

SDPⁱ Parser Design Specification

Girish Moodalbail
network-sip-discuss@opensolaris.org

Revision 1.4
Sep 19, 2007

ⁱ Session Description Protocol – RFC 4566

Contents

SDP Parser Design Specification.....	1
1 Introduction.....	4
2 Requirements	5
3 Architectural Overview	5
4 SDP fields & corresponding data structures.....	8
4.1 Protocol Version (“v=”).....	9
4.2 Origin (“o=”).....	9
4.3 Session Name (“s=”).....	9
4.4 Session Information (“i=”).....*	9
4.5 URI (“u=”).....*	10
4.6 Email Address (“e=”).....*	10
4.7 Phone Number (“p=”).....*	10
4.8 Connection Data (“c=”).....	10
4.9 Bandwidth (“b=”).....*	11
4.10 Timing (“t=”).....	11
4.11 Repeat time (“r=”).....*	12
4.12 Time Zones (“z=”).....*	12
4.13 Encryption Keys (“k=”).....*	13
4.14 Attributes (“a=”).....	13
4.15 Media Descriptions (“m=”).....	14
5 Functions to generate SDP description	14
6 Public functions	15
6.1 sdp_find_media()	15
6.2 sdp_find_attr().....	15
6.3 sdp_get_media_rtpmap().....	16
6.4 sdp_clone_session().....	16
6.5 sdp_delete_all_field().....	16
6.6 sdp_delete_all_media_field().....	17
6.7 sdp_delete_attribute().....	17
6.8 sdp_delete_media()	17
6.9 sdp_new_session().....	18
6.10 sdp_add_origin()	18
6.11 sdp_add_name()	18
6.12 sdp_add_information()	19
6.13 sdp_add_uri()	19
6.14 sdp_add_email()	19
6.15 sdp_add_phone()	20
6.16 sdp_add_connection().....	20
6.17 sdp_add_bandwidth()	20
6.18 sdp_add_repeat()	21
6.19 sdp_add_time().....	21
6.20 sdp_add_zone()	21
6.21 sdp_add_key().....	22
6.22 sdp_add_attribute().....	22
6.23 sdp_add_media()	22

6.24	<code>sdp_session_to_str()</code>	23
6.25	<code>sdp_free_session()</code>	23
7	Impact on <code>sip.h</code> & <code>libsip.so.1</code>	23
8	Appendix	23
8.1	<code><sdp.h></code>	23
8.2	Example usage scenario ()	28
8.2.1	Normal case	28
8.2.2	SIP usecase	31
9	Other open source SDP library	33
10	References	33

1 Introduction

Session Description Protocol (SDP), as the name suggests, is intended for describing multimedia sessions for the purposes of session announcement, session invitation, and other forms of multimedia session initiation. SDP conveys media details like type (audio, video, etc.) and encoding (MPEG video, DivX video, etc), transport protocol (RTP/UDP/IP, H.320, etc.), addresses and ports, session name, purpose and owner and other session description metadata to the participants. However it is not intended to support negotiation of session content or media encodings.

An SDP session description uses text encoding and consists of a number of lines of text of the form:

```
<type>=<value>
```

where <type> is a single lower-case letter and <value> is structured text whose format depends on <type>. In general, <value> is either a number of fields delimited by a single space character or a free format string, and is case-significant unless a specific field defines otherwise. To simplify parsing the lines of text need to be in a particular order. An example SDP description, from RFC 4566, is as shown below

```
v=0
o=jdoe 2890844526 2890842807 IN IP4 10.47.16.5
s=SDP Seminar
i=A Seminar on the session description protocol
u=http://www.example.com/seminars/sdp.pdf
e=j.doe@example.com (Jane Doe)
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
a=recvonly
m=audio 49170 RTP/AVP 0
m=video 51372 RTP/AVP 99
a=rtpmap:99 h263-1998/90000
```

SDP is used to convey session information in Session Initiation Protocol (SIP), Streaming Media (Real Time Streaming Protocol, RTSP), Email and World Wide Web and Multicast Session Announcement.

In this project we provide public interfaces that parses the SDP description and checks for syntax conformance (as defined in the Section 9 (SDP Grammar) of RFC 4566). Public interfaces are also provided to generate SDP and convert it to byte-string, which will be used as a payload in the aforementioned protocols.

SDP is used predominately by SIP. The Session Initiation Protocol (SIP) is an application-layer control protocol for creating, modifying, and terminating sessions such as Internet multimedia conferences, Internet telephone calls, and multimedia distribution.

The SIP messages used to create sessions carry session descriptions that allow participants to agree on a set of compatible media types. These session descriptions are commonly formatted using SDP.

The current Solaris SIP library does not provide any support for parsing, generating and syntax checking of the SDP info inside SIP messages. The application retrieves the SDP info from SIP messages using `sip_get_content()`, parses it and checks for the syntax itself. Further to generate SDP the application has to create a buffer filled with SDP info and call `sip_add_content()` to add the info to a SIP message.

With this library Solaris SIP developers can leverage these interfaces in developing SIP apps. This would also expedite the acceptance of our SIP stack in the community as other open source SIP stack already provides such functionality.

2 Requirements

The technical requirements for the SDP parser are as described below:

- 1) Must parse SDP description as per the RFC 4566.

To aid application developers developing SIP application and other related applications which use SDP, we need to provide API's to parse the SDP description into various fields as described in RFC 4566.

- 2) Must check for malformed SDP description.

SDP description may be transported via very unreliable means or damaged by an intermediate caching server, so the design should take care of identifying malformed SDP descriptions by checking for strict order of fields and formatting rules.

- 3) Must provide API's to generate SDP description to be later used in applications like SIP or other applications (RTSP, Multicast session announcement), which use SDP.
- 4) Must be generic enough that all the applications, which use SDP, benefit from these API's, i.e. it should not be tied down to SIP.

3 Architectural Overview

An application developer would pass raw byte array, which represents SDP description, as input to the API shown below. For example: In the case of SIP, after ensuring that a SIP message contains SDP info (by checking the content-type for "application/sdp"), he/she would extract that info from the message body (using `sip_get_content()`) and pass it to the API below.

```
int sdp_parse(const char *sdp_info, int len, int flags, sdp_session_t
**session, int *p_error);
```

The above API parses the `sdp_info` and populates the structure `sdp_session_t` (see below) with all the SDP field information. It internally allocates memory for each individual field (either a structure or char buffer) and sub-fields (mostly char buffer) within a field. An API `sdp_free_session(sdp_session_t *)` will be provided for freeing up the memory allocated to entire `sdp_session_t` structure.

