of the data structures OSP_ADDRESS, OSP_SERVICE_INFO and OSP_CLIENT_ID, see section 3.4, Common data structures and types.

Parameters:

corr	A correlator to be passed back to OpenOSP on the subsequent call to osp_ar_pricing_confirm().

timestamp	A null-terminated string containing the message's timestamp.
source_info	Pointer to a structure containing details about the source of the call, e.g. the originating phone number.
dest_info	Pointer to a structure containing details about the destination of the call, e.g. the destination phone number.
currency	A null-terminated string describing the currency used for pricing.
amount	A null-terminated string containing the price of the number of units specified in 'increment.'
increment	A null-terminated string containing the number of usage units that are available for the specified amount of currency.
unit	A null-terminated string containing the units of use by which pricing is measured, for example, seconds, packets, bytes.
service	A pointer to an OSP_SERVICE_INFO structure describing the type of service. A null pointer indicates basic internet telephony service, which is the only type of service currently defined by [OSP].
valid_after	A null-terminated string containing the time that new pricing begins; a null pointer means immediately.
valid_until	A null-terminated string containing the duration of pricing validity; a null pointer means that it is indefinite.
client_id	A structure containing the client IP address and the 'cookie' from any earlier client verification.

Example: If 'currency' is 'USD,' 'amount' is '0.5,' 'increment' is '60,' and 'unit' is 's,' this indicates a cost of 50 cents per minute to the destination specified in 'dest_info.'

6.9 osp_ar_pricing_confirm

When OpenOSP calls the settlement application's implementation of POSP_AR_PRICING_IND_CALLBACK(), the application must respond by calling osp_ar_pricing_confirm(). This function uses the information provided to construct an OSP **PricingConfirmation** message, to be sent to the OSP client from which the indication originated. This indicates acceptance or rejection of the information in the indication.

```
OSP_RC OSPAPI osp_ar_pricing_confirm(OSP_CORRELATOR  corr,
                                     OSP_STATUS     *status,
                                     unsigned int    sig_required);
```

Parameters:

corr	The correlator supplied by OpenOSP on the corresponding pricing indication callback.
status	A pointer to an OSP_STATUS structure that specifies the status code and description that should be used to form the **PricingConfirmation** response.
sig_required	Set this parameter to 1 if the server should S/MIME sign the response, 0 if it should not.

6.10 POSP_AR_CAPS_IND_CALLBACK

The application provides this function to handle OSP **CapabilitiesIndication** messages, and registers it during initialization by supplying the address of the function on the osp_ar_register() call.

OpenOSP then calls this callback routine each time it receives a **CapabilitiesIndication** from an OSP client. The parameters passed to the routine provide information about the indication.

The application must respond to the callback by calling osp_ar_caps_confirm().

```
OSP_RC (OSPAPI *POSP_AR_CAPS_IND_CALLBACK)
                              (OSP_CORRELATOR    corr,
                               OSP_DEVICE_INFO  *dev_info_list,
                               char             *osp_version,
                               OSP_CAPS         *caps_list,
                               OSP_RESOURCE     *resource_list,
                               OSP_CLIENT_ID    *client_id);
```

For details of the data structure OSP_CLIENT_ID, see section 3.4, Common data structures and types.

Parameters:

corr	A correlator to be passed back to OpenOSP on the subsequent call to osp_ar_caps_confirm().
dev_info_list	A pointer to the first item in a list of OSP_DEVICE_INFO structures, each of which describes an address type that is supported by the client.
osp_version	A null-terminated string containing the highest version of the OSP protocol that the client supports.
caps_list	A pointer to the first item in a list of OSP_CAPS structures, each of which describes an OSP service supported by the client e.g. **AuthorizationIndication**, **UsageIndication**.

resource_list	A pointer to the first item in a list of OSP_RESOURCE structures, each of which describes a protocol that is supported by the client.
client_id	A structure containing the client IP address and the 'cookie' from any earlier client verification.

6.10.1 OSP_DEVICE_INFO

This structure provides data corresponding to the **DeviceInfo** element described in [OSP].

```
typedef struct osp_device_info
{
  struct osp_device_info *next;
  char                   *type;
  char                   *data;
} OSP_DEVICE_INFO;
```

Members:

next	A pointer to the next item if this structure is in a list, or a null pointer if there are no further items.
type	A null-terminated string describing the type of the device. The following values are valid: e164, h323, url, email, transport, serialnumber, or customerId.
data	A null-terminated string containing the data supplied in the **DeviceInfo** element.

6.10.2 OSP_CAPS

This structure contains the name of a single OSP service.

```
typedef struct osp_caps
{
  struct osp_caps *next;
  char            *cap_name;
} OSP_CAPS;
```

Members:

next	A pointer to the next item if this structure is in a list, or a null pointer if there are no further items.
cap_name	A null-terminated string that contains the name of a particular OSP service; for example, **AuthorizationRequest** or **UsageIndication**.

6.10.3 OSP_RESOURCE

This structure provides data corresponding to the **Resources** element described in [OSP].

```
typedef struct osp_resource
{
  struct osp_resource *next;
  OSP_DATA_RATE       *data_rate;
  char                *almost_out;
  OSP_SUPP_PROTOCOL   *supp_protocol_list;
} OSP_RESOURCE;
```

Members:

next	A pointer to the next item if this structure is in a list, or a null pointer if there are no further items.
data_rate	A pointer to the first item in a list of OSP_DATA_RATE structures, each of which contains details of a supported data rate.
almost_out	A null-terminated string indicating whether or not the originating client is almost out of resources. This field should take one of the values “true” and “false”.
supp_protocol_list	A pointer to the first item in a list of OSP_SUPP_PROTOCOL structures, each of which contains details of a supported protocol.

6.10.4 OSP_SUPP_PROTOCOL

This structure provides data corresponding to the **SupportedProtocol** element described in [OSP].

```
typedef struct osp_supp_protocol
{
  struct osp_supp_protocol *next;
  OSP_PROTOCOL_TYPE        *type;
  OSP_DATA_RATE            *data_rates;
} OSP_SUPP_PROTOCOL;
```

Members:

next	A pointer to the next item if this structure is in a list, or a null pointer if there are no further items.
type	A null-terminated string describing the type of protocol. The following values are valid: H323, SIP, SS7 or Other.
data_rates	A pointer to the first item in a list of OSP_DATA_RATE structures, each of which contains details of an available data rate for this protocol.