The above API while parsing `sdp_info`, checks for the strict order of fields, checks for missing mandatory fields and checks for strict formatting rules (all defined in the section 9 (SDP Grammar) of RFC 4566). Any aforementioned errors will be recorded in `p_error`. It also identifies the field(s) that had parsing error by setting a bit in `p_error`. Possible values for `p_error` are:

SDP_VERSION_ERROR	0x00000001
SDP_ORIGIN_ERROR	0x00000002
SDP_NAME_ERROR	0x00000004
SDP_INFO_ERROR	0x00000008
SDP_URI_ERROR	0x00000010
SDP_EMAIL_ERROR	0x00000020
SDP_PHONE_ERROR	0x00000040
SDP_CONNECTION_ERROR	0x00000080
SDP_BANDWIDTH_ERROR	0x00000100
SDP_TIME_ERROR	0x00000200
SDP_REPEAT_TIME_ERROR	0x00000400
SDP_ZONE_ERROR	0x00000800
SDP_KEY_ERROR	0x00001000
SDP_ATTRIBUTE_ERROR	0x00002000
SDP_MEDIA_ERROR	0x00004000
SDP_FIELDS_ORDER_ERROR	0x00008000
SDP_MISSING_FIELDS	0x00010000

If there are no errors while parsing, `p_error` will be set to “0”. Further if an error occurs while parsing a SDP field, we mark that field to have parsing error and continue parsing of remaining fields. That is we don’t exit from parsing. This is where the “flags” comes into picture. The argument `flags` can be used to control parser behavior.

- Strict parsing (current behavior, RFC mandates 0x20 (space) between sub-fields). For e.g.: `o=<username> <SP> <id> <SP> <version> <SP>`
- Loose parsing (Consider any white spaces between sub-fields, ignore trailing and leading white spaces; FUTURE)
- Early Exit: (current behavior is to continue parsing, even if one of the SDP fields has parsing error, with this flag we can exit parsing as soon as we hit error; FUTURE)

SDP session structure is as defined below. Each member of the structure map to a unique SDP field.

```
typedef struct sdp_session {
    int            sdp_session_version; /* SDP session version */
    int            s_version;           /* SDP version field */
    sdp_origin_t  *s_origin;           /* SDP origin field */
    char          *s_name;              /* SDP name field */
    char          *s_info;              /* SDP info field */
    char          *s_uri;               /* SDP uri field */
    sdp_list_t    *s_email;            /* SDP email field */
    sdp_list_t    *s_phone;            /* SDP phone field */
    sdp_conn_t    *s_conn;             /* SDP connection field */
    sdp_bandwidth_t *s_bw;             /* SDP bandwidth field */
    sdp_time_t    *s_time;             /* SDP time field */
    sdp_zone_t    *s_zone;             /* SDP zone field */
    sdp_key_t     *s_key;              /* SDP key field */
    sdp_attr_t    *s_attr;             /* SDP attribute field */
    sdp_media_t   *s_media;            /* SDP media field */
} sdp_session_t;
```

The session structure (`sdp_session_t`) contains entire SDP description in various structures (explained in next section). This structure is public and the individual members can be directly accessed/modified. Of course, one has to be careful while performing modification as they can easily overflow buffer. Best is to delete the entire field and add a new field. (API's provided for that)

Now for the sake of the example, if the passed `sdp_info` has just one field, say,

```
o=jdoe 23423423 234234234 IN IP4 192.168.1.1\r\n
```

then the `s_origin` (`sdp_origin_t`) in session (`dp_session_t`) will be;

```
session {
    sdp_session_version = 1
    s_version = 0
    s_origin {
        o_username = "jdoe"
        o_id = 23423423ULL
        o_ver = 234234234ULL
        o_nettype = "IN"
        o_addrtype = "IP4"
        o_address = "192.168.1.1"
    }
    s_name = (nil)
    s_info = (nil)
    ...
    (omitted info)
}
```

Application uses the above structure as required i.e. retrieves the values from the structures et al. To send a response back to the received SDP description, the developer could clone the above structure, modify it accordingly (i.e. delete and add fields), convert it to byte-string (`sdp_session_to_str()`) and send it across as response.

To generate new SDP info, the developer gets a pointer to allocated session structure (`sdp_session_t`) using `sdp_new_session()` and populates the structure by calling field specific `sdp_add_<fieldname>(sdp_session_t, <field specific arguments>)` functions. Once the structure is populated he/she would pass that structure to `sdp_session_to_str()` which returns a pointer to a character buffer holding SDP information. In the case of SIP he/she would then add this character buffer into SIP message using `sip_add_content(sip_msg_t, char *)`. Then he needs to set the content type to "application/sdp". Check section 8.1 for `sdp.h`. Also section 8.2 contains an example, which captures a use case.

4 SDP fields & corresponding data structures

Following section describes each SDP field, format of the field and the data structure associated with the field. These data structures are public and will be defined in `sdp.h` header file. Each of these structures corresponds to a particular field in SDP description. For more information on the individual fields please refer to RFC 4566. The "*" next to field name indicates that the field is OPTIONAL in a SDP description.

Note: The design is based on RFC 4566. For eg: As per RFC,

(i) "The set of <type> letters is deliberately small and not intended to be extensible -- an SDP parser MUST completely ignore any session description that contains a type letter that it does not understand. The attribute mechanism ("a=") is the primary means for extending SDP and tailoring it to particular applications or media."

This means that the session structure (`sdp_session_t`) is pretty much fixed with members representing <type>'s from the standard and the attribute structure has been designed to be a linked list to hold extended attributes.

(ii) the <value> part of <type>=<value>, is a structured text whose format depends on <type>. These values are again defined to be different structures (`sdp_origin_t`, `sdp_conn_t`, `sdp_time_t`, `sdp_bandwidth_t`, `sdp_attr_t`, et al) which will be part of structure `sdp_session_t`.

4.1 Protocol Version ("v=")

Format: v=<number>

Description: The "v=" field gives the version of the SDP. The structure `sdp_session_t` holds the value of this field in its "s_version" member. It's defined to be an integer.

4.2 Origin ("o=")

Format: o=<username> <sess-id> <sess-version> <nettype> <addrtype>
<unicast-address>

Structure:

```
typedef struct sdp_origin {
    char      *o_username;
    uint64_t  o_id;
    uint64_t  o_version;
    char      *o_nettype;
    char      *o_addrtype;
    char      *o_address;
} sdp_origin_t;
```

"o_username" holds the username of the originating host

"o_id" holds the session id

"o_version" holds the version number of this session description

"o_nettype" holds type of network

"o_addrtype" holds type of the address

"o_address" holds the address of the machine from which session was created.

4.3 Session Name ("s=")

Format: s=<session name>

Description: The "s=" field is the textual session name.

Data type: char *

4.4 Session Information ("i=")*

Format: i=<session description>

Description: The "i=" field provides the textual information about the session.

Data type: char *

4.5 URI (“u=”)*

Format: u=<uri>

Description: The "u=" field points to additional information about the session through URI.

Data type: char *

4.6 Email Address (“e=”)*

Format: e=<email-address>

Description: The "e=" field specify contact information for the person responsible for the conference. Since there could be several such fields, we define the structure to be a linked list.

Structure:

```
typedef struct sdp_list {
    void                *value;
    struct sdp_list     *next;
} sdp_list_t;
```

4.7 Phone Number (“p=”)*

Format: p=<phone-number>

Description: The "p=" field specify contact information for the person responsible for the conference. Since there could be several such fields, we define the structure to be a linked list.

Structure:

```
typedef struct sdp_list {
    void                *value;
    struct sdp_list     *next;
} sdp_list_t;
```

4.8 Connection Data (“c=”)

Format: c=<nettype> <addrtype> <connection-address>

Description: The "c=" field contains connection data. Since there could be several such fields, we define the structure to be a linked list

Structure:

```

typedef struct sdp_conn {
    char          *c_nettype;
    char          *c_addrtype;
    char          *c_address;
    int           c_addrcount;
    struct sdp_conn *c_next;
    uint8_t       c_ttl;
} sdp_conn_t;

```

"c_nettype" holds type of network

"c_addrtype" holds type of the address

"c_address" holds unicast-address or multicast address

"c_addrcount" holds number of addresses (case of multicast address with layered encodings)

"c_ttl" holds TTL value for IPV4 multicast address

"c_next" pointer to next connection structure as there could be several connection fields in SDP description

4.9 Bandwidth ("b=")*

Format: b=<bwtype>:<bandwidth>

Description: The "b=" field denotes the proposed bandwidth to be used by the session or media. Since there could be multiple such fields we define the structure to be a linked list.

Structure:

```

typedef struct sdp_bandwidth {
    char          *b_type;
    uint64_t       b_value;
    struct sdp_bandwidth *b_next;
} sdp_bandwidth_t;

```

"b_type" holds info needed to interpret "b_value"

"b_value" holds bandwidth value

"b_next" pointer to next bandwidth structure as there could be several bandwidth fields in SDP description

4.10 Timing ("t=")

Format: t=<start-time> <stop-time>

Description: The "t=" lines specify the start and stop times for a session. Since there could be multiple such fields we define the structure to be a linked list.

Structure:

```

typedef struct sdp_time {
    uint64_t      t_start;
    uint64_t      t_stop;
    sdp_repeat_t  *t_repeat;
    struct sdp_time *t_next;
} sdp_time_t;

```

"t_start" holds the start-time for a session

"t_end" holds the end-time for a session

"t_repeat" points to the SDP repeat field (see sdp_repeat_t)

"t_next" is a pointer to next time field as there could be several time fields in SDP description

4.11 Repeat time ("r=")*

Format: r=<repeat interval> <active duration> <offsets from start-time>

Description: The "r=" fields specify repeat times for a session. Since there could be multiple fields we define the structure to be a linked list.

Structure:

```

typedef struct sdp_repeat {
    uint64_t      r_interval;
    uint64_t      r_duration;
    sdp_list_t    *r_offset;
    struct sdp_repeat *r_next;
} sdp_repeat_t;

```

"r_interval" holds the repeat interval, for eg: 86400 meaning every 1 day

"r_duration" holds the duration of session, for eg: 3600 meaning 1 hour.

"r_offset" is a linked list of "offset" values and each value represents offset from <start-time> in SDP time field.

"r_next" pointer to next repeat structure as there could be several repeat fields in SDP description.

This structure will always be part of the time structure (sdp_time_t) as repeat field does not appear alone in SDP description and is always associated with time field.

4.12 Time Zones ("z=")*

Format: z=<adjustment time> <offset> <adjustment time> <offset>

Structure:

```

typedef struct sdp_zone {
    uint64_t      z_time;
    char          *z_offset;
    struct sdp_zone *z_next;
} sdp_zone_t;

```

"z_time" represents the base time
"z_offset" represents the offset that will be added to "z_time" to determine session time. Mainly used for daylight saving time conversions.
"z_next" is a pointer to next zone field as there could be several "<adjustment-time> <offset>" pairs within a zone field.

4.13 Encryption Keys ("k=")*

Format: k=<method>
k=<method>:<encryption key>

Description: The "k=" field contains encryption keys.

Structure:

```
typedef struct sdp_key {  
    char      *k_method;  
    char      *k_enckey;  
} sdp_key_t;
```

"k_method" represents the key type
"k_enckey" holds the encryption key.

4.14 Attributes ("a=")

Format: a=<attribute>
a=<attribute>:<value>

Description: The "a=" field are primary means for extending SDP. They may be defined to be used as "session-level" attributes, "media-level" attributes, or both. There could be several such fields and hence this structure is defined to be a linked list.

Structure:

```
typedef struct sdp_attr {  
    char      *a_name;  
    char      *a_value;  
    struct sdp_attr *a_next;  
} sdp_attr_t;
```

"a_name" holds the name of the attribute
"a_value" holds the value of the attribute
"a_next" holds the pointer to the next attribute structure as there could be several attribute fields within SDP description.

4.15 Media Descriptions (“m=”)

Format: <media> <port>/<number of ports> <proto> <fmt> ...