6.10.5 OSP_DATA_RATE

This structure provides data corresponding to the **DataRate** element described in [OSP].

```
typedef struct osp_data_rate
{
  struct osp_data_rate *next;
  char                 *num_channels;
  char                 *bandwidth;
} OSP_DATA_RATE;
```

Members:

next	A pointer to the next item if this structure is in a list, or a null pointer if there are no further items.
num_channels	A null-terminated string indicating the number of channels available. This parameter may be null, which indicates that the number of channels was not specified.
bandwidth	A null-terminated string indicating the bandwidth, in bits/sec, of each channel, or the total bandwidth if the number of channels is not specified.

6.11 osp_ar_caps_confirm

When OpenOSP calls the settlement application's implementation of POSP_AR_CAPS_CALLBACK(), the application must respond by calling osp_ar_caps_confirm(). This function uses the information provided to construct an OSP **CapabilitiesConfirmation** message, to be sent to the OSP client from which the indication originated. This allows the application to specify the OSP capabilities that clients should use to communicate with the server.

```
OSP_RC OSPAPI osp_ar_caps_confirm(OSP_CORRELATOR    corr,
                                  OSP_STATUS       *status,
                                  char             *osp_version,
                                  OSP_SERVICE      *service_list,
                                  OSP_CERTIFICATE  *cert_list,
                                  char             *device_id,
                                  unsigned int      sig_required);
```

Parameters:

corr	The correlator supplied by OpenOSP on the corresponding capabilities indication callback.
status	A pointer to an OSP_STATUS structure that specifies the status code and description that should be used to form the **CapabilitiesConfirmation** response.
osp_version	A null-terminated string containing the negotiated version of the OSP protocol that will be used.
service_list	A pointer to the first item in a list of OSP_SERVICE structures, each of which contains details of a particular service supported by the OSP server.

cert_list	A pointer to the first item in a list of OSP_CERTIFICATE structures, each of which contains a public key certificate. The first certificate in the list is the certificate that the server will use to sign any authorization tokens that it supplies. This is followed by any intermediate certificates, in order, and the list concludes with the root authority's certificate.
device_id	Server device identifier.
sig_required	Set this parameter to 1 if the server should S/MIME sign the response, 0 if it should not.

6.11.1 OSP_SERVICE

This structure provides data corresponding to the **OSPService** element described in [OSP].

```
typedef struct osp_service
{
  struct osp_service *next;
  char               *capability;
  OSP_SERVICE_URL    *url_list;
  unsigned int        sig_required;
} OSP_SERVICE;
```

Members:

next	A pointer to the next item if this structure is in a list, or a null pointer if there are no further items.
capability	A null-terminated string containing the name of an OSP service supported by the server e.g. **AuthorizationIndication**, **UsageIndication**.
url_list	A pointer to the first item in a list of OSP_SERVICE_URL structures, each of which contains a URL that the client may use to obtain the OSP service specified in the 'capability' member. This member may be null.
sig_required	1 indicates that the client should sign requests for the service specified in 'capability' when it sends them to the URLs identified in 'url_list.' This member is otherwise set to zero.

6.11.2 OSP_SERVICE_URL

This structure contains a single URL that points to an OSP server.

```
typedef struct osp_service_url
{
  struct osp_service_url *next;
  char                   *url;
} OSP_SERVICE_URL;
```

Members:

next	A pointer to the next item if this structure is in a list, or a null pointer if there are no further items.
url	A null-terminated string that contains a URL, which identifies a particular OSP server.

6.11.3 OSP_CERTIFICATE

This structure provides data corresponding to the **Certificate** element described in [OSP].

```
typedef struct osp_certificate
{
  struct osp_certificate *next;
  char                   *encoding;
  int                     cert_len;
  char                   *cert_data;
} OSP_CERTIFICATE;
```

Members:

next	A pointer to the next item if this structure is in a list, or a null pointer if there are no further items.
encoding	A null-terminated string that describes the encoding in which the certificate is stored. One of the following: • cdata (XML CDATA format) • base64
cert_len	The size in bytes of the certificate data.
cert_data	A string containing the certificate data.

7 Subscriber Authentication API

The Subscriber Authentication (SA) API allows OpenOSP to request your application to authenticate subscribers and check whether they are entitled to use the facilities of the OSP server (for example to authenticate users with mobile telephones who are calling from outside their usual network, and to verify whether they are permitted to do so).

7.1 osp_sa_register

The application calls this function to register a callback routine that will handle information from OSP **SubscriberAuthenticationRequest** messages. OpenOSP then calls this SA callback routine each time it receives a **SubscriberAuthenticationRequest** from an OSP client.

If the application issues more than one call to this function, the parameter provided in the most recent call overrides those provided in previous calls.

The application may deregister the callback at any time by calling this function with a null pointer in place of the function pointer. After deregistering the callback, the application will not receive any further subscriber authentication requests unless it registers a new SA callback routine.

```
OSP_RC OSPAPI osp_sa_register(POSP_SA_CALLBACK sa_cb);
```

sa_cb	A pointer to the settlement application's SA callback routine. See section 7.2, POSP_SA_CALLBACK, for details of this routine. To deregister the existing callback routine, the application supplies a null pointer.

7.2 POSP_SA_CALLBACK

The application provides this function to handle information from OSP **SubscriberAuthenticationRequest** messages, and registers it during initialization by supplying the address of the function on the osp_sa_register() call.

OpenOSP then calls this function each time it receives a **SubscriberAuthenticationRequest** from an OSP client.

The settlement application must respond to the callback by calling osp_sa_response().

```
OSP_RC (OSPAPI *POSP_SA_CALLBACK) (OSP_CORRELATOR    corr,
                                   char             *timestamp,
                                   OSP_ADDRESS      *source_info,
                                   OSP_ADDRESS      *source_alt_list,
                                   OSP_ADDRESS      *dest_info,
                                   OSP_SERVICE_INFO *service,
                                   OSP_AUTH_INFO    *subs_authen_info,
                                   OSP_CLIENT_ID    *client_id);
```

For details of the data structures OSP_ADDRESS, OSP_SERVICE_INFO, OSP_AUTH_INFO and OSP_CLIENT_ID, see section 3.4, Common data structures and types.