Description: The "m=" field describes the media to be used for the session. There could be several such fields so this structure is defined to be a linked list

Structure:

```
typedef struct sdp_media {
    char          *m_name;
    uint_t       m_port;
    int          m_portcount;
    char          *m_proto;
    sdp_list_t   *m_format;
    char          *m_info;
    sdp_conn_t   *m_conn;
    sdp_bandwidth_t *m_bw;
    sdp_key_t    *m_key;
    sdp_attr_t   *m_attr;
    struct sdp_media *m_next;
    sdp_session_t *m_session;
} sdp_media_t;
```

"m_name" holds the name of the media like "audio", "video", "message" et al

"m_port" holds the transport layer port information

"m_portcount" holds number of ports in case of hierarchically encoded streams

"m_proto" holds the transport protocol

"m_format" holds the media format description

"m_next" is a pointer to next media structure as there could be several media sections in SDP description.

"m_session" is a pointer back to the session structure.

Each Media section inside SDP description could contain other fields too. They being information, connection, bandwidth, key and attribute.

5 Functions to generate SDP description

The developer gets a pointer to allocated session structure (`sdp_session_t`) using `sdp_new_session()` and populates the structure by calling field specific `sdp_add_<fieldname>(sdp_session_t, <field specific arguments>)` functions. Once the structure is populated he/she would pass that structure to `sdp_session_to_str()` which returns a pointer to a character buffer holding SDP information. In the case of SIP he/she would then add this character buffer into SIP message using `sip_add_content(sip_msg_t, char *)`. Then he needs to set the content type to "application/sdp". Check section 8.1 for `sdp.h`. Also section 8.2 contains an example, which captures a use case.

6 Public functions

This section describes functions that will be exported through SDP library. Note that this information is very preliminary and will change as the implementation takes form.

6.1 sdp_find_media()

```
sdp_media_t *sdp_find_media(sdp_media_t *media, char *name);
```

SDP info can have several media sections. For example,

```
m=audio 49230 RTP/AVP 96          #MediaSection - 1
a=rtpmap:96 L8/8000
m=video 51372 RTP/AVP 99         #MediaSection - 2
a=rtpmap:99 h263-1998/90000
m=audio 4980 RTP/AVP 97          #MediaSection - 3
a=rtpmap:97 L16/8000
```

Function: Given a media list and media name ("audio", "video", et al), it searches the list for that media. (case-insensitive search)

Arguments:

"media" - media list from session structure.
"name" - media name

Return Values:

NULL - If media not found or arguments were incorrect
sdp_media_t * - pointer to matched media in the list.

6.2 sdp_find_attr()

```
sdp_attr_t * sdp_find_attr(sdp_attr_t *attr, char *name);
```

SDP session/media section can have several attribute fields. For example

```
m=audio 49230 RTP/AVP 96 97 98
a=rtpmap:96 L8/8000
a=rtpmap:97 L16/8000
a=sendrecv
```

Function: Given a attribute list and name of the attribute ("rtpmap", "fmt", et al), this API searches the list for that attribute. (case-insensitive search)

Arguments:

"attr" - attribute list from media or session structure
"name" - name of the attribute to be searched

Return Values:

NULL - If attribute not found or arguments were incorrect
sdp_attr_t * - pointer to matched attribute in the list

6.3 sdp_get_media_rtpmap()

```
sdp_attr_t *sdp_find_media_rtpmap(sdp_media_t *media, char *format)
```

Function: Given a media structure and a format description, this API will return the rtpmap attribute matching that format description.

Arguments:

"media" - media section in SDP within which "rtpmap" attribute need to be searched
"format" - media format description

Return Values:

NULL - If rtpmap attribute not found or arguments were incorrect
sdp_attr_t * - pointer to the matched rtpmap attribute.

6.4 sdp_clone_session()

```
sdp_session_t *sdp_clone_session(const sdp_session_t *s);
```

Function: Given a session structure it clones (deep copy) and returns the cloned copy

Arguments:

"session" - represents the session structure to be cloned.

Return values:

NULL - If an error occurred during cloning
sdp_session_t * - pointer to copy of the passed session structure.

Note: user has to free the cloned session using sdp_free_session()

6.5 sdp_delete_all_field()

```
int sdp_delete_all_field(sdp_session_t *, char fieldname);
```

Function: Given a session structure and the field ('v', 'o', 's', et al), this API deletes the corresponding structure element. If there are multiple fields of same time, it deletes them too. It frees the memory and sets that pointer to NULL

Arguments:

session = session structure whose member needs to be deleted
field = SDP field that needs to be deleted.

Return Values:

0 = Success

EINVAL = session is NULL or unknown field type

6.6 sdp_delete_all_media_field()

```
int sdp_delete_all_media_field(sdp_media_t *media, char field)
```

Function: Given a media structure and the field ('i', 'b', 'c', et al), this API deletes the corresponding structure element. If there are multiple fields of same time then this API deletes them too. It frees the memory and sets the pointer to NULL.

Arguments:

media = media structure whose member needs to be deleted.

field = SDP field that needs to be deleted.

Return Values:

0 = Success

EINVAL = session is NULL or unknown field type

6.7 sdp_delete_attribute()

```
int sdp_delete_attribute(sdp_attr_t **a, sdp_attr_t *del);
```

Function: Given an attribute list and an attribute, this API deletes that attribute from the list. It frees the memory corresponding to that attribute.

Arguments:

l_attr = attribute list from which attribute needs to be removed

attr = attribute that needs to be removed from the list. This attribute is obtained using sdp_find_attribute().

Return Values:

0 = Success

EINVAL = *l_attr or attr is NULL

6.8 sdp_delete_media()

```
int sdp_delete_media(sdp_media_t **m, sdp_media_t *old, sdp_media_t *new);
```

Function: Given a media list and the media, this API deletes that media from the list. It frees the memory corresponding to that media.

Arguments:

`l_media` = media list from which media needs to be removed
`media` = media that needs to be removed from the list. This media is obtained using `sdp_find_media()`

Return Values:

0 = Success
EINVAL = `*l_media` or `media` is NULL

6.9 `sdp_new_session()`

`sdp_session_t *sdp_new_session()`

Function: Allocates memory for new SDP session structure and assigns a version number to it.

Arguments:

None.

Return values:

`sdp_session_t *` = pointer to allocated session structure
NULL = if memory allocation failed or version was incorrect.

6.10 `sdp_add_origin()`

`int sdp_add_origin(sdp_session_t *session, char *name, uint64_t id, uint64_t ver, char *nettype, char *addrtype, char *address)`

Function: Adds origin field to the session structure.

Arguments:

See the definition of `sdp_origin_t` structure above.

Return Values:

0 = Success.
EINVAL = if arguments are NULL
ENOMEM = failed memory allocation
EPROTO = if origin structure is already part of session, as there can be only one origin field in a SDP description.