Parameters:

corr	A correlator to be passed back to OpenOSP on the subsequent call to osp_sa_response().
timestamp	A null-terminated string containing the message's timestamp.
source_info	Pointer to a structure containing details about the source of the call, e.g. the originating phone number.
source_alt_list	A pointer to the first item in a list of OSP_ADDRESS structures, each of which contains alternate information about the source of the call, e.g. the address of the source gateway.
dest_info	Pointer to a structure containing details about the destination of the call, e.g. the destination phone number.
service	A pointer to an OSP_SERVICE_INFO describing the type of service requested. A null pointer indicates basic internet telephony service, which is the only type of service currently defined by [OSP].
subs_authen_info	A pointer to an OSP_AUTH_INFO structure containing a subscriber authentication token. This parameter may be NULL, which indicates that the corresponding **SubscriberAuthenticationRequest** did not include a subscriber authentication token.
client_id	A structure containing the client IP address and the 'cookie' from any earlier client verification.

7.3 osp_sa_response

When OpenOSP calls the settlement application's implementation of POSP_SA_CALLBACK(), the application must respond by calling osp_sa_response(). This function uses the information provided to construct a **SubscriberAuthenticationResponse** message, to be sent to the OSP client from which the request originated.

```
OSP_RC OSPAPI osp_sa_response(OSP_CORRELATOR     corr,
                              OSP_STATUS        *status,
                              OSP_TOKEN         *subs_token_list,
                              OSP_CREDIT_AMOUNT *credit_amount,
                              OSP_CREDIT_TIME   *credit_time,
                              unsigned int       sig_required);
```

For details of the data structures OSP_STATUS and OSP_TOKEN, see section 3.4, Common data structures and types.

Parameters:

corr	The correlator supplied by OpenOSP on the corresponding subscriber authentication callback.
status	A pointer to an OSP_STATUS structure that specifies the status code and description that should be used to form the **SubscriberAuthenticationResponse** response.
subs_token_list	A pointer to the first item in a list of OSP_TOKEN structures, each of which contains a subscriber authentication token.
credit_amount	A pointer to an OSP_CREDIT_AMOUNT structure indicating the amount of prepaid credit remaining for the authenticated subscriber.
credit_time	A pointer to an OSP_CREDIT_TIME structure indicating the length of time for which the authenticated prepaid subscriber can maintain a call to the destination that was specified by the corresponding **SubscriberAuthenticationRequest**.
sig_required	Set this parameter to 1 if the server should S/MIME sign the response, 0 if it should not.

7.3.1 OSP_CREDIT_AMOUNT

This structure provides data describing the amount of credit remaining for a prepaid subscriber.

```
typedef struct osp_credit_amount
{
  char               *currency;
  char               *amount;
} OSP_CREDIT_AMOUNT;
```

Members:

currency	A null-terminated string specifying the currency in which the amount is expressed, for example “USD”.
amount	A null-terminated string specifying the amount of credit remaining as a decimal number.

7.3.2 OSP_CREDIT_TIME

This structure provides data describing the call duration that is available for a prepaid subscriber, given the call destination and the subscriber’s remaining credit.

```
typedef struct osp_credit_time
{
  char                  *amount;
  char                  *increment;
  char                  *unit;
} OSP_CREDIT_TIME;
```

Members:

amount	A null-terminated string containing a number which, when multiplied by ‘increment,’ gives the number of units of service available.
increment	A null-terminated string containing a number which, when multiplied by ‘amount,’ gives the number of units of service available.
unit	A null-terminated string containing the units of use by which available credit time is expressed, for example seconds or minutes.

8 Security API

The Security API is implemented by the Crypto and Certificates Manager (CCM). It is used by the OpenOSP stack and the sample application to

- create and verify signatures
- retrieve the server's certificate authority (CA) certificate.

If the OSP server is also acting as a CA for SCEP clients, then the API will also be used to create certificates.

8.1 ccm_init

The OpenOSP stack calls this function to initialize the CCM component. Apart from performing its own internal initialization during this call, CCM also configures various aspects of the SSL library. The SSL library is configured to retrieve certificates using CCM's certificate store and to use the configured certificate, private key, cipher list and client verification mode.

```
OSP_RC ccm_init(SSL_CTX                  *ssl_ctx,
                PCCM_SSL_VERIFY_CALLBACK  ssl_verify_cb);
```

Parameters:

ssl_ctx	A pointer to the SSL_CTX structure to configure.
ssl_verify_cb	A callback function that CCM will call to verify clients that connect via SSL.

8.2 PCCM_SSL_VERIFY_CALLBACK

The OpenOSP stack provides this function to receive callbacks from CCM when a client is attempting to connect via SSL. The callbacks are used to implement part of the client verification functionality that OpenOSP provides to the application. See section 4.5, POSP_CLIENT_VERIFY_CALLBACK, for a description of the client verification facilities that are available to the application.

```
int (*PCCM_SSL_VERIFY_CALLBACK) (SSL          *ssl,
                                 char         *cert,
                                 unsigned int  cert_len,
                                 unsigned int  chain_len,
                                 unsigned int  chain_pos);
```

Parameters:

ssl	A pointer to the SSL connection structure associated with the connection being made.
cert	One of the certificates in the client's certificate chain, in ASN.1 DER-encoded X.509 format.

cert_len	The size in bytes of the certificate data.
chain_len	The number of certificates in the client's certificate chain.
chain_pos	The position of the supplied certificate in the chain, with 0 being the client's certificate.

8.3 ccm_term

The OpenOSP stack calls this function to terminate the CCM component.

```
OSP_RC ccm_term(void);
```

Parameters:

None.

8.4 ccm_pkcs7_sign

OpenOSP uses this function to sign OSP responses, if required by the application, and to sign SCEP responses. The application may also use this function to sign tokens and other data.

The caller supplies a distinguished name that identifies the certificate and private key that are to be used for the signing operation. This distinguished name corresponds to one of the identities set up in the OpenOSP configuration file.

```
OSP_RC ccm_pkcs7_sign(char                     *subj_name,
                      char                     *sdata,
                      unsigned int              sdata_len,
                      unsigned int              digest,
                      STACK_OF(X509_ATTRIBUTE) *auth_attribs,
                      unsigned int              flags,
                      char                     **sig_buf,
                      unsigned int             *sig_len);
```

Parameters:

subj_name	The distinguished name of the identity to sign as.
sdata	A pointer to the data that is to be signed.
sdata_len	The number of bytes to be signed.
digest	The digest algorithm to be used. The following values are valid: • CCM_DIGEST_SHA1: the SHA-1 digest. • CCM_DIGEST_MD5: the MD5 digest. • CCM_DIGEST_DEFAULT: CCM's default digest, which is MD5.

auth_attribs	A list of authenticated attributes to include in the resulting PKCS#7 SignedData structure. The X509_ATTRIBUTE type and the STACK_OF(…) macro are imported from OpenSSL.
flags	Flags that control the behavior of the function. This parameter may be zero, or it may take the following value: • CCM_FLAG_PKCS7_D: the resulting structure should be detached (in other words, it contains no data).
sig_buf	Receives a pointer to a buffer containing the resulting ASN.1 DER-encoded PKCS#7 SignedData structure. The caller must free this memory.
sig_len	Receives the number of bytes in the returned PKCS#7 SignedData structure.

8.5 ccm_pkcs7_verify

OpenOSP uses this function to verify signed OSP requests and SCEP requests. The application may also use this function to verify signatures on tokens and other data.

```
OSP_RC ccm_pkcs7_verify(char                        *p7_asn1,
                        unsigned int                 p7_asn1_len,
                        char                        *indata_buf,
                        unsigned int                 indata_len,
                        unsigned int                 flags,
                        STACK_OF(X509_ATTRIBUTE) **auth_attribs,
                        char                       **ver_buf,
                        unsigned int                *ver_len,
                        char                       **cert_buf,
                        unsigned int                *cert_len);
```

Parameters:

p7_asn1	A pointer to the ASN.1 DER-encoded PKCS#7 SignedData structure that is to be verified.
p7_asn1_len	The number of bytes in the PKCS#7 SignedData structure.
indata_buf	Raw data to be verified (required only if a detached SignedData structure is supplied).
indata_len	The number of bytes of raw data supplied.
flags	Flags that control the behavior of the function. This parameter may be zero, or it may be any combination (logical OR) of the following values: • CCM_FLAG_PKCS7_D: signifies a detached PKCS#7 SignedData structure.

- CCM_FLAG_SELF_SIGNED: allows a single self-signed certificate in the certificate chain, as found in an SCEP PKCSReq message. A single self-signed certificate that is not a CA certificate would normally fail the certificate verification process.

auth_attribs	Receives a list of authenticated attributes from the supplied PKCS#7 SignedData structure. The X509_ATTRIBUTE type and the STACK_OF(…) macro are imported from OpenSSL.
ver_buf	Receives a pointer to a buffer containing the raw data that was verified. The caller must free this memory. This parameter may be a null pointer.
ver_len	Receives the number of bytes of raw data that were verified. This parameter may be a null pointer.
cert_buf	Receives a pointer to a buffer containing the single self-signed certificate, if present. This is in ASN.1 DER-encoded X.509 format, and is returned only if CCM_FLAG_SELF_SIGNED is specified. The caller must free this memory. This parameter may be a null pointer.
cert_len	Receives the number of bytes in the single self-signed certificate, if present. This parameter may be a null pointer.

8.6 ccm_pkcs7_encrypt

OpenOSP uses this function to encrypt SCEP responses. The application may also use this function to encrypt tokens and other data.

The caller supplies a distinguished name that identifies the certificate and private key that are to be used for the encryption operation. This distinguished name corresponds to one of the identities set up in the OpenOSP configuration file.

```
OSP_RC ccm_pkcs7_encrypt(char                   *subj_name,
                         char                   *cert,
                         unsigned int            cert_len,
                         char                   *edata,
                         unsigned int            edata_len,
                         unsigned int            cipher,
                         char                  **enc_buf,
                         unsigned int           *enc_len);
```

Parameters:

subj_name	The distinguished name of the identity to encrypt as.
cert	The certificate to use for encryption, in ASN.1 DER-encoded X.509 format. This is used only if subj_name is a null pointer.

cert_len	The number of bytes in the supplied certificate. This is used only if subj_name is a null pointer.
edata	A pointer to the raw data to be encrypted.
edata_len	The number of bytes to be encrypted.
cipher	The cipher algorithm to be used. The following values are valid: • CCM_CIPHER_DES_CBC: the DES-CBC cipher. • CCM_CIPHER_RC2_CBC: the RC2-CBC cipher.
enc_buf	Receives a pointer to a buffer containing the resulting ASN.1 DER-encoded PKCS#7 EnvelopedData structure. The caller must free this memory.
enc_len	Receives the number of bytes in the returned PKCS#7 EnvelopedData structure.

8.7 ccm_pkcs7_decrypt

OpenOSP uses this function to decrypt SCEP requests. The application may also use this function to decrypt tokens and other data.

```
OSP_RC ccm_pkcs7_decrypt(char          *p7_asn1,
                         unsigned int   p7_asn1_len,
                         char          *subj_name,
                         char         **dec_buf,
                         unsigned int  *dec_len);
```

Parameters:

p7_asn1	A pointer to the ASN.1 DER-encoded PKCS#7 EnvelopedData structure that is to be decrypted.
p7_asn1_len	The number of bytes in the PKCS#7 EnvelopedData structure.
subj_name	The distinguished name of the identity to decrypt as (a PKCS#7 EnvelopedData structure may be encoded to enable decryption by multiple parties).
dec_buf	Receives a pointer to a buffer containing the decrypted data. The caller must free this memory.
dec_len	Receives the number of bytes in the decrypted data.

8.8 ccm_get_cert

This function is called by OpenOSP to obtain certificates from the CCM certificate store. In particular, it is used to obtain the server's certificate authority (CA) certificate, which is returned in response to an SCEP GetCACert request. The application may also use this function to obtain certificates if required.

```
OSP_RC ccm_get_cert(char          *subj_name,
                    char         **cert_buf,
                    unsigned int  *cert_len);
```

Parameters:

subj_name	The distinguished name of the entity whose certificate is to be returned.
cert_buf	Receives a pointer to a buffer containing the required X.509 certificate, in ASN.1 DER-encoded form.
cert_len	Receives the number of bytes in the certificate.