6.11 `sdp_add_name()`

`int sdp_add_name(sdp_session_t *session, char *name)`

Function: Adds session name field to the session.

Arguments:

"name" points to a character buffer containing session name.

Return Values:

0 = Success.

EINVAL = if arguments are NULL

ENOMEM = failed memory allocation

EPROTO = if name is already assigned to session, as there can be only one name field.

6.12 sdp_add_information()

```
int sdp_add_information(char **information, char *value)
```

Function: Adds session information field to the session or media section of SDP

Arguments:

"value" points to a character buffer containing info

Return Values:

0 = Success.

EINVAL = if arguments are NULL

ENOMEM = failed memory allocation

EPROTO = if information is already assigned to session or media, as there can be only one information field.

6.13 sdp_add_uri()

```
int sdp_add_uri(sdp_session_t *session, char *uri)
```

Function: Adds uri field to the session.

Arguments:

"uri" - points to a character buffer containing uri

Return Values:

0 = Success.

EINVAL = if arguments are NULL

ENOMEM = failed memory allocation

EPROTO = if uri is already assigned to session, as there can be only one uri field.

6.14 sdp_add_email()

```
int sdp_add_email(sdp_session_t *session, char *email)
```

Function: Adds email field to the session.

Arguments:

"email" - points to a character buffer containing email

Return Values:

0 = Success.
EINVAL = if arguments are NULL
ENOMEM = failed memory allocation

6.15 sdp_add_phone()

```
int sdp_add_phone(sdp_session_t *session, char *phone)
```

Function: Adds phone field to the session

Arguments:

"phone" - points to a character buffer containing phone

Return Values:

0 = Success.
EINVAL = if arguments are NULL
ENOMEM = failed memory allocation

6.16 sdp_add_connection()

```
int sdp_add_connection(sdp_conn_t **conn, char *nettype, char *addrtype, char *address, uint8_t ttl, int addrcount)
```

Function: Adds connection field to the session or media section of SDP

Arguments:

Please see the sdp_conn_t structure above.

Return Values:

0 = Success.
EINVAL = if arguments are NULL
ENOMEM = failed memory allocation

6.17 sdp_add_bandwidth()

```
int sdp_add_bandwidth(sdp_bandwidth_t **bw, char *type, uint64_t value)
```

Function: Adds bandwidth field to the session or media section of SDP

Arguments:

Please see the sdp_bandwidth_t structure above.

Return Values:

0 = Success.
EINVAL = if arguments are NULL
ENOMEM = failed memory allocation

6.18 sdp_add_repeat()

```
int sdp_add_repeat(sdp_time_t *time, uint64_t *interval, uint64_t
*duration, char *offset)
```

Function: Adds repeat field to the time structure of session

Arguments:

Please see the sdp_repeat_t structure above.

Return Values:

0 = Success.
EINVAL = if arguments are NULL
ENOMEM = failed memory allocation

6.19 sdp_add_time()

```
int sdp_add_time(sdp_session_t *session, uint64_t starttime,
uint64_t stoptime, sdp_time_t **time)
```

Function: Adds time field to the session

Arguments:

Please see the sdp_time_t structure above.
sdp_time_t * = pointer to the time structure just created. This is needed by API
sdp_add_repeat(). It will be set to NULL if API fails

Return Values:

0 = Success.
EINVAL = if arguments are NULL
ENOMEM = failed memory allocation

6.20 sdp_add_zone()

```
int sdp_add_zone(sdp_session_t *session, uint64_t time, char *offset)
```

Function: Adds time zone field to the session

Arguments:

Please see the sdp_zone_t structure above.

Return Values:

0 = Success.
EINVAL = if arguments are NULL
ENOMEM = failed memory allocation

6.21 sdp_add_key()

```
int sdp_add_key(sdp_key_t **key, char *method, char *enckey)
```

Function: Adds key field to session or media section of SDP.

Arguments:

Please see the sdp_key_t structure above.

Return Values:

0 = Success
EINVAL = if arguments are NULL
ENOMEM = failed memory allocation
EPROTO = if more than one key is being added to session, as there can be only one key field in SDP

6.22 sdp_add_attribute()

```
int sdp_add_attribute(sdp_attr_t **attr, char *name, char *value)
```

Function: Adds attribute field to session or media section of SDP

Arguments:

Please see the sdp_attr_t structure above.

Return Values:

0 = Success
EINVAL = if arguments are NULL
ENOMEM = failed memory allocation

6.23 sdp_add_media()

```
int sdp_add_media(sdp_session_t *session, char *name, int port, int portcount, char *protocol, char *format, sdp_media_t **)
```

Function: Adds media field to the session.

Arguments:

please see the sdp_media_t structure above.
sdp_media_t * = pointer to the media structure just created. To this user can further add other media specific fields like connection, bandwidth et al

will be set to NULL, if the API fails.

Return Values:

0 = Success
EINVAL = if arguments are NULL
ENOMEM = failed memory allocation

6.24 sdp_session_to_str()

```
char* sdp_session_to_str(sdp_session_t *, int *error);
```

Function: Given a session structure, this API converts it into byte-string which will be used as a payload later on.

Arguments:

session = pointer to a session structure which needs to be converted into byte-string
error = if error != NULL then it will be 0 (no failure) or will be EINVAL or ENOMEM, after returning from function.

Return Values:

NULL = if session was NULL
char * = pointer to the character buffer containing structure contents.

6.25 sdp_free_session()

```
void sdp_free_session(sdp_session_t *);
```

Given a sdp_session_t this API frees the memory allocated for it.

7 Impact on sip.h & libsip.so.1

All the aforementioned data structures and API's will be exported through a separate header file <sdp.h> and yet to be determined library. Therefore there is NO impact whatsoever on either sip.h or libsip.so.1.