8.9 ccm_get_cert_chain

This function is called by the sample application to obtain the full certificate chain used for token signing. The certificate chain is returned to clients in OSP **CapabilitiesConfirmation** messages.

```
OSP_RC ccm_get_cert_chain(char             *subj_name,
                          OSP_CERTIFICATE **cert_chain);
```

Parameters:

subj_name	The distinguished name of the entity whose certificate is to be returned.
cert_chain	Receives a pointer to a null-terminated list of OSP_CERTIFICATE structures, containing all of the certificates in the specified entity's certificate chain. The caller must free this memory by calling ccm_free_cert_chain().

8.10 ccm_free_cert_chain

This function is called by the sample application to free a certificate chain that was previously obtained from a call to ccm_get_cert_chain().

```
void ccm_free_cert_chain(OSP_CERTIFICATE *chain);
```

Parameters:

chain	A pointer to a null-terminated list of OSP_CERTIFICATE structures that are to be freed.

8.11 ccm_request_new_cert

This function is called by OpenOSP to issue a new certificate on behalf of an SCEP client. CCM does the processing asynchronously and returns the resulting certificate on a callback. As long as ccm_request_new_cert() returns OSP_SUCCESS, the new certificate callback is called regardless of whether or not the certificate was issued successfully.

```
OSP_RC ccm_request_new_cert(char                  *req_asn1,
                            unsigned int           req_asn1_len,
                            char                  *trans_id,
                            char                  *ca_name,
                            PCCM_NEW_CERT_CALLBACK  cert_cb,
                            void                  *corr);
```

Parameters:

req_asn1	A pointer to a buffer that contains a PKCS #10 certificate request, in ASN.1 DER-encoded form.
req_asn1_len	The number of bytes in the request.
trans_id	A SCEP transaction ID, represented as a string of ASCII hex digits. This value is checked against the MD5 digest of the public key in the certificate request and an error is returned if the two do not match. This parameter may be a null pointer if the check is not required.
ca_name	The distinguished name of the certificate authority that is to issue the new certificate.
cert_cb	A callback function that CCM will call when it has completed the process of issuing a new certificate.
corr	A correlator that CCM will pass back when it calls the supplied callback function.

8.12 PCCM_NEW_CERT_CALLBACK

The OpenOSP stack provides this function to receive a callback from CCM when the process of issuing a new certificate is complete. The callback provides the newly-issued certificate and the OpenOSP stack returns it to a waiting SCEP client.

```
int (*PCCM_NEW_CERT_CALLBACK) (void         *corr,
                               char         *x509_asn1,
                               unsigned int  x509_asn1_len);
```

Parameters:

corr	The correlator passed to ccm_request_new_cert().

x509_asn1	If a new certificate was successfully issued, this parameter points to it, in ASN.1 DER-encoded X.509 format. The memory is valid only for the duration of this function call and is freed by CCM. If the certificate was not issued successfully, this parameter is a null pointer.
x509_asn1_len	The length of the newly-issued certificate.

8.13 ccm_random

This function is called by OpenOSP and the sample application to obtain random bytes for various purposes. The OpenOSP sample implementation of this function provides an abstraction from OpenSSL's random number generator.

If a hardware random number generator is available, ccm_random() can be easily modified to access that directly. Alternatively, OpenSSL's RAND API could be redirected to the hardware random number source if this is more convenient.

```
void ccm_random(void *buffer,
                int   buf_len);
```

Parameters:

buffer	A pointer to a buffer that will receive random bytes.
buf_len	The number of bytes of random data to be placed in the supplied buffer.

9 Resource API

The core OpenOSP stack uses the Resource API for memory management and to control thread creation and destruction. The API is implemented by the Resource Manager (RM) and provides functions to

- allocate and free memory for internal data structures
- allocate, reallocate, and free memory for general use
- determine whether a new thread can be created and indicate that a thread is to be ended, giving RM control over the total number of threads in existence.

9.1 rm_init

OpenOSP calls this function to initialize the RM component.

```
OSP_RC rm_init(void);
```

Parameters:

None.

9.2 rm_term

OpenOSP calls this function to terminate the RM component.

```
void rm_term(void);
```

Parameters:

None.

9.3 rm_get_mem

This function provides general purpose memory allocation to the OpenOSP stack and its associated components. If the function succeeds, the return value is a pointer to a memory block of at least the requested size. If the function fails, the return value is a null pointer.

```
void* rm_get_mem(unsigned int size);
```

Parameters:

size	The number of bytes of memory required.

9.4 rm_release_mem

This function allows the OpenOSP stack or one of its associated components to release memory that was obtained with a call to rm_get_mem().

```
OSP_RC rm_release_mem(void *mem);
```

Parameters:

mem	A pointer to the memory that is no longer required.

9.5 rm_realloc_mem

The OpenOSP stack or an external application calls this function to change the amount of memory previously allocated in a call to rm_get_mem().

If the function succeeds, the return value is a pointer to a memory block of at least the specified size. In this case, the new memory block contains a copy of the contents of the original memory block (this is truncated if the new memory block is smaller) and the original block is released automatically. If the function fails, the return value is a null pointer and the original memory block remains valid.

```
void* rm_realloc_mem(void         *mem,
                     unsigned int  new_size);
```

Parameters:

mem	A pointer to the previously-allocated memory.
new_size	The number of bytes required for the new memory block.

9.6 rm_get_structure

The OpenOSP stack calls this function to obtain memory for its internal data structures. If the function succeeds, the return value is a pointer to a memory block of at least the specified size. If the function fails, the return value is a null pointer.

This function allows RM to limit the total number of structures of each type that are in existence, if required.

```
void* rm_get_structure(unsigned int size,
                       unsigned int type);
```

Parameters:

size	The number of bytes of memory required.
type	The type of internal structure for which the memory will be used. The following values are valid: • RM_THREAD_INFO: a thread information structure; • RM_CONN_INFO: a connection information structure; • RM_TRANS_INFO: a transaction information structure.

9.7 rm_release_structure

The OpenOSP stack calls this function to release memory that was obtained with a call to rm_get_structure().

```
OSP_RC rm_release_structure(void         *mem,
                            unsigned int  type);
```

Parameters:

mem	A pointer to the memory that is no longer required.
type	The type of internal structure that is being freed. Any of the 'type' values specified for rm_get_structure() are valid here.

9.8 rm_request_thread_create

The OpenOSP stack calls this function to request permission from RM to create a new thread. This allows RM to limit the total number of threads of each type that are in existence, if required.

```
OSP_RC rm_request_thread_create(int type);
```

Parameters:

type	The type of thread that the caller wishes to create. The following values are valid: • RM_WORKER_THREAD: an internal OpenOSP thread; • RM_LAM_THREAD: an internal LDAP access manager (LAM) thread.

9.9 rm_notify_thread_exit

The OpenOSP stack calls this function to notify RM that a thread is about to exit. This allows RM to keep its thread counts up-to-date.

```
OSP_RC rm_notify_thread_exit(int type);
```

Parameters:

type	The type of thread that is about to exit. Any of the 'type' values specified for rm_request_thread_create() are valid here.

10 Secure Sockets Layer (SSL) API

The SSL API is used to communicate with OSP clients using SSL or TLS. It is a subset of the API defined by the OpenSSL open source SSL / TLS implementation. If you intend to use a different SSL / TLS implementation with OpenOSP, the implementation must provide this API.

Unless otherwise indicated, all of the SSL library functions return 1 on success and 0 on failure.

10.1 Configuration functions

This section describes the SSL API functions that OpenOSP uses during initialization and termination to set up the SSL library.

10.1.1 SSL_library_init

This function initializes the SSL library.

```
int SSL_library_init(void);
```

Parameters:

None.

10.1.2 SSL_load_error_strings

This function instructs the library to load into memory a set of plain-text descriptions of error codes that ERR_print_errors_fp() can use later.

```
void SSL_load_error_strings(void);
```

Parameters:

None.

10.1.3 CRYPTO_set_mem_functions

This function sets the allocation, reallocation and freeing functions that the SSL library should use for memory management. If this function is not called, the SSL library should default to using the standard C functions malloc(), realloc() and free().

```
void CRYPTO_set_mem_functions(char *(*malloc_fn)(),
                              char *(*realloc_fn)(),
                              void  (*free_fn)());
```

Parameters:

malloc_fn	A pointer to the replacement function for malloc(). The prototype is the same as that of the standard C routine.

realloc_fn	A pointer to the replacement function for realloc(). The prototype is the same as that of the standard C routine.
free_fn	A pointer to the replacement function for free(). The prototype is the same as that of the standard C routine.

10.1.4 CRYPTO_set_id_callback

This function sets a callback that the SSL library should use to find the current thread ID.

```
void CRYPTO_set_id_callback(unsigned long (*func)(void));
```

Parameters:

func	The callback function. This takes no parameters and returns the current thread ID.

10.1.5 CRYPTO_set_locking_callback

This function sets a callback that the SSL library should use to lock memory accesses in a multithreaded environment. The SSL library should define CRYPTO_NUM_LOCKS to be the number of locks that it requires so that OpenOSP may initialize them at startup.

```
void CRYPTO_set_locking_callback(void (*func)(int mode, int type, const char
*file, int line));
```

Parameters:

func	The callback function. This takes the following parameters: • mode: this is set to CRYPTO_LOCK if the SSL library wishes to lock the specified lock; CRYPTO_UNLOCK otherwise. • type: a zero-based index identifying the lock to manipulate. • file: a null-terminated string containing the name of the file in which the caller's source code resides (for debug purposes only). • line: the line number on which the caller's source code resides (for debug purposes only).

10.1.6 SSLv23_server_method

This function returns a library-defined pointer to a set of 'methods' that may be passed to SSL_CTX_new(). These methods define the SSL library's internal functions that are to be used for handshaking and data transfer using SSLv3 and TLSv1.

```
SSL_METHOD *SSLv23_server_method(void);
```

Parameters:

None.

10.1.7 SSL_CTX_new

This function returns a pointer to a new (library-defined) SSL_CTX structure. If the function fails, the return value is a null pointer.

Only one SSL_CTX structure is created by OpenOSP; it contains the SSL / TLS session ID cache and is also used as a template for new SSL structures created with SSL_new().

```
SSL_CTX *SSL_CTX_new(SSL_METHOD *meth);
```

Parameters:

meth	A pointer to a method structure, obtained from SSLv23_server_method().

10.1.8 SSL_CTX_free

This function frees all the memory associated with an SSL_CTX structure.

```
void SSL_CTX_free(SSL_CTX *ctx);
```

Parameters:

ctx	A pointer to the SSL_CTX structure to be freed.

10.1.9 SSL_CTX_set_options

This functions allows various options to be set that affect the operation of connections based on a particular SSL_CTX structure. The return value is the new state of the options flags.

```
long SSL_CTX_set_options(SSL_CTX *ctx, long larg);
```

Parameters:

ctx	A pointer to the SSL_CTX structure on which to operate.
larg	The options to set. This may be zero, or it may take the following value: • SSL_OP_NO_SSLv2 disables SSLv2 support

10.1.10 SSL_CTX_sess_set_cache_size

This function sets the size of the session cache for a particular SSL_CTX structure. The session cache is used by the SSL library to implement SSL / TLS session re-use. The return value is the previous size of the session cache.

```
long SSL_CTX_sess_set_cache_size (SSL_CTX *ctx,
                                  long     larg);
```

Parameters:

ctx	A pointer to the SSL_CTX structure on which to operate.
larg	The maximum number of sessions allowed in the cache.

10.1.11 SSL_CTX_set_cipher_list

This function sets the list of ciphers that the SSL library should use for connections based on a particular SSL_CTX structure.

```
int SSL_CTX_set_cipher_list(SSL_CTX *ctx,
                            char    *str);
```

Parameters:

ctx	A pointer to the SSL_CTX structure on which to operate.
str	A null-terminated string containing a colon-separated list of cipher descriptions, in decreasing order of preference. Each cipher description defines a paticular combination of key exchange method, type of certificates, encryption method and type of message authentication code (MAC). The following cipher descriptions are valid: • DES-CBC-SHA: unlimited-bit RSA key exchange, RSA certificates, 56-bit DES encryption in CBC mode and a SHA-1 MAC. • EXP-DES-CBC-SHA: 512-bit RSA key exchange, RSA certificates, 40-bit DES encryption in CBC mode and a SHA-1 MAC. • EDH-DSS-DES-CBC-SHA: unlimited-bit ephemeral Diffie-Hellman key exchange, DSA certificates, 56-bit DES encryption in CBC mode and a SHA-1 MAC. • EXP-EDH-DSS-DES-CBC-SHA: 512-bit ephemeral Diffie-Hellman key exchange, DSA certificates, 40-bit DES encryption in CBC mode and a SHA-1 MAC. • NULL-MD5: unlimited-bit RSA key exchange, RSA certificates, no encryption and an MD5 MAC.