8 Appendix

8.1 <sdp.h>

The <sdp.h> header file will contain all the new data structures and function declarations described in section 5 and 6. Note that this information is very preliminary and will change as the implementation takes form.

```

#define SDP_CRLF          "\r\n"
#define SDP_LF           "\n"
#define SDP_SKIP_CRLF(msg_ptr) ((msg_ptr) = (msg_ptr) + 2)
#define SDP_SKIP_LF(msg_ptr)  ((msg_ptr) = (msg_ptr) + 1)

#define SDP_VERSION_FIELD 'v'
#define SDP_ORIGIN_FIELD  'o'
#define SDP_NAME_FIELD    's'
#define SDP_INFO_FIELD    'i'
#define SDP_URI_FIELD     'u'
#define SDP_EMAIL_FIELD   'e'
#define SDP_PHONE_FIELD   'p'
#define SDP_CONNECTION_FIELD 'c'
#define SDP_BANDWIDTH_FIELD 'b'
#define SDP_TIME_FIELD    't'
#define SDP_REPEAT_FIELD  'r'
#define SDP_ZONE_FIELD    'z'
#define SDP_KEY_FIELD     'k'
#define SDP_ATTRIBUTE_FIELD 'a'
#define SDP_MEDIA_FIELD   'm'

/* SDP Parse errors */
#define SDP_VERSION_ERROR      0x00000001
#define SDP_ORIGIN_ERROR      0x00000002
#define SDP_NAME_ERROR        0x00000004
#define SDP_INFO_ERROR        0x00000008
#define SDP_URI_ERROR         0x00000010
#define SDP_EMAIL_ERROR       0x00000020
#define SDP_PHONE_ERROR       0x00000040
#define SDP_CONNECTION_ERROR  0x00000080
#define SDP_BANDWIDTH_ERROR   0x00000100
#define SDP_TIME_ERROR        0x00000200
#define SDP_REPEAT_TIME_ERROR 0x00000400
#define SDP_ZONE_ERROR        0x00000800
#define SDP_KEY_ERROR         0x00001000
#define SDP_ATTRIBUTE_ERROR   0x00002000
#define SDP_MEDIA_ERROR       0x00004000
#define SDP_FIELDS_ORDER_ERROR 0x00008000
#define SDP_MISSING_FIELDS   0x00010000

#define SDP_AUDIO            "audio"
#define SDP_VIDEO            "video"
#define SDP_TEXT             "text"
#define SDP_APPLICATION      "application"
#define SDP_MESSAGE          "message"

```

```

#define SDP RTPMAP                "rtpmap"

#define SDP_SESSION_VERSION_1    1

typedef struct sdp_list {
    void                *value;
    struct sdp_list    *next;
} sdp_list_t;

/*
 * SDP origin field.
 * o=<username> <sess-id> <sess-version> <nettype> <addrtype> <address>
 */
typedef struct sdp_origin {
    char                *o_username;
    uint64_t            o_id;
    uint64_t            o_version;
    char                *o_nettype;
    char                *o_addrtype;
    char                *o_address;
} sdp_origin_t;

/*
 * SDP connection field.
 * c=<nettype> <addrtype> <connection-address>[/ttl]/<number addresses>
 */
typedef struct sdp_conn {
    char                *c_nettype;
    char                *c_addrtype;
    char                *c_address;
    int                 c_addrcount;
    struct sdp_conn    *c_next;
    uint8_t             c_ttl;
} sdp_conn_t;

/*
 * SDP repeat field. Always found in time structure.
 * r=<repeat interval> <active duration> <offsets from start-time>
 */
typedef struct sdp_repeat {
    uint64_t            r_interval;
    uint64_t            r_duration;
    sdp_list_t         *r_offset;
    struct sdp_repeat  *r_next;
} sdp_repeat_t;

/*
 * SDP time field.

```

```

    * t=<start-time> <stop-time>
    */
typedef struct sdp_time {
    uint64_t          t_start;
    uint64_t          t_stop;
    sdp_repeat_t      *t_repeat;
    struct sdp_time   *t_next;
} sdp_time_t;

/*
 * SDP time zone field.
 * z=<adjustment time> <offset> <adjustment time> <offset> ....
 */
typedef struct sdp_zone {
    uint64_t          z_time;
    char              *z_offset;
    struct sdp_zone   *z_next;
} sdp_zone_t;

/*
 * SDP attribute field.
 * a=<attribute> or a=<attribute>:<value>
 */
typedef struct sdp_attr {
    char              *a_name;
    char              *a_value;
    struct sdp_attr   *a_next;
} sdp_attr_t;

/*
 * SDP bandwidth field.
 * b=<bwtype>:<bandwidth>
 */
typedef struct sdp_bandwidth {
    char              *b_type;
    uint64_t          b_value;
    struct sdp_bandwidth *b_next;
} sdp_bandwidth_t;

/*
 * SDP key field to session or media section of SDP.
 * k=<method> or k=<method>:<encryption key>
 */
typedef struct sdp_key {
    char              *k_method;
    char              *k_enckey;
} sdp_key_t;

```

```

typedef struct sdp_session      sdp_session_t;

/*
 * SDP media section, contains media fields and other fields within
 * media section.
 * m=<media> <port>[/portcount] <proto> <fmt> ...
 */
typedef struct sdp_media {
    char          *m_name;
    int           m_port;
    int           m_portcount;
    char          *m_proto;
    sdp_list_t    *m_format;
    char          *m_info;
    sdp_conn_t    *m_conn;
    sdp_bandwidth_t *m_bw;
    sdp_key_t     *m_key;
    sdp_attr_t    *m_attr;
    struct sdp_media *m_next;
    sdp_session_t *m_session;
} sdp_media_t;

struct sdp_session {
    int           sdp_session_version;
    int           s_version;
    sdp_origin_t *s_origin;
    char          *s_name;
    char          *s_info;
    char          *s_uri;
    sdp_list_t    *s_email;
    sdp_list_t    *s_phone;
    sdp_conn_t    *s_conn;
    sdp_bandwidth_t *s_bw;
    sdp_time_t    *s_time;
    sdp_zone_t    *s_zone;
    sdp_key_t     *s_key;
    sdp_attr_t    *s_attr;
    sdp_media_t   *s_media;
};

extern int  sdp_parse(const char *, int, int, sdp_session_t **, int *);
extern sdp_media_t *sdp_find_media(sdp_media_t *, char *);
extern sdp_attr_t *sdp_find_attribute(sdp_attr_t *, char *);
extern sdp_attr_t *sdp_find_media_rtpmap(sdp_media_t *, char *);
extern sdp_session_t *sdp_clone_session(const sdp_session_t *);
extern sdp_session_t *sdp_new_session();
extern int sdp_add_origin(sdp_session_t *, char *, uint64_t, uint64_t,
                          char *, char *, char *);