- NULL-SHA: unlimited-bit RSA key exchange, RSA certificates, no encryption and a SHA-1 MAC.

10.1.12 SSL_CTX_use_PrivateKey

This function instructs the SSL library to use the private key contained in the specified file for connections based on the specified SSL_CTX structure.

```
int SSL_CTX_use_PrivateKey_file(SSL_CTX    *ctx,
                                EVP_PKEY   *pkey);
```

Parameters:

ctx	A pointer to the SSL_CTX structure on which to operate.
pkey	The private key to use. The EVP_PKEY type is defined by OpenSSL.

10.1.13 SSL_CTX_use_certificate

This function instructs the SSL library to use the X.509 certificate contained in the specified file for connections based on the specified SSL_CTX structure.

```
int SSL_CTX_use_certificate_file(SSL_CTX *ctx,
                                 X509    *x);
```

Parameters:

ctx	A pointer to the SSL_CTX structure on which to operate.
x	The certificate to use. The X509 type is defined by OpenSSL.

10.1.14 SSL_CTX_set_verify

OpenOSP calls this function to set the certificate verification mode and specify a callback routine that the SSL library will call to verify each certificate during the SSL / TLS handshake.

```
void SSL_CTX_set_verify(SSL_CTX *ctx,
                        int      mode,
                        int (*callback)(int ok, X509_STORE_CTX *store_ctx));
```

Parameters:

ctx	A pointer to the SSL_CTX structure on which to operate.
mode	The certificate verification mode to use. The following values are valid: • SSL_VERIFY_NONE: no verification is performed. • SSL_VERIFY_PEER: verification is performed.

If SSL_VERIFY_PEER is used, it may also be ORed with one or both of the following values:

- SSL_VERIFY_FAIL_IF_NO_PEER_CERT: the verification will fail if the client does not present a certificate.
- SSL_VERIFY_CLIENT_ONCE: the server will not ask for the client's certificate if a previous session is re-used.

callback — A pointer to a callback function that the SSL library will call with details of each certificate received from a client during the SSL / TLS handshake. Note that this function definition is specific to OpenSSL, and its implementation within OpenOSP will need to be changed appropriately if another SSL library is used. The parameters are as follows:

- ok: 1 indicates that the SSL library has determined that the certificate is consistent and within-date. 0 indicates that the SSL library has identified a problem with the certificate.
- store_ctx: a pointer to a certificate store. The type of this parameter, X509_STORE_CTX, is defined by the SSL library; please refer to the OpenSSL code for details on what it contains.

The callback function returns 1 if it accepts the certificate and 0 otherwise.

10.1.15 SSL_CTX_set_cert_store

This function instructs the SSL library to use the X.509 certificate store specified for connections based on the specified SSL_CTX structure.

```
void SSL_CTX_set_cert_store(SSL_CTX *ctx,
                            X509_STORE *store);
```

Parameters:

ctx — A pointer to the SSL_CTX structure on which to operate.

x — The store to use. The X509_STORE type is defined by OpenSSL.

10.1.16 SSL_CTX_set_ session_id_context

This function sets the session ID context for connections based on the specified SSL_CTX structure.

```
int SSL_CTX_set_session_id_context(SSL_CTX *ctx,
                                   const unsigned char *sid_ctx,
                                   unsigned int sid_ctx_len);
```

Parameters:

ctx	A pointer to the SSL_CTX structure on which to operate.
sid_ctx	The session ID context.
sid_ctx_len	The size of the context in bytes.

10.1.17 SSL_CTX_set_ tmp_dh_callback

This function sets the callback for generating temporary DH keys for the specified SSL_CTX structure.

```
void SSL_CTX_set_tmp_dh_callback(SSL_CTX *ctx,
                                 DH *(*cb)(SSL *ssl,
                                           int is_export,
                                           int keylength) );
```

Parameters:

ctx	A pointer to the SSL_CTX structure on which to operate.
cb	The callback function.

10.1.18 SSL_CTX_set_ tmp_rsa_callback

This function sets the callback for generating temporary RSA keys for the specified SSL_CTX structure.

```
void SSL_CTX_set_tmp_dh_callback(SSL_CTX *ctx,
                                 RSA *(*cb)(SSL *ssl,
                                           int is_export,
                                           int keylength) );
```

Parameters:

ctx	A pointer to the SSL_CTX structure on which to operate.
cb	The callback function.

10.1.19 RAND_set_rand_method

This function allows the caller to specify an alternate set of random number functions, e.g. to support an external random number generator. Note that the supplied RAND_bytes() replacement function must never fail, i.e. if there is a failure in the external RNG, then it should call onto a backup such as the default OpenSSL RNG function.

```
void RAND_set_rand_method(RAND_METHOD *meth);
```

Parameters:

meth	Pointer to the structure containing the new RNG functions. The type of this parameter is defined by the SSL library.

10.1.20 RAND_load_file

This function loads the random state information from file.

```
int RAND_load_file(const char *file_name,
                   long bytes_to_read);
```

Parameters:

file_name	Name of the random state file.
bytes_to_read	Number of bytes to read from the file, or –1 for the entire file.

10.1.21 RAND_write_file

This function writes the random state information to file.

```
int RAND_write_file(const char *file_name);
```

Parameters:

file_name	Name of the random state file.

10.2 Operational functions

This section describes the SSL API functions that OpenOSP uses during normal operation. These include functions that create a new SSL connection context and implement SSL / TLS handshaking and data transfer.

10.2.1 SSL_new

This function returns a pointer to a new (library-defined) SSL structure, using the supplied SSL_CTX structure as a template. If the function fails, the return value is a null pointer.

An SSL structure contains all of the SSL library's state information relating to a single SSL / TLS connection.

```
SSL *SSL_new(SSL_CTX *ctx);
```

Parameters:

ctx	A pointer to the SSL_CTX structure to use as a template.

10.2.2 SSL_free

This function frees all the memory associated with an SSL structure.

```
void SSL_free(SSL *ssl);
```

Parameters:

ssl	A pointer to the SSL structure to be freed.

10.2.3 SSL_set_fd

This function associates an SSL structure with a socket. All data sent or received using this SSL structure will go via the specified socket.

```
int SSL_set_fd(SSL *s,
               int  fd);
```

Parameters:

ssl	A pointer to the SSL structure.
fd	The file descriptor of the socket to be associated with the SSL structure.