```

```

extern int          sdp_add_name(sdp_session_t *, char *);
extern int          sdp_add_information(char **, char *);
extern int          sdp_add_uri(sdp_session_t *, char *);
extern int          sdp_add_email(sdp_session_t *, char *);
extern int          sdp_add_phone(sdp_session_t *, char *);
extern int          sdp_add_connection(sdp_conn_t **, char *,
char *, char *, uint8_t, int);
extern int          sdp_add_bandwidth(sdp_bandwidth_t **, char *,
uint64_t);
extern int          sdp_add_repeat(sdp_time_t *, char *, char *,
char *);
extern int          sdp_add_time(sdp_session_t *, uint64_t,
uint64_t, sdp_time_t **);
extern int          sdp_add_zone(sdp_session_t *, uint64_t, char *);
extern int          sdp_add_key(sdp_key_t **, char *, char *);
extern int          sdp_add_attribute(sdp_attr_t **, char *, char *);
extern int          *sdp_add_media(sdp_session_t *, char *, uint_t,
int, char *, char *, sdp_media_t **);
extern int          sdp_delete_all_field(sdp_session_t *, char);
extern int          sdp_delete_all_media_field(sdp_media_t *, char);
extern int          sdp_delete_media(sdp_media_t **, sdp_media_t *);
extern int          sdp_delete_attribute(sdp_attr_t **, sdp_attr_t *);
extern void         sdp_free_session(sdp_session_t *);
extern char         *sdp_session_to_str(const sdp_session_t *, int *);

```

8.2 Example usage scenario ()

8.2.1 Normal case

```

#include <stdio.h>
#include <string.h>
#include <sdp.h>

/* SDP Message, we will be using in request
"v=0\r\n\
o=Alice 2890844526 2890842807 IN IP4 10.47.16.5\r\n\
s=-\r\n\
i=A Seminar on the session description protocol\r\n\
u=http://www.example.com/seminars/sdp.pdf\r\n\
e=alice@example.com (Alice smith)\r\n\
p=+1 911-345-1160\r\n\
c=IN IP4 10.47.16.5\r\n\
b=CT:1024\r\n\
t=2854678930 2854679000\r\n\
r=604800 3600 0 90000\r\n\
z=2882844526 -1h 2898848070 0h\r\n\
a=recvonly\r\n\
m=audio 49170 RTP/AVP 0\r\n\
i=audio media\r\n\
b=CT:1000\r\n\
k=prompt\r\n\
m=video 51372 RTP/AVP 99 90\r\n\

```

```

i=video media\r\n\
a=rtpmap:99 h232-199/90000\r\n\
a=rtpmap:90 h263-1998/90000\r\n";
*/

/* SDP Message, we will be using in response
"v=0\r\n\
o=Bob 2890844536 2890844540 IN IP4 10.47.16.6\r\n\
s=-\r\n\
i=A Seminar on the session description protocol\r\n\
u=http://www.example.com/seminars/sdp.pdf\r\n\
e=bob@example.com (Bob Doe)\r\n\
p=+1 612-802-7489\r\n\
c=IN IP4 10.47.16.6\r\n\
b=CT:1024\r\n\
t=2854678930 2854679000\r\n\
r=604800 3600 0 90000\r\n\
m=audio 49270 RTP/AVP 0\r\n\
i=audio media\r\n\
m=video 51572 RTP/AVP 99 90\r\n\
i=video media\r\n\
a=rtpmap:99 h232-199/90000\r\n\
a=rtpmap:90 h263-1998/90000\r\n";
*/
int main() {
    int error = 0;                /* indicates EINVAL or ENOMEM
                                situation, while parsing */

    int parse_error = 0;        /* indicates field(s) for which
                                parsing error exists */

    int flags = 0;              /* Not used now */
    sdp_session_t *my_sess;     /* constructed session */
    sdp_media_t *my_media;
    sdp_time_t *my_time;
    char *b_sdp;
    sdp_session_t *p_sess;      /* parsed session */

    my_sess = sdp_new_session();
    if (my_sess == NULL) {
        return (ENOMEM);
    }
    my_sess->version = 0;
    if (sdp_add_name(my_sess, "-") != 0)
        goto err_ret;
    if (sdp_add_origin(my_sess, "Alice", 2890844526ULL, 2890842807ULL, "IN", "IP4", "10.47.16.5")
        != 0)
        goto err_ret;
    if (sdp_add_information(&my_sess->info, "A Seminar on the session description protocol") != 0)
        goto err_ret;
    if (sdp_add_uri(my_sess, "http://www.example.com/seminars/sdp.pdf") != 0)
        goto err_ret;
    if (sdp_add_email(my_sess, "alice@example.com (Alice smith)") != 0)
        goto err_ret;
    if (sdp_add_phone(my_sess, "+1 911-345-1160") != 0)
        goto err_ret;
    if (sdp_add_connection(&my_sess->connection, "IN", "IP4", "10.47.16.5", 0, 0) != 0)
        goto err_ret;
}

```

```

if (sdp_add_bandwidth(&my_sess->bandwidth, "CT", 1024) != 0)
    goto err_ret;
if ( sdp_add_time(my_sess, 2854678930ULL, 2854679000ULL, &my_time) != 0)
    goto err_ret;
if (sdp_add_repeat(my_time, "604800", "3600", "0 90000") != 0)
    goto err_ret;
if (sdp_add_zone(my_sess, 2882844526ULL, "-1h") != 0)
    goto err_ret;
if (sdp_add_zone(my_sess, 2898848070ULL, "0h") != 0)
    goto err_ret;
if (sdp_add_attribute(&my_sess->attribute, "sendrecv", NULL) != 0)
    goto err_ret;
if ( sdp_add_media(my_sess, "audio", 49170, 1, "RTP/AVP", "0", &my_media) != 0)
    goto err_ret;
if (sdp_add_information(&my_media->info, "audio media") != 0)
    goto err_ret;
if (sdp_add_bandwidth(&my_media->bandwidth, "CT", 1000) != 0)
    goto err_ret;
if (sdp_add_key(&my_media->key, "prompt", NULL) != 0)
    goto err_ret;
if ( sdp_add_media(my_sess, "video", 51732, 1, "RTP/AVP", "99 90", &my_media) != 0)
    goto err_ret;
if (sdp_add_information(&my_media->info, "video media") != 0)
    goto err_ret;
if (sdp_add_attribute(&my_media->attribute, "rtpmap", "99 h232-199/90000") != 0)
    goto err_ret;
if (sdp_add_attribute(&my_media->attribute, "rtpmap", "90 h263-1998/90000") != 0)
    goto err_ret;
b_sdp = sdp_session_to_str(my_sess, &error);

/*
 * Now send the above character buffer to destination
 */
printf("Sending message:\n%s\n", b_sdp);
/*
 * Now parse the recieved buffer.
 */
if(sdp_parse(b_sdp, strlen(b_sdp), flags, &p_sess, &parse_error) != 0)
    goto parse_err_ret;
/*
 * Note we parse other fields even if some fields have error. The user
 * can determine if he wants to use other fields. Using flags (future)
 * we could change this behavior.
 */
if (parse_error != 0)
    printf("Parsing error exists: %d\n", parse_error);
/*
 * we have now the SDP info in p_sess. Application uses it as
 * required i.e. retrieves the values from the structures et al.
 */

/*
 * Clone's the session and modify's the structure OR directly modify's
 * the above structure.
 */
sdp_delete_all_field(p_sess, SDP_ORIGIN_FIELD);