10.2.4 SSL_accept

This function initiates the SSL / TLS handshake. If the handshake is completed successfully, this function returns 1; if there was an error, it returns –1. For a non-blocking socket, the handshake does not complete immediately. In this case, SSL_accept() returns 0 and OpenOSP then calls SSL_get_error() to find out what to do next.

```
int SSL_accept(SSL *ssl);
```

Parameters:

ssl	A pointer to the SSL structure on which to operate.

10.2.5 SSL_get_error

This function provides information on what OpenOSP should do after calling SSL_accept(), SSL_read() or SSL_write().

```
int SSL_get_error(SSL *ssl,
                  int  ret_code);
```

Parameters:

ssl	A pointer to the SSL structure on which to operate.
ret_code	The return code obtained from SSL_accept(), SSL_read() or SSL_write().

Return codes:

SSL_ERROR_NONE	The operation completed successfully (only possible if 'ret_code' is greater than zero).
SSL_ERROR_ZERO_RETURN	The connection was closed cleanly.
SSL_ERROR_WANT_READ	The operation did not complete and OpenOSP should select() on the associated socket for a read operation.
SSL_ERROR_WANT_WRITE	The operation did not complete and OpenOSP should select() on the associated socket for a write operation.
SSL_ERROR_SYSCALL	An I/O error occurred.
SSL_ERROR_SSL	An SSL protocol error occurred.

10.2.6 SSL_write

This function sends data over the SSL / TLS connection associated with the specified SSL structure. After calling this function, OpenOSP calls SSL_get_error() to find out what to do next.

```
int SSL_write(SSL        *ssl,
              const char *buf,
              int         num);
```

Parameters:

ssl	A pointer to the SSL structure on which to operate.
buf	A pointer to the data to be sent.
num	The number of bytes to be sent.

10.2.7 SSL_read

This function receives data from the SSL / TLS connection associated with the specified SSL structure. After calling this function, OpenOSP calls SSL_get_error() to find out what to do next. If the return value is greater than zero, it indicates the number of bytes that have been received into the supplied buffer.

```
int SSL_read(SSL  *ssl,
             char *buf,
             int   num);
```

Parameters:

ssl	A pointer to the SSL structure on which to operate.
buf	A pointer to the buffer into which received data should be placed.

num — The number of bytes available in the supplied buffer.

10.2.8 SSL_renegotiate

This function initiates a renegotiation on the SSL / TLS connection associated with the specified SSL structure. The renegotiation does not necessarily complete immediately, but subsequent calls to SSL_read() and SSL_write() will automatically send or receive outstanding renegotiation messages until the renegotiation is complete.

```
int SSL_renegotiate(SSL *ssl);
```

Parameters:

ssl — A pointer to the SSL structure on which to operate.

10.2.9 SSL_shutdown

This function sends a notification to the client that the SSL / TLS connection associated with the specified SSL structure is about to go down. A return value of 0 indicates that further calls to SSL_shutdown() are required.

```
int SSL_shutdown(SSL *ssl);
```

Parameters:

ssl — A pointer to the SSL structure on which to operate.

10.2.10 SSL_set_app_data

This function allows OpenOSP to associate arbitrary data with an SSL structure.

```
int SSL_set_app_data(SSL *ssl,
                     void *data);
```

Parameters:

ssl — A pointer to the SSL structure on which to operate.

data — An OpenOSP-defined value.

10.2.11 SSL_get_app_data

This function allows OpenOSP to retrieve the data that it previously associated with an SSL structure using SSL_set_app_data(). The return value is the data.

```
void *SSL_get_app_data(SSL *ssl);
```

Parameters:

ssl — A pointer to the SSL structure on which to operate.

10.2.12 ERR_print_errors_fp

This function prints error information to the specified file in a format defined by the SSL library. OpenOSP calls this function when an SSL library call returns an error code.

```
void ERR_print_errors_fp(FILE *fp);
```

Parameters:

fp	A file pointer that identifies the file to which error information should be printed.

10.2.13 RAND_bytes

This function returns the requested number of random bytes.

```
int RAND_bytes(unsigned char *buf,
               int num);
```

Parameters:

buf	buffer to receive the random data.
num	Number of bytes to return.

References

The following references provide further information on relevant subjects:

PO	OpenOSP Product Overview, version 1.2 Data Connection Limited, September 2000 MSM-0005-0102
OSP	Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON); Open Settlement Protocol (OSP) for Inter-Domain pricing, authorization and usage exchange. ETSI TS 101 321 V2.1.0 (2000-05).
SCEP	Cisco Systems' Simple Certificate Enrollment Protocol (SCEP) Cisco Systems, August 2000 http://search.ietf.org/internet-drafts/draft-nourse-scep-03.txt
SSL	The SSL Protocol Version 3.0 Netscape Communications Corporation, November 1996 http://oem.netscape.com/eng/ssl3/draft302.txt
TLS	RFC 2246: The TLS Protocol Version 1.0 T. Dierks and C. Allen, January 1999 http://www.ietf.org/rfc/rfc2246.txt
HTTP/1.0	RFC 1945: Hypertext Transfer Protocol – HTTP/1.0 T. Berners Lee, R. Fielding and H. Frystyk, May 1996 http://www.ietf.org/rfc/rfc1945.txt
HTTP/1.1	RFC 2616: Hypertext Transfer Protocol – HTTP/1.1 R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee, June 1999 http://www.ietf.org/rfc/rfc2616.txt
S/MIME	RFC 2311: S/MIME Version 2 Message Specification S. Dusse, P. Hoffman, B. Ramsdell, L. Lundblade and L. Repka, March 1998 http://www.ietf.org/rfc/rfc2311.txt
XML	Extensible Markup Language (XML) 1.0 W3C, February 1998 http://www.w3.org/TR/REC-xml
PKCS #7	PKCS #7: Cryptographic Message Syntax Standard (version 1.5) RSA Laboratories, November 1993 http://www.rsasecurity.com/rsalabs/pkcs/pkcs-7/index.html
PKCS #10	PKCS #10: Certification Request Syntax Standard (version 1.0) RSA Laboratories, November 1993 http://www.rsasecurity.com/rsalabs/pkcs/pkcs-10/index.html