```

```

if (sdp_add_origin(p_sess, "Bob", 2890844536ULL, 2890844540ULL, "IN",
    "IP4", "10.47.16.6") != 0)
    goto parse_err_ret;
sdp_delete_all_field(p_sess, SDP_EMAIL_FIELD);
if (sdp_add_email(p_sess, "bob@example.com (Bob Doe)") != 0)
    goto parse_err_ret;
sdp_delete_all_field(p_sess, SDP_CONNECTION_FIELD);
if (sdp_add_connection(&p_sess->connection, "IN", "IP4", "10.47.16.6",0, 0) != 0)
    goto parse_err_ret;
/* to change media ports you could do this */
if ((my_media = sdp_find_media(p_sess->media, SDP_AUDIO)) == NULL)
    goto parse_err_ret;
my_media->port = 49270;
if ((my_media = sdp_find_media(p_sess->media, SDP_VIDEO)) == NULL)
    goto parse_err_ret;
my_media->port = 51572;
/*
 * on the other hand if user wants to add media or delete media he can
 * do so with respective sdp_add_media or sdp_delete_media functions
 */
/* Now we have the structure. convert it to string and send */
free(b_sdp);
b_sdp = sdp_session_to_str(p_sess, &error);
printf("sending the following response:\n%s\n", b_sdp);
return (0);

err_ret:
    sdp_free_session(my_sess);
    printf("Error while adding\n");
    return (1);

parse_err_ret:
    free(b_sdp);
    sdp_free_session(p_sess);
    sdp_free_session(my_sess);
    printf("Error while parsing the recieved buffer\n");
    return (1);
}

```

8.2.2 SIP usecase

Alice sends Bob an INVITE message with SDP info. Bob replies back to Alice with 200 OK and negotiated SDP info.

Alice's INVITE message to Bob contains following SDP info: (UAC)

```

v=0
o=alice 2890844526 2890844526 IN IP4 host.anywhere.com
s=Daily conference
c=IN IP4 host.anywhere.com
t=0 0
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000
m=video 51372 RTP/AVP 31
a=rtpmap:31 H261/90000

```

```
m=video 53000 RTP/AVP 32
a=rtpmap:32 MPV/90000
```

Bob's Response message (200 OK) to Alice contains following SDP info: (UAS)

```
v=0
o=bob 2890844730 2890844730 IN IP4 host.example.com
s=Daily Conference
c=IN IP4 host.example.com
t=0 0
m=audio 49920 RTP/AVP 0
a=rtpmap:0 PCMU/8000
m=video 0 RTP/AVP 31
m=video 53000 RTP/AVP 32
a=rtpmap:32 MPV/90000
```

/ Assuming sip_msg is the request delivered by the stack, the SDP parsing at Bob is as shown below*/*

```
sip_str_t *sdp_info;
sdp_session_t *session;

if ( strcmp(sip_get_content_sub_type(sip_msg, &error), "sdp") == 0) {
    sdp_info = sip_get_content(sip_msg, &error)
    error =sdp_parse(sdp_info->sip_str_ptr, sdp_info->sip_str_len, 0,
                    &session, &perror);
    if (error != 0) {
        /* This error is due to EINVAL, ENOMEM et al. */
        goto error_exit;
    }
    if (perror != 0) {
        /* we have parse error so check which field and take decision */
    }
} else {
    return; // SDP info not present.
}
}
```

/ At this point we have access to sdp_session_t and it's members contains SDP field info. Bob can determine which SDP attributes he wants to keep or change (negotiation) and accordingly send appropriate SDP info back to Alice.*

```
*/
/* Generating SDP descriptions to be sent with SIP message */
/* Create a SIP response (200 OK) using existing library function */
if ((sip_msg_resp = sip_create_response()) == NULL)
    goto error_exit;
sdp_session_t *new_session = sdp_clone_session(session, &error);

// delete unwanted fields using sdp_delete_field() API
// add the new fields using sdp_add_xxxxx() API's
// convert session structure to character buffer

char *sdp_info = sdp_session_to_str(new_session, &error);
sip_add_content(sip_msg_resp, sdp_info);

// Message with SDP description successfully created, send the new sip // message;
```

9 Other open source SDP library

We looked at other open source SDP library to see if we could use them. The two common problems in using any of them are:

- (a) Most of them are packaged as part of a SIP library and they were closely tied with SIP library. So to just use the SDP module we might require the corresponding SIP module or other libraries as some of the data structures and helper functions in SDP module comes from those libraries. So we might have to include those SIP modules, which does not make sense as we already have our own SIP library.
- (b) No flexibility in naming convention and design. Some of the library had the project name at the beginning of every function. Having our own gives us the flexibility of designing in a way that adheres to our existing API's naming convention and designs.
- (c) All the libraries do not provide means to generate SDP description. Our SDP library is unique that way and provides API's to build/construct SDP description.

10 References

[RFC 4566] "SDP: Session Description Protocol", M. Handley, V. Jacobson and C. Perkins, July 2006.

[RFC 3261] "SIP: Session Initiation Protocol", J. Rosenberg, H. Schulzrinne, et al, June 2002.

[RFC 3264] "An Offer/Answer Model with SDP", Rosenberg J. and H. Schulzrinne, June 2001.

[RFC 4234] "Augmented BNF for Syntax Specifications: ABNF", D. Crocker and P. Overell, October 2005